Addressing the Computational Cost of the Immersed Boundary Method through Multi-Implicit and Multi-Rate Strategies with Time Parallelism

Lauren E Fovargue

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Mathematics.

Chapel Hill
2012

Approved by:

Michael L Minion

Laura A Miller

Jingfang Huang

Jan F Prins

M. Gregory Forest

Abstract

LAUREN E FOVARGUE: Addressing the Computational Cost of the Immersed Boundary Method through Multi-Implicit and Multi-Rate Strategies with Time Parallelism

(Under the direction of Michael L Minion)

Many problems in biological fluid-structure interaction have been studied with Peskin's *immersed boundary method* (IBM). This method defines a relatively simple mathematical modeling framework, and allows for the use of standard fluid solvers. However, when evaluated computationally in many applications, a very small time step is required. This time step is not restricted by accuracy, but stability, causing the temporal problem to be stiff. Previous attempts to address the stability restriction of IBM use semi or fully implicit schemes that require the code, including the fluid solver, to be rewritten, and these have yet to be implemented in application focused studies.

Here, new ideas for addressing the computational cost of IBM are presented, which rely on a novel method for splitting the spatial field into stiff and non-stiff components. With this splitting, the impact on the largest stable time step and computational cost for stiff problems is investigated through multi-implicit, multirate and time parallel techniques. All of the algorithms presented here improve upon the stability restriction of IBM, whether with a larger stable time step or overall speedup. In addition, they are focused on the treatment of the immersed boundary and no changes to the fluid solver are necessary, making them more accessible to applications.

ACKNOWLEDGMENTS

I want to thank my wonderful husband Dan for his constant support and reassurance and for building a life with me, without your help, both in math and at home, I wouldn't have made it to this point. Thank you for being the strongest part of me.

Thank you to Dr. Minion who was a great advisor and mentor, your guidance and help has taken me from a clueless graduate student to a confident novice scientist. Thank you for not giving up on me and pushing me to take my own research by the horns, I can't express how valuable that will be to my future. Don't worry, I will continue to push for SDC and have an answer to "you can't do parallel in time". Thank you so much for all your time and effort over the last few years, I will always be proud to be one of Minion's minions.

Thank you to Laura Miller, for being a mentor, travel companion and most of all, friend. Thank you for showing me the beauty of biology, which ignited my interest in it, and being a constant source of encouragement, I couldn't have done it without you.

Thank you to all my professors here at UNC, especially those who taught computational classes. Dr. Minion, Dr. Miller, Dr. Huang, Dr. Adalsteinsson, and Dr. Mitran. You have inspired a pure math background into a love of numerics and for that I cannot thank you enough.

A big thank you to my window for getting me through my 4th and 5th years and my 4th year office mate Christy Hamlet, who shall always hold the title of office genius. Thank you to Brandyn Lee, my 5th year office mate, whose brilliance is (hopefully) contagious and stories priceless. Thank you to Emily Braley, for being a workout partner,

support network and best friend rolled into one person. For all our walks, fun times, and library time, I cant thank you enough. Finally, thank you to my wonderful supportive family, my extended family, and my network of friends, the most amazing group of people I could have asked for, for keeping me sane the last 5 years. UNC and Chapel Hill, with its people and spirit will forever be revered as a magical place where my life truly began and blossomed.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

There are two categories of interaction for dynamic systems that involve a viscous incompressible fluid and a flexible, dynamic object, one-way and two-way coupling. A system is considered to have a one-way coupling when the dynamics of the fluid or object are prescribed and unaffected by the others motion. Two way coupling occurs when the motion of the fluid and object are mutually dependent on each other, creating a coupled system of equations that are difficult to extract information from, both analytically and numerically. Here the focus is on two-way coupling, specifically problems of biological fluid-structure interaction (FSI), where an elastic, biological structure interacts with a viscous incompressible fluid.

This introduction aims to motivate this class of problems and provide an overview of a particular framework for modeling such problems. This framework, the *immersed boundary method*, is a popular method for studying biological FSI problems, however when discretized, it can become very computationally expensive. It has been well documented that the time step of this method for many applications has to be small for the purposes of stability, not accuracy, causing it to be computationally expensive.

Although some attempts have been made to address the small time step needed for IBM, widespread use with applications are not yet seen. Thus there is still a need for techniques which address the computational bottlenecks seen in IBM without large rewrites of existing IB implementations. The purpose of thesis is to present new ways of addressing this computational expense, using techniques that are more accessible to applications. This introduction will conclude by describing this purpose in more detail, and mapping out the progression of this thesis.

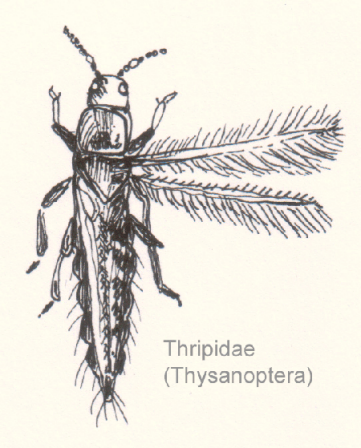


FIGURE 1.1. *Thysanoptera*, or a thrip, is a tiny wasp [**47**].

1.1. Motivation

Although the application of mathematics to biology can be dated back at least to the days of Gregor Mendel (1822-1884), recently there has been a surge in the field of mathematical biology [**52**]. From biologically inspired design, to cancer research, to improving medical procedures, mathematics and computation have helped advance biological research. Not only does using mathematics answer fundamental questions, aiding the direction of biological research, but it advances the field of mathematics, as new techniques are being developed to tackle such problems [**52**]. Here, contributions to the field of mathematical biology are made by expanding upon current techniques in computational biological fluid (biofluid) dynamics.

There are many questions in biofluid dynamics, such as 'how do tiny insects fly?', 'what happens in the endothelial surface layer', and 'what is the optimal design for a mitral valve replacement?', and computational techniques have helped to provide some understanding [**69, 67, 42**].

Consider three examples, shown in Figures 1.1, 1.2, 1.3. In each of these cases, there is a coupled dynamic between a viscous incompressible fluid and an elastic biological structure. The first figure (1.1) shows the thrip, a small wasp believed to achieve lift based on the unique structure and Weis-Fogh, "clap and fling", motion of its wings [**67, 68**]. The second figure, Fig. 1.2, shows a schematic of a heart, highlighting a mitral

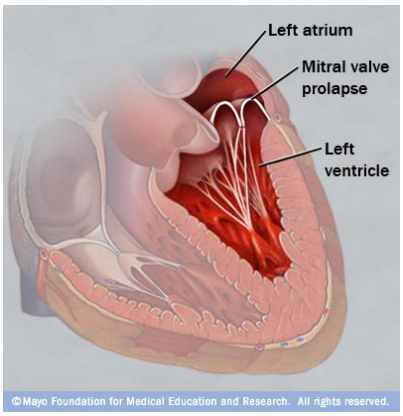


FIGURE 1.2. Heart with prosthetic mitral valve [**35**].

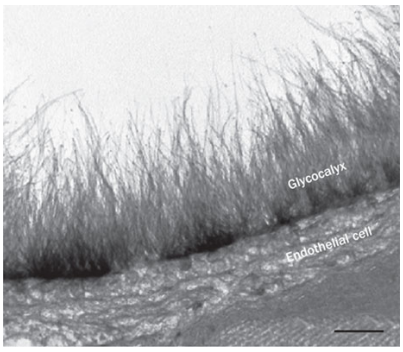


FIGURE 1.3. Glycocalyx in a rat capillary [**75**].

valve prolapse, where the leaflets of the heart fall back into the left atrium. This condition may be treated with an artificial replacement of this valve, the optimal design for which has been studied mathematically [**42**]. Finally, Figure 1.3 shows the endothelial surface layer, or glycocalyx, in a rat capillary. This hair-like structure lines the lumen, or inner region, of capillaries and is widely believed to be a mechanotransducer, that is, it converts mechanical hemodynamic stress into a chemical response [**70**].

In each of these three examples, there are thin elastic structures (e.g. the wing on the thrip, the 'strings' on the mitral valve and the glycocalyx itself) with complicated geometry, interacting with a fluid. Fluid behavior can be categorized by the value of the Reynolds number, a dimensionless ratio of inertial forces to viscous forces. It is defined as

$$Re = \frac{\rho u L}{\mu},$$

where u is a characteristic velocity, L is a characteristic length scale, ρ the density of the fluid and μ the dynamic viscosity. This definition allows for dynamic similarity, i.e. flows categorized by the same Reynolds number behave in the same way. For example, when viscous forces are increasingly dominant and inertial forces less noticeable, the Re becomes small. This is sometimes mathematically modeled by using the Stokes equations, which neglect the inertial terms. However, fluid where both viscous and inertial forces cannot be ignored, can be modeled with the Navier-Stokes equations. Applications in this fluid regime are targeted here.

Creating a computational simulation for particular FSI problems may requires many discretization points and high resolution, becoming highly computationally expensive. This overwhelming cost leads to the utilization of simplified studies, without the full complexity of the geometry [**59, 70, 68, 67, 69**]. Although these studies are a necessary first pass to looking at the underlying biological phenomena, more robust simulations could help to further understanding.

1.2. The Immersed Boundary Model

The immersed boundary method (IBM) is a framework to model and simulate biological FSI problems that has been used in many applications, from hearts to insects [**79, 69**]. Its popularity with application studies can be pointed to the 'simplicity' of the framework and the ability to the use standard fluid solvers. However, there are well known computational bottlenecks with IBM, as the time step required for numerical stability can be very small. That is, the problem becomes stiff, a term meaning that the time step is restricted not by accuracy but stability. This has inspired semi and fully implicit implementations of IBM, to increase the largest stable time step [**85, 61, 74, 50, 58, 15**]. Most of these methods, though they may achieve a larger stable time step, are more computationally expensive per time step, resulting in little gain in overall run time. Additionally, to date, there has been virtually no use of these methods in applications, which is likely a result of the work it takes to implement these solutions.

In the literature, there tends to be a division between those who work on the algorithms for IBM and those that apply the IBM framework to get biologically significant results, with some important exceptions [**81, 62, 42**]. In the case of the latter, there is more emphasis on the biological modeling and results, with less focus on involved computing. Once one has implemented the IB framework, significant changes do not need to be made to study different applications, and in particular no changes need to be made to the fluid solver. Thus it may not be efficient to spend time, for example, implementing a nonlinear solver or preconditioning for a moderate run time reduction. Additionally, many of these semi and fully implicit techniques require an entire rewrite of the IBM code, including the fluid solver, and the time it takes to do so may not be worth it to those interested in application. There does not appear, to date, to be a sufficient approach that can reduce simulation run time and not require great changes to existing IBM code.

1.3. Thesis statement

What follows is an investigation into effective ways of addressing the large computational cost of the IBM framework without making changes to the fluid solver, providing algorithms that are accessible for applications. It is known that the source of stiffness in the IBM is due to forces generated on the immersed boundary, meaning it can be computationally addressed separately from the fluid solver. Here new ideas on how to handle the stiffness of IBM problems is presented, based on a new splitting the spatial field into stiff and non-stiff parts, coupled with the use of mutli-rate and multi-implicit strategies. In addition, time parallelization will be considered to address the computational cost of IBM applications. Although the ideas presented here are not trivial to implement, they can be used with existing fluid solvers, eliminating the need to rewrite this part of the code. Each of the proposed new ideas are improvements over current techniques, whether it be in the largest stable time step allowed, the amount of numerical accuracy gained from a given time step, or run time savings.

Spatial splitting, multi-rate, multi-implicit methods, and time parallelization have been used in other types of stiff problems, and the success of these implementations is what motivated the work presented here. Bouzarth et. al. split the spatial field for FSI problems where viscous forces overpower inertial forces, known as the Stokes regime, and employed multirate methods to increase the efficiency of the *method of regularized Stokeslets* [**8**]. Effectively, the work here is an extension of the ideas of Bouzarth, to flow where inertial forces cannot be ignored and the use of IBM, as well as multi-implicit and time parallelism considerations. Other studies, such as Bourlouix et. al. found that using a multi-implicit technique for advection-diffusion-reaction equations yielded a computational advantage when the reaction term was significantly stiffer than the other parts of the equation [**7**]. Time parallelization with the *parallel full approximation scheme in space and time (PFASST)* has shown better convergence and lower error, although without increased efficiency, in the work of Emmett and Minion, where the Viscous Burgers equation and the Kuramoto-Silvashinky equation were considered [**29**]. Potenetially, a combination of these methods is promising to reducing computational time while keeping error low. The critical common denominator in all of these works is the use of *spectral deferred corrections (SDC)*, a stable, high order method for time integration. Thus it is employed here as well, and details about this choice and the particular benefits of using this method with the IB model will be addressed in Chapter 3.

The new algorithms and implementations will be presented after first establishing necessary background material. The details of the IB model and relevant variations are presented first in Chapter 2, followed by a discussion of SDC in Chapter 3. Chapter 4 begins the new material with a discussion of splitting of the spatial field. This is followed by Chapters 5 and 6 and 7, where the new multi-implicit, multi-rate and time parallel implementations are presented, respectively. This discussion will include numerical efficiency tests, discussions on convergence and accuracy, as well as a comparison to the existing techniques, done with a representative two dimensional test problem. Finally, a discussion of these approaches is found in Chapter 8.

CHAPTER 2

The Immersed Boundary Method and Variants

In general, fluid structure interaction (FSI) problems are difficult to study computationally because of the mutually dependent motion of the fluid and structure. The *immersed boundary method (IBM)* is a popular and relatively simple mathematical framework used to study such problems. IBM is also the basis for the *blob projection method (BPM)*, which is utilized in subsequent chapters, the reason for which will be stated later.

These mathematical models define an elastic boundary, Γ, submerged in a viscous incompressible fluid in a domain, Ω. This is shown in two dimensions in Fig. 2.1 for visualization purposes, although the extension to three dimensions is natural and discussed later. It is prudent to establish notation for the quantities defined on the fluid

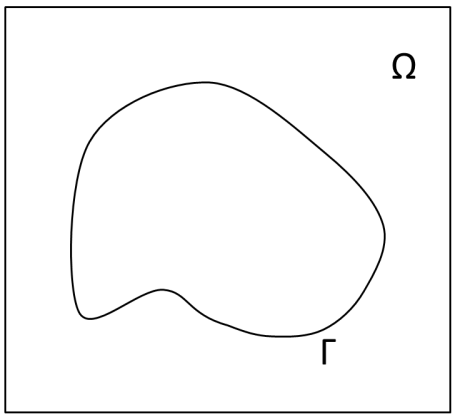


FIGURE 2.1. The two dimensional FSI problem with domain Ω and immersed fiber Γ

domain, Ω and those defined on the immersed boundary domain, Γ. As is conventional, all quantities defined on Ω are notated by lower case letters, and all quantities defined on Γ are capitalized. Additionally, it should be noted that from here on, all bold variables indicate vectors. A full list of notation can be found in **??**.

Both IBM and BPM rely on a projection formulation for the incompressible Navier-Stokes equations, and IBM can easily be explained in this formulation, thus this is presented first in this chapter. Following this, the IBM model is given, with its computational formulation, and finally BPM is discussed. This will complete the spatial methods necessary for later discussions.

2.1. The Projection Formulation

Projection methods for computational fluids were originated by Chorin [**16, 17**], and their implementation has been a frequent topic in the the literature, for examples see Brown or Guermond and Shen, [**10, 43**]. It is based on the idea that the time integration of the incompressible Navier-Stokes equations

$$\begin{aligned} \boldsymbol{u}_t + (\boldsymbol{u} \cdot \nabla) &= -\nabla p + \boldsymbol{u} \frac{1}{Re} \Delta \boldsymbol{u} + \boldsymbol{f}, \\ \nabla \cdot \boldsymbol{u} &= 0, \end{aligned} \tag{2.1}$$

can be made divergence free with the use of the Hodge decomposition, given proper treatment of the boundary conditions. Commonly, boundary conditions are taken to be the no flow conditions

$$\hat{n} \cdot \boldsymbol{u} = 0,$$

which is used in the definition of the Hodge condition, however it should be noted that the following holds for periodic boundary conditions as well.

THEOREM 2.1.1. *The Hodge Decomposition*

A vector field $\boldsymbol{v}$ in a bounded domain $\Omega \in \mathbb{R}^3$, with boundary $\partial\Omega$, can be written uniquely as the sum of a divergence free vector, $\boldsymbol{u}$, and the gradient of a scalar, η.

$$\boldsymbol{v} = \boldsymbol{u} + \nabla \eta, \tag{2.2}$$

where $\boldsymbol{u}$ satisfies $\nabla \cdot \boldsymbol{u} = 0$ in Ω and on the boundary, there is no flow in the normal direction, $\hat{n} \cdot \boldsymbol{v}|_{\partial\Omega} = 0$.

PROOF. Consider taking the divergence of both sides of equation (2.2). This gives

$$\nabla^2 \eta = \quad \nabla \cdot \boldsymbol{v} \quad \text{in } \Omega, \tag{2.3}$$

$$\hat{n} \cdot \nabla \eta = \quad \hat{n} \cdot \boldsymbol{v} \quad \text{on } \partial\Omega. \tag{2.4}$$

$\boldsymbol{u}$ is the projection of $\boldsymbol{v}$ onto a divergence free space and denoted equally as $\boldsymbol{u} = \mathbb{P}(\boldsymbol{v}) = \boldsymbol{v} - \nabla\eta$. In addition, the compatibility conditon for a Poisson equation with Neumann boundary conditions is nessicarily satisified by the divergence theroem.

Thus, defining the scalar function η as the solution equation (2.3) provides the proof of the Theorem as well as a way to construct the divergence free projection of $\boldsymbol{f}$, $\mathbb{P}\boldsymbol{f}$. □

With this definition, we can uphold the incompressibility constraint,$\nabla \cdot \boldsymbol{u} = 0$, of the Navier-Stokes Equations, (2.1), by taking the projection of both sides,

$$\boldsymbol{u}_t = \mathbb{P}(\boldsymbol{u}_t) = \mathbb{P}\left((\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nabla p + \frac{1}{Re}\nabla^2 \boldsymbol{u} + \boldsymbol{f}\right). \tag{2.5}$$

Further, the projection operator is linear and thus can be distributed across the right hand side terms of (2.5). Additionally, the projection of the gradient of the pressure, ∇p, is zero, and thus the term can be dropped from (2.5). With this, (2.5) can be rewritten as

$$\boldsymbol{u}_t = \mathbb{P}\left((\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \frac{1}{Re}\nabla^2 \boldsymbol{u}\right) + \mathbb{P}\boldsymbol{f}. \tag{2.6}$$

The pressure does not need to be tracked for the purposes of the work presented here, thus this form of the Navier-Stokes equations will subsequently be used instead of (2.1).

2.2. The Immersed Boundary Method

The *immersed boundary method (IBM)* was initially introduced by Peskin with the intention of studying cardiac fluid dynamics [**79, 81**]. It has, however, subsequently been applied to diverse problems in biofluid dynamics and other applications in which an elastic structure is immersed in a viscous incompressible fluid. Examples range from blood flow

[**66, 62, 63, 64, 65**], aquatic locomotion [**32, 31, 49, 91, 20**] , platelet aggregation during clotting [**33, 86, 34**], parachutes [**55**], foams [**88**], cell motility [**27**], insect flight [**69, 67, 68**] and jellyfish pulsation [**48**]. This list is by no means comprehensive, but does show the diversity of applications that utilize IBM. The accessibility of this method to so many applications is a result of its relatively 'simple' approach. It relies on treating the immersed boundary in a Lagrangian sense, while defining the fluid in a Eulerian sense and then coupling the two domains. This allows for the fluid to be discretized on a standard Cartesian mesh and the use of grid based fluid solvers, without direct consideration of the domain Γ. The mathematical model for the IBM is now presented.

Again consider Fig. 2.1 with an elastic boundary, Γ, immersed in a viscous incompressible fluid domain Ω. The immersed boundary, also referred to here as a fiber, is taken to be a two dimensional line. This simplification is often seen in IBM literature, as even the most complicated complex immersed surfaces in three dimensions, e.g. a heart model [**81**], are made of a network of such fibers. Additionally, the IBM assumes the fiber to be mass-less, occupy zero volume fracture and be neutrally buoyant, thus moving at the local fluid velocity [**79**].

The motion of fluid is described by the non-dimensional, incompressible Navier-Stokes Equations, given in (2.1). The equation of motion for the immersed boundary is a consequence of the no slip condition, which states that at the boundary between a structure and fluid, the velocity of the fluid, $\boldsymbol{u}(\boldsymbol{x},t)$, is equal to the velocity of the structure, defined as $\boldsymbol{U}(\boldsymbol{X},t)$. This is stated mathematically by the advection equation

$$\boldsymbol{X}_t \;=\; \boldsymbol{U}\left(\boldsymbol{X},t\right), \tag{2.7}$$

where $\boldsymbol{U}\left(\boldsymbol{X},t\right)$ is the velocity of the fluid at the structure position, $\boldsymbol{X}$.

To complete the equations, a definition for $\boldsymbol{f}\left(\boldsymbol{x},t\right)$, the body force on Ω, is required. In the IBM model, this force is a result of the forces, $\boldsymbol{F}\left(\boldsymbol{X},t\right)$, generated and defined on the fiber. The fiber is taken to be an elastic material and thus it is assumed that the

elastic energy is determined by an energy functional $E[\boldsymbol{X}(t)]$, which is the energy stored in the material at time t [**80**].

Consider a perturbation $\mathfrak{d}\boldsymbol{X}$ of a fiber with configuration $\boldsymbol{X}$. The perturbation of the energy functional, up to first order terms, is a linear functional of the perturbation in the material position [**80**]. This can be written as

$$\mathfrak{d}E[\boldsymbol{X}] = \int (-\boldsymbol{F}(q,r,s,t)) \cdot \mathfrak{d}\boldsymbol{X}(q,r,s,t) dqdrds, \tag{2.8}$$

where $-\boldsymbol{F}$ is the Fréchet derivative of E evaluated on $\boldsymbol{X}$. $\boldsymbol{F}$ is interpreted as the force density, with respect to q, r, s and is written

$$\boldsymbol{F} = -\frac{\mathfrak{d}E}{\mathfrak{d}X}. \tag{2.9}$$

As an example, consider a fiber where q, r are constant along the fiber. Then the force density comes from a resistance to bending and stretching. Resistance to torque could be included in these forces, however when using a network of fibers, it is unlikely that it is necessary to include [**80**]. Thus

$$\boldsymbol{F} = -\left(\frac{\mathfrak{d}E_S}{\mathfrak{d}X} + \frac{\mathfrak{d}E_B}{\mathfrak{d}X}\right) = \boldsymbol{F}_B + \boldsymbol{F}_S,$$

where the subscripts S and B denote the energy and forces due to stretching and bending, respectively. The force per unit length of bending is given by

$$\boldsymbol{F}_B = -k_B \frac{\partial^4 \boldsymbol{X}}{\partial^4 s}, \tag{2.10}$$

where k_B is stiffness coefficient. Similarly, the force due to stretching is

$$\boldsymbol{F}_S = -k_S \frac{\partial}{\partial s}\left(\left(\frac{\partial \boldsymbol{X}}{\partial s} - 1\right)\tau\right), \tag{2.11}$$

where $\tau = (\partial \boldsymbol{X}/\partial s)/|\partial \boldsymbol{X}/\partial s|$ is the unit directional vector and k_s the spring stiffness coefficient [**69**].

These forces are defined on the Lagrangian domain and thus they can be taken as a distribution, and defined on the fluid domain by the convolution,

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_{\Gamma} \boldsymbol{F}(\boldsymbol{X},t)\delta\left(\boldsymbol{x}-\boldsymbol{X}\right) d\boldsymbol{X}. \tag{2.12}$$

This definition of the forces is singular, as it is a line integral of a delta function.

Similarly, the velocity of the immersed boundary can also be defined with a convolution, since it is to be taken as the local fluid velocity. This defines $\boldsymbol{U}(\boldsymbol{X},t)$ as

$$\boldsymbol{U}(\boldsymbol{X},t) = \int_{\Omega} \boldsymbol{u}(\boldsymbol{x},t)\delta\left(\boldsymbol{u}\left(\boldsymbol{x},t\right)-\boldsymbol{X}\right) d\boldsymbol{x} \tag{2.13}$$

Equations (2.1), (2.7) - (2.9), and (2.12) and (2.13) give a closed, coupled system of equations. For the sake of future reference, the continuous equations that make up the mathematical model of the IBM are restated here.

$$\boldsymbol{u}_t(\boldsymbol{x},t) = \mathbb{P}\left(\left(\boldsymbol{u}(\boldsymbol{x},t)\cdot\nabla\right)\boldsymbol{u}(\boldsymbol{x},t) + \frac{1}{Re}\nabla^2\boldsymbol{u}(\boldsymbol{x},t)\right) + \mathbb{P}\boldsymbol{f}, \tag{2.14}$$

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_{\Gamma} \boldsymbol{F}(\boldsymbol{X},t)\delta\left(\boldsymbol{x}-\boldsymbol{X}\right) d\boldsymbol{X}, \tag{2.15}$$

$$\begin{aligned} \boldsymbol{X}_t &= \boldsymbol{U}\left(\boldsymbol{X},t\right), \\ &= \int_{\Omega} \boldsymbol{u}(\boldsymbol{x},t)\delta\left(\boldsymbol{x}-\boldsymbol{X}\right) d\boldsymbol{x}, \end{aligned} \tag{2.16}$$

$$\boldsymbol{F}\left(\boldsymbol{X},t\right) = -\frac{\mathfrak{d}E}{\mathfrak{d}X} \tag{2.17}$$

where equation (2.14) is the fluid equation, (2.17) is the fiber equation and $(2.15)-(2.17)$ are the interaction equations.

2.2.1. Computational Formulation. Consider a standard three dimensional Cartesian computational domain Ω defined on the unit cube $[0,1]\times[0,1]\times[0,1]$, with grid spacing $h_x = 1/N_x, h_y = 1/N_y, h_z = 1/N_z$, where the fluid is defined at each grid point $\boldsymbol{x}_{i,j,k} = (ih_X, jh_y, kh_z) \in \Omega$. In conjunction with this grid, an immersed boundary is defined on an independent Lagrangian grid by N_Γ points, $\boldsymbol{X}_\gamma(q,r,s) = \boldsymbol{X}_{q,r,s}$, $\gamma = 1...N_\Gamma$, with curvilinear coordinates q,r,s, separated by a distance of Δl. It is again assumed

that the geometry of any immersed boundary is made up of a network of q fibers, thus all discussion of the immersed equations are in this context. The discretization of Ω and Γ is shown in figure 2.2 in two dimensions for ease of visualization.

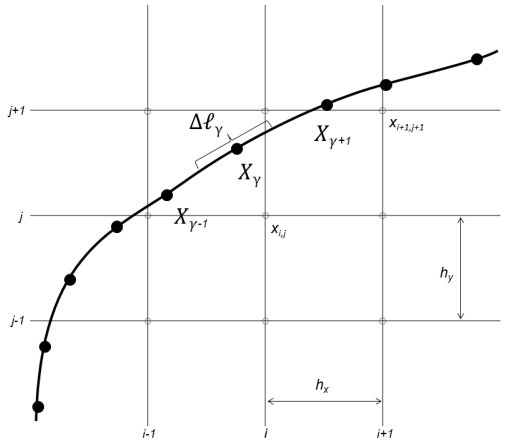


FIGURE 2.2. Close up on part of Ω and Γ, discretized in 2D.

Here the popular *method of lines* approach (MOL) for PDEs is used, discretizing PDE spatially first, resulting in a system of coupled temporal ODEs. The spatial discretization of equations $(2.14) - (2.17)$ gives the following system of ODEs.

$$\frac{d\boldsymbol{u}_{i,j,k}}{dt} = \mathbb{P}\left(-\left(\boldsymbol{u}_{i,j,k}\cdot\nabla_h\right)\boldsymbol{u}_{i,j,k} + \frac{1}{Re}\nabla_h^2\boldsymbol{u}_{i,j,k} + \boldsymbol{f}_{i,j,k}\right), \tag{2.18}$$

$$\boldsymbol{f}_{i,j,k} = \sum_{\gamma=0}^{N_\Gamma} \boldsymbol{F}_\gamma\, \mathcal{D}_h\left(\boldsymbol{x}_{i,j,k} - \boldsymbol{X}_\gamma\right)\Delta l_\gamma, \tag{2.19}$$

$$\frac{d\boldsymbol{X}_{q,r,s}}{dt} = \boldsymbol{U}_{q,r,s}, \tag{2.20}$$

$$= \sum_{i,j,k} \boldsymbol{u}_{i,j,k}\, \mathcal{D}_h\left(\boldsymbol{x}_{i,j,k} - \boldsymbol{X}_{q,r,s}\right) h_x h_y h_z, \tag{2.21}$$

$$\boldsymbol{F}_{q,r,s} = \boldsymbol{F}_{\mathrm{Str},q,r,s} + \boldsymbol{F}_{\mathrm{Bend},q,r,s}. \tag{2.22}$$

Where Δ_h and ∇_h are the discretized forms of their respective continuous operators, $\mathcal{D}_h$ is a discretized delta function.

The forces, $\boldsymbol{F}_{\text{Str}}$ and $\boldsymbol{F}_{\text{Bend}}$ are the resistance to stretching and bending, given by

$$\boldsymbol{F}_{\text{Str},q,r,s} = -k\left(\left(\boldsymbol{X}_{q,r,s+1} - \boldsymbol{X}_{q,r,s} - L_s\right)\hat{n}_s \right. \tag{2.23}$$

$$\left. + \left(\boldsymbol{X}_{q,r,s} - \boldsymbol{X}_{q,r,s-1} - L_{s-1}\right)\hat{n}_s\right), \tag{2.24}$$

$$\boldsymbol{F}_{\text{Bend},q,r,s} = \sigma\kappa\hat{n}. \tag{2.25}$$

L_s is the resting length of the spring located between $\boldsymbol{X}_{q,r,s+1}$ and $\boldsymbol{X}_{q,r,s}$, k is the spring constant, $\hat{n}$ is the normal vector, σ is the bending stiffness and κ is the curvature. This model reflects a commonly used IBM technique, to define the fiber as a series of points attached by simple linear springs, resulting in the force given by Hooke's law [**82, 81**], shown in Figure 2.3.

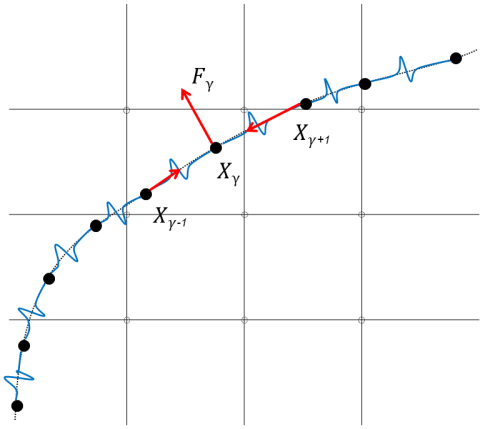


FIGURE 2.3. Springs connecting points on Γ, producing a force, $\boldsymbol{F}_\gamma$ at point $\boldsymbol{X}_\gamma$.

There are many choices for the discrete delta function; an example of a definition see frequently is

$$\mathcal{D}_h(\boldsymbol{x}) = \delta_h(x)\delta_h(y)\delta_h(z), \tag{2.26}$$

$$\text{where} \quad \delta_h(x) = \begin{cases} \frac{1}{4h}\left(1 + \cos\frac{\pi x}{2h}\right), & |x| \le 2h \\ 0 & |x| > 2h. \end{cases} \tag{2.27}$$

This definition satisfies the conditions for a delta function approximation, as it is $C^1(\mathbb{R})$, has unit mass and satisfies $\int_0^\infty x\delta_h(x)dx = 0$. Note that this definition of $\mathcal{D}_h$ is dependent on grid size, and as the grid spacing decreases, it becomes more singular. This is a common trait of the spreading functions used for the immersed boundary method, although there have been smoother versions used in recent publications [**40**].

Immersed Boundary method is generally implimented with a second order implimentation, although higher order schemes have been presented [**56, 41, 87**]. Excluding the fully and semi-implicit implimentations, when IBM is used for applications, diffusion is often evaluated implicitly and the movement of the boundary (2.22) explictly [**41, 68**]. This, along with high forces due to large values of k or σ, contribute to the need for small time steps in the interest of stability.

2.3. The Blob Projection Method

The *blob projection method*, as introduced by Cortez and Minion (2000), is a high-order finite-difference method for modeling the interaction between an elastic structure and an incompressible fluid [**21**]. This method follows the same basic framework as the immersed boundary method, presented in 2.2.

There are two key differences between IBM and BPM, the first being how the spreading function is defined. In IBM the spreading function is dependent on the grid size, where the regularized delta function or 'blob' in BPM is grid independent. The other major difference is that with classic IBM, the effect of the immersed boundary's motion on the fluid, the projection of the force, $\mathbb{P}\boldsymbol{f}$, is evaluated with discretized finite difference operators, by solving the Poisson equation (2.3). BPM evaluates the projection of the force with an analytic function, which reduces error and gives flexibility in how this term is computationally evaluated. These differences improve the loss of formal order at the immersed boundary and reduce leaking across the boundary.

BPM is selected here because of the analytic function for $\mathbb{P}\boldsymbol{f}$, which gives the potential to split the forcing into stiff and non-stiff parts. The drawback of this approach is that

the analytic $\mathbb{P}\boldsymbol{f}$ evaluation is expensive, so a hybrid approach is taken, details of which are discussed in 2.3.2

2.3.1. The Spreading Function. With IBM, forces are spread with the use of a grid dependent approximation to the Dirac delta function, $\delta \approx \mathcal{D}_h$ (see (2.26)). The subscript, h, refers to this grid dependence. Instead consider a regularized function, or blob, denoted $B_\delta(\boldsymbol{X})$, dependent on a grid independent cutoff parameter δ. This gives the following approximation for the forces

$$\boldsymbol{f}(\boldsymbol{x},t) \approx \int_\Gamma \boldsymbol{F}(\boldsymbol{X},t)\, B_\delta\left(\boldsymbol{x}-\boldsymbol{X}_\gamma\right) d\boldsymbol{X}, \tag{2.28}$$

where

$$B_\delta(\boldsymbol{X}) = \frac{1}{\delta^2} B(|\boldsymbol{X}|/\delta), \tag{2.29}$$

and $B(r)$ is a radially symmetric cutoff function that satisfies the proper conditions for an approximation to the delta function, mentioned in Section 2.2.1. An example of the blob on a two dimensional grid is given in Figure 2.4.

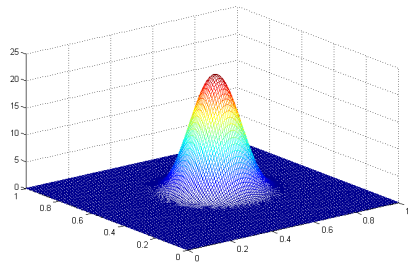


FIGURE 2.4. Regularized blob, $B_\delta(r)$

Choosing a grid independent spreading function gives an analytic expression for the forces, allowing for the derivation of an analytic formula for the projection of the forces, $\mathbb{P}\boldsymbol{f}$. Additionally, the spreading radius can be varied to minimized error at the boundary.

From here on, a specific, compact, cutoff function will be used, given by

$$B_\delta(\boldsymbol{X}) = \frac{1}{\delta^2} B(|\boldsymbol{X}|/\delta), \tag{2.30}$$

$$B(\chi) = \frac{6}{\pi}(1-\chi^2)^5 \quad \text{for} \quad \chi \leq 1 \tag{2.31}$$

Although this blob is chosen here, the derivation of $\mathbb{P}\boldsymbol{f}$ and its resulting formula, discussed below is independent of specific blob function choice.

2.3.2. Projection of the Forces, $\mathbb{P}\boldsymbol{f}$. Again refer to Equations (2.14) – (2.17), specifically the evaluation of the term, $\mathbb{P}\boldsymbol{f}$ in (2.14). In traditional IBM implementations, this is evaluated by first solving the Poisson equation, (2.3), with discrete operators, (denoted by subscript h, as before)

$$\nabla_h^2 \eta = \nabla_h \cdot \boldsymbol{f}, \tag{2.32}$$

$$\hat{n} \cdot \nabla \eta = \hat{n} \cdot \boldsymbol{f}. \tag{2.33}$$

Then, subtracting the gradient of the resulting solution η from $\boldsymbol{f}$ gives the final form of the term,

$$\mathbb{P}\boldsymbol{f} = \boldsymbol{f} - \nabla_h \eta.$$

In this process, discrete approximations, commonly finite difference, are used for the divergence, gradient and Laplacian operators. It is well known that to obtain an exact projection, where the divergence of $\mathbb{P}\boldsymbol{f}$ is identically zero, an adjoint condition is necessary [**17**]. There are ways to formulate the discrete operators such that an exact, not approximate, projection is found, however one must be careful of this condition.

Instead of computing $\nabla_h \eta$ through finite difference operators, including taking the divergence of a discretized delta function, the BPM uses an analytic formula for $\mathbb{P}\boldsymbol{f}$. This is derived from the analytic formula for $\boldsymbol{F}$, (2.28) with the idea from vortex and impulse methods [**5, 12, 19, 22**]. The full derivation of this formula can be found in the Appendix, here only the main result is given. The analytic projection of the forces at

any grid point $\boldsymbol{x}$ is

$$\mathbb{P}\boldsymbol{f}(\boldsymbol{x}) = \sum_{\gamma=0}^{N_\Gamma} \left(\frac{r\mathbb{F}'(r) - \mathbb{F}(r)}{2\pi r^2} \right) \boldsymbol{f}_\gamma \Delta l_- \left(\frac{r\mathbb{F}'(r) - 2\mathbb{F}(r)}{2\pi r^2} \right) \hat{\boldsymbol{x}}_\gamma (\boldsymbol{f}_\gamma \Delta l . \hat{\boldsymbol{x}}_\gamma), \tag{2.34}$$

where

$$r = |\boldsymbol{x} - \boldsymbol{X}|, \quad \hat{\boldsymbol{x}} = (\boldsymbol{x} - \boldsymbol{X})/r, \text{ and } \mathbb{F}(r) = 2\pi \int_0^r s B_\delta(s)\, ds.$$

Note that this function is global, however the force can defined locally, as is done here through the choice of a compact blob function, $B_\delta()$.

As mentioned, evaluating this term is expensive, the cost is on the order of $O(N_\Gamma N_x N_y N_z)$ where N_Γ is the number of points that discritize Γ, and the number of grid points in the x, y and z directions are given by $N_x = N_y = N_z$. Therefore, using it directly is not practical when the goal is to reduce computational run time. Thus a hybrid approach is taken where, like (2.32), a discrete operator is used to solve the Poisson problem, however the right hand side is the analytic expression for $\nabla \cdot \boldsymbol{f}$. That is,

$$\nabla_h^2 \eta = (\nabla \cdot \boldsymbol{f})_{analytic}, \tag{2.35}$$

$$\hat{n} \cdot \nabla \eta = 0. \tag{2.36}$$

This still avoids the error from taking the discrete divergence of a delta function and is not as costly as evaluating (2.34) outright. There are other choices that could be made here, for example, using the *fast multipole method (FMM)*, which is used for fast summations for the n-body problem [**39**]. Even though the choice made here does not use the full equation, (2.34) is still useful because it can be used locally to split up the spatial field into near and far field components, which is presented in detail in Chapter 4.

2.4. Stiffness in IBM

Stiffness, which is described more in Section 3.4, is the notion that the computational time step for explicit schemes is restricted not by accuracy, but stability. The system

of time ODEs given in (2.18)-(2.22) for IBM (and likewise BPM) is a stiff system of ODE's. This has been documented many times in the literature and directly linked to the magnitude of the forcing term [**79, 85**]. This causes the computational run time to be very costly, and has restricted the physical problems that can be studied.

Investigations into the cause of this stiffness for IBPM were conducted by Stockie and Wetton [**83, 82**]. In their first attempt to isolate the cause of stiffness, Stockie and Wetton showed that the restriction on the time step was dominated by the presence of the force generated on the fiber, not the usual limitation from the Reynolds number [**83**]. Next, they used the Stokes' equations and a local linearization of the fiber position with force spreading to examine the eigenvalues, λ, of an idealized solution. This analysis determined that stiffness in IBM comes from two sources:

(1) Large variation in $Re(\lambda)$, indicating fiber mode components that decay on a wide variety of time scales.

(2) Large variation in $Im(\lambda)$, indicating disparate frequencies of physical oscillations in the fiber solution.

Additionally, they stated the combination of large fiber forces and small viscosity were the components that effected numerical stability the most. Finally, they conclude that stiffness is likely traced to the tangential oscillations of the fiber motion, and recommend that a potential solver could be more efficient if it directly isolated these oscillations. Here these recommendations are considered by the forces and motion of the fiber, using both spatial and temporal splitting. The spatial splitting is presented in detail in Chapter 4 and the temporal splitting, by means of multi-rate methods, are discussed next in Chapter 3.

CHAPTER 3

Spectral Deferred Corrections Methods

3.1. Spectral Deferred Corrections

Spectral deferred corrections (SDC) is a class of algorithms that provides high order, stable solutions to initial value ODE problems. It was introduced by Dutt, Greengard and Rokhlin in 2000 as a stable improvement to classical defect or deferred correction methods, for stiff ODEs [**28**]. Defect corrections first appeared in the work of P. Zadunaisky in the 1960's, and is frequently seen in the literature thereafter [**89, 90, 6, 78**]. The general strategy is to start with a low-order approximation of the solution and through successive iterations, to correct the previous approximation, with a low-order estimate of the error, giving a more accurate solution each time.

SDC methods are comparable to Runge-Kutta (R-K) and multi-step methods, however these other methods are not easily extend to multiple time scales [**11, 38, 44, 45**]. Additionally, in contrast to R-K methods, SDC methods are easily constructed for any order of accuracy [**71**].

Here, SDC methods are used for the ease of handling implicit-explicit (IMEX) implementations, multiple terms in the ODE, and different time scales. In the following section SDC is derived, followed by extensions of the method, and a discussion of SDC in stiff problems.

Consider the general initial value ODE

$$\begin{aligned} \phi'(t) &= f(\phi(t), t) \qquad t \in [0, T], \\ \phi(0) &= \phi_0, \end{aligned} \tag{3.1}$$

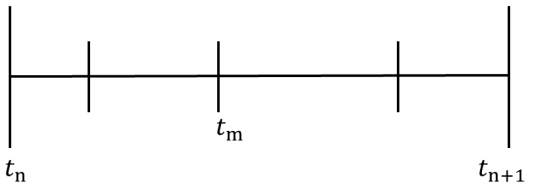


FIGURE 3.1. A full time step, $[t_n, t_{n+1}]$ with SDC nodes t_m

where $\phi(t) \in \mathbb{C}^N$ is continuous in t, $\phi_0 \in \mathbb{C}^N$, and $f(\phi(t), t)$, assumed to be Lipschitz continuous, maps $\mathbb{R}$ x $\mathbb{C}^N \to \mathbb{C}^N$. The time domain, $[0, T]$, is discretized into N sub-steps with intervals $[t_n, t_{n+1}] = [t_n, t_n + \Delta t_n]$. Further, each interval is subdivided into M SDC intervals, where $t_n = t_{m=1} < t_{m=1} < ... < t_{m=M-2} < t_{m=M} = t_{n+1}$ and $t_{m+1} = t_m + \Delta t_m$, shown in Figure 3.1. Note that the SDC nodes, are not necessarily uniformly distributed, this will be important to the formulation of the method. In the interest of simple notation, t_n will refer to the larger interval nodes and t_m, the SDC nodes, even though each t_m is a node belonging to a particular n interval.

Let $\phi^{[0]} = [\phi^{[0]}(t_0), \phi^{[0]}(t_1), ..., \phi^{[0]}(t_M)]$ denote a provisional solution to (7.1) on the SDC nodes, computed with a standard numerical method, e.g. forward Euler for explicit problems. The superscript [0] indicates that this is the initial iteration, also sometimes referred to as the predictor. The error of this solution is given by

$$E(t) = \phi(t) - \phi^{[0]}(t) \tag{3.2}$$

A better solution is $\phi^{[1]}$, where $\phi^{[1]}(t) = \phi^{[0]}(t) + \tilde{E}(t)$, with $\tilde{E}(t)$ approximating the error. In classic defect corrections, the error, $E(t)$, is approximated by differentiating $\phi^{[0]}$ and formulating an ODE for $E(t)$. However, SDC employs the Picard integral equation formulation for (7.1),

$$\phi(t) = \phi_0 + \int_0^t f(\phi(\tau), \tau) d\tau. \tag{3.3}$$

Using (3.3), the residual, R, of the provisional solution at time t_m is given by

$$R(\phi^{[0]}, t_m) = \phi_0 + \int_0^{t^m} q^{[0]}(\tau) d\tau - \phi^{[0]}, \tag{3.4}$$

where $q^{[0]}(t_m)$ is the polynomial that interpolates $f(\phi^{[0]}, t^m)$. Subsequently the resulting error E at time t_m satisfies

$$E(t_m) = E_m = \phi(t_m) - \phi_m^{[0]} = \int_0^{t_m} f(\phi(\tau), \tau) - q^{[0]}(\tau) d\tau + R(\phi^{[0]}, t_m). \tag{3.5}$$

With (5.40), it is possible to write the error at time t_{m+1} as an update from the error at time t_m

$$E_{m+1} = E_m + \int_{t_m}^{t_{m+1}} f(\phi(\tau), \tau) - q^{[0]}(\tau) d\tau + R(\phi^{[0]}, t_{m+1}) - R(\phi^{[0]}, t_m). \tag{3.6}$$

In the integral, $f(\phi(t), t)$ is unknown, however it is equivalent to $f(\phi(t), t) = f(\phi^{[0]}(t) + E(t), t)$, which gives

$$E_{m+1} = E_m + \int_{t_m}^{t_{m+1}} f(\phi^{[0]}(\tau) + E(\tau), \tau) - q^{[0]}(\tau) d\tau + R(\phi^{[0]}, t_{m+1}) - R(\phi^{[0]}, t_m). \tag{3.7}$$

Combining (5.40) and (3.7), the difference in the residuals is defined by

$$R(\phi^{[0]}, t_{m+1}) - R(\phi^{[0]}, t_m) = \int_{t_m}^{t_{m+1}} q^{[0]}(\tau) d\tau - (\phi_{m+1}^{[0]} - \phi_m^{[0]}), \tag{3.8}$$

and equation (3.7) can be written as

$$\begin{aligned} E_{m+1} &= E_m + \int_{t_m}^{t_{m+1}} f(\phi^{[0]}(\tau) + E(\tau), \tau) - q^{[0]}(\tau) d\tau + \int_{t_m}^{t_{m+1}} q^{[0]}(\tau) d\tau \\ &- (\phi_{m+1}^{[0]} - \phi_m^{[0]}), \end{aligned} \tag{3.9}$$

which is a full expression for the error, E_{m+1}, at t_{m+1}.

As mentioned, a better approximation to the solution, $\phi^{[1]}$, is obtained by updating the provisional solution $\phi^{[0]}$ at each t_m with an discritized estimate of the error, $\tilde{E}_m$

$$\phi_m^{[1]} = \phi_m^{[0]} + \tilde{E}_m. \tag{3.10}$$

Further, equation 3.9 is equivalent to

$$\phi_{m+1}^{[0]} + E_{m+1} = \phi_m^{[0]} + E_m + \int_{t_m}^{t_{m+1}} f(\phi^{[0]}(\tau) + E(\tau), \tau) - q^{[0]}(\tau) d\tau + \int_{t_m}^{t_{m+1}} q^{[0]}(\tau) d\tau, \tag{3.11}$$

and combining 3.10 and 3.9 results in

$$\phi_{m+1}^{[1]} = \phi_m^{[1]} + \int_{t_m}^{t_{m+1}} f(\phi^{[1]}(\tau), \tau) - q^{[0]}(\tau) d\tau \int_{t_m}^{t_{m+1}} q^{[0]}(\tau) d\tau, \tag{3.12}$$

a direct update for $\phi^{[1]}$ on the t_m nodes. This can be extended iteratively, $k = 1...K$ times, giving the solution $\phi^{[k+1]}$ as follows

$$\phi_{m+1}^{[k+1]} = \phi_m^{[k+1]} + \int_{t_m}^{t_{m+1}} f(\phi^{[k+1]}(\tau), \tau) - q^{[k]}(\tau) d\tau + \int_{t_m}^{t_{m+1}} q^{[k]}(\tau) d\tau. \tag{3.13}$$

Equation 3.13 is sometimes called the correction equation and the evaluation of the integrals should be remarked on. The first

$$\int_{t_m}^{t_{m+1}} f(\phi^{[k+1]}(\tau), \tau) - q^{[k]}(\tau) d\tau, \tag{3.14}$$

requires values at the $k+1$ iteration, and thus is evaluated with a simple rule, such as a left hand rule. The second,

$$\int_{t_m}^{t_{m+1}} q^{[k]}(\tau) d\tau, \tag{3.15}$$

only requires previously computed values, thus can be evaluated with a spectral integration rule, giving the *Spectral* in SDC [**28**]. This implies that the best estimation is found by choosing the t_m nodes with a spectral quadrature rule, such as Gaussian, or a high precision rule like Clenshaw-Curtis quadrature [**84**]. That is,

$$\int_{t_m}^{t_{m+1}} q^{[k]}(\tau) d\tau \approx \sum_{m=1}^{M} w_m q^{[k]}(t_m), \tag{3.16}$$

where t_m and w_m are the nodes and weight functions associated with the quadrature rule. This a highly accurate estimation of the error gives a good approximation to the solution.

To clarify, consider evaluating the first integral in Equation (3.13) with a forward Euler type approximation. The resulting predictor-corrector scheme for applying k iterative corrections to $\phi^{[0]}$, is

$$\phi_{m+1}^{[0]} = \phi_m^{[0]} + \Delta t_m f(\phi_m^{[0]} t_m) \tag{3.17}$$

$$\phi_{m+1}^{[k+1]} = \phi_m^{[k+1]} + \Delta t_m \left(f(\phi_m^{[k+1]} t_m) - f(\phi_m^{[k]}, t_m) \right) + \mathbb{I}_m^{m+1} f(\phi^{[k]}), \tag{3.18}$$

where $\mathbb{I}_m^{m+1} f(\phi^{[k]})$ is used to denote the quadrature approximation given in (3.16). Each k iteration of correction raises the formal order of the initial solution by the order of the approximation to (3.14). Thus the order for this example's predictor (3.17) is one , and applying (3.18), which is also order one, K times will increase the formal order to $K+1$. Additionally, as the iterations increase $f(\phi_m^{[k+1]}, t_m) - f(\phi_m^{[k]}, t_m)$ goes to zero, leaving only the error from the quadrature rule. Note that this means high-order results simply from iterating low order methods in a particular way, one of the main benefits of using SDC. Additionally, upon convergence of SDC, (3.18) becomes

$$\phi_{m+1}^{[k+1]} = \phi_m^{[k+1]} + \mathbb{I}_m^{m+1} f(\phi^{[k]}),$$

which is equivilent to the fully implicit collocation method, or the implicit Runge-Kutta Method [**44, 45**].

This method is chosen here not only because of the aforementioned property, but that extensions to various terms on the right hand side are quite easy. This will be evident in the next few sections, where extensions to multi-implicit and multi-rate formulations of SDC are given.

3.2. Semi-Implicit Spectral Deferred Corrections

Semi-Implicit Spectral Deferred Corrections (SISDC) is an extension of SDC published by Minion, to target the incompressible Navier-Stokes equations (2.1) which have both stiff and non-stiff components [**71**]. The stiff components are treated with an implicit scheme, such as backward Euler, and the non-stiff terms with an explicit scheme such as forward Euler. SISDC treats all terms with the same time step and is comparable to IMEX or additive Runge-Kutta methods (*e.g.* [**3, 54, 13**]) or linear multi-step schemes (*e.g.* [**4, 37, 1**]). Unlike these methods, SISDC can easily be constructed to any order of accuracy, since it uses the SDC approach of successive low order iterations.

To formulate SISDC, consider the general initial value ODE

$$\begin{aligned} \phi'(t) &= f_1(\phi(t),t) + f_2(\phi(t),t) \qquad t \in [0,T], \\ \phi(0) &= \phi_0, \end{aligned} \tag{3.19}$$

where $f_1(\phi(t),t)$ is non-stiff and $f_2(\phi(t),t)$ is stiff. Then the $k+1$ update of an initial solution $\phi^{[0]}$ is given by

$$\begin{aligned} \phi_{m+1}^{[k+1]} &= \phi_m^{[k+1]} + \int_{t_m}^{t_{m+1}} f_1(\phi^{[k+1]}(\tau),\tau) - q_1^{[k]}(\tau)d\tau \\ &+ \int_{t_m}^{t_{m+1}} f_2(\phi^{[k+1]}(\tau),\tau) - q_2^{[k]}(\tau)d\tau + \int_{t_m}^{t_{m+1}} q_1^{[k]}(\tau) + q_2^{[k]}(\tau)d\tau, \end{aligned} \tag{3.20}$$

which is exactly equation (3.13) with the right hand side given in (3.19). Now, since $f_1(\phi(t),t)$ is non-stiff, it is evaluated explicitly and the stiff $f_i(\phi(t),t)$ is evaluated implicitly. Thus the predictor-corrector formula is formulated using an IMEX approach,

$$\phi_{m+1}^{[0]} = \phi_m^{[0]} + \Delta t_m \left(f_1(\phi_m^{[0]} t_m) + f_2(\phi_{m+1}^{[0]} t_{m+1}) \right) \tag{3.21}$$

$$\begin{aligned} \phi_{m+1}^{[k+1]} &= \phi_m^{[k+1]} + \Delta t_m \left(f_1(\phi_m^{[k+1]} t_m) - f_1(\phi_m^{[k]}, t_m) \right) \\ &+ \Delta t_m \left(f_2(\phi_{m+1}^{[k+1]} t_{m+1}) - f_2(\phi_{m+1}^{[k]}, t_{m+1}) \right) + \mathbb{I}_m^{m+1} \left(f_1(\phi^{[k]}) + f_2(\phi^{[k]}) \right). \end{aligned} \tag{3.22}$$

This idea was extended to a multi-implicit implementation of SISDC (MISDC) by Bourlioux *et. al.* for advection-diffusion-reaction equations and applied to gas dynamics [**7, 57**]. A multi-implicit implementation of SDC follows the same principal and derivation as SISDC, with an extra stiff right hand side term, evaluated implicitly. Operator splitting is used to handle the different terms in both SISDC and MISDC

Applications such as incompressible flow, turbulent flow and advection-diffusion-reaction systems with complex chemistry, that have a combination of stiff and non-stiff terms, have benefited from using SISDC [**71, 2, 76**]. However, there are some applications that call for the use of multiple time scales, which is the motivation for the next section.

3.3. Multi-rate Spectral Deferred Corrections

In addition to handling IMEX schemes, SDC is easily extended to use different time scales. Published by Bourlioux et. al. , and used by Bouzarth to evaluate stiff terms explicitly on refined time scales, this approach subdivides the time domain further with P sub-SDC nodes in each SDC interval [**7, 8**] . This is referred to as multi-rate spectral deferred corrections (MRSDC), where the term *multi-rate* is used to imply that the ODE has terms which are evaluated on different time scales.

Each SDC interval consists of P sub-SDC nodes, t_p, such that $[t_m, t_{m+1}] = [t_m = t_{p=0}, t_1, ..., t_{p=P} = t_{m+1}]$, $p = 1, ..., P$, and $t_{p+1} = t_p + \Delta t_p$. The t_p's, are also chosen according to a spectral integration quadrature and this setup is shown in the context of a full time step $[t_n, t_{n+1}]$ in Fig. 3.2. The same simplification in notation is used, where t_p denotes the sub-SDC node for a particular $[t_m, t_{m+1}] \in [t_n, t_{n+1}]$ interval.

Consider the ODE

$$\begin{aligned} \phi'(t) &= f_1(\phi(t), t) + f_2(\phi(t), t) \qquad t \in [0, T], \\ \phi(0) &= \phi_0. \end{aligned} \tag{3.23}$$

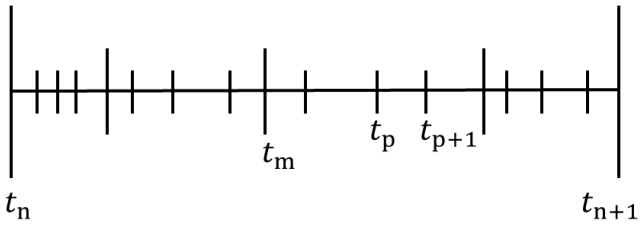


FIGURE 3.2. A full time step, $[t_n, t_{n+1}]$ with SDC nodes t_m and sub-SDC nodes, t_p

where $f_2(\phi(t), t)$ is stiff and $f_1(\phi(t), t)$ is non-stiff. Both terms are evaluated with an explicit scheme, however $f_2(\phi(t), t)$ will be evaluated on multirate nodes, since it is stiff. Again let $\phi^{[0]}$ be the provisional solution where $f_1(\phi(t), t)$ is evaluated on the t_m nodes, and $f_2(\phi(t), t)$ on the t_p nodes. This results in the the $k+1$ update to the initial solution $\phi^{[0]}$

$$\begin{aligned} \phi_{m+1}^{[k+1]} &= \phi_m^{[k+1]} + \int_{t_m}^{t_{m+1}} f_1(\phi^{[k+1]}(\tau), \tau) - q_1^{[k]}(\tau) d\tau \\ &+ \int_{t_m}^{t_{m+1}} f_2(\phi^{[k+1]}(\tau), \tau) - q_2^{[k]}(\tau) d\tau + \int_{t_m}^{t_{m+1}} q_1^{[k]}(\tau) + q_2^{[k]}(\tau) d\tau. \end{aligned} \tag{3.24}$$

Since $t_{p=1} = t_m$ and $t_{p=P} = t_{m+1}$, the integral

$$\int_{t_m}^{t_{m+1}} f_2(\phi^{[k+1]}(t), \tau)$$

can easily be evaluated as the sum of integrals over the sub-SDC nodes.

$$\int_{t_m}^{t_{m+1}} f_2(\phi^{[k+1]}(t), \tau) = \int_{t_{p=1}=t_m}^{t_{p=2}} f_2(\phi^{[k+1]}(t), \tau) + ... + \int_{t_{p=P-1}}^{t_{p=P}=t_{m+1}} f_2(\phi^{[k+1]}(t), \tau). \tag{3.25}$$

Similarly,

$$\int_{t_m}^{t_{m+1}} q_1^{[k]}(\tau) + q_2^{[k]}(\tau) d\tau = \int_{t_m}^{t_{m+1}} q_1^{[k]}(\tau) d\tau + \sum_{p=1}^{P-1} \int_{t_p}^{t_{p+1}} q_2^{[k]}(\tau) d\tau \tag{3.26}$$

To construct a predictor-corrector scheme for MRSDC, first consider the initial solution update on the SDC nodes,

$$\phi_{m+1}^{[0]} = \phi_m^{[0]} + \Delta t_m \left(f_1(\phi_m^{[0]}, t_m) + f_2(\phi_m^{[0]}, t_m) \right), \tag{3.27}$$

Fixing $f_N(\phi_m^{[0]}, t_m)$ across the sub-SDC interval gives the refined update on the t_p nodes,

$$\phi_{p+1}^{[0]} = \phi_m^{[0]} + \sum_{\zeta=1}^{\zeta=p} \Delta t_\zeta \left(f_1(\phi_m^{[0]}, t_m) + f_2(\phi_\zeta^{[0]}, t_\zeta) \right) \tag{3.28}$$

This is equivalent to

$$\phi_{p=1}^{[0]} = \phi_m^{[0]}, \tag{3.29}$$

$$\phi_{p+1}^{[0]} = \phi_p^{[0]} + \Delta t_p \left(f_1(\phi_m^{[0]}, t_m) + f_2(\phi_p^{[0]}, t_p) \right), \tag{3.30}$$

whose result is $f_2(\phi_P^{[0]}, t_P) = f_2(\phi_{m+1}^{[0]}, t_{m+1})$.

Extending Eq. 3.29 with 3.25 and 3.26 gives the k^{th} correction equation with a forward Euler implementation,

$$\begin{aligned} \phi_{m+1}^{[k+1]} &= \phi_m^{[k+1]} + \Delta t_m \left(f_1(\phi_m^{[k+1]}, t_m) - f_1(\phi_m^{[k]}, t_m) \right) \\ &+ \Delta t_m \left(f_2(\phi_m^{[k+1]}, t_m) + f_2(\phi_m^{[k]}, t_m) \right) + \mathbb{I}_m^{m+1} q_1^{[k]}(t) + q_2^{[k]}(t) \end{aligned} \tag{3.31}$$

$$\phi_{p=1}^{[k+1]} = \phi_m^{[k+1]} \tag{3.32}$$

$$\begin{aligned} \phi_{p+1}^{[k+1]} &= \phi_p^{[k+1]} + \Delta t_p \left(f_1(\phi_m^{[k+1]}, t_m) - f_1(\phi_m^{[k]}, t_m) \right) \\ &+ \Delta t_p \left(f_2(\phi_p^{[k+1]}, t_p) + f_2(\phi_p^{[k]}, t_p) \right) + \mathbb{I}_p^{p+1} q_1^{[k]}(t) + q_2^{[k]}(t) \end{aligned} \tag{3.33}$$

In the later, new approaches, a combination of SISDC and MRSDC will be used, since there will be multiple terms defined implicitly, as well as terms defined on different time scales. This will be discussed in detail in Chapter 6, however since the problems targeted here are stiff, it is prudent to make a note about SDC in stiff problems first.

3.4. Stiff Problems and SDC

To discuss the notion of a stiff equation, it must first be established that there is not a precise mathematical definition for the term. Thus, what will follow is a discussion of how stiffness is generally characterized.

The term 'stiff' was first used in 1952 by Curtiss and Hirschfelder [**24**], when comparing the preformance of implicit and explicit Euler methods to problems in chemical kinematics. However, the behavior of stiff problems had been seen prior to that in the works of Crank and Nicholson and Fox and Goodwin [**23, 36, 46**]. Generally, differential equation is said stiff when solving it with an explicit scheme requires a very small time step in order to be numerically stable. That is, the time step size is not determined by considerations for accuracy, but stability. The effect of stiffness on order-reduction and stability is seen throughout the literature, whether in conjunction with R-K methods or SDC methods [**25, 14, 71, 51**].

The interest here is in how stiffness effects SDC in particular. As mentioned in Section 3.1, SDC was initially published to target both non-stiff and stiff problems. The work of Dutt *et.al.* suggests that compared to the best initial value ODE solvers, SDC methods are at least equal, possibly favorable, when solving stiff problems where high accuracy is desired. However, subsequent work has found that when applied to stiff problems, SDC methods loses the formal order of accuracy at large time steps [**7, 71, 51**]. Consider Eq. (3.34),

$$\begin{aligned}
\phi_{m+1}^{[0]} &= \phi_m^{[0]} + \Delta t_m f(\phi_m^{[0]} t_m) \\
\phi_{m+1}^{[k+1]} &= \phi_m^{[k+1]} + \Delta t_m \left(f(\phi_m^{[k+1]} t_m) - f(\phi_m^{[k]}, t_m) \right) + \mathbb{I}_m^{m+1} f(\phi^{[k]}).
\end{aligned}$$

As the number of SDC correction iterations increases,

$$\lim_{k \to \infty} f(\phi_m^{[k+1]} t_m) - f(\phi_m^{[k]}, t_m) = 0,$$

meaning the solution converges to the spectral approximation of the Picard integral,

$$\mathbb{I}_m^{m+1} f(\phi^{[k]}).$$

However, when problems are stiff, more SDC iterations are needed to converge, giving order reduction.

To illustrate this, consider the ODE

$$\begin{aligned} \phi'(t) &= g'(t) + \frac{1}{\epsilon}\left(g(t) - \phi(t)\right), \\ \phi(0) &= \phi_0, \end{aligned} \tag{3.34}$$

where $g(t)$ is a slowly varying function. It is easy to verify that the exact solution to this ODE is $\phi(t) = g(t)$, As ϵ decreases, (3.34) becomes stiff because there are varying time scales, that for $g'(t)$ that for $\frac{1}{\epsilon}(g(t) - \phi(t))$.

Equation (3.34) is evaluated by disrictizing the time domain $[0, 1]$ with $N = 100$ steps, where $t_n = n/N$ and $\Delta t_n = 1/N$. M SDC steps were then defined on every $[t_n, t_{n+1}]$ interval, with $t_n = t_{m=1}, ..., t_{m=M} = t_{n+1}$. The function $g(t) = \sin(2\pi t)$ is used along with a forward Euler (FE) scheme, giving the following update on the t_m nodes,

$$\phi_{m+1} = \phi_m + \Delta t_m \left(g'(t_m) + \frac{1}{\epsilon}\left(g(t_m) - \phi_m\right)\right).$$

This is then extended to a full SDC predictor-corrector scheme and run for different values of ϵ to time $t = 1$. Figure 3.3 shows the error, compared to the exact solution $g(t)$, and it is clear that as the stiffness of an ODE increases, more SDC iterations are necessary for full convergence, if it will converge at all.

There are a two key points that this example illustrates. First, that order of convergence for SDC methods applied to stiff problems is a function of the number of iterations and the stiffness of the problem. Second, that this dependence on stiffness appears to be on a continuum, i.e. the stiffer the problem, the more SDC iterations needed for convergence, if it will converge at all.

Additionally, it should be noted that this issue was addressed by Huang *et. al.*, where GMRES Krylov subspace methods were employed to successfully accelerate the convergence of SDC, giving near full order under certain conditions [**51**]. The order is still reduced, because the problem is stiff, however it is not an artifact of the treatement of the method [**45**]. Although this technique is not used here, a more robust picture of

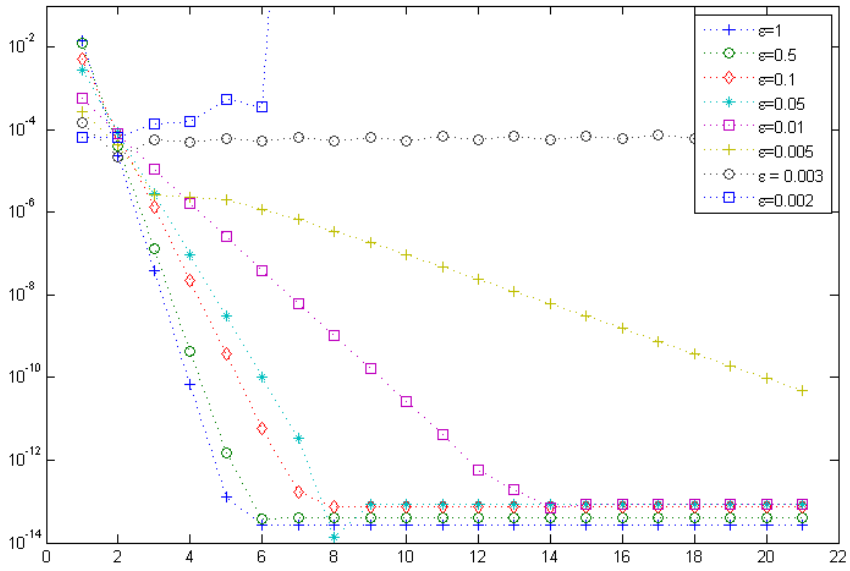


FIGURE 3.3. Number of SDC correction sweeps vs. error, solving (3.34) with different ϵ's

SDC's behavior with stiff problems is found in that work. Here, it is enough to note this property about SDC, as it will appear again in later analysis.

CHAPTER 4

Spatial Splitting in BPM

As previously discussed, the source of stiffness for both IBM and BPM is due to the presence of fiber forces. The force itself is local to the fiber and although the projection of the force is a global function, most of the magnitude of $\mathbb{P}\boldsymbol{f}$ is near the fiber. Thus the effects of $\mathbb{P}\boldsymbol{f}$ away from the fiber is not as large. Overall, the locally spatial behavior of the projection of the forces lends itself to the idea of splitting up the spatial field. This can be done with BPM, unlike IBM, since $\mathbb{P}\boldsymbol{f}$ can be analytically defined everywhere. This chapter will present a spatial splitting of $\mathbb{P}\boldsymbol{f}$, with the intention that the local, stiff part will be evaluated with a different technique than that used for the non-stiff, non-local part.

4.1. Splitting $\mathbb{P}\boldsymbol{f}$

In order to decompose how $\mathbb{P}\boldsymbol{f}$ effects the FSI equations, a quantitative metric for stiffness from this term is necessary. Here, spring forces and bending forces are used, as have been seen in application [**21, 68**]. The dependence of stiffness is linked to the spring constant, and in turn the spring constant scales directly with both the magnitude of $\mathbb{P}\boldsymbol{f}$,$|\mathbb{P}\boldsymbol{f}|$, and the magnitude of its gradient, $|\nabla\mathbb{P}\boldsymbol{f}|$, (see Figure 4.1). Thus it is reasonable to assume that these quantities can mirror the amount of stiffness in the ODE. The intent of splitting this term is to evaluate the stiff parts implicitly, and/or on a refined time step. Although adding another time scale will increase the computations, these will be computations on the Lagrangian domain, Γ, and thus are less expensive compared to the Eulerian evaluations on Ω. Additionally, it should be the case that the added cost is much less than the savings gained from increasing the maximum stable time step.

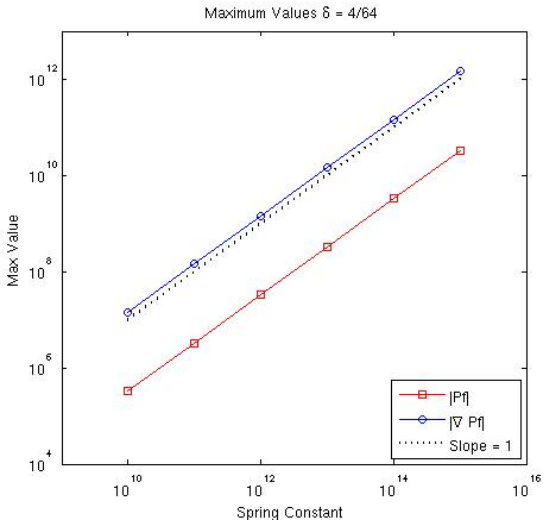


FIGURE 4.1. Scaling of $\mathbb{P}\boldsymbol{f}$ and $\nabla\mathbb{P}\boldsymbol{f}$ with the spring constant k

The splitting discussed in this section will be of the spatial domain ($\Gamma \cup \Omega$, with definitions on both Γ and Ω, and will consist of two domains, the *near field* and *far field*. The near field is taken as the compact area near the boundary and designed to capture most of the stiffness. The far field is a smooth representation of $\mathbb{P}\boldsymbol{f}$ on Ω near Γ and away from it does not change rapidly, thus can be considered non-stuff. Thus, $\mathbb{P}\boldsymbol{f}$ is decomposed into

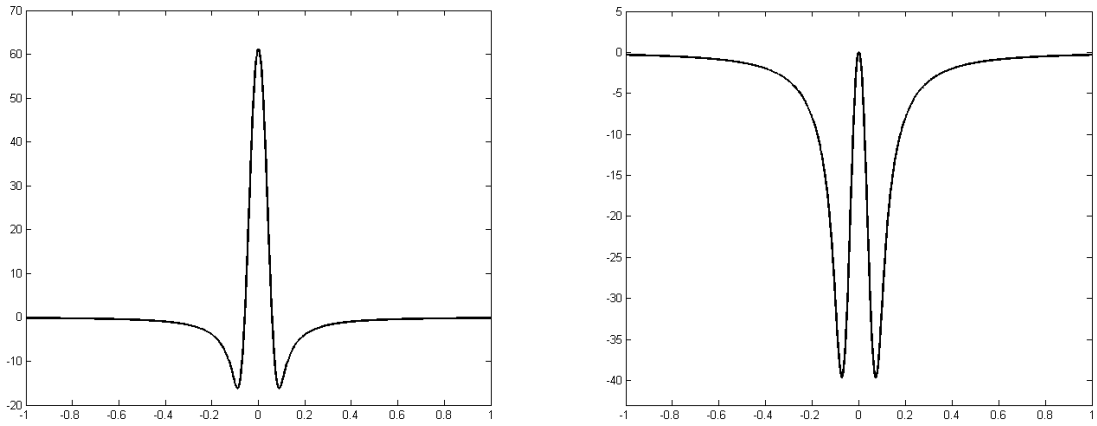
$$\mathbb{P}\boldsymbol{f} = \mathfrak{nf} + \mathfrak{ff}, \tag{4.1}$$

with

$$\text{near field}: \quad \mathfrak{nf} = \mathbb{P}\boldsymbol{f} - H_{\mathcal{R}}(r), \tag{4.2}$$

$$\text{far field}: \quad \mathfrak{ff} = \mathbb{P}\boldsymbol{f} - \mathfrak{nf} = H_{\mathcal{R}}(r), \tag{4.3}$$

where $H_{\mathcal{R}}(r)$ is an approximation function and $\mathcal{R}$ is the radius of the near field in terms of δ. That is, for a near field defined as the area within 3δ of each fiber point, $\mathcal{R} = 3$ and the corresponding approximation function is denoted, $H_{\mathcal{R}=3}(r)$. Note that both the near and far field are defined with the choice of $H_{\mathcal{R}}(r)$, thus this term will be examined with careful and detailed analysis, presented below.



(A) $p_1(r)$ (B) $p_2(r)$

FIGURE 4.2. The two radially symmetric terms of $\mathbb{P}\boldsymbol{f}$, for a fixed δ

4.1.1. Defining $H_{\mathcal{R}}(r)$. There are two main constraints that guide the construction of the near field. First, this quantity should capture the majority of the stiffness. Second, when it is used to construct the far field, $\mathfrak{ff} = \mathbb{P}\boldsymbol{f} - \mathfrak{nf}$, the subtraction of the local $\mathfrak{nf}$ from the global $\mathbb{P}\boldsymbol{f}$, should not leave any analytic discontinuities.

As discussed in section 2.3, BPM analytically defines $\mathbb{P}\boldsymbol{f}$ at any arbitrary point $\boldsymbol{x}$ on the grid, using the following sum over the N_Γ Lagrangian fiber points, with corresponding position $\boldsymbol{X}$. This is given here again for reference, for a fiber $\boldsymbol{X} \in \mathbb{R}^2$ or $\boldsymbol{X} \in \mathbb{R}^3$with force $\boldsymbol{f} \in \mathbb{R}^3$

$$\mathbb{P}\boldsymbol{f} = \sum_{\gamma=0}^{N_\Gamma} \left(\frac{r\mathbb{F}'(r) - \mathbb{F}(r)}{2\pi r^2} \right) \boldsymbol{f}_\gamma \Delta l_- \left(\frac{r\mathbb{F}'(r) - 2\mathbb{F}(r)}{2\pi r^2} \right) \hat{\boldsymbol{x}}_\gamma (\boldsymbol{f}_\gamma \Delta l . \hat{\boldsymbol{x}}_\gamma), \tag{4.4}$$

$$\text{where } \mathbb{F}(r) = 2\pi \int_0^r s B_\delta(s)\, ds, \quad r = |\boldsymbol{x} - \boldsymbol{X}_\gamma|, \quad \hat{\boldsymbol{x}} = (\boldsymbol{x} - \boldsymbol{X}_\gamma)/r.$$

To define $H_{\mathcal{R}}(r)$, first denote each radially symmetric term in (4.4), as $p_1(r)$ and $p_2(r)$, where

$$\mathbb{P}\boldsymbol{f} = \sum_{\gamma=0}^{N_\Gamma} p_1(r) \boldsymbol{f}_\gamma \Delta l - p_2(r) \hat{\boldsymbol{x}}_\gamma (\boldsymbol{f}_\gamma \Delta l . \hat{\boldsymbol{x}}_\gamma),$$

and

$$p_1(r) = \frac{r\mathbb{F}'(r) - \mathbb{F}(r)}{2\pi r^2}, \qquad p_2(r) = \frac{r\mathbb{F}'(r) - 2\mathbb{F}(r)}{2\pi r^2}. \tag{4.5}$$

These are shown in Fig. 4.2 and gives the following general form of $H_{\mathcal{R}}(r)$:

$$H_{\mathcal{R}}(r) = \begin{cases} H_1(r)\boldsymbol{f}_\gamma \Delta l - H_2(r)\hat{\boldsymbol{x}}_\gamma(\boldsymbol{f}_\gamma \Delta l \cdot \hat{\boldsymbol{x}}_k) & \text{if } \frac{r}{\delta} > \mathcal{R}, \\ \mathbb{P}\boldsymbol{f} & \text{if } \frac{r}{\delta} \leq \mathcal{R}. \end{cases} \tag{4.6}$$

We choose $H_1(r)$ and $H_2(r)$ to be fourth degree polynomials of form

$$H_{1,2}(r) = a_{1,2} + b_{1,2}\left(\left(\frac{r}{\delta}\right)^2 - (\mathcal{R})^2\right) + c_{1,2}\left(\left(\frac{r}{\delta}\right)^2 - (\mathcal{R})^2\right)^2, \tag{4.7}$$

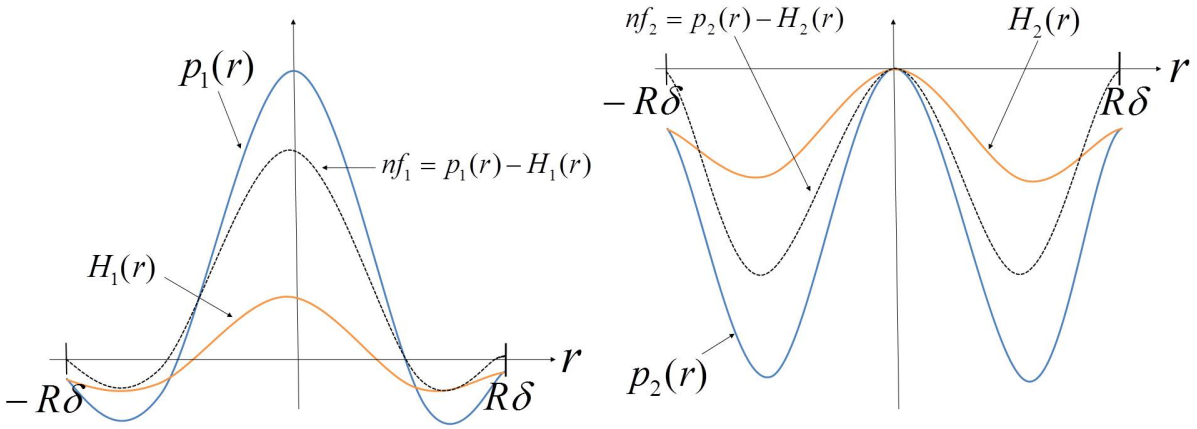
with coefficients $a_{1,2}, b_{1,2}, c_{1,2}$ for $H_1(r)$ and $H_2(r)$ respectively. These are constructed by the polynomial approximation technique of matching conditions. $H_1(r)$ will be constructed matching the conditions of $p_1(r)$ and $H_2(r)$ matching $p_2(r)$, which are given in detail in the following sections. This definition of $H_{\mathcal{R}}(r)$ gives the following definition of the near field, which is indeed compact and local.

$$nf = \begin{cases} \mathbb{P}\boldsymbol{f} - H_{\mathcal{R}}(r) & \text{if } \frac{r}{\delta} \leq \mathcal{R}, \\ 0 & \text{if } \frac{r}{\delta} > \mathcal{R}. \end{cases} \tag{4.8}$$

An example of $H_1(r)$, $p_1(r)$, $H_2(r)$, $p_1(r)$, $p_2(r)$ and their the resulting near field components are depicted in Figure 4.3.

4.1.2. Polynomial Approximation. To ensure that the near field is local, the matching value

$$H_{1,2}(\mathcal{R}\delta) = p_{1,2}(\mathcal{R}\delta) \tag{4.9}$$



(A) $H_1(r), p_1(r)$ (B) $H_2(r), p_2(r)$

FIGURE 4.3. Components of the splitting function

is always used. To define the other matching conditions, the following parameters are needed for both $H_1(r)$ and $H_2(r)$

(4.10) $\mathcal{R}$: the ratio of the radius of the near field and δ from each point on Γ,

(4.11) α_1 : matching point where $H_{1,2}(\alpha_1) = p_{1,2}(\alpha_1)$, respectively,

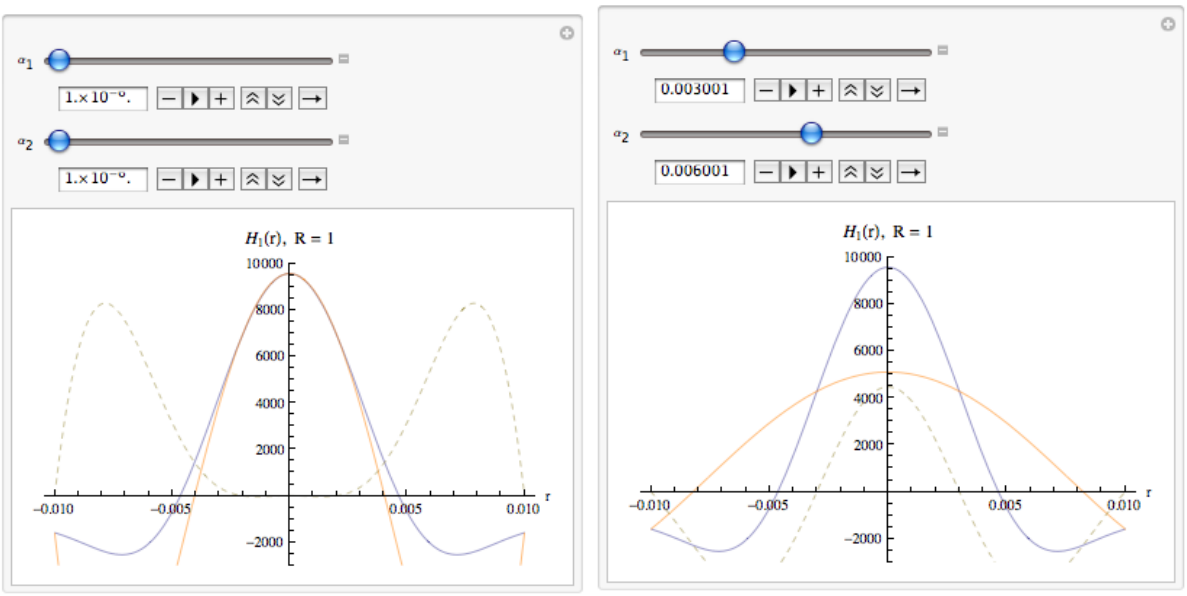
(4.12) α_2 : matching point where $H'_{1,2}(\alpha_2) = p'_{1,2}(\alpha_2)$, respectively.

To explore this parameter space, a *Mathematica* program was created, where $\mathcal{R}$, α_1 and α_2 were variables. The range for α_1 and α_2 was from 0 to $\mathcal{R}\delta$, stepping by increments of $\frac{1}{10}\mathcal{R}\delta$ and for visual purposes $\delta = 1/100$ was used to how α_1 and α_2 effected the scaling of the $\mathfrak{nf}$. Figures 4.4 - 4.5 show the result of this program with $\mathcal{R} = 1$. In these figures, the orange line represents $H_{1,2}(r)$, the blue line is $p_{1,2}(r)$ and the dotted line is the resulting near field component $\mathfrak{nf}_{1,2}$. Additional figures for different cases of $\mathcal{R}$ can be found in the Appendix.

The dependence of the computational cost of the near field on $\mathcal{R}$ should be noted. The near field each is evaluated at each point defining Γ, $\boldsymbol{X}_\gamma$, where the contribution from the points $\boldsymbol{X}_{\tilde{\gamma}}$, such that

$$||\boldsymbol{X}_\gamma - \boldsymbol{X}_{\tilde{\gamma}}|| \leq \mathcal{R}\delta.$$

As $\mathcal{R}$ increases, a larger percentage of points $\boldsymbol{X}_{\tilde{\gamma}}/N_\Gamma$ contributes to this sum, thus the computational cost increases. This is revisited later when talking about specific techniques, but should be mentioned in context of an analysis of the near field.

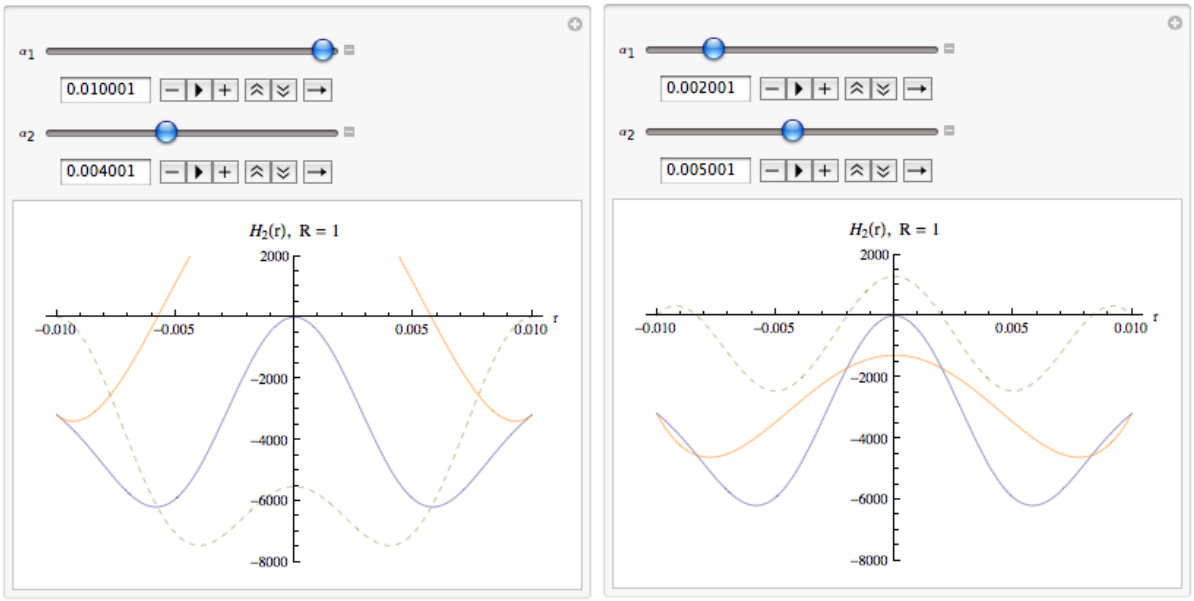


(A) $\alpha_1 = 0$ and $\alpha_2 = 0$ (B) $\alpha_1 = 0.3\delta$ and $\alpha_2 = 0.6\delta$

FIGURE 4.4. $H_1(r)$ with $\mathcal{R} = 1$ and labeled matching conditions

For each $\mathcal{R}$, four sets of matching conditions were chosen to compare. The choice was made for those values of α_1 and α_2 that captured the majority of the magnitude of the near field, while reducing the magnitude of the far field. These new values, as well as the coefficients $a_{1,2}, b_{1,2}, c_{1,2}$ are given in Tables 4.1 and 4.2 and graphs of these conditions with $\mathcal{R} = 1$ are given in Figures 4.6 and 4.7. Graphs of $\mathcal{R} = 2, 3$ with the conditions listed in Tables 4.1 and 4.2, respectively, can be found in the Appendix.

The ratio of $|\mathbb{P}\boldsymbol{f}|/|\mathbb{f}\mathbb{f}|$ and $|\nabla\mathbb{P}\boldsymbol{f}|/|\nabla\mathbb{f}\mathbb{f}|$ is considered with all the possible coefficients for varying grid sizes and δ's, in two dimensions. To do so, a perturbed ellipse test problem, as seen in Cortez and Minion, is used [**21**]. This is chosen because forces are generated immediately in the system, allowing for the relative magnitude of the near and far field to be determined quickly.



(A) $\alpha_1 = \delta$ and $\alpha_2 = 0.4\delta$ (B) $\alpha_1 = 0.2\delta$ and $\alpha_2 = 5\delta$

FIGURE 4.5. $H_2(r)$ with $\mathcal{R} = 1$ and labeled matching conditions

Initially the ellipse, immersed in a fluid at rest, is defined by the parametrization

$$\boldsymbol{X}(\theta) = (r(\theta)\cos(\theta - \theta_0), r(\theta)\sin(\theta - \theta_0)), \tag{4.13}$$

where

$$r^2(\theta) = a^2\cos^2(\theta) + b^2\sin(\theta) + \epsilon\left(-\frac{4}{3}e^{-3(\theta-\pi)^2} - e^{-5(\theta-\theta_1)^2} + e^{-8(\theta-\theta_2)^2}\right).$$

The parameters used are $a = 0.2, b = 0.25, \epsilon = 0.012, \theta_0 = \pi/3, \theta_1 = 4\pi/5$, and $\theta_2 = 2\pi/3$. The grid is subdivided into $N_x = N_y = 128$ grid cells and the immersed boundary is discretized by $N_\Gamma = 400$ points, unless otherwise noted. Forces are defined by linear springs, with resting length zero and spring constant $k_s = 10^{10}$, attaching each point on the immersed boundary.

SDC is used to preform the time integration with $K = 3$ SDC sweeps and $M = 5$ SDC nodes. The time step used is $\Delta t_n = 0.00006$, and the simulation evolves to $T = 0.0006s$. The choice of time scale and k_s are arbitrary, as using a smaller spring constant and

TABLE 4.1. Coefficients for $H_1(r)$

	$H_1(r)$	a_1	b_1	c_1	Figure
	$\mathcal{R} = 1$				
(i)	$\alpha_1 = 0$ $\alpha_2 = \delta$	$-\frac{1}{2\pi\delta^2}$	$\frac{1}{2\pi\delta^2}$	$\frac{1}{\pi\delta^2}$	4.6a
(ii)	$\alpha_1 = 0.6\delta$ $\alpha_2 = \delta$	$-\frac{1}{2\pi\delta^2}$	$\frac{1}{2\pi\delta^2}$	$\frac{13031}{31250\pi\delta^2}$	4.6b
(iii)	$\alpha_1 = 0.5\delta$ $\alpha_2 = \delta$	$-\frac{1}{2\pi\delta^2}$	$\frac{1}{2\pi\delta^2}$	$\frac{149}{128\pi\delta^2}$	4.6c
(iv)	$\alpha_1 = 0.9\delta$ $\alpha_2 = 0.7\delta$	$-\frac{1}{2\pi\delta^2}$	$\frac{13170879083}{16600000000\pi\delta^2}$	$\frac{20107253}{20750000\pi\delta^2}$	4.6d
	$\mathcal{R} = 2$				
(i)	$\alpha_1 = 0$ $\alpha_2 = 2\delta$	$-\frac{1}{8\pi\delta^2}$	$\frac{1}{32\pi\delta}$	$\frac{1}{64\pi\delta}$	9.5a
(ii)	$\alpha_1 = 0.8(2\delta)$ $\alpha_2 = 0.9(2\delta)$	$-\frac{1}{8\pi\delta^2}$	$\frac{26275}{373248\pi\delta^2}$	$\frac{100625}{6718464\pi\delta^2}$	9.5b
(iii)	$\alpha_1 = .9(2\delta)$ $\alpha_2 = 2\delta$	$-\frac{1}{8\pi\delta^2}$	$\frac{1}{32\pi\delta}$	$-\frac{25}{2592\pi\delta^2}$	9.5c
(iv)	$\alpha_1 = 0.2(2\delta)$ $\alpha_2 = 2\delta$	$-\frac{1}{8\pi\delta^2}$	$\frac{1}{32\pi\delta}$	$\frac{6311441}{128000000\pi\delta}$	9.5d
	$\mathcal{R} = 3$				
(i)	$\alpha_1 = 0$ $\alpha_2 = 3\delta$	$-\frac{1}{18\pi\delta^2}$	$\frac{1}{162\pi\delta^2}$	$\frac{1}{729\pi\delta^2}$	9.6a
(ii)	$\alpha_1 = .5(3\delta)$ $\alpha_2 = 3\delta$	$-\frac{1}{18\pi\delta^2}$	$\frac{1}{162\pi\delta^2}$	$-\frac{2}{729\pi\delta^2}$	9.6b
(iii)	$\alpha_1 = .9(3\delta)$ $\alpha_2 = 3\delta$	$-\frac{1}{18\pi\delta^2}$	$\frac{1}{162\pi\delta^2}$	$-\frac{50}{59049\pi\delta^2}$	9.6c
(iv)	$\alpha_1 = 0.7(3\delta)$ $\alpha_2 = 0.6(3\delta)$	$-\frac{1}{18\pi\delta^2}$	$-\frac{175025}{16503102\pi\delta^2}$	$-\frac{1126250}{222791877\pi\delta^2}$	9.6d

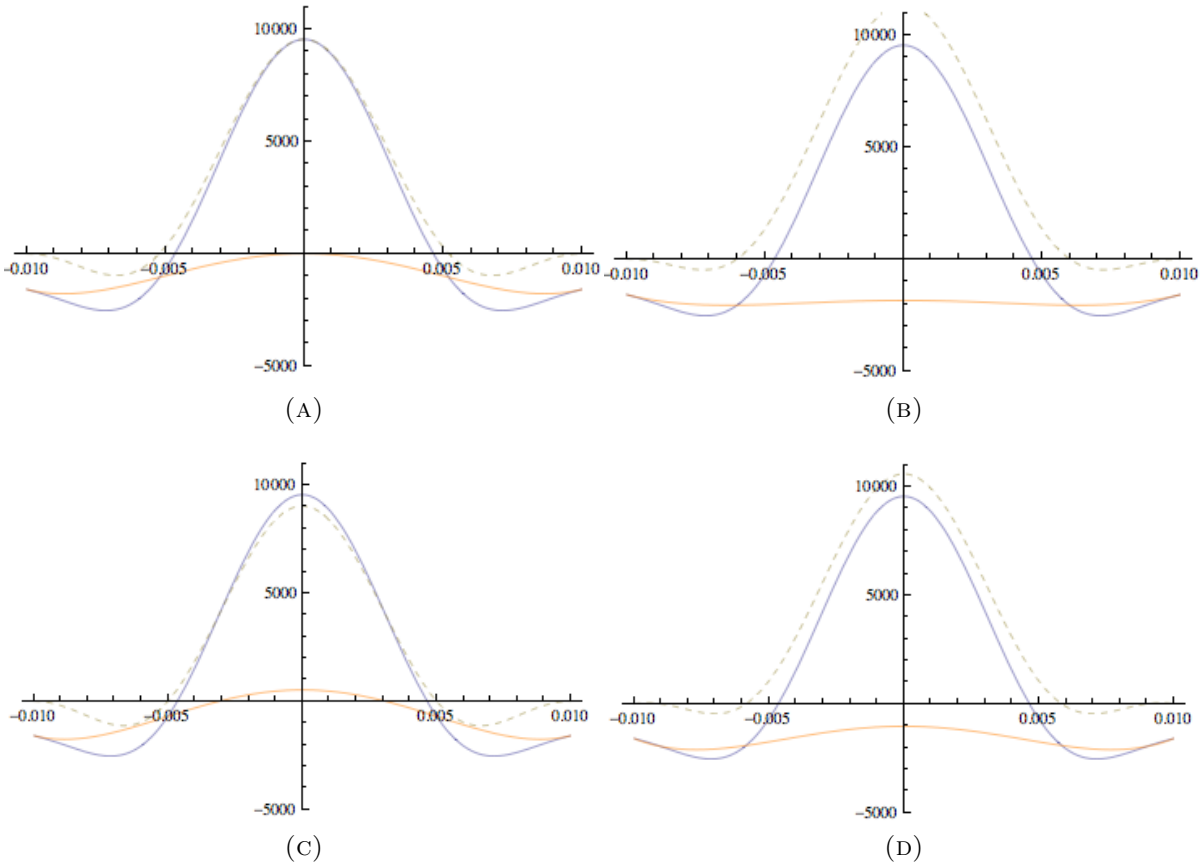


FIGURE 4.6. $H_1(r)$ with $\mathcal{R} = 1$ and matching conditions (i)-(iv), given in Table 4.1

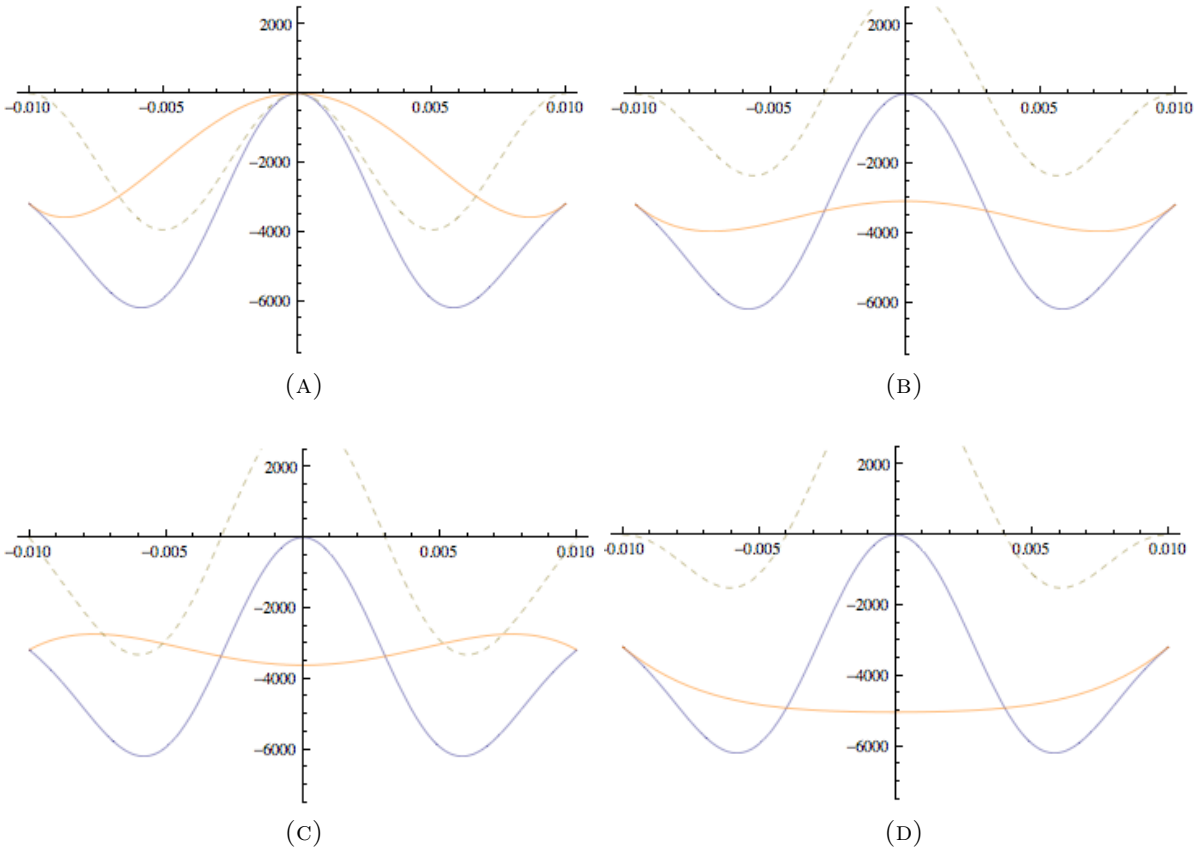
longer time scale would produce the same problem. The initial configuration and the motion captured at $T = 0.0006s$ later are shown in Figure 4.8.

Two examples of the coefficient comparison tests are shown in Figures 4.9 and 4.10, more are seen in the Appendix. In these figures, the x axis is an index given by the condition case, C, such that if the cases for $H_1(r)$ are notated $c_1 = 1, 2, 3, 4$, corresponding to cases i,ii,iii,iv, and similarly for $H_2(r)$, $c_2 = 1, 2, 3, 4$, then $C = c_2 + 4(c_1 - 1)$. The optimal coefficients, those that produce the largest ratio of $|\mathbb{P}\boldsymbol{f}|/|\mathfrak{f}\mathfrak{f}|$ and $|\nabla\mathbb{P}\boldsymbol{f}|/|\nabla\mathfrak{f}\mathfrak{f}|$, are reported in Tables 4.3 - 4.6 for different cases, where Δx is the grid size.

Tables 4.3 and 4.4 show that α_1 does not vary with change in grid size, and does not vary across $\mathcal{R}$ at all, with the exception of $\delta = 2\Delta x$ on a 256 grid. This is not an unusual case to see a difference because a blob radius of that size is more singular and not well

TABLE 4.2. Coefficients for $H_2(r)$

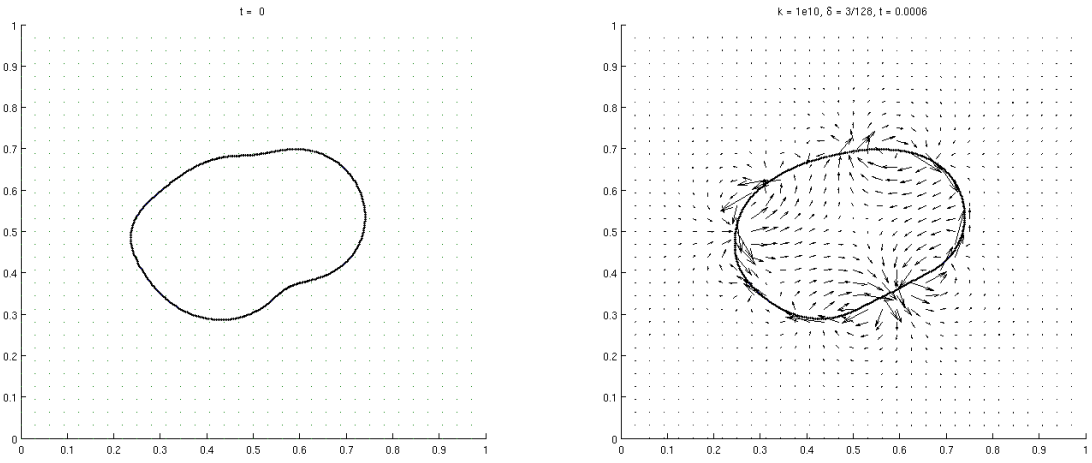
$H_2(r)$		a_2	b_2	c_2	Figure
$\mathcal{R}=1$					
(i)	$\alpha_1=0$, $\alpha_2=\delta$	$-\frac{1}{\pi\delta^2}$	$\frac{1}{\pi\delta^2}$	$\frac{2}{\pi\delta^2}$	4.7a
(ii)	$\alpha_1=0.3\delta$, $\alpha_2=\delta$	$-\frac{1}{\pi\delta^2}$	$\frac{1}{\pi\delta^2}$	$\frac{205951}{200000\pi\delta^2}$	4.7b
(iii)	$\alpha_1=0.3\delta$, $\alpha_2=0.6\delta$	$-\frac{1}{\pi\delta^2}$	$-\frac{3903971}{5781250\pi\delta^2}$	$-\frac{6002981}{7400000\pi\delta^2}$	4.7c
(iv)	$\alpha_1=0.4\delta$, $\alpha_2=\delta$	$-\frac{1}{\pi\delta^2}$	$\frac{1}{\pi\delta^2}$	$\frac{1306}{3125\pi\delta^2}$	4.7d
$\mathcal{R}=2$					
(i)	$\alpha_1=0$, $\alpha_2=2\delta$	$-\frac{1}{4\pi\delta^2}$	$\frac{1}{16\pi\delta^2}$	$\frac{1}{32\pi\delta^2}$	9.7a
(ii)	$\alpha_1=0.9(2\delta)$, $\alpha_2=\delta$	$-\frac{1}{4\pi\delta^2}$	$\frac{1}{16\pi\delta^2}$	$-\frac{25}{1296\pi\delta^2}$	9.7b
(iii)	$\alpha_1=0.5(2\delta)$, $\alpha_2=0.5(2\delta)$	$-\frac{1}{4\pi\delta^2}$	$-\frac{1}{2\pi\delta^2}$	$-\frac{1}{4\pi\delta^2}$	9.7c
(iv)	$\alpha_1=0.1(2\delta)$, $\alpha_2=0.3(2\delta)$	$-\frac{1}{4\pi\delta^2}$	$-\frac{38964269}{142656250\pi\delta^2}$	$-\frac{997349}{11412500\pi\delta^2}$	9.7d
$\mathcal{R}=3$					
(i)	$\alpha_1=0$, $\alpha_2=3\delta$	$-\frac{1}{9\pi\delta}$	$\frac{1}{81\pi\delta^2}$	$\frac{2}{729\pi\delta^2}$	9.8a
(ii)	$\alpha_1=0.9(3\delta)$, $\alpha_2=3\delta$	$-\frac{1}{9\pi\delta}$	$\frac{1}{81\pi\delta^2}$	$-\frac{100}{59049\pi\delta^2}$	9.8b
(iii)	$\alpha_1=0.5(3\delta)$, $\alpha_2=0.5(3\delta)$	$-\frac{1}{9\pi\delta}$	$-\frac{8}{81\pi\delta^2}$	$-\frac{16}{729\pi\delta^2}$	9.8c
(iv)	$\alpha_1=0.7(3\delta)$, $\alpha_2=0.7(3\delta)$	$-\frac{1}{9\pi\delta}$	$-\frac{200}{194481\pi\delta^2}$	$-\frac{10000}{1750329\pi\delta^2}$	9.8d



(A) (B)

(C) (D)

FIGURE 4.7. $H_2(r)$ with $\mathcal{R} = 1$ and matching conditions (i)-(iv), given in Table 4.2



(A) $t = 0$ (B) $t = 0.0006$

FIGURE 4.8. Perturbed Ellipse.

resolved. For the case of α_2, it has less uniform behavior. With large near field areas, such as $\mathcal{R} = 2$ and $\delta = 4\Delta x$ on a 128×128 grid, or $\mathcal{R} = 3$ and $\delta = 4\Delta x$ on a 256×256 grid, the optimal α_2 almost always condition (i).

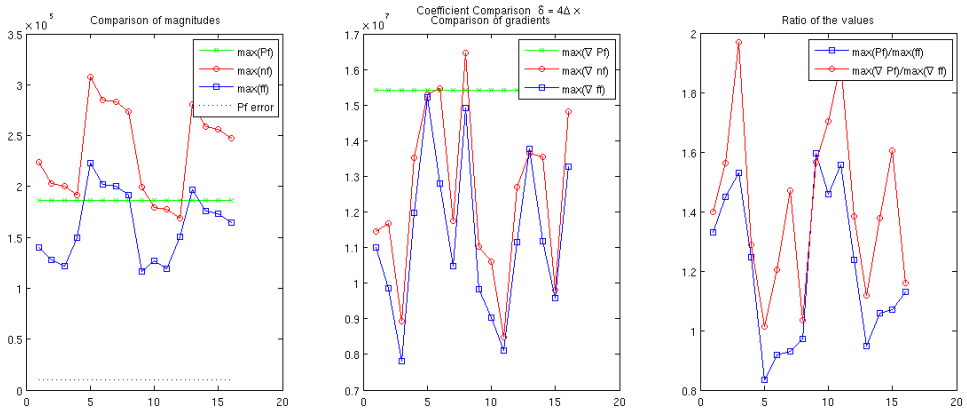


FIGURE 4.9. Coefficients with 128×128 grid, $N_\Gamma = 400$ with $\delta = 4/128$, and $\mathcal{R} = 1$

Tables 4.5 and 4.6 show different grid sizes vs. gain from the optimal coefficients, measured by the ratio of $|\mathbb{P}\boldsymbol{f}/\mathfrak{f}\mathfrak{f}|$ and $|\nabla\mathbb{P}\boldsymbol{f}/\nabla\mathfrak{f}\mathfrak{f}|$, with a δ that varied with grid size (Table 4.5), and with fixed $\delta = 4/128$ (Table 4.6). Table 4.5 shows that the blob radius does not have a large effect on the on the gain from the splitting. Conversely, Table 4.12 shows that the gain achieved is proportional to the near field size, which would mean it

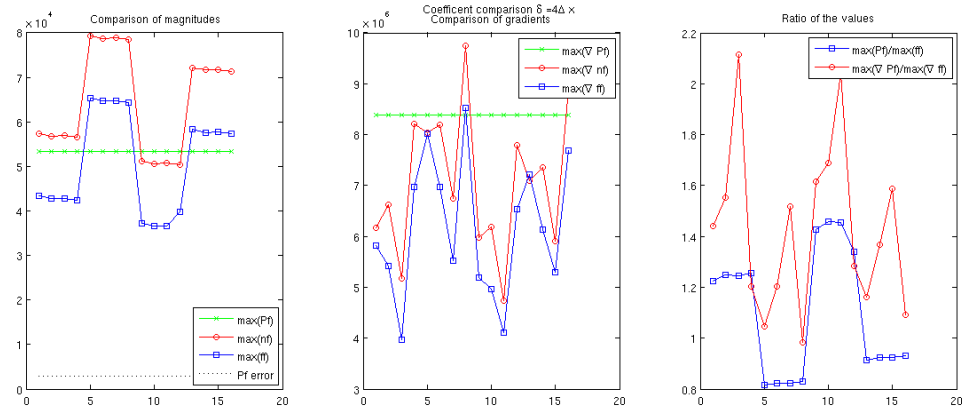


FIGURE 4.10. Coefficiens with 256×256 grid with $\delta = 4/256$, $N_\Gamma = 800$ and $\mathcal{R} = 1$

TABLE 4.3. Optimal coefficients for $H_1(r)$, $H_2(r)$ on a 128 x 128 grid

	$\delta = 2\Delta x$	$\delta = 3\Delta x$	$\delta = 4\Delta x$	$\delta = 8\Delta x$
$\mathcal{R} = 1$	α_1: (iii)	α_1 : (iii)	α_1: (iii)	α_1 : (iii)
	α_2: (iii)	α_2 : (i)	α_2: (i)	α_2 : (i)
$\mathcal{R} = 2$	α_1 : (i)	α_1 : (i)	α_1 : (i)	α_1 : (i)
	α_2: (iv)	α_2 : (iv)	α_2 : (i)	α_2 : (i)
$\mathcal{R} = 3$	α_1 : (i)	α_1 : (i)	α_1 : (i)	α_1: (i)
	α_2 : (ii)	α_2 : (i)	α_2 : (i)	α_2 : (i)

TABLE 4.4. Optimal coefficients for $H_1(r)$, $H_2(r)$ on a 256 x 256 grid

	$\delta = 2\Delta x$	$\delta = 3\Delta x$	$\delta = 4\Delta x$	$\delta = 8\Delta x$
$\mathcal{R} = 1$	α_1: (i)	α_1 : (iii)	α_1: (iii)	α_1 : (iii)
	α_2: (ii)	α_2 : (iii)	α_2: (ii)	α_2 : (iii)
$\mathcal{R} = 2$	α_1 : (ii)	α_1 : (i)	α_1 : (i)	α_1 : (i)
	α_2: (ii)	α_2 : (iv)	α_2 : (iv)	α_2 : (i)
$\mathcal{R} = 3$	α_1 : (i)	α_1 : (i)	α_1 : (i)	α_1: (i)
	α_2 : (ii)	α_2 : (ii)	α_2 : (i)	α_2 : (i)

does depend on $\mathcal{R}$. This is to expected since a larger near field captures more effect from $\mathbb{P}\boldsymbol{f}$.

TABLE 4.5. Optimal coefficients for $H_1(r)$, $H_2(r)$ for $\delta = 4\Delta x$

N_x	64	128	256	512
$\mathcal{R} = 1$	α_1: (iii)	α_1 : (iii)	α_1: (iii)	α_1 : (iii)
	α_2: (i)	α_2 : (i)	α_2: (ii)	α_2 : (ii)
$\mathcal{R} = 2$	α_1 : (i)	α_1 : (i)	α_1 : (i)	α_1 : (i)
	α_2: (i)	α_2 : (i)	α_2 : (iv)	α_2 : (iv)
$\mathcal{R} = 3$	α_1 : (i)	α_1 : (i)	α_1 : (i)	α_1: (i)
	α_2 : (ii)	α_2 : (ii)	α_2 : (ii)	α_2 : (ii)

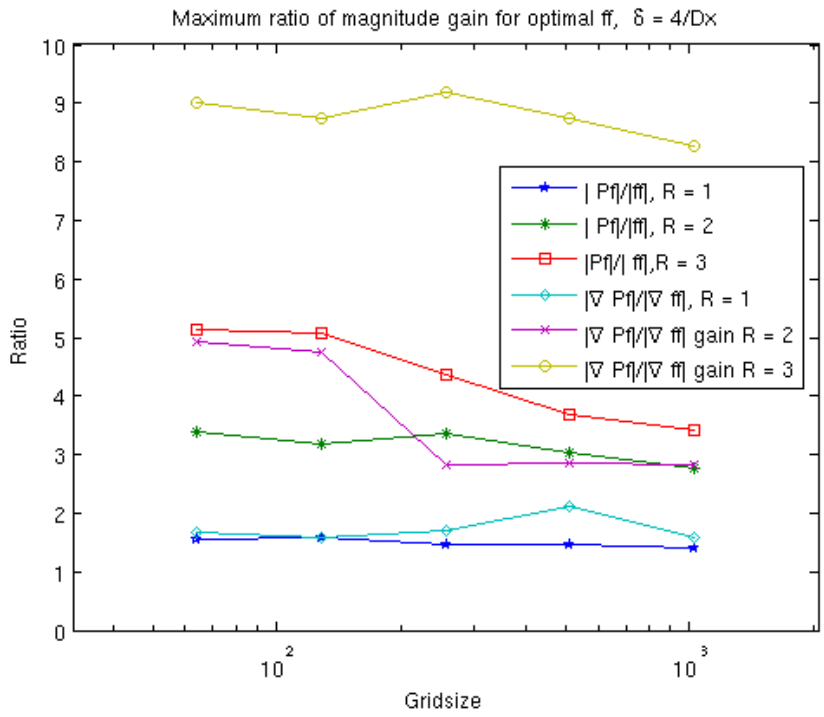


FIGURE 4.11. Grid size vs. Ratio of $|\mathbb{P}\boldsymbol{f}/\mathfrak{f}\mathfrak{f}|$ and $|\nabla\mathbb{P}\boldsymbol{f}/\nabla\mathfrak{f}\mathfrak{f}|$, $\delta = 4/128$

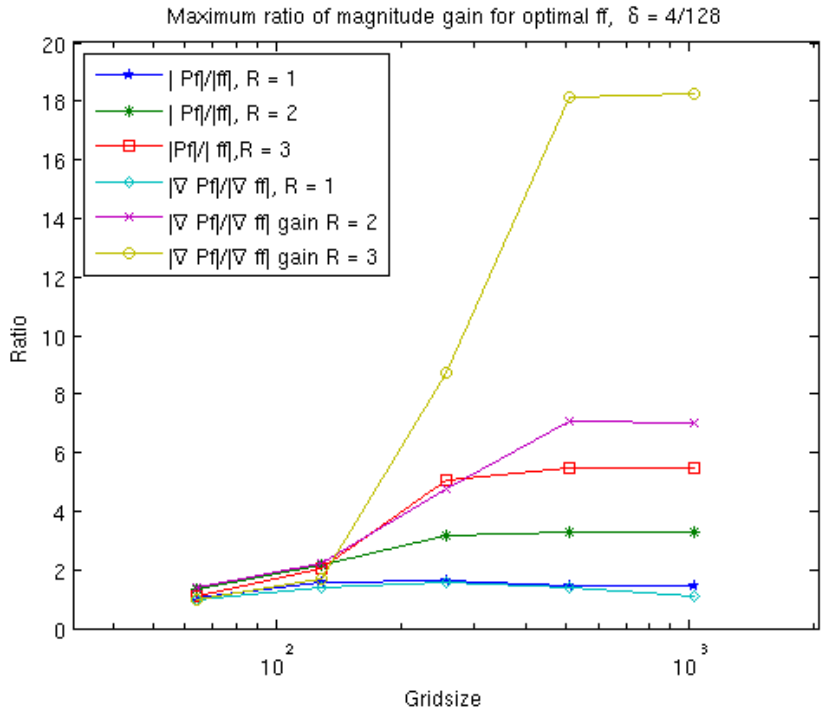


FIGURE 4.12. Grid size vs. Ratio of $|\mathbb{P}\boldsymbol{f}/\mathfrak{f}\mathfrak{f}|$ and $|\nabla\mathbb{P}\boldsymbol{f}/\nabla\mathfrak{f}\mathfrak{f}|$, $\delta = 4\Delta x$

TABLE 4.6. Optimal coefficients for $H_1(r)$, $H_2(r)$ for $\delta = 4/128$

N_x	64	128	256	512
$\mathcal{R}=1$	α_1: (iii)	α_1 : (iii)	α_1: (iii)	α_1 : (iii)
	α_2: (iv)	α_2 : (iii)	α_2: (i)	α_2 : (iii)
$\mathcal{R}=2$	α_1 : (ii)	α_1 : (i)	α_1 : (i)	α_1 : (i)
	α_2: (iii)	α_2 : (iv)	α_2 : (i)	α_2 : (i)
$\mathcal{R}=3$	α_1 : (iii)	α_1 : (i)	α_1 : (i)	α_1: (i)
	α_2 : (iii)	α_2 : (ii)	α_2 : (i)	α_2 : (i)

The choice of coefficients for $H_1(r)$ and $H_2(r)$ in the implementation will be remarked on for the different new strategies in the next three chapters. The last consideration to make is the evaluation of the near field quantities on the two domains, Γ and Ω. The near field on the grid is evaluated by spreading the near field equations to the grid. That is, for each fiber point, $\boldsymbol{X}$, the near field is evaluated analytically at the grid points, $\boldsymbol{x}$ within its radius, $\mathcal{R}$, where $r = |\boldsymbol{X} - \boldsymbol{x}|$ determines the strength of the near field terms.

For the Lagrangian fiber points,the near field can be evaluated directly at each fiber point. At each $\boldsymbol{X}_\gamma \in N_\Gamma$, adding the contributions from all the other points is straight forward, however the force from $\boldsymbol{X}_\gamma$ at $\boldsymbol{X}_\gamma$, causes $p_{1,2}(r)$ to become singular and therefore the limit is used. This limit as $r \to 0$ of (4.4) is,

$$p_1(r=0) = \lim_{r->0} \frac{r\mathbb{F}'(r) - \mathbb{F}(r)}{2\pi r^2} = \frac{3}{\pi\delta^2}, \tag{4.14}$$

$$p_2(r=0) = \lim_{r->0} \frac{r\mathbb{F}'(r) - 2\mathbb{F}(r)}{2\pi r^2} = 0. \tag{4.15}$$

This concludes the spatial splitting discussion.

CHAPTER 5

Multi-implicit techniques

With the background methodology and spatial splitting discussed in the previous chapters, the new multi-implicit approach can now be presented. Since the stiffness in the IBM equations are a result of the forces generated on the boundary, handling these implicitly could increase the largest stable time step. With the spatial splitting presented in the previous chapter, these forces can be split into near and far field effects, allowing for a local definition of the stiffest part of the forces. Thus, the idea presented in this chapter is to combine the spatial splitting with a multi-implicit algorithm, where the stiff terms are defined on Γ and evaluated implicitly.

The numerical results that are presented in this section will use BPM and SDC together with a new multi-implicit implementation. In Cortez and Minion, the original BPM implementation, all terms in the fluid time update are evaluated explicitly with a standard Runge-Kutta algorithm [**21**]. Here, a new, slightly different algorithm is used for comparison and called the *standard algorithm*.

5.1. The Standard Algorithm

To define this standard algorithm, again the MOL is used. Recall the time evolution equations for BPM,

$$\boldsymbol{u}_t(\boldsymbol{x},t) = \mathbb{P}\left((-\boldsymbol{u}(\boldsymbol{x},t)\cdot\nabla)\,\boldsymbol{u}(\boldsymbol{x},t) + \frac{1}{Re}\nabla^2\boldsymbol{u}(\boldsymbol{x},t)\right) + \mathbb{P}\boldsymbol{f}(\boldsymbol{x},t), \tag{5.1}$$

$$\boldsymbol{X}_t = \boldsymbol{U}\left(\boldsymbol{X},t\right). \tag{5.2}$$

From here on, the following notation will be used for spatially discretized advection and diffusion term in the Navier Stokes Equations (N-S),

$$\mathbb{A}_{i,j,k} = -\left(\boldsymbol{u}(\boldsymbol{x}_{i,j,k},t)\cdot\nabla_h\right)\boldsymbol{u}(\boldsymbol{x}_{i,j,k},t), \tag{5.3}$$

$$\mathbb{D}_{i,j,k} = \frac{1}{Re}\nabla_h^2\boldsymbol{u}(\boldsymbol{x}_{i,j,k},t). \tag{5.4}$$

Again, ∇_h and ∇_h^2 are the discrete operators of their continuous counterparts. In addition, it is assumed that the velocity $\boldsymbol{U}(\boldsymbol{X}_{q,r,s},t)$ at point $\boldsymbol{X}_{q,r,s}$ is the interpolation of the surrounding $\boldsymbol{u}(\boldsymbol{x}_{i,j,k},t)$'s at time t.

To discretize in time, the time interval $[0,T]$ is divided into N sub-steps, divided further into M SDC nodes, as given in Section 3.1. Additionally, let the following notation simplifications be made, $\boldsymbol{u}(\boldsymbol{x},t_m)=\boldsymbol{u}_m$, $\boldsymbol{f}(\boldsymbol{X},t_m)=\boldsymbol{f}_m$ and $\boldsymbol{U}(\boldsymbol{x},t_m)=\boldsymbol{U}_m$. The multi-implicit technique presented here, as well as the multi-rate and time parallel methods, mainly deal with methods of temporal integration, thus unless otherwise specified, spatial indices's are assumed and not written.

The standard algorithm is constructed with an first order, IMEX scheme, where diffusion is evaluated implicitly. Diffusion is sometimes considered a stiff term, however with IBM, it seems to have less of an effect on stability than $\mathbb{P}\boldsymbol{f}$ [**83, 82**]. Nevertheless, this will be evaluated implicitly here to focus the stability discussion solely on $\mathbb{P}\boldsymbol{f}$. This approach results in the following zero-th order SDC iteration, or predictor, along with the corresponding SDC corrector.

5.1.1. Predictor.

$$\boldsymbol{u}_{m+1}^{[0]} = \boldsymbol{u}_m^{[0]} + \Delta t_m\mathbb{P}\left(\mathbb{A}_m^{[0]}+\mathbb{D}_{m+1}^{[0]}\right)+\Delta t_m\mathbb{P}\boldsymbol{f}_m^{[0]}, \tag{5.5}$$

$$\boldsymbol{X}_{m+1}^{[0]} = \boldsymbol{X}_m^{[0]} + \Delta t_m\boldsymbol{U}_m^{[0]}. \tag{5.6}$$

5.1.2. Corrector.

$$\begin{aligned} \boldsymbol{u}_{m+1}^{[k+1]} &= \boldsymbol{u}_m^{[k+1]} + \Delta t_m \mathbb{P}\left(\mathbb{A}_m^{[k+1]} - \mathbb{A}_m^{[k]} + \mathbb{D}_{m+1}^{[k+1]} - \mathbb{D}_{m+1}^{[k]}\right) \\ &+ \Delta t_m \left(\mathbb{P}\boldsymbol{f}_m^{[k+1]} - \mathbb{P}\boldsymbol{f}_m^{[k]}\right) + \mathbb{I}_m^{m+1}\mathbb{P}\left(\mathbb{A}^{[k]} + \mathbb{D}^{[k]}\right) + \mathbb{P}\boldsymbol{f}^{[k]}, \end{aligned} \tag{5.7}$$

$$\boldsymbol{X}_{m+1}^{[k+1]} = \boldsymbol{X}_m^{[k+1]} + \Delta t_m \left(\boldsymbol{U}_m^{[k+1]} - \boldsymbol{U}_m^{[k]}\right) + \mathbb{I}_m^{m+1}\boldsymbol{U}^{[k]}. \tag{5.8}$$

From here on the standard algorithm will refer to the algorithms given in 5.1.1 and 5.1.2.

5.2. Multi-Implicit Algorithm

The purpose of the multi implicit technique is to use implicit methods for the stiff terms, along with the splitting $\mathbb{P}\boldsymbol{f}$, presented in Chapter 4.

In the fluid equation (5.1), the stiff terms are $\mathbb{P}\boldsymbol{f}$ and $\mathbb{D}$. Thus the IMEX fluid update is given by

$$\boldsymbol{u}_{m+1} = \boldsymbol{u}_m + \Delta t_m \mathbb{P}\left(\mathbb{A}_m + \mathbb{D}_{m+1}\right) + \Delta t_m \mathbb{P}\boldsymbol{f}_{m+1}, \tag{5.9}$$

where the only actual implicit evaluation is $\mathbb{D}_{m+1}$, as $\mathbb{P}\boldsymbol{f}_{m+1}$ is not explicitly dependent on $\boldsymbol{u}_{m+1}$, but computed from the $\boldsymbol{X}_{m+1}$ position. To construct $\boldsymbol{X}_{m+1}$, the fluid velocity is also evaluated implicitly, keeping consistent with (5.9). That is,

$$\boldsymbol{X}_{m+1} = \boldsymbol{X}_m + \Delta t_m \boldsymbol{U}_{m+1}, \tag{5.10}$$

$$= \boldsymbol{X}_m + \Delta t_m \mathcal{I}\left(\boldsymbol{u}_{m+1}, \boldsymbol{X}_{m+1}\right), \tag{5.11}$$

where $\mathcal{I}(\boldsymbol{u}, \boldsymbol{X})$ is the interpolation of the fluid to the points on the fiber. To avoid fully implicitly coupling (5.9) and (5.11), $\boldsymbol{u}_{m+1}$ in (5.11) is replaced with a estimation of $\boldsymbol{u}_{m+1}$.

To formulate this estimate, consider $\boldsymbol{U}_{m+1}$,

$$\boldsymbol{U}_{m+1} = \mathcal{I}\left(\boldsymbol{u}_{m+1}, \boldsymbol{X}_{m+1}\right), \tag{5.12}$$

$$= \mathcal{I}\left(\boldsymbol{u}_m + \Delta t_m \mathbb{P}\left(\mathbb{A}_m + \mathbb{D}_{m+1}\right) + \Delta t_m \mathbb{P}\boldsymbol{f}_{m+1}, \boldsymbol{X}_{m+1}\right), \tag{5.13}$$

$$= \mathcal{I}\left(\boldsymbol{u}_m + \Delta t_m \mathbb{P}\left(\mathbb{A}_m + \mathbb{D}_{m+1}\right) + \Delta t_m \left(\mathfrak{nf}_{m+1} + \mathfrak{ff}_{m+1}\right), \boldsymbol{X}_{m+1}\right). \tag{5.14}$$

The goal is to only evaluate (5.12) implicitly with quantities defined on Γ, and it is assumed that the far field does not change significantly over the course of one SDC step, thus reasonable approximations for $\mathbb{D}_{m+1}$ and $\mathfrak{ff}_{m+1}$ are made in the form

$$\mathfrak{ff}_{m+1} \approx \mathfrak{ff}_m \quad \text{and} \quad \mathbb{D} \approx \tilde{\mathbb{D}}, \tag{5.15}$$

where the estimate of the diffusion is given by,

$$\tilde{\boldsymbol{u}}^*_{m+1} = \boldsymbol{u}_m + \Delta t_m \left(\mathbb{A}_m + \tilde{\mathbb{D}}_{m+1}(\tilde{\boldsymbol{u}}^*_{m+1} + \mathfrak{ff}_m\right). \tag{5.16}$$

Therefore, $\boldsymbol{U}_{m+1}$ is approximated by

$$\boldsymbol{U}_{m+1} \approx \mathcal{I}\left(\boldsymbol{u}_m + \Delta t_m \mathbb{P}\left(\mathbb{A}_m + \tilde{\mathbb{D}}_{m+1}\right) + \Delta t_m \left(\mathfrak{nf}_{m+1} + \mathfrak{ff}_m\right), \boldsymbol{X}_{m+1}\right), , \tag{5.17}$$

$$\approx \mathcal{I}\left(\boldsymbol{u}_m + \Delta t_m \mathbb{P}\left(\mathbb{A}_m + \tilde{\mathbb{D}}_{m+1}\right) + \Delta t_m \mathfrak{ff}_m, \boldsymbol{X}_{m+1}\right) + \Delta t_m \mathfrak{nf}_{m+1}. \tag{5.18}$$

where $\mathfrak{nf}$ is analytically defined on Γ.

Letting $\tilde{\boldsymbol{u}}_{m+1} = \boldsymbol{u}_m + \Delta t_m \left(\mathbb{P}\left(\mathbb{A}_m + \tilde{\mathbb{D}}_{m+1}\right) + \mathfrak{ff}_m\right)$, gives the following predictor-corrector SDC update. It is important to note that the implicit terms in (**??**) are only evaluated on the immersed boundary points, not the grid.

5.2.1. Predictor.

$$
\begin{aligned}
\tilde{\boldsymbol{u}}_{m+1}^{[0]} &= \boldsymbol{u}_m^{[0]} + \Delta t_m \left(\mathbb{P} \left(\mathbb{A}_m^{[0]} + \tilde{\mathbb{D}}_{m+1}^{[0]} \right) + \mathfrak{ff}_m^{[0]} \right), \\
\boldsymbol{U}_{m+1}^{[0]} &= \mathcal{I} \left(\tilde{\boldsymbol{u}}_{m+1}^{[0]}, \boldsymbol{X}_{m+1} \right) + \Delta t_m \left(\mathfrak{nf}_{m+1}^{[0]} \right) \\
\boldsymbol{X}_{m+1}^{[0]} &= \boldsymbol{X}_m^{[0]} + \Delta t_m \boldsymbol{U}_{m+1}^{[0]}, \\
\boldsymbol{u}_{m+1}^{[0]} &= \boldsymbol{u}_m^{[0]} + \Delta t_m \left(\mathbb{P} \left(\mathbb{A}_m^{[0]} + \mathbb{D}_{m+1}^{[0]} \right) + \mathbb{P} \boldsymbol{f}_{m+1}^{[0]} \right).
\end{aligned}
$$

5.2.2. Corrector.

$$
\begin{aligned}
\tilde{\boldsymbol{u}}_{m+1}^{[k+1]} &= \boldsymbol{u}_m^{[k+1]} + \Delta t_m \mathbb{P} \left(\mathbb{A}_m^{[k+1]} - \mathbb{A}_m^{[k]} + \tilde{\mathbb{D}}_{m+1}^{[k+1]} - \tilde{\mathbb{D}}_{m+1}^{[k]} \right) \\
&+ \Delta t_m \left(\mathfrak{ff}_m^{[k+1]} - \mathfrak{ff}_m^{[k]} \right) + \mathbb{I}_m^{m+1} \mathbb{P} \left(\mathbb{A}^{[k]} + \mathbb{D}^{[k]} \right) + \mathbb{P} \boldsymbol{f}^{[k]}, \\
\boldsymbol{U}_{m+1}^{[k+1]} &= \mathcal{I} \left(\tilde{\boldsymbol{u}}_{m+1}^{[k+1]}, \boldsymbol{X}_{m+1} \right) + \Delta t_m \left(\mathfrak{nf}_{m+1}^{[k+1]} - \mathfrak{nf}_{m+1}^{[k]} \right), \\
\boldsymbol{X}_{m+1}^{[k+1]} &= \boldsymbol{X}_m^{[k+1]} + \Delta t_m \left(\boldsymbol{U}_{m+1}^{[k+1]} - \boldsymbol{U}_{m+1}^{[k]} \right) + \mathbb{I}_m^{m+1} \boldsymbol{U}^{[k]}, \\
\boldsymbol{u}_{m+1}^{[k+1]} &= \boldsymbol{u}_m^{[k+1]} + \Delta t_m \mathbb{P} \left(\mathbb{A}_m^{[k+1]} - \mathbb{A}_m^{[k]} + \mathbb{D}_{m+1}^{[k+1]} - \mathbb{D}_{m+1}^{[k]} \right) \\
&+ \Delta t_m \left(\mathbb{P} \boldsymbol{f}_{m+1}^{[k+1]} - \mathbb{P} \boldsymbol{f}_{m+1}^{[k]} \right) + \mathbb{I}_m^{m+1} \mathbb{P} \left(\mathbb{A}^{[k]} + \mathbb{D}^{[k]} \right) + \mathbb{P} \boldsymbol{f}^{[k]}
\end{aligned}
$$

In first equation of 5.2.2 the SDC integration term is the integration of the actual velocity, $\boldsymbol{u}_{m+1}$, not the integration of the right had side of (**??**). This is done because as the SDC iterations converge, $\tilde{\boldsymbol{u}}_{m+1}$ will converge to the actual velocity at t_{m+1}, $\boldsymbol{u}_{m+1}$, instead of its estimate.

The cost in terms of operations for one SDC step of the multi-implicit algorithm compared with the standard algorithm in d dimensions is shown in Table 5.1. The number of nonlinear solver operations, denoted κ, is not known a priori but is dependent on the formulation of the initial guess, and has been observed to depend on the time step. Thus, this table gives a good comparison for the cost of the algorithms, but is not a complete pitcture. The number of iterations needed for the nonlinear solver will be looked at in the following stiff test problems, to quantify the computational cost per time step for this algorithm.

TABLE 5.1. Operation count for the standard and multi-implicit algorithms.

Term	Standard Alg.	Multi-implicit Alg.
$\mathbb{A}$	$O(N^d)$	$O(N^d)$
$\mathbb{D}$	$O(N^d)$	$O(N^d)$
$\tilde{\mathbb{D}}$	-	$O(N^d)$
$\mathbb{P}\boldsymbol{f}$	$O(N^d)$	$O(N^d)$
$\boldsymbol{U}$	$O(N_\Gamma)$	$O(\kappa N_\Gamma)$
$\mathfrak{nf}$	-	$O((2\mathcal{R})^d N_\Gamma)$
$\mathbb{P}$	$O(N^d)$	$2O(N^d)$
Total:	$4(O(N^d)) + O(N_\Gamma)$	$6(O(N^d)) + O(\kappa N_\Gamma) + O((2\mathcal{R})^d N_\Gamma)$

With the multi-implicit algorithm and its associated costs established, it is now examined in the context of a stiff and a non-stiff test problem.

5.3. Numerical Implementation

To compare the multi-implicit and standard algorithms, (5.2.1)-(5.2.2) and (5.1.1)-(5.1.2) respectively, they are implemented on a 2-D grid with fourth order finite difference and semi-spectral schemes in both space and time. It should be noted that the parameters and choices presented in this section will be valid for the numerical comparisons done in Chapter 6 and 7 as well.

5.3.1. Spatial discretization. The Eulerian domain, $\Omega = [0,1] \times [0,1]$ is discretized with a 128×128 grid, where N denotes $N_x = N_y = 128$ and the spacing is denoted $h = h_x = h_y = 1/128$. The Lagrangian domain Γ is discretized with $N_\Gamma = 400$ points, separated by an initially fixed distance of Δl. To prevent leaking across the immersed boundary, the initial distance Δl is taken to be less than $h/2$. This is a common metric with immersed boundary simulations [**79, 21**].

Periodic boundary conditions on Ω are prescribed so the FFT can be utilized. Thus, the advection and diffusion terms are computed with spectral operators, and along with $\boldsymbol{u}$, are defined at the grid nodes $\boldsymbol{x}_{i,j} = (ih, jh)$.

The spring force density model, given in 2.2.1, is used with a fourth order stencil to find the arc length between points, Δl. The blob is taken as the compact blob, given in (2.30) and the cutoff parameter is fixed at $\delta = 3h$. The forces, and projection of the forces, $\mathbb{P}\boldsymbol{f}$, are evaluated with the semi-analytic formulation given in Section 2.3.2. That is, $\mathbb{P}\boldsymbol{f}$ is solved by computing

$$\nabla_h^2 \eta = (\nabla \cdot \boldsymbol{f})_{analytic}, \tag{5.19}$$

$$\hat{n} \cdot \nabla \eta = 0, \tag{5.20}$$

where

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_{\gamma=1}^{N_\Gamma} \boldsymbol{F}_\gamma B_\delta \left(\boldsymbol{x} - \boldsymbol{X}_\gamma\right) \Delta l_\gamma, \tag{5.21}$$

$$\nabla \cdot \boldsymbol{f} = \sum_{\gamma=1}^{N_\Gamma} \boldsymbol{F}_\gamma \left(\nabla \cdot B_\delta \left(\boldsymbol{x} - \boldsymbol{X}_\gamma\right)\right) \Delta l_\gamma. \tag{5.22}$$

Equation (5.22) can be determined by using the analytic expression for the blob (2.30). With the choice of a compact blob, both (5.21) and (5.22) are local and thus spread to the grid on a compact patch of size $L_{patch} = 2\delta \times 2\delta$. With these definitions, the Poisson equation (5.19) is solved with spectral methods and the computation of $\nabla_h \eta$ is computed with a spectral operator, resulting in $\mathbb{P}\boldsymbol{f}$.

The choice of near field splitting should now be revisited. Recall that there were different cases of matching conditions for the near field approximation polynomials, $H_1(r)$ and $H_2(r)$ (See Tables 4.1 and 4.2). Each case (i) - (iv) for each $\mathcal{R} = 1, 2, 3$ was tested using the standard algorithm and the perturbed ellipse, described in Chapter 4, the resulting errors, of which are reported in Figure 5.1. It appears that the first case for all values of $\mathcal{R}$ results in the lowest error. This does not line up exactly with the analysis done in Chapter 4, however that analysis was done based on the criteria of the magnitude of $|\mathbb{P}\boldsymbol{f}|$. This suggests that the effectiveness of the near field is still in need of exploration.

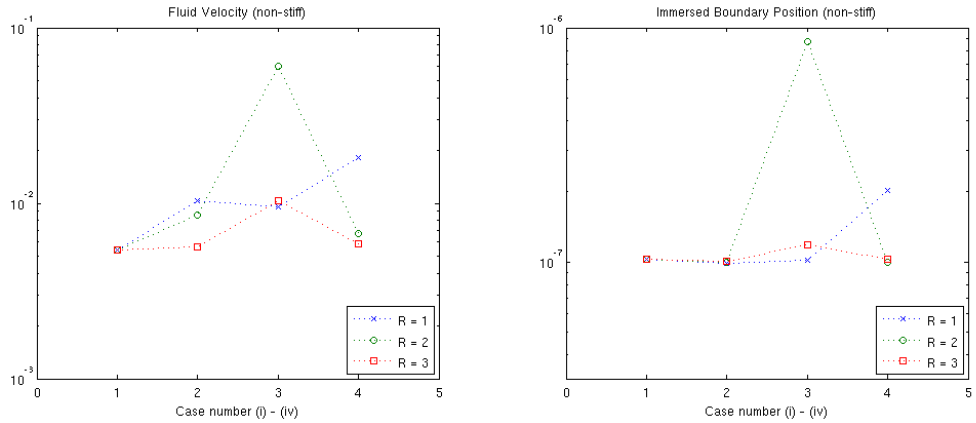


FIGURE 5.1. Error of different near field matching condition cases.

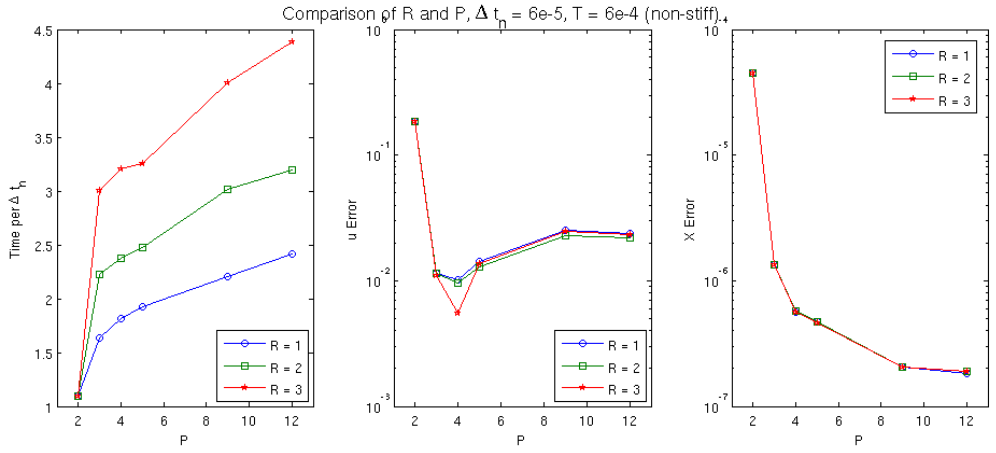


FIGURE 5.2. Error and computational cost for different values $\mathcal{R}$

Additionally, in this case, using $\mathcal{R} = 1$ is sufficient, which is significant because the cost of using this splitting is effected by size of the near field, reflected in Figure 5.2. This figure uses the mutlirate algorithm, where P denotes the number of multi-rate nodes and is slightly out of context here, however it does display the cost associated with the choice of $\mathcal{R}$, which is given by the data at $P = 2$, and will be revisited in the next chapter. Thus from here on, unless specified, $\mathcal{R} = 1$ with matching conditions (i) will be used, given the information in Figures 5.1 and 5.2.

5.3.2. Temporal discretization. The temporal domain, $[0, T]$, where T is the final time, is subdivided into N uniform intervals with M SDC nodes, as mentioned in 5.1 and detailed in 3.1.

The SDC nodes are prescribed by Clenshaw-Curtis (CC) quadrature [**18**] which makes the change of variable,

$$t = \cos(\theta). \tag{5.23}$$

This choice of quadrature was made with the multi-rate implementation in mind, since the nodes would easily correspond with different levels of refinement. To define the weight functions, such that on each time interval $[t_n, t_{n+1}]$,

$$\int_{t_n}^{t_{n+1}} f(\tau)d\tau = \sum_{m=1}^{M} w_m(t) f(t_m), \tag{5.24}$$

$$\text{where } t_m = \cos(\theta_m), \qquad \theta_m = \frac{2\pi m}{M} \tag{5.25}$$

first consider the integration of $\int_{-1}^{1} f(\tau)d\tau$ with the change of variables given by (5.23). This gives

$$\int_{-1}^{1} f(\tau)d\tau = -\int_{\pi}^{0} f(\cos(\theta))\sin(\theta)d\theta \tag{5.26}$$

$$= \frac{1}{2}\int_{-\pi}^{\pi} f(\cos(\theta))\sin(\theta)d\theta. \tag{5.27}$$

Since $f(\cos(\theta) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k cos(k\theta)$, this gives

$$\int_{-\pi}^{\pi} f(\cos(\theta))\sin(\theta)d\theta = \int_{-\pi}^{\pi}\left(\frac{a_0}{2} + \sum_{k=1}^{\infty} a_k cos(k\theta)\right)\sin(\theta)d\theta \tag{5.28}$$

$$= \left(\int_{-\pi}^{\pi}\frac{a_0}{2}\sin(\theta)d\theta + \sum_{k=1}^{\infty}\int_{-\pi}^{\pi} a_k cos(k\theta)\sin(\theta)d\theta\right), \tag{5.29}$$

where only the even k contribute.

To determine the numerical weight functions, first (5.27) is numerically evaluated for $f = \mathfrak{e}_1, \mathfrak{e}_2, ..., \mathfrak{e}_M$, where

$$(\mathfrak{e}_i)_j = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases} \tag{5.30}$$

Then equation (5.27) is computed on $t_{m=1}, ..., t_{m=M}$, using spectral techniques. First notice that for the contributing wave numbers,

$$f(\cos(\theta)) = \hat{f}_{N/2} \cos\left(\frac{N}{2}\theta\right) + \sum_{k=-N/2+1}^{N/2+1} \hat{f}_k e^{ik\theta}. \tag{5.31}$$

Combining this with the identity $\sin(\theta) = \left(e^{ik\theta} - e^{-ik\theta}\right)/2i$, (5.31) results in

$$f(\cos(\theta))\sin(\theta) = \hat{f}_{N/2} \cos\left(\frac{N}{2}\theta\right)\sin(\theta) + \sum_{k=-N/2+1}^{N/2+1} \hat{f}_k \frac{e^{i(k+1)\theta} - e^{i(k-1)\theta}}{2i}. \tag{5.32}$$

The first term in (5.32) can be integrated exactly and the summation can be integrated spectrally. This will result in the needed weight functions, w_m, to evaluate (5.24).

Here, the values $K = 3$ and $M = 5$, unless otherwise specified, which gives a formally fourth-order method in the temporal direction.

5.3.3. Implicit solver. Solving Equations (5.19) and (5.19) requires the use of an implicit, nonlinear solver, since equation (5.19), $\boldsymbol{U}_{m+1}$ is dependent on $\boldsymbol{X}_{m+1}$. Here, C. T. Kelley's Newton-Krylov solver, available through *www.siam.org*, is used [**53**]. This code solves the inexact Newton condition [**26, 53, 77**]

$$||\mathbb{F}'(\boldsymbol{X}_\psi)d_\psi + \mathbb{F}(\boldsymbol{X}_\psi)|| \leq c||\mathbb{F}(\boldsymbol{X}_\psi)|| \tag{5.33}$$

where $\mathbb{F}'$ is the Jacobian of the function $\mathbb{F}$, $\boldsymbol{X}_\psi$ is the solution increment at the ψ^{th} step and $c \in [0, 1)$ is a forcing term to enhance the convergence. Additionally, $\mathbb{F}(\boldsymbol{X})$ is defined as

$$\mathbb{F}(\boldsymbol{X}) \;=\; \boldsymbol{X} - \Delta t \mathbb{V}(\boldsymbol{X}) - \mathbb{H} = 0, \tag{5.34}$$

where

$$\boldsymbol{X} - \Delta t \mathbb{V}(\boldsymbol{X}) \;=\; \mathbb{H}, \tag{5.35}$$

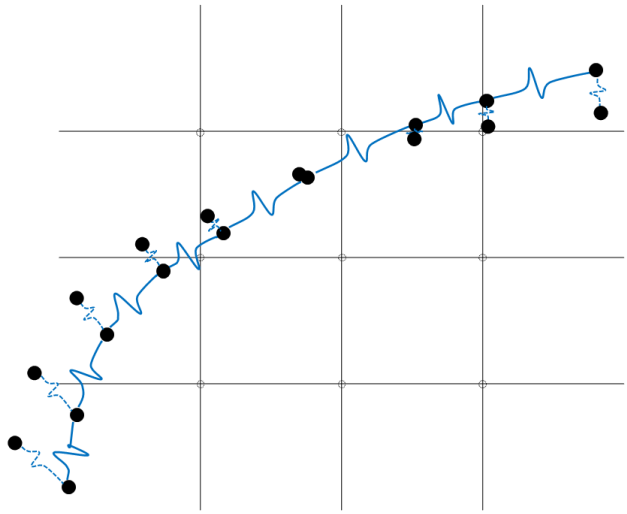


FIGURE 5.3. Spring force model for the immersed boundary.

and $\mathbb{H}$ is known. Iterations of GMRES are applied until the solution to (5.33) is within a specified tolerance, $\tau = \tau_a + \tau_r||\mathbb{F}||$, with τ_a and τ_r referred to as the absolute and relative tolerance, respectively.

5.4. Results

In an effort to fully examine the algorithms, two test problem were chosen, a *forced circle*, that is non-stiff and *straight wings*, that is stiff. The non-stiff problem is used to establish convergence and accuracy properties of the algorithms considered here, and the stiff problem to compare stability and computational cost. These two problems will be used to compare not only the algorithm presented in this chapter, but those presented in the next two chapters. Thus, in the following subsections the test problem is described first, followed by the results of using the multi-implicit algorithm.

5.4.0.1. *Non-stiff problem.* The *forced circle* begins with a periodic, circular immersed boundary, immersed in a fluid at rest in the unit square, $[0,1] \times [0,1]$. The points on Γ are 'attached' with a linear spring model, and the immersed boundary is initially defined in polar coordinates as

$$\boldsymbol{X}(\theta) = (0.5 + r\cos(\theta), 0.5 + r\sin(\theta)),$$

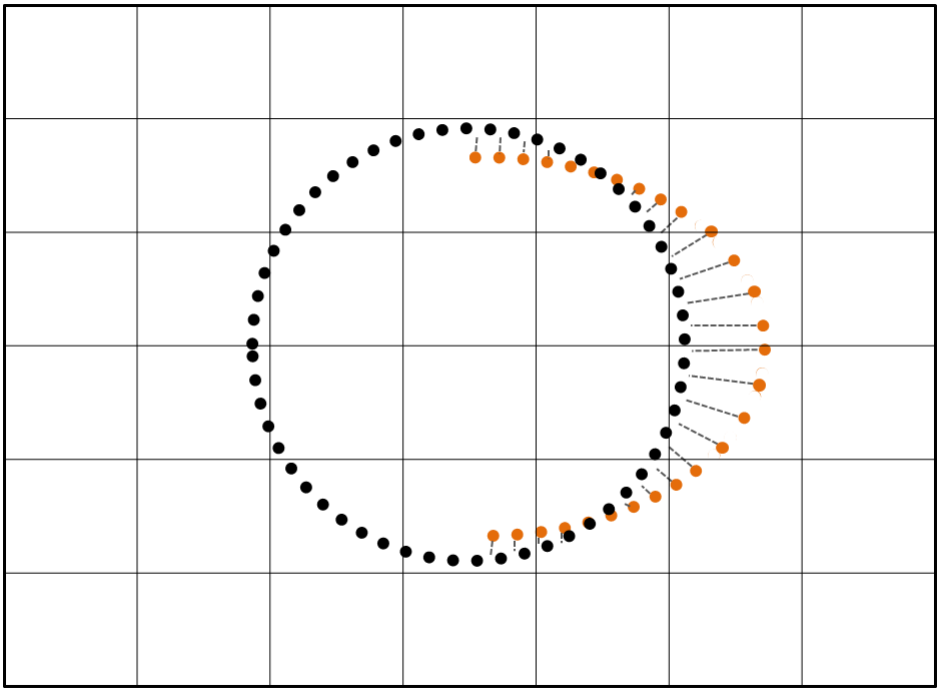


FIGURE 5.4. Depiction of target points attached to half of the immersed boundary.

where the value $r = 0.2$ is used and $(0.5, 0.5)$ is the center of the computational box.

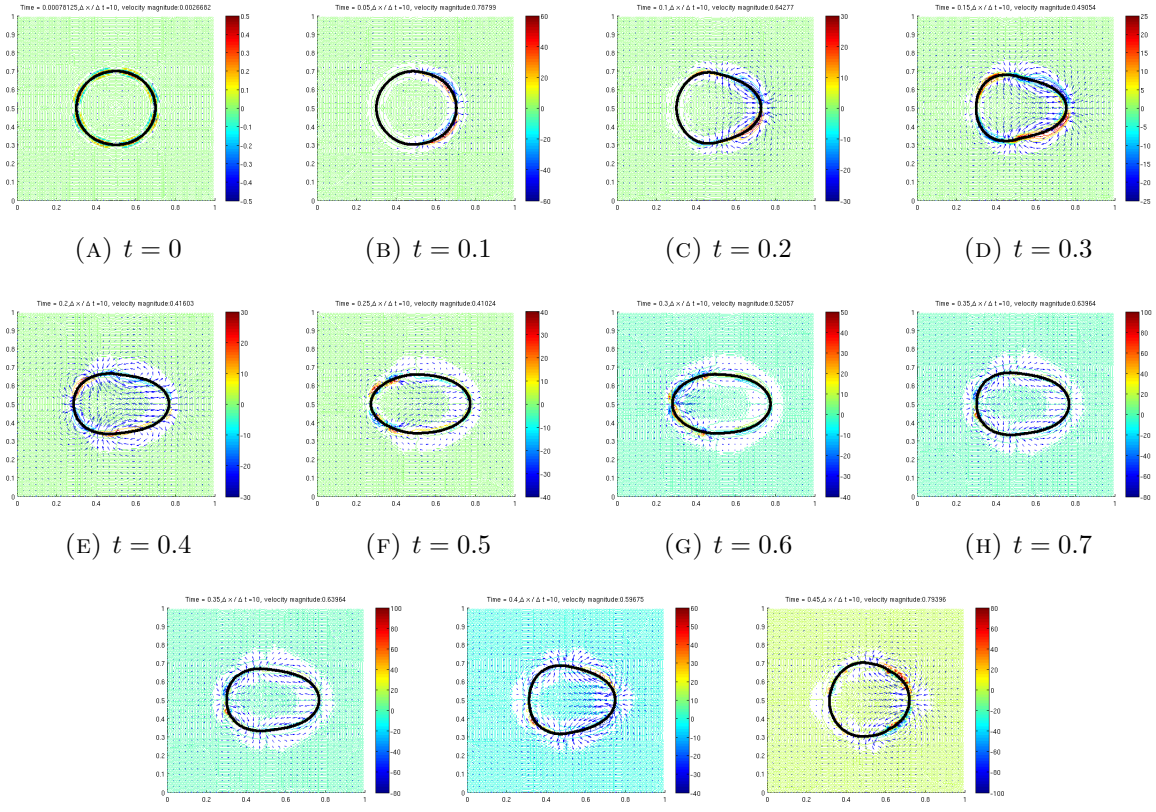
The immersed boundary is then forced by 'target points', a common technique used to prescribe motion in IBM [**48, 69, 67, 68**]. This technique also uses springs to model a connection from the points on Γ to the target points, as shown in Figure 5.3. Here, the target points, denoted $\boldsymbol{T}$, are based on a time dependent function, chosen to force the right half of the immersed boundary ($\theta \in [-\pi/2, \pi/2]$) into the shape of an ellipse, conserving area, this is depicted in Figure 5.4.

That is,

$$\boldsymbol{T}(\theta, t) = (0.5 + a(t)\cos(\theta), 0.5 + b(t)\sin(\theta)),$$

where,

$$\begin{aligned} a(t) &= c(1 + \epsilon \sin 2\pi t), \\ b(t) &= c/(1 + \epsilon \sin 2\pi t), \end{aligned}$$



(A) $t = 0$ (B) $t = 0.1$ (C) $t = 0.2$ (D) $t = 0.3$

(E) $t = 0.4$ (F) $t = 0.5$ (G) $t = 0.6$ (H) $t = 0.7$

(I) $t = 0.8$ (J) $t = 0.9$ (K) $t = 1$

FIGURE 5.5. The *forced circle,* non-stiff test problem

and $c^2 = ab$ to preserve area inside Γ. The values $c = 0.2$ and $\epsilon = 0.4$ are used, such that half of the circle will be stretched by 40% of its original radius, in the x direction from $t = 0$ to $t = T/2$, and then pushed back toward its original shape from $t = T/2 + \Delta t_n$ to $t = T = 1$, as shown in Figure 5.5.

The resulting force due to the the target points are modeled by a neo-Hookean linear spring model, with resting length zero. The left half of the circle is not forced, but passively follows due to the forces keeping the ellipse together. Additionally, a resistance to bending is added using the curvature of the immersed boundary, giving the following

complete definition for the force function,

$$\boldsymbol{F} = \boldsymbol{F}_B + \boldsymbol{F}_S + \boldsymbol{F}_T, \tag{5.36}$$

$$\boldsymbol{F}_B = \sigma\kappa\hat{n}, \tag{5.37}$$

$$\boldsymbol{F}_S = -k_S\frac{\partial}{\partial s}\left(\left(\frac{\partial \boldsymbol{X}}{\partial s} - 1\right)\tau\right), \tag{5.38}$$

$$\boldsymbol{F}_T = -k_T|\boldsymbol{X} - \boldsymbol{T}|. \tag{5.39}$$

Finally, the fluid viscosity is chosen to be $\nu = 0.0001$, giving a Reynolds number of $Re = 1000$.

The error is measured with the velocity of the fluid, $\boldsymbol{u}$, and the position of the immersed boundary $\boldsymbol{X}$. These are calculated using a inf-norm relative error, where a refined, reference solution $\boldsymbol{u}_r$ is computed to be the 'exact' solution. Additionally, the error in the fluid is normalized by the maximum fluid value, so to show the percentage of error. Thus the form of the two errors, $E_{\boldsymbol{u}}$ and $E_{\boldsymbol{X}}$, for respective approximate solutions $\tilde{\boldsymbol{u}}$ and $\tilde{\boldsymbol{X}}$ are given by

$$E_{\boldsymbol{u}} = \max_{u\in\boldsymbol{u},\tilde{u}\in\tilde{\boldsymbol{u}}} \frac{|u-\tilde{u}|}{|\tilde{u}|}, \qquad E_{\boldsymbol{X}} = \max_{X\in\boldsymbol{X},\tilde{X}\in\tilde{\boldsymbol{X}}} \left|X - \tilde{X}\right|. \tag{5.40}$$

Applying the standard and multi-implicit algorithm to the non-stiff test problem produces the fourth order convergence rate, seen in Figure 5.6. Additionally, the convergence of multi-rate algorithm and the combination of the multi-implicit and multi-rate algorithms is shown, however this is better discussed in the next chapter. The splitting of the multi-rate algorithm gives a higher error than with the standard method, however since the goal is to increase the largest stable time step, this should not be a large factor in the overall results.

As previously mentioned, the non-stiff case was only used to establish the convergence of the methods, the main goal will now be addressed with the stiff problem.

5.4.0.2. *Stiff problem.* The purpose of the work done here is to address the computational cost of the immersed boundary method. To properly do so, the algorithms are tested

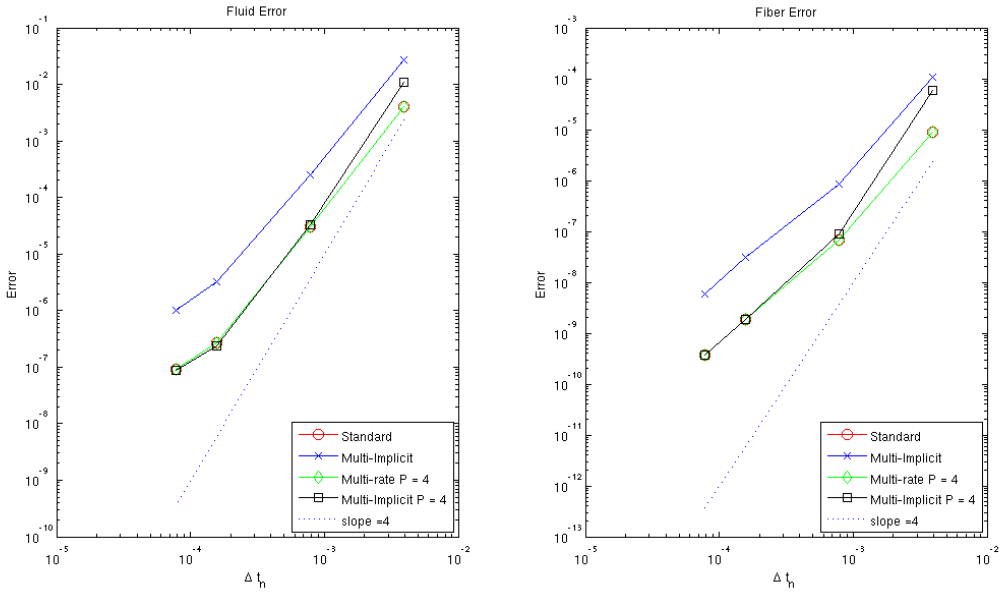


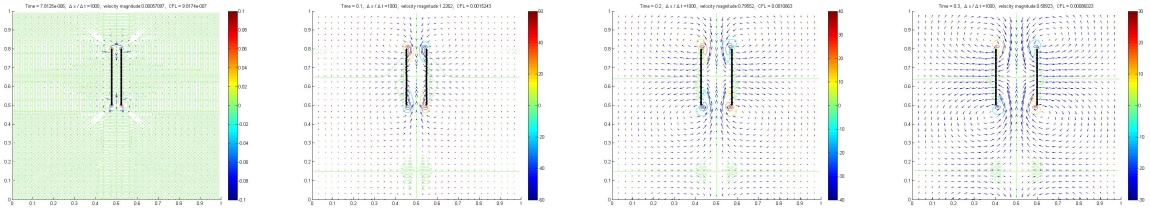
FIGURE 5.6. Convergence of the standard and multi-implicit algorithm.

on a problem designed to mimic the numerical properties of Miller and Peskin's insect wing [**69, 67, 68**]. This application, which used IBM to study the dynamics thrip wing, discussed briefly in Chapter 1, has a time step dominated by stability over accuracy. Here, a simplified problem, called the *straight wings*, is used to provide a stiff example for which to test the multi-implicit, and later multi-rate and time-parallel, algorithms.

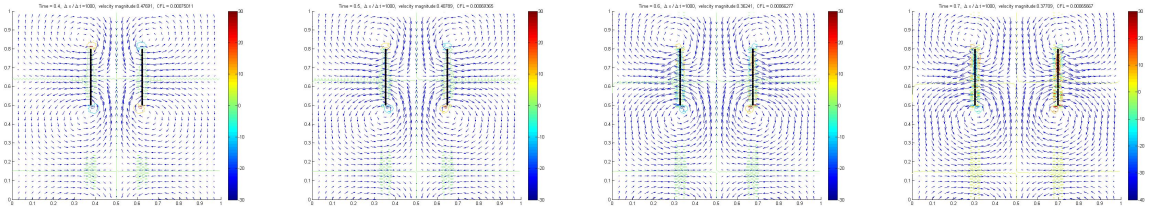
Initially, two flat vertical lines (wings), separated by a distance of $6\Delta x$ are submerged in a fluid at rest. Each line is length $L_w = 0.3$ and discretized by 50 points, giving $N_\Gamma = 100$. All other discretization parameters remain the same as the previous example.

Motion is then induced by the use of target points, defined to have a one to one correspondence with each point on Γ, that is each point on Γ follows a specific target point of $\boldsymbol{T}$. Note that there is no resistance to stretching or bending introduced in this particular test. At time $t \in T$, $\boldsymbol{T}$ is defined on the left wing by

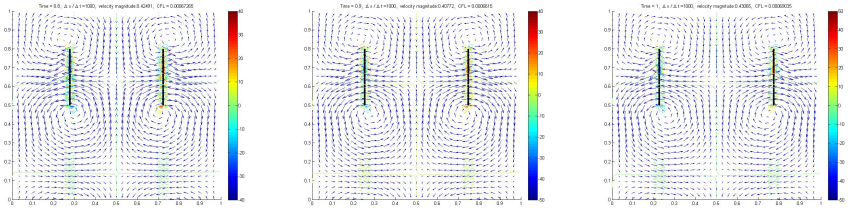
$$\boldsymbol{T}_{\text{left}}(t) = \left(\boldsymbol{X}_0 - \frac{Lt}{4T}, \boldsymbol{X}_0\right),$$



(A) $t = 0$ (B) $t = 0.1$ (C) $t = 0.2$ (D) $t = 0.3$



(E) $t = 0.4$ (F) $t = 0.5$ (G) $t = 0.6$ (H) $t = 0.7$



(I) $t = 0.8$ (J) $t = 0.9$ (K) $t = 1$

FIGURE 5.7. The *straight wings* stiff test problem.

and on the right wing by,

$$\boldsymbol{T}_{\text{right}}(t) = \left(\boldsymbol{X}_0 + \frac{Lt}{4T}, \boldsymbol{X}_0\right).$$

This causes the target points to move the length of $L/4$ over the time period, pulling the wings in opposite directions, the evolution of which is seen in Figure 5.7.

The result of using the standard and multi-implicit algorithms with this test case are shown in Figure 5.8. The left plot shows the size of the time step versus the relative error $E_{\boldsymbol{u}}$ and the center plot shows the same comparison with the error $E_{\boldsymbol{X}}$. In order to understand how the algorithm affects run time, the number of nonlinear solver solves are tracked over the entire simulation and shown in the third plot. The simulation time is dependent on the nonlinear solver, which is in turn dependent on the tolerance τ, and thus this could be optimized for a particular desired error. Note that it is not in general

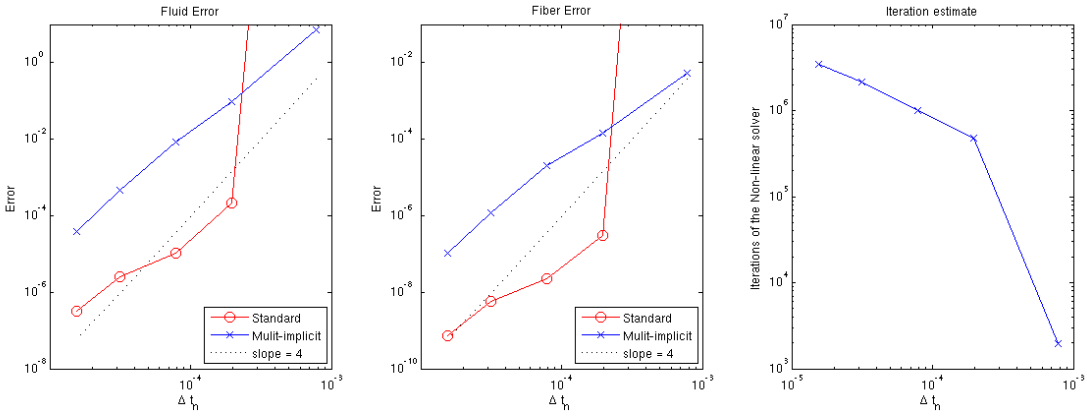


FIGURE 5.8. Multi-implicit and standard algorithms on the stiff test problem.

true that a smaller tolerance on the nonlinear solver gives a smaller error. In this case, $\tau_r = 10^{-5}$ is used for all the simulations except the largest time step, where $\tau_a = 1$ is used.

From Figure 5.8, it is clear that the time step for the standard algorithm is restricted by stability, not accuracy. Additionally, the opposite seems to be true, at least in this particular test, for the multi-implicit algorithm. The multi-implicit algorithm does have higher error than the standard algorithm, however $O(1)$ error is acceptable for many immersed boundary application studies, as the error put into the system due to modeling tends to overshadows the numerical error. That is, when examining applications, small relative errors, e.g. $O(10^-4)$, are simply an artifact of the numerical implementation, not an accurate description of the precision for which the simulation captures the physical FSI system.

The largest stable time step is increased by the multi-implicit algorithm by roughly a factor of $2 - 3$ in this stiff problem. It is possible in a more extreme case, where the time step is further restricted, there could be a larger gain. Even if this is not the case, a factor of 2-3 for application problems that have run times on the order of days and weeks can still be considered significant. Clearly, the size of the largest stable time step is not the only thing to consider, as it is only part of the overall computational cost. The cost of the multi-implicit algorithm per time step is equally significant, and this quantity is not easily determined. Optimal use of the nonlinear solver has yet to be explored, thus

it remains to be seen if the multi-implicit method reduces the overall run time. It should be stated again that this algorithm only evaluates quantities defined on Γ, and that one iteration of the nonlinear solver is much less expensive than one grid evaluation. Thus this could provide a reduction in overall cost in the long run.

Overall, it is clear that the multi-implicit algorithm increases the largest stable time step, making this a viable avenue of further research into numerical methods to address the computational cost of the immersed boundary method.

CHAPTER 6

Multi-rate techniques

After using multi-implicit techniques, and observing that in some cases the nonlinear solver had trouble computing the solution at large time steps, the focus shifted to using a multi-rate implementation. This, either in conjunction with the multi-implicit algorithm or alone, is again targeted at the stiff parts of the coupled equations. The near field forces on Γ and its position update will be calculated on a refined time step, compared to the calculations of the fluid on Ω, again with the purpose of increasing the largest stable time step.

Here, multi-rate techniques will be implemented in various ways, relying mainly on the MRSDC framework presented in Section 3.3, as well as the splitting given in Chapter 4. Recall that MRSDC splits the time domain into three levels of refinement, the coarsest being the uniform time nodes t_n, the next the SDC nodes t_m and finally the fine t_p nodes. This is shown again here in Figure 3.2 for reference.

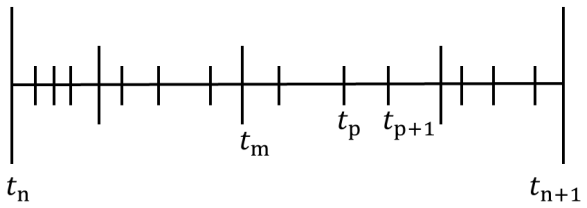


FIGURE 6.1. One time step $[t_n, t_n + 1]$ with SDC nodes and sub-SDC nodes

Initially, to present the basics of this chapter, the position of the boundary will be updated explicitly on the sub-SDC time scale. This will be compared against the standard algorithm, given in 5.1 with the same implementation used in 5.3. Next, the ideas of Chapter 5 will be combined with the multi-rate technique and the velocity at the

boundary will be evaluated implicitly. This too will be compared against the standard case with the same implementation.

6.1. Multi-rate Algorithm, Explicit U

Again, the splitting of $\mathbb{P}\boldsymbol{f}$ (Chapter 4) is employed, however here the near field, $\mathfrak{nf}$, and the position of the fiber, $\boldsymbol{X}$, are updated on a refined timescale.

The fluid equation with the stiff terms $\mathbb{P}\boldsymbol{f}$ and $\mathbb{D}$ evaluated implicitly in a first order IMEX update is given by

$$\boldsymbol{u}_{m+1} = \boldsymbol{u}_m + \Delta t_m \mathbb{P}\left(\mathbb{A}_m + \mathbb{D}_{m+1}\right) + \Delta t_m \mathbb{P}\boldsymbol{f}_{m+1}. \tag{6.1}$$

The goal is to update the position of the fiber with an explicit scheme on the sub-SDC nodes, without updating the fluid equation, since the purpose is to reduce computational cost, while increasing the largest stable time step. Evaluating the fluid equations on the refined time nodes would require full grid evaluations, which are expensive, $O(N^d)$ where $d = 2, 3$ is the dimension, compared to the updates done on the immersed fiber $O(N_\Gamma)$. Therefore, it is descry to construct a good approximation to the fluid at t_p. Using the notation presented in Chapter 5, the sub-SDC first order update for the immersed boundary position is given by

$$\boldsymbol{X}_{p+1} = \boldsymbol{X}_p + \Delta t_p \boldsymbol{U}_p, \tag{6.2}$$

$$= \boldsymbol{X}_p + \Delta t_p \mathcal{I}\left(\tilde{\boldsymbol{u}}_p, \boldsymbol{X}_p\right), \tag{6.3}$$

where $\tilde{\boldsymbol{u}}_p$ is the approximation to $\boldsymbol{u}_p$, constructed next.

Consider (6.1) with $\mathbb{P}\boldsymbol{f}$ split, as if it were evaluated at t_p,

$$\boldsymbol{u}_{p+1} = \boldsymbol{u}_p + \Delta t_p \mathbb{P}\left(\mathbb{A}_p + \mathbb{D}_{p+1}\right) + \Delta t_p \left(\mathfrak{nf}_{p+1} + \mathfrak{ff}_{p+1}\right). \tag{6.4}$$

The terms $\mathbb{A}$, $\mathbb{D}$ and $\mathfrak{ff}$, are evaluated on Ω, whereas $\mathfrak{nf}$ can be analytically evaluated on Γ. Therefore, since it is assumed $\mathbb{A}$, $\mathbb{D}$ and $\mathfrak{ff}$ do not change as much as $\mathfrak{nf}$ on small time

steps, these quantities can be fixed at t_m, giving the approximation $\tilde{\boldsymbol{u}}$, on the t_p nodes ,

$$\tilde{\boldsymbol{u}}_{p+1} = \tilde{\boldsymbol{u}}_p + \Delta t_p \left(\mathbb{P} \left(\mathbb{A}_m + \tilde{\mathbb{D}}_{m+1} \right) + \mathfrak{ff}_m \right) + \Delta t_p \mathfrak{nf}_{p+1}. \tag{6.5}$$

Again, $\tilde{\mathbb{D}}_{m+1}$ is used to denote an approximate diffusion term, evaluated prior to the sub-SDC time stepping, that solves (5.16). Combining (6.2) and (6.5) gives a full update for the fiber position, on the sub-SDC nodes,

$$\tilde{\boldsymbol{u}}_{p=1} = \boldsymbol{u}_m \tag{6.6}$$

$$\boldsymbol{X}_{p+1} = \boldsymbol{X}_p + \Delta t_p \boldsymbol{U}_p \tag{6.7}$$

$$\boldsymbol{U}_{p+1} = \mathcal{I} \left(\tilde{\boldsymbol{u}}_{p+1}, \boldsymbol{X}_{p+1} \right) + \Delta t_p \mathfrak{nf}_{p+1}. \tag{6.8}$$

A few notes about this formulation. First, there are no grid operations being done on the multi-rate nodes, since $\mathbb{P} \left(\mathbb{A}_m + \tilde{\mathbb{D}}_{m+1} \right) + \mathfrak{ff}_m$, is fixed on the SDC nodes and $\mathfrak{nf}_p$ can be defined on the immersed boundary. Second, the near field is computed for each point defining Γ by summing the contributions from each of the other particles that fall within the near field radius. For example, when the near field radius is $\mathcal{R}\delta = 1\delta$, the percentage of particles that contribute to each particles near field is observed to be $3 - 7\%$ of N_Γ. Even though as $\mathcal{R}$ grows, this percentage grows, computing $\mathfrak{nf}_p$ is of the order $O(N_\Gamma)$ operations, not $O(N_\Gamma^2)$, as the definition, given in Equation (4.4), would suggest.

The predictor-corrector scheme for the multi-implicit implementation with $\boldsymbol{U}$ evaluated explicitly is given by the following

6.1.1. Predictor.

$$\tilde{\boldsymbol{u}}_{p=1}^{[0]} = \boldsymbol{u}_m^{[0]},$$

Loop $p = 1...P$

$$\begin{aligned}
\boldsymbol{X}_{p+1}^{[0]} &= \boldsymbol{X}_p^{[0]} + \Delta t_p \boldsymbol{U}_p^{[0]}, \\
\tilde{\boldsymbol{u}}_{p+1}^{[0]} &= \tilde{\boldsymbol{u}}_p^{[0]} + \Delta t_p \left(\mathbb{P} \left(\mathbb{A}_m^{[0]} + \tilde{\mathbb{D}}_{m+1}^{[0]} \right) + \mathfrak{ff}_m^{[0]} \right), \\
\boldsymbol{U}_{p+1}^{[0]} &= \mathcal{I} \left(\tilde{\boldsymbol{u}}_{p+1}^{[0]}, \boldsymbol{X}_{p+1}^{[0]} \right) + \Delta t_p \mathfrak{nf}_{p+1}^{[0]},
\end{aligned}$$

$$\boldsymbol{X}_{m+1}^{[0]} = \boldsymbol{X}_P^{[0]},$$

$$\boldsymbol{u}_{m+1}^{[0]} = \boldsymbol{u}_m^{[0]} + \Delta t_m \mathbb{P} \left(\mathbb{A}_m^{[0]} + \mathbb{D}_{m+1}^{[0]} \right) + \Delta t_m \mathbb{P} \boldsymbol{f}_{m+1}^{[0]}.$$

6.1.2. Corrector.

$$\tilde{\boldsymbol{u}}_{p=1}^{[k+1]} = \boldsymbol{u}_m^{[k+1]},$$

Loop $p = 1...P$

$$\begin{aligned}
\boldsymbol{X}_{p+1}^{[k+1]} =& \boldsymbol{X}_p^{[k+1]} + \Delta t_p \left(\boldsymbol{U}_p^{[k+1]} - \boldsymbol{U}_p^{[k]} \right) + \mathbb{I}_p^{p+1} \boldsymbol{U}^{[k]}, \\
\tilde{\boldsymbol{u}}_{p+1}^{[k+1]} =& \tilde{\boldsymbol{u}}_p^{[k+1]} + \Delta t_p \mathbb{P} \left(\mathbb{A}_m^{[k+1]} - \mathbb{A}_m^{[k]} + \tilde{\mathbb{D}}_{m+1}^{[k+1]} - \tilde{\mathbb{D}}_{m+1}^{[k]} \right) \\
&+ \Delta t_p \left(\mathfrak{ff}_m^{[k+1]} - \mathfrak{ff}_m^{[k]} \right) + \mathbb{I}_p^{p+1} \mathbb{P} \left(\mathbb{A}^{[k]} + \mathbb{D}^{[k]} \right) + \mathbb{P} \boldsymbol{f}^{[k]}, \\
\boldsymbol{U}_{p+1}^{[k+1]} =& \mathcal{I} \left(\tilde{\boldsymbol{u}}_{p+1}^{[k+1]}, \boldsymbol{X}_{p+1}^{[k+1]} \right) + \Delta t_p \left(\mathfrak{nf}_{p+1}^{[k+1]} - \mathfrak{nf}_{p+1}^{[k]} \right)
\end{aligned}$$

$$\boldsymbol{X}_{m+1}^{[k+1]} = \boldsymbol{X}_P^{[k+1]}$$

$$\begin{aligned}
\boldsymbol{u}_{m+1}^{[k+1]} &= \boldsymbol{u}_m^{[k+1]} + \Delta t_m \mathbb{P} \left(\mathbb{A}_m^{[k+1]} - \mathbb{A}_m^{[k]} + \mathbb{D}_{m+1}^{[k+1]} - \mathbb{D}_{m+1}^{[k]} \right) \\
&+ \Delta t_m \left(\mathbb{P} \boldsymbol{f}_{m+1}^{[k+1]} - \mathbb{P} \boldsymbol{f}_{m+1}^{[k]} \right) + \mathbb{I}_m^{m+1} \mathbb{P} \left(\mathbb{A}^{[k]} + \mathbb{D}^{[k]} \right) + \mathbb{P} \boldsymbol{f}^{[k]}
\end{aligned}$$

Finally, with the multirate algorithm, there is the impact the cost to consider. Table 6.1 shows the cost of the standard algorithm and the multirate algorithm in one initial SDC step, in two dimensions. In this table, it is assumed that $N = N_x = N_y$, and the cost presented is computed M times per time step. This table shows two things, first that

TABLE 6.1. Operation count for the standard and multirate algorithms

Term	Standard Alg.	Multirate Alg.
$\mathbb{A}$	$O(N^d)$	$O(N^d)$
$\mathbb{D}$	$O(N^d)$	$O(N^d)$
$\tilde{\mathbb{D}}$	-	$O(N^d)$
$\mathbb{P}\boldsymbol{f}$	$O(N^d)$	$O(N^d)$
$\mathfrak{nf}$	-	$O((P+(2R)^d)N_\Gamma)$
$\boldsymbol{U}$	$O(N_\Gamma)$	$O(PN_\Gamma)$
$\mathbb{P}$	$O(N^d)$	$2O(N^d)$
Total:	$4(O(N^d))+O(N_\Gamma)$	$6(O(N^d)+2(O(2PN_\Gamma))$

there is an increase in cost from the extra grid computations of $\mathbb{P}$ and $\tilde{\mathbb{D}}$ computations. Second, since $O(N^d) > O(PN_\Gamma) \approx O(N_\Gamma)$, it is the case that adding more multirate steps does not necessarily increase the cost of this algorithm significantly. Thus, there is a 'start up' cost to use any amount of multi-rate nodes due to the two grid operations, but after that the cost increase is not substantial. Although this was done for one SDC predictor step, the cost for the remaining k steps of SDC corrections is roughly the same, as each term is updated to its $[k+1]$ value.

6.1.3. Results. The multirate algorithm given in 6.1.1 and 6.1.2 is now compared against the standard algorithm given by 5.1.1 and 5.1.2, using both the non-stiff *forced circle* and the stiff *straight wing* tests. The implementation used for this comparison is identical to that given in 5.3, without the need for a nonlinear solver (5.3.3). Here, it is important to note that $P = 2$ corresponds to having no multirate refinement, as $t_{p=1} = t_m$ and $t_{p=2} = t_{m+1}$, and therefore $P = 3$ corresponds to having one multirate node, $P = 2$ to two multirate nodes, and so on. Additionally, run time for the algorithms will be reported by calculating the average cost per time step, as determined by the 'wall clock', and extrapolated to $T = 1s$, for purposes of comparison.

Figure 6.2 shows the effect of the number of multirate nodes and the size of the near field, $\mathcal{R}$, on the the error and run time. This figure was shown in the previous chapter but

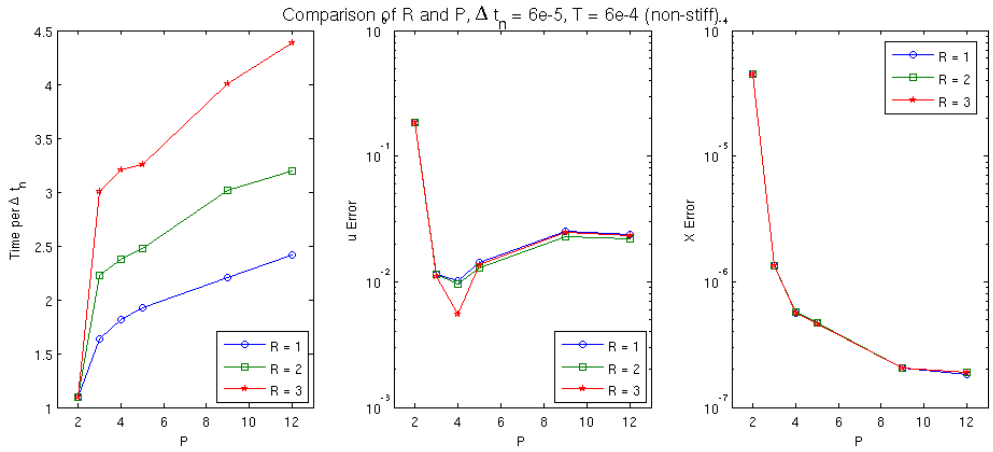


FIGURE 6.2. Effect of $\mathcal{R}$ and P on error, with cost, in a non-stiff problem

is again presented. Using the perturbed ellipse, presented in Chapter 4, this shows the error does decrease with the increase in P. It has been observed that using $P > 10$ only increases the run time, without actually reducing the error. Therefore, the comparisons made here will focus on using the values of $P = 4, 5, 9$.

6.1.3.1. *Non-stiff problem.* The non-stiff *forced ellipse*, presented originally in Section 5.4.0.1, is used to establish the convergence and accuracy of the multirate algorithm, compared to the standard algorithm. Figure 6.3 shows the time step versus the error for both the fluid velocity and the fiber position. It is evident that the error of the multirate algorithm in both cases is lower than that of the standard algorithm for different values of P, which is to be expected. The fourth order convergence of the multirate algorithm is also shown in Figure 6.3, which rounds out the purpose of using the non-stiff test. The target application for the multirate algorithm, the stiff test problem, is now discussed.

6.1.3.2. *Stiff problem.* The multirate algorithm is now compared against the standard algorithm using the stiff *straight wing* test problem (see Section 5.4.0.2 for details). Figures 6.4 and 6.5 show the time step vs. error and the run time vs. error for the fluid velocity and fiber position, respectively. The largest stable time step of the multirate algorithm is larger than that of the standard algorithm, although not significantly.

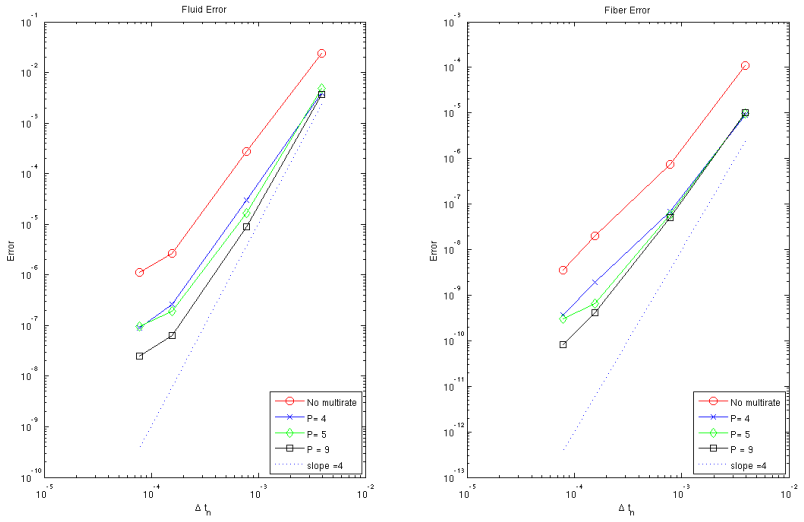


FIGURE 6.3. Convergence of the multirate algorithm on the non-stiff problem

In contrast to what was seen in the non-stiff problem, the error of the multirate algorithm is larger than that of the standard algorithm. Although this is not altogether intuitive, it does not seriously detract from the goal of reducing the computational cost for stiff IBM applications. As stated in Section 5.4.0.2, the modeling error usually overshadows the numerical error of IBM application problems. Thus the efficiency of the algorithm is more important in the goal of this work than the accuracy, granted that accuracy is O(1) or lower. Additionally, this error behavior for the multi-rate algorithm is believed to be the result of the splitting, because in the stiff regime, SDC does not fully converge, and thus the velocity for which the boundary advects is not converging to the true velocity. This phenomena of SDC failing to converge for stiff problems was detailed in Section 3.4.

Although the multirate algorithm does not see the magnitude of increase in the largest stable time step as the multi-implicit algorithm did, the time step was still an improvement over that of the standard algorithm. In this particular case, the cost per time step was high enough that the overall run time for the multirate algorithm was still larger than that of the standard algorithm. Although, because the cost of the multirate algorithm is fixed with the choice of P, and thus always known for future tests with this algorithm,

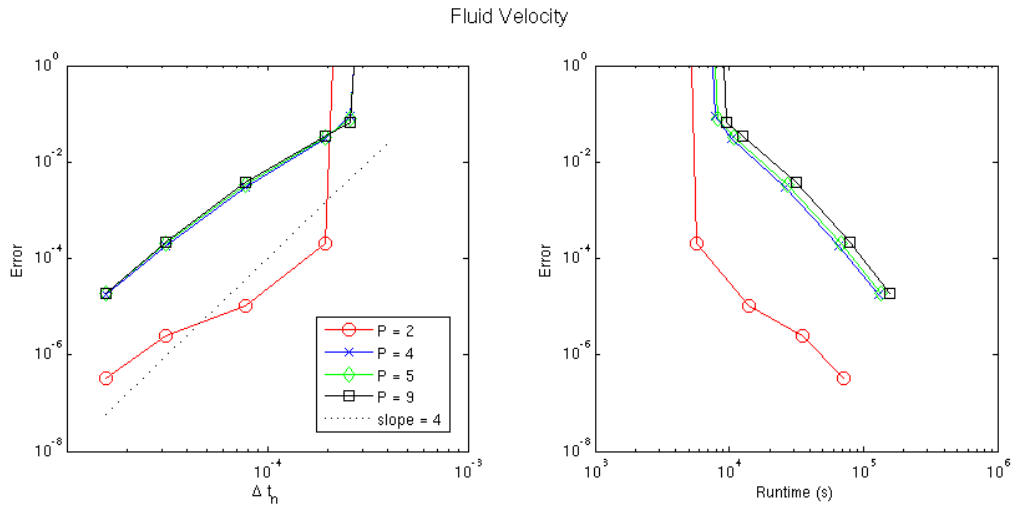


FIGURE 6.4. Relative fluid velocity error for the stiff problem with the multirate algorithm.

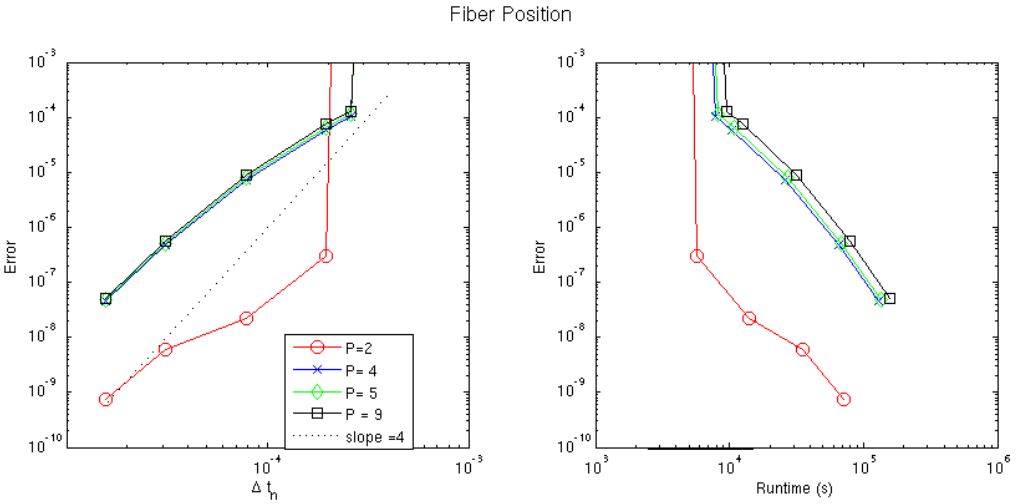


FIGURE 6.5. Fiber position error for the stiff problem with the multirate algorithm.

there could be problems for which it is more efficient. Altogether, the multirate algorithm does provide an avenue toward the goal of reducing the computational cost of stiff immersed boundary applications.

6.2. Multi-rate Algorithm, Implicit U

With the results of the multi-implicit and multirate algorithms presented, a next natural step is to look at combining the two algorithms. The multi-implicit algorithm

increased the largest stable time step by a larger amount than the multirate algorithm did, but its run time is dependent on the performance of the nonlinear solver, which is known to take more iterations when larger time steps are used. Thus, there is the potential that combining it with the multirate algorithm will decrease the overall number of nonlinear solves per time step, reducing the computational cost. This combination is presented next, along with results from both the stiff and non-stiff test problems.

The multirate algorithm (6.1.1-6.1.2) can easily be used in conjunction with the multi-implicit algorithm (5.2.1-5.2.2), by modifying the update on the t_p nodes. The update for $\boldsymbol{X}_{p+1}^{[0]}$ simply changes from

$$\boldsymbol{X}_{p+1}^{[0]} = \boldsymbol{X}_{p}^{[0]} + \Delta t_p \boldsymbol{U}_{p}^{[0]},$$

to

$$\boldsymbol{X}_{p+1}^{[0]} = \boldsymbol{X}_{p}^{[0]} + \Delta t_p \boldsymbol{U}_{p+1}^{[0]}.$$

This gives a predictor corrector scheme of the form,

6.2.1. Predictor.

$$\begin{aligned}
\tilde{\boldsymbol{u}}_{p=1}^{[0]} &= \boldsymbol{u}_{m}^{[0]}, \\
t_p \text{ nodes} & \quad \begin{cases}
\tilde{\boldsymbol{u}}_{p+1}^{[0]} = & \tilde{\boldsymbol{u}}_{p}^{[0]} + \Delta t_p \left(\mathbb{P} \left(\mathbb{A}_{m}^{[0]} + \tilde{\mathbb{D}}_{m+1}^{[0]} \right) + \mathfrak{ff}_{m}^{[0]} \right), \\
\boldsymbol{U}_{p+1}^{[0]} = & \mathcal{I} \left(\tilde{\boldsymbol{u}}_{p+1}^{[0]}, \boldsymbol{X}_{p+1}^{[0]} \right) + \Delta t_p \mathfrak{nf}_{p+1}^{[0]}, \\
\boldsymbol{X}_{p+1}^{[0]} = & \boldsymbol{X}_{p}^{[0]} + \Delta t_p \boldsymbol{U}_{p+1}^{[0]},
\end{cases} \\
\boldsymbol{X}_{m+1}^{[0]} &= \boldsymbol{X}_{P}^{[0]}, \\
\boldsymbol{u}_{m+1}^{[0]} &= \boldsymbol{u}_{m}^{[0]} + \Delta t_m \mathbb{P} \left(\mathbb{A}_{m}^{[0]} + \mathbb{D}_{m+1}^{[0]} \right) + \Delta t_m \mathbb{P} \boldsymbol{f}_{m+1}^{[0]}.
\end{aligned}$$

6.2.2. Corrector.

$$
\begin{aligned}
\tilde{\boldsymbol{u}}_{p=1}^{[k+1]} &= \boldsymbol{u}_m^{[k+1]}, \\
t_p \text{ nodes} & \begin{cases}
\tilde{\boldsymbol{u}}_{p+1}^{[k+1]} = & \tilde{\boldsymbol{u}}_p^{[k+1]} + \Delta t_p \mathbb{P}\left(\mathbb{A}_m^{[k+1]} - \mathbb{A}_m^{[k]} + \tilde{\mathbb{D}}_{m+1}^{[k+1]} - \tilde{\mathbb{D}}_{m+1}^{[k]}\right) \\
& \quad + \Delta t_p \left(\mathfrak{ff}_m^{[k+1]} - \mathfrak{ff}_m^{[k]}\right) + \mathbb{I}_p^{p+1}\mathbb{P}\left(\mathbb{A}^{[k]} + \mathbb{D}^{[k]}\right) + \mathbb{P}\boldsymbol{f}^{[k]}, \\
\boldsymbol{U}_{p+1}^{[k+1]} = & \mathcal{I}\left(\tilde{\boldsymbol{u}}_{p+1}^{[k+1]}, \boldsymbol{X}_{p+1}^{[k+1]}\right) + \Delta t_p \left(\mathfrak{nf}_{p+1}^{[k+1]} - \mathfrak{nf}_{p+1}^{[k]}\right) \\
\boldsymbol{X}_{p+1}^{[k+1]} = & \boldsymbol{X}_p^{[k+1]} + \Delta t_p \left(\boldsymbol{U}_{p+1}^{[k+1]} - \boldsymbol{U}_{p+1}^{[k]}\right) + \mathbb{I}_p^{p+1}\boldsymbol{U}^{[k]},
\end{cases} \\
\boldsymbol{X}_{m+1}^{[k+1]} &= \boldsymbol{X}_P^{[k+1]} \\
\boldsymbol{u}_{m+1}^{[k+1]} &= \boldsymbol{u}_m^{[k+1]} + \Delta t_m \mathbb{P}\left(\mathbb{A}_m^{[k+1]} - \mathbb{A}_m^{[k]} + \mathbb{D}_{m+1}^{[k+1]} - \mathbb{D}_{m+1}^{[k]}\right) \\
&+ \Delta t_m \left(\mathbb{P}\boldsymbol{f}_{m+1}^{[k+1]} - \mathbb{P}\boldsymbol{f}_{m+1}^{[k]}\right) + \mathbb{I}_m^{m+1}\mathbb{P}\left(\mathbb{A}^{[k]} + \mathbb{D}^{[k]}\right) + \mathbb{P}\boldsymbol{f}^{[k]}
\end{aligned}
$$

With these definitions for the multi-rate, multi-implicit (MM) algorithm established, the performance of this algorithm is now shown for the non-stiff *forced circle* and the stiff *straight wing* test problems.

6.2.2.1. *Non-stiff problem.* With this combined algorithm, the result of using it on the non-stiff problem, along with using the standard algorithm and the non-multirate multi-implicit algorithm are shown in Figure 6.6. The MM algorithm has lower error than that of the multi-implicit algorithm for both the fluid velocity and the fiber position, and the convergence of the algorithm is established with this figure.

6.2.2.2. *Stiff problem.* The MM algorithm is now tested on the stiff problem and the results of time step vs. error and number of iterations needed are shown in Figure 6.7. The behavior of the algorithm seems to follow that of the multi-implicit algorithm, where the choice of time step is restricted by accuracy, not stability, at least in this case. In 'more stiff' examples, the multi-implicit and MM algorithms may see the time step dominated by stability, but according to the tests done in this work, it will be a larger step than the use of a standard algorithm.

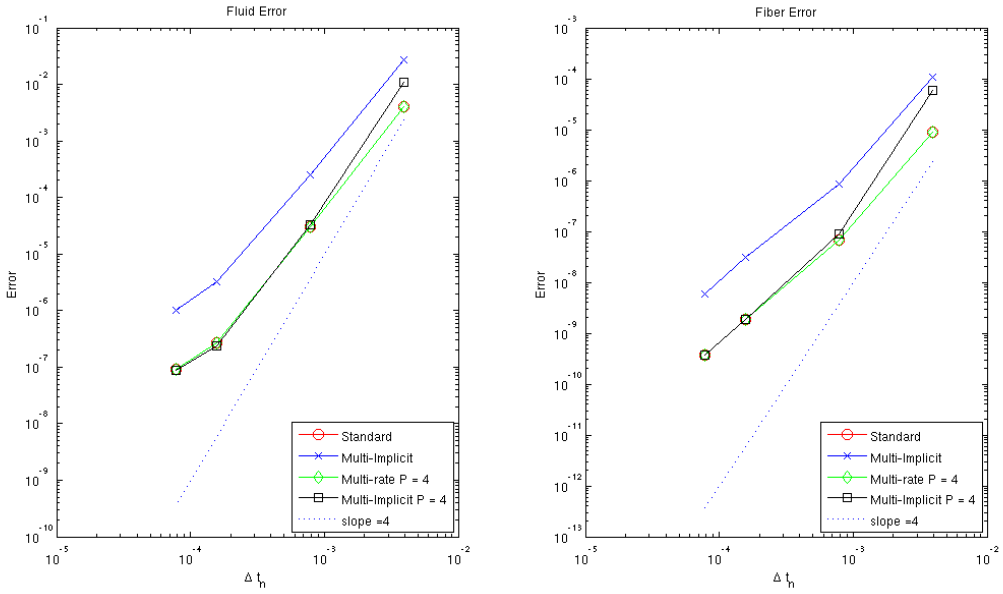


FIGURE 6.6. Convergence of the multi-rate, multi-implicit algorithm.

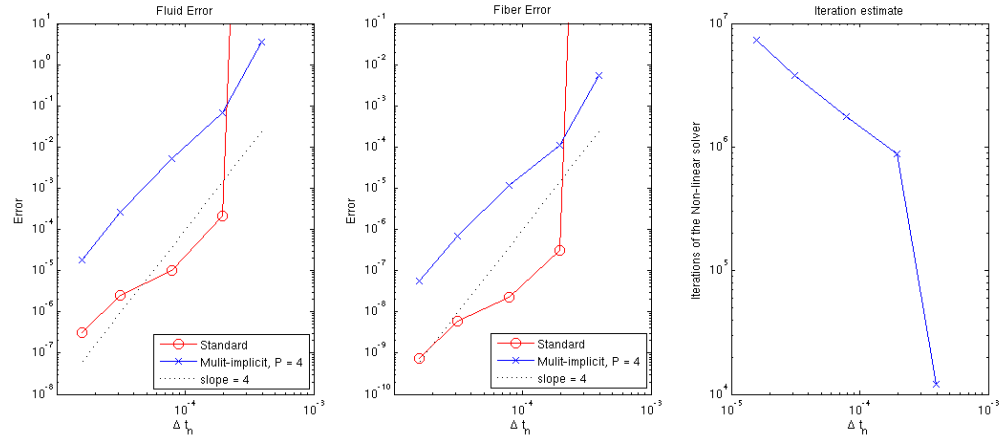


FIGURE 6.7. Result of using the multi-implicit, multirate algorithm on the stiff problem

The iteration count, however, does not seem to be significantly lower than using the multi-implicit algorithm, but this could change for other applications. Again, if the problem is more stiff, and the time step increases further, the nonlinear solver needs more iterations to resolve its solution and thus there could be a significant difference in the iteration count between the MM and multi-implicit algorithms.

The algorithms presented in this chapter, along with that presented in Chapter 5, increase the largest stable time step, but do not see an overall computational cost gain

due to the cost per time step. In the next chapter, a method for amortizing the cost per time step over multiple processors will be discussed in an effort to combine the largest stable time step with a lower cost per time step, reducing the overall computational cost.

CHAPTER 7

Time Parallelization

In the previous two chapters the multi-implicit and multirate algorithms had a high cost per time step, compared with the standard algorithm. This high cost can be amortized by using these techniques in conjunction with the *parallel full approximation scheme in space and time* (PFASST) method, presented by Emmett and Minion [**29**]. PFASST distributes the cost per time step over processors, thus combining it with multirate and multi-implicit methods could increase the computational efficiency.

The PFASST method has three main components, the *parareal* algorithm for temporal paralellization, spectral deferred corrections , and the *full approximation scheme* (FAS). To properly introduce PFASST, a short review of these components will be presented first, then PFASST itself is detailed.

7.1. Background

7.1.1. Parareal. With the introduction of the parareal algorithm in 2001, research into parallel methods for ODEs and PDEs in the temporal direction has increased [**60**]. Parareal, and the related PITA scheme [**30**], take an iterative approach, where in each iteration, each processor (assigned a distinct time step) uses both an accurate (or fine) method and a coarse method to propagate an improved solution through time. The coarse method is less computationally expensive than the fine method, and parallel speedup is achieved because the fine solutions are computed in parallel.

To clarify, consider the initial value ODE

$$\phi'(t) = f(\phi, t), \qquad \phi(0) = \phi_0 \tag{7.1}$$

Let the time domain $[0, T]$ be discretized into N intervals, where the solution is evaluated in each interval by $\phi_n \approx \phi(t_n)$. Further, assign each interval to a different processor and let k denote the iteration number for the successive approximations of ϕ_n^k.

The two numerical methods used to construct the parareal algorithm are typically denoted $\mathcal{F}$ and $\mathcal{G}$. These methods, although they can be any self-starting ODE method (such as forward Euler), are typically chosen such that $\mathcal{G}$ is computationally less expensive than $\mathcal{F}$, for efficiency purposes. Thus, $\mathcal{G}$ is generally a low order method and $\mathcal{F}$ is a high order method that limits the accuracy of parareal, and may use a smaller time step than $\mathcal{G}$. Thus $\mathcal{F}$ is referred to as the fine propagator and $\mathcal{G}$, the coarse propagator.

The algorithm begins by computing an initial solution ϕ^0 in serial for each n, often with the coarse propagator $\mathcal{G}$, giving $\phi_{n+1}^0 = \mathcal{G}(t_n, t_{n+1}, \phi_n^0)$. It then proceeds iteratively with $k = 1, ..., K$ iterations, alternating between computing $\mathcal{F}(t_n, t_{n+1}, \phi_n^0)$ in parallel and updating the initial conditions at each processor by

$$\phi_{n+1}^{k+1} = \mathcal{G}(t_n, t_{n+1}, \phi_n^{k+1}) + \mathcal{F}(t_n, t_{n+1}, \phi_n^k) - \mathcal{G}(t_n, t_{n+1}, \phi_n^k). \tag{7.2}$$

Notice that the right hand side of (7.2) is the coarse approximation plus a correction in the initial condition. Additionally, this can be allocated to processors in such a way that the parallel computation $\mathcal{G}(t_n, t_{n+1}, \phi_n^{k+1})$ is done upon completion of $\mathcal{F}(t_n, t_{n+1}, \phi_n^k)$, computed in serial [**72**].

After k iterations, the solution ϕ_m^k for $m \leq k$ is equal to the solution of ϕ_m^k computed by the fine propagator. Therefore, after N iterations, the solution converges to the solution using $\mathcal{F}$ in serial, although in practice this can happen in less than N iterations [**29**].

Parareal's main drawback is that the parallel efficiency is low. The cost is dominated by the computation of $\mathcal{F}$, and its parallel efficiency is bounded by $1/K$. Combining parareal with SDC (see Chapter 3), as done in [**73, 72**], reduces this cost in two ways. First, the accuracy achieved in one step of SDC, because of the spectral implementation, is much better after one iteration than would be with standard methods, like Runge-Kutta.

Second, the high cost of SDC per time step is distributed over the parallel iterations, since it is approximately M steps of the first-order method, not many more steps of $\mathcal{F}$. The result of combining these two methods, presented in [**72**], which suggests that the hybrid scheme does have similar convergence behavior as parareal, while reducing the parallel cost and achieving higher parallel efficiency. The details of this combination will be seen in conjunction with PFASST, in Section 7.2.

Although there are disadvantages to the parareal/SDC approach, in this context of the work done here, it is an appropriate way to look at the computational bottleneck of IBM. The drawbacks include the need for large storage and the dependence on SDC, which means it cannot function as a black box algorithm, like parareal.

7.1.2. FAS. When parallelizing PDEs in time, a way to reduce the cost using the above method, is to coarsen the spatial discretization, along with the temporal. PFASST utilizes the *full approximation scheme (FAS)* to do so [**29**]. FAS is a popular method in multigrid approaches for nonlinear problems, a short description is presented next, further details are found in [**9**].

Consider a non-linear equation of the form

$$A(\boldsymbol{x}) = \boldsymbol{b} \tag{7.3}$$

where $\boldsymbol{x}$ is the solution, and $\boldsymbol{b}$ is the spatial discretization of some function. The corresponding residual equation, for an approximate solution, $\tilde{\boldsymbol{x}}$, is given by

$$A(\tilde{\boldsymbol{x}} + \boldsymbol{e}) = \boldsymbol{r} + A(\tilde{\boldsymbol{x}}), \tag{7.4}$$

where $\boldsymbol{e}$ is the error, and $\boldsymbol{r} = \boldsymbol{b} - A(\tilde{\boldsymbol{x}})$. In a multigrid approach, (7.4) is solved on a coarse level with the use of T_F^G, an operator that translates between the fine and coarse grids. Letting the superscript G denote evaluations on the coarse grid, i.e. A^G is the

approximation of A on the coarse level, Equation (7.4) is

$$A^G(\tilde{\boldsymbol{x}}^G + \boldsymbol{e}^G) = A^G(\tilde{\boldsymbol{x}}^G) + \boldsymbol{r}^G \tag{7.5}$$

$$= A^G(\tilde{\boldsymbol{x}}^G) + T_F^G(\boldsymbol{b} - A(\tilde{\boldsymbol{x}}) \tag{7.6}$$

$$= \boldsymbol{b}^G + A^G(\tilde{\boldsymbol{x}}^G) - T_F^G(A(\tilde{\boldsymbol{x}}). \tag{7.7}$$

Letting $\boldsymbol{y}^G = \tilde{\boldsymbol{x}}^G + \boldsymbol{e}^G$, the residual equation (7.5) is equivalent to

$$A^G\boldsymbol{y}^G = \boldsymbol{b}^G + \boldsymbol{\tau}, \tag{7.8}$$

where $\boldsymbol{\tau}$ is the FAS correction term, given by

$$\boldsymbol{\tau} = A^G(\tilde{\boldsymbol{x}}^G) - T_F^G A^G(\tilde{\boldsymbol{x}}). \tag{7.9}$$

This allows for a similar degree of accuracy at the coarse level as that of the fine level solution, using the resolution of the coarse level [**9**]. Once this coarse solution $\boldsymbol{y}$ has been computed, the fine solution is updated by

$$\tilde{\boldsymbol{x}} = T_G^F(\boldsymbol{y}^G - \tilde{\boldsymbol{x}}^G). \tag{7.10}$$

To detail how FAS interacts with SDC, first recall Equation (3.18), the direct $k+1$ SDC update for a provisional solution $\phi^{[0]}$ to (7.1), again stated here

$$\phi_{m+1}^{[k+1]} = \phi_m^{[k+1]} + \Delta t_m \left(f(\phi_m^{[k+1]}t_m) - f(\phi_m^{[k]}, t_m)\right) + \mathbb{I}_m^{m+1} f(\phi^{[k]}). \tag{7.11}$$

As previously mentioned, if SDC converges, it converges to the solution

$$\phi_{m+1}^{[k+1]} = \phi_m^{[k+1]} + \mathbb{I}_m^{m+1} f(\phi^{[k]}). \tag{7.12}$$

Letting $\boldsymbol{\phi}$ denote $\boldsymbol{\phi} = [\phi_1, ..., \phi_M]$, as well as $\boldsymbol{f} = [f(\phi_1, t_1), ..., f(\phi_M, t_M)]$ and $\mathbb{I} = [\mathbb{I}_{t_1}^{t_2}, ..., \mathbb{I}_{t_{M-1}}^{t_M}]$, a matrix formulation of (7.12) is given by,

$$\boldsymbol{\phi} = \boldsymbol{\phi}_0 + \Delta t \boldsymbol{S}\boldsymbol{f}, \tag{7.13}$$

or equivalently

$$\boldsymbol{\phi} - \Delta t \boldsymbol{S}\boldsymbol{f} = \boldsymbol{\phi}_0, \tag{7.14}$$

where $\boldsymbol{\phi}_0 = [\phi_0, ..., \phi_0]$. This can be combined with (7.9), to give the FAS correction for coarse SDC iterations,

$$\boldsymbol{\tau} = \Delta t \left(\mathbb{I}^G \boldsymbol{f}^G - T_F^G \mathbb{I} \boldsymbol{f} \right). \tag{7.15}$$

Here, $\mathbb{I}^G$ is the iteration matrix defined on the coarse nodes and $\boldsymbol{f}^G$ is the vector of function values at the coarse level. This allows the coarse SDC to achieve the accuracy of the fine SDC iterations, at the coarse level resolution.

7.2. PFASST

The PFASST algorithm is constructed from the three previously presented methods, parareal, SDC and FAS, and thus can now be properly described.

First, the time domain $[0, T]$ is partitioned into N uniform intervals, $[t_n, t_{n+1}]$ and assigned to a unique processor, P_n, for $n = 1, ..., N$. Next, each n interval is discretized by $M + 1$ SDC nodes, $\boldsymbol{t}_n = [t_{n,0}, ..., t_{n,M}]$, where $t_n = t_{n,0} < ... < t_{n+1} = t_{n,M}$; as well as $\tilde{M} + 1$ coarse SDC nodes $\tilde{\boldsymbol{t}}$, such that $t_n = \tilde{t}_{n,0} < ... < \tilde{t}_{n,M} = t_{n+1}$. The coarse nodes are chosen as a subset of the fine nodes, in the interest of translating between the coarse and fine levels. The solution at the m^{th} fine node on the P_n processor during iteration k^{th} is denoted $\phi_{n,m}^{[k]}$. Similarly, the corresponding fine node solution is $\phi_{n,\tilde{m}}^{[k]}$. Additionally, the notation $\boldsymbol{\phi}_n^k = [\phi_{n,0}^{[k]}, ..., \phi_{n,M}^{[k]}]$ and $\boldsymbol{f}_n^k = [f(\phi_{n,0}^{[k]}, t_{n,0}), ..., f(\phi_{n,M}^{[k]}, t_{n,M})]$ is used, with analogous notion for the coarse level ($\tilde{m}$ replaces m).

7.2.1. Initialization. PFASST initializes each processor by beginning coarse SDC sweeps while the initial conditions are passed in serial. This is given in the following steps where the two super-scripts denote PFASST iteration and initialization iteration, respectively.

(1) Receive the new initial value $\tilde{\phi}_{n,0}^{0,j}$ from processor $\boldsymbol{P}_{n-1}$ if $n > 0$ and $j > 1$.

(2) Do one or more coarse SDC sweeps using the values $\tilde{\boldsymbol{f}}_n^{0,j-1}$ computed previously and the FAS correction $\boldsymbol{\tau}_n^0$. This results in updated values for $\boldsymbol{\phi}_n^{0,j}$ and $\tilde{\boldsymbol{f}}_n^{0,j}$.

(3) Send $\phi_{n,M}^{0,j}$ to processor $\boldsymbol{P}_{n+1}$ (if $n < N-1$). This will become the new initial condition $\tilde{\phi}_{n+1,0}^{0,j+1}$ in the next iteration.

Once each processor $\boldsymbol{P}_n$ has finished computing the value $\tilde{\phi}_{n,\tilde{M}}^{0,n}$ and sent it to $\boldsymbol{P}_{n+1}$, the fine grid value $\boldsymbol{\phi}_n^0$ is computed by means of the correction $\tilde{\boldsymbol{\phi}}_n^{0,n} - \tilde{\boldsymbol{\phi}}_n^{0,0}$ interpolated from the coarse grid to the fine grid. With this new value, the PFASST iterations on the processors are immediately started.

7.2.2. PFASST iterations. The PFASST iterations for $k = 1, ...K$ are given on each processor as follows. It is assumed that the fine solution and values $\phi nk - 1$ and $\boldsymbol{f}_n^{k-1}$ are previously computed.

(1) Compute one fine SDC correction using the values $\boldsymbol{f}_n^{k-1}$ to yield provisional solutions $\boldsymbol{\phi_n^{k'}}$ and $\boldsymbol{f}_n^{k'}$.

(2) Construct $\tilde{\boldsymbol{\phi}}_n^{k'}$ by restricting the fine nodes to the coarse nodes and compute $\tilde{\boldsymbol{f}}_n^{k'}$

(3) Compute the FAS correction τ_n^k with $\boldsymbol{f}_n^{k'}$ and $\tilde{\boldsymbol{f}}_n^{k'}$

(4) Receive the updated initial value $\phi_{n,0}^k$ from $\boldsymbol{P}_{n-1}$ if $n > 0$.

(5) Perform n_G coarse SDC correction steps with $\tilde{\boldsymbol{f}}_n^{k'}$, τ_n^k and the restriction of the new initial value $\phi_{n,0}^k$, yielding $\tilde{\phi nk}$ and $\tilde{\boldsymbol{f}}_n^k$.

(6) Interpolate the spatial coarse correction $\tilde{\phi}_{n,\tilde{M}}^{k'}$ and add to $\tilde{\phi}_{n,\tilde{M}}^k$, giving $\phi_{n,M}^k$.

(7) Send $\phi_{n,M}^k$, to processor $\boldsymbol{P}_{n+1}$ (for $n < N-1$), where it is the new initial condition $\phi_{n+1,0}^{k+1}$ for the next iteration.

(8) Interpolate $\boldsymbol{\phi}_n^{k'} - \boldsymbol{\phi}_n^k$, the coarse grid correction, in space and time on the remaining fine time nodes ($0 < m < M$) and add to $\boldsymbol{\phi}_n^{k'}$, yielding $\boldsymbol{\phi}_n^k$. Compute new $\boldsymbol{f}_n^k$ values.

Most of the cost in this approach is associated with FAS is done in step 8, which does not start until the new initial condition is received, minimizing the amount of computation done in each iteration.

In Emmett and Minion [**29**], the theoretical efficiency is predicted to be higher than that of the standard parareal method. That is, the efficiency was bounded above by K_s/K_p rather than $1/K_p$, where K_s is the number of SDC iterations needed to compute a solution to desired accuracy in serial, and K_p is the number of PFASST/parareal iterations. Additionally, the results suggest that it is possible to achieve some parallel efficiency in the temporal direction. The main drawback of this approach is the high cost of interpolation and recomputing explicit function values of FAS.

7.3. PFASST and BPM

Here, the PFASST algorithm is combined with BPM and compared against the standard algorithm. PFASST can be implemented in conjunction with BPM because δ is not tied to grid size. The implementation presented here uses the PyPFASST package, written by Emmett and was done in conjunction with Emmett.

In this implementation, the spatial direction is defined on the coarse level as $N_x = N_y = 64$ and on the fine level $N_x = N_y = 128$. Only the grid is coarsened, since coarsening the immersed boundary, although possible, would take a knowledge of forces on a coarse level. However, the cost of operations on the immersed boundary is less than that of the grid, this is not troublesome at this point, and can be left for future work. The temporal coarsening is done with the number of SDC nodes, $M = 5$ at the fine time level, and $M = 3$ at the coarse.

The results of applying PFASST to BPM for the non-stiff, forced circle problem are shown in Figures 7.1 and 7.2. Using the stiff problem here is sufficient, as the goal in using PFASST is to see the amortization of cost per time step, not an increase in the stability. Figure 7.1 shows the error vs. iteration number for three cases, the fine level (level 0), coarse level (level 1), and PFASST using 2 levels of refinement. Here, the coarse and fine levels refers to the coarse and fine level in time and space respectively. This shows that PFASST fully converges after only 4-5 iterations of the method.

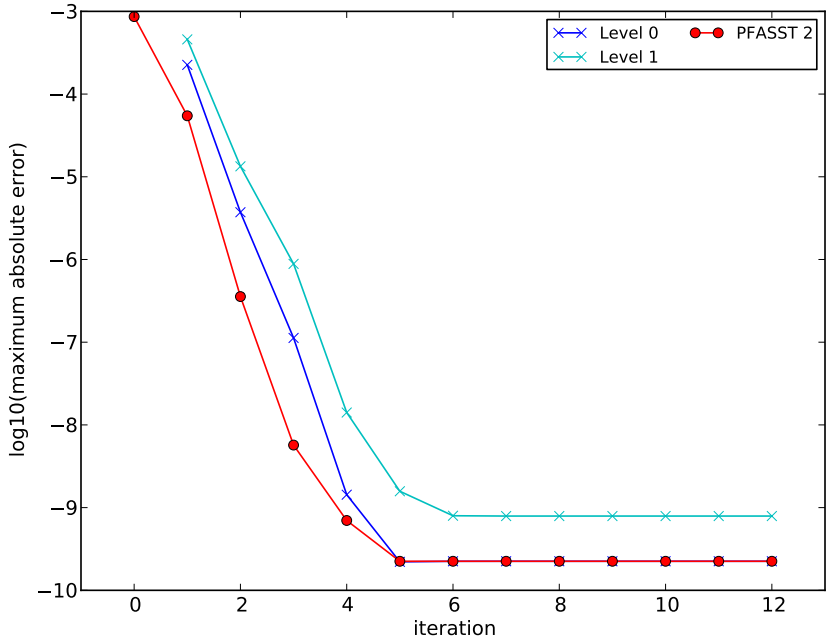


FIGURE 7.1. Error vs. iteration number for PFASST/BPM

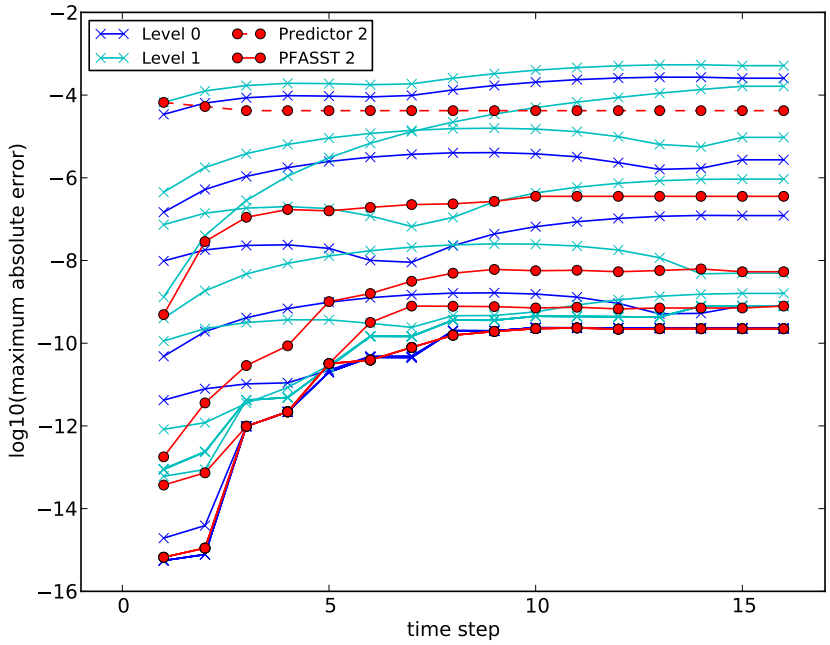


FIGURE 7.2. Error vs. time for PFASST/BPM

Figure 7.2 shows the error for the three cases, as well as just using a predictor for the PFASST iterations (no SDC) for the first 16 time steps of the simulation ($\Delta t = \frac{1}{10}\Delta x$). Additionally, each case is shown for each successive iteration. This shows that PFASST produces lower error than the alternative cases.

Theoretically, combining the spatial and temporal coarsening, this implementation defines a $\mathcal{G}$ that is 2^{d+1} times faster than $\mathcal{F}$ in d dimensions. This is the key relationship, as the speedup in the method is proportional to the ratio of the cost of $\mathcal{G}$ to $\mathcal{F}$. When PFASST and BPM were used together the actual speedup gained from using 16 time processors was about 6.2.

Recall that the purpose of introducing PFASST into the pursuit of addressing the computational bottle neck of the IBM was to amortize the cost per time step of the multi-implicit and multirate implementations, as they decrease the number of time steps necessary, but are more expensive per time step. Thus, this initial test shows it is in fact possible to use PFASST with BPM to decrease the overall run time, by reducing the cost per time step, without compromising accuracy. This suggests that a combination of using PFASST with the multi-implicit and multi-rate algorithms will produce significant savings for the computational cost of IBM.

CHAPTER 8

Conclusion

This work has focused on the stability restriction and subsequent high computational cost of applications which use IBM. Specifically targeting the treatment of the immersed boundary, the methods presented in this work do not require changes to the fluid solver, making these methods more accessible to application studies.

The new spatial splitting introduced Chapter 4, allows one to isolate the stiff terms in the time update for IBM. This is a novel approach to IBM problems, and there is still much more potential for exploiting this splitting in the interest of capturing the stiffness. Chapters 5 and 6 used this splitting along with multi-implicit and multirate techniques to handle the stiff terms. The multi-implicit algorithm allowed for the treatment of the stiff terms implicitly, without a fully coupled system or nonlinear implicit solves on the grid. The multirate algorithm evaluated the stiff terms explicitly, on a refined timescale, using only the values defined on the immersed boundary, not the grid. Both methods, and the combination of the two, increased the largest stable time step for the particular representative stiff application test case. Although the cost per time step was too high to achieve a reduction in the overall computational cost, this is an important result in addressing the stability restriction for IBM. Further, the use time parallelism, shown in Chapter 7, showed promise as it amortized the cost per time step, giving a reduction of the overall computational time. The results presented here strongly suggest that a combination of these three temporal methods could greatly reduce the computational time for IBM application problems.

A robust picture of the stability properties for the temporal methods discussed here is still necessary. The stiff problem used to test the methods did give results that suggest that these methods are successful in increasing the time step, but did not find the limit

of their stability. It is not immediately clear how the increase in the time step will scale with different applications and this is an avenue for further investigation.

CHAPTER 9

Appendix

9.1. Derivation of analytic formulation of $\mathbb{P}(F)$

9.1.1. Introduction. Derivation of the analytic formula for the projection of forces, $\mathbb{P}(F)$ which appears as equation 14 in *The Blob Projection Method for Immersed Boundary Problems* by Cortez and Minion, 2000. This particular verbose derivation was done in January 2010 by Lauren Cooper and followed the steps of the shortened derivation from *An impulse-based approximation of fluid motion due to boundary forces*, Cortez 1996. This particular derevation is done in 2D but extends to 3D. Equation 14 is given by

$$\mathbb{P}(\vec{F}) = \Delta l \sum_{k=0}^{K} \frac{1}{2} \boldsymbol{f}_k \phi_\delta(r) + \frac{1}{2\pi} [\boldsymbol{f}_k - 2(\boldsymbol{f}_k \cdot \hat{\boldsymbol{x}}_k)\hat{\boldsymbol{x}}_k] \frac{rF'(r) - 2F(r)}{2r^2}$$

9.1.2. Definitions. Let us quickly define a few terms we will be using in the derivation.

Blob : $B_\delta(r) = \delta^{-2} B(|\boldsymbol{x}|/\delta)$

Force : $\vec{F} = \Delta l \sum_{k=0}^{K} \boldsymbol{f}_k B_\delta(\boldsymbol{x} - \boldsymbol{z}_k)$

$$\hat{\boldsymbol{x}}_k = \boldsymbol{x} - \boldsymbol{z}_k \qquad r = |\boldsymbol{x} - \boldsymbol{z}_k|$$

9.1.3. Derivation.

9.1.3.1. *Impulse Method Formulation.* Recall that the Helmholtz decomposition tells us that any sufficiently smooth vector F can be decomposed into the sum of a divergence free vector and the gradient of a scalar function :

$$\vec{F} = \mathbb{P}(\vec{F}) + \nabla\phi$$

Where $\nabla \cdot \mathbb{P}(\vec{F}) = 0$ and ϕ satisfies the equation $\Delta\phi = \nabla \cdot F$

To find an analytic expression for $\mathbb{P}(\vec{F})$ lets start by solving $\Delta\phi = \nabla \cdot F$. Note that

$$\Delta\phi = \nabla \cdot \vec{F} = \nabla \cdot (\Delta l \sum_{k=0}^{K} \boldsymbol{f}_k B_\delta(\boldsymbol{x} - z_k)) = \Delta l \sum_{k=0}^{K} \boldsymbol{f}_k \nabla \cdot B_\delta(\boldsymbol{x} - z_k)$$

Now if ψ is a function that satisfies $\Delta\psi = B_\delta(r)$ then we can write ϕ as

$$\phi = \Delta l \sum_{k=0}^{K} \boldsymbol{f}_k \cdot \nabla\psi$$

Or letting $\boldsymbol{m}_k = \Delta l \boldsymbol{f}_k$ we have the following system to solve

$$\phi = \sum_{k=0}^{K} \boldsymbol{m}_k \cdot \nabla\psi \tag{9.1}$$

$$\Delta\psi = B_\delta(r) \tag{9.2}$$

The proof of this identity is done (tediously) in the Appendix, section 1.

9.1.3.2. *Solving equation* (9.2). We want to solve $\Delta\psi = B_\delta(r)$ to find the gradient of ψ, thus giving ϕ as given in (9.1).

First let us transform the laplacian into polar coordinates

$$\Delta\psi = \frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\psi}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2\psi}{\partial\theta^2}$$

Now since B_δ is a radially symmentric function and ϕ is also radially symmentric because

$$\begin{aligned}\phi &= \sum_{k=1}^{K} \boldsymbol{m}_k \cdot \nabla\psi = \sum_{k=1}^{K} \boldsymbol{m}_k \cdot \left(\frac{\partial\psi}{\partial r}\ , \frac{1}{r}\frac{\partial\psi}{\partial\theta}\right) \\ &= \sum_{k=1}^{K} \boldsymbol{m}_k \cdot \left(\frac{\partial\psi}{\partial r}\ , 0\right)\end{aligned}$$

We can solve

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\psi}{\partial r}\right) = B_\delta(r)$$

for $\frac{\partial\psi}{\partial r}$ since all we are looking for is the gradient of ψ to get ϕ.

$$\frac{\partial\psi}{\partial r} = \frac{1}{r}\int_0^r 2\pi r B_\delta(r)dr = \frac{1}{2\pi r}F(r)$$

where $F(r)$, called the 'Shape Factor' in Cortez 1996, is defined as

$$F(r) = \int_0^r 2\pi r B_\delta(r)dr = \int_0^{2\pi}\int_0^r r B_\delta(r)drd\theta = \int_{|\boldsymbol{x}|} B_\delta(|\boldsymbol{x}|)d\boldsymbol{x}$$

9.1.3.3. *Solving for $\phi, \nabla\phi$.* Now that we have solved equation 2 , we can find ϕ and $\nabla\phi$

$$\begin{aligned}\phi &= \sum_{k=1}^K \boldsymbol{m}_k \cdot \nabla\psi = \sum_{k=1}^K \boldsymbol{m}_k \cdot \left(\frac{\partial\psi}{\partial x}, \frac{\partial\psi}{\partial y}\right) \\ &= \sum_{k=1}^K \boldsymbol{m}_k \cdot \left(\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial x}, \frac{\partial\psi}{\partial r}\frac{\partial r}{\partial y}\right)\end{aligned}$$

Given that $r^2 = (x_1 - z_{k,1})^2(x_2 - z_{k,2})^2$ we know that

$$\frac{\partial r}{\partial x} = \frac{x_1 - z_{k,1}}{r}$$

$$\frac{\partial r}{\partial y} = \frac{x_2 - z_{k,2}}{r}$$

Therefore

$$\phi = \sum_{k=1}^K m_1 \frac{F(r)}{2\pi r}\frac{x_1 - z_{k,1}}{r} + m_2\frac{F(r)}{2\pi r}\frac{x_2 - z_{k,2}}{r}$$

$$\phi = \sum_{k=1}^K (\boldsymbol{m}\cdot\hat{\boldsymbol{x}})\frac{F(r)}{2\pi r}$$

Now let us find $\nabla\phi$.

$$\begin{aligned}\nabla\phi = \sum_{k=1}^K \boldsymbol{m}_k \cdot \nabla\psi &= \nabla\sum_{k=1}^K \boldsymbol{m}_k \cdot \left(\frac{\partial\psi}{\partial x}, \frac{\partial\psi}{\partial y}\right) \\ =&\nabla\sum_{k=1}^K \left(m_1\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial x} + m_2\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial y}\right) \\ =&\sum_{k=1}^K \left(m_1\frac{\partial}{\partial x}\left(\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial x}\right) + m_2\frac{\partial}{\partial x}\left(\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial y}\right), m_1\frac{\partial}{\partial y}\left(\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial x}\right) + m_2\frac{\partial}{\partial y}\left(\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial y}\right)\right)\end{aligned}$$

Since we have the following equalities (proved in the Appendix)

$$\begin{aligned}
\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial x}\right) &= \left(\frac{rF'(r)-2F(r)}{2\pi r^2}\right)\hat{x}_1^2 + \frac{F(r)}{2\pi r^2} \\
\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial y}\right) &= \frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial x}\right) = \left(\frac{rF(r)-2F(r)}{2\pi r^2}\right)\hat{x}_1\hat{x}_2 \\
\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial y}\right) &= \left(\frac{rF'(r)-2F(r)}{2\pi r^2}\right)\hat{x}_2^2 + \frac{F(r)}{2\pi r^2}
\end{aligned}$$

we can find the components of $\nabla\phi$

$$\begin{aligned}
(\nabla\phi)_1 =& m_1\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial x}\right) + m_2\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial y}\right) \\
=& m_1\left(\left(\frac{rF'(r)-2F(r)}{2\pi r^2}\right)\hat{x}_1^2 + \frac{F(r)}{2\pi r^2}\right) + m_2\frac{rF'(r)-2F(r)}{2\pi r^2}\hat{x}_1\hat{x}_2 \\
=& m_1\frac{F(r)}{2\pi r^2} + (m_1\hat{x}_1^2 + m_2\hat{x}_1\hat{x}_2)\frac{rF'(r)-2F(r)}{2\pi r^2}
\end{aligned}$$

$$\begin{aligned}
(\nabla\phi)_2 =& m_1\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial x}\right) + m_2\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial y}\right) \\
=& m_1\left(\frac{rF'(r)-2F(r)}{2\pi r^2}\right)\hat{x}_1\hat{x}_2 + m_2\left(\frac{rF'(r)-2F(r)}{2\pi r^2}\hat{x}_2^2 + \frac{F(r)}{2\pi r^2}\right) \\
=& m_2\frac{F(r)}{2\pi r^2} + (m_1\hat{x}_1\hat{x}_2 + m_2\hat{x}_2^2)\frac{rF'(r)-2F(r)}{2\pi r^2}
\end{aligned}$$

Note that (also in Appendix)

$$\hat{\boldsymbol{x}}(\boldsymbol{m}\cdot\hat{\boldsymbol{x}}) = \left(m_1\hat{x}_1^2 + m_2\hat{x}_1\hat{x}_2 \;\;,\;\; m_1\hat{x}_1\hat{x}_2 + m_2\hat{x}_2^2\right)$$

Therefore we can write $\nabla\phi$ compactly as

$$\nabla\phi = \sum_{k=0}^{K}\frac{F(r)}{2\pi r^2}\boldsymbol{m} + \left(\frac{rF'(r)-2F(r)}{2\pi r^2}\right)\hat{\boldsymbol{x}}(\boldsymbol{m}\cdot\hat{\boldsymbol{x}})$$

9.1.3.4. *Solving for* $\mathbb{P}(F)$. Now that we have an analytic expression for $\nabla\phi$ we can put this back into the Helmholtz Decomposition to finish equation 14

$$\begin{aligned}\mathbb{P}(F) =& \vec{F} - \nabla\phi \\ =& \sum_{k=0}^{K} \vec{F} - \frac{F(r)}{2\pi r^2}\boldsymbol{m} + \left(\frac{rF'(r) - 2F(r)}{2\pi r^2}\right)\hat{\boldsymbol{x}}(\boldsymbol{m}\cdot\hat{\boldsymbol{x}})\end{aligned}$$

Note that

$$\vec{F} = \sum_{k=0}^{K} \boldsymbol{m}_j B_\delta(\boldsymbol{x} - \boldsymbol{z}_k) = \sum_{k=0}^{K} \frac{m}{2\pi r} F'(r)$$

Therefore

$$\mathbb{P}(F) = \sum_{k=0}^{K} \frac{m}{2\pi r} F'(r) - \left(\frac{F(r)}{2\pi r^2}\boldsymbol{m} + \left(\frac{rF'(r) - 2F(r)}{2\pi r^2}\right)\hat{\boldsymbol{x}}(\boldsymbol{m}\cdot\hat{\boldsymbol{x}})\right)$$

In this work the blob was taken as a compact blob

$$B_\delta(r) = \frac{6}{\pi\delta^2}\left(1 - \left(\frac{r}{\delta}\right)\right)^5 \tag{9.3}$$

This gives the following formulas

$$\mathbb{F}(r) = -\left(1 - \left(\frac{r}{\delta}\right)^2\right)^6 + 1 \tag{9.4}$$

$$\frac{\partial B_\delta(r)}{\partial x} = \frac{\partial B_\delta(r)}{\partial r}\frac{\partial r}{\partial x} \tag{9.5}$$

$$= -\frac{60r}{\pi\delta^4}\left(1 - \left(\frac{r}{\delta}\right)^2\right)^4 \frac{|x|}{r} \tag{9.6}$$

$$\frac{\partial B_\delta(r)}{\partial y} = \frac{\partial B_\delta(r)}{\partial r}\frac{\partial r}{\partial x} \tag{9.7}$$

$$= -\frac{60r}{\pi\delta^4}\left(1 - \left(\frac{r}{\delta}\right)^2\right)^4 \frac{|y|}{r} \tag{9.8}$$

9.1.4. Used Identities.

9.1.4.1. *Vector Calculus Identity.* Proposition : If $\Delta\psi = B_\delta$ and $\Delta\phi = \sum \boldsymbol{m}_k \nabla\cdot B_\delta$ then $\phi = \sum \boldsymbol{m}_k \cdot \nabla\psi$

Proof :

$$
\begin{aligned}
\Delta\phi &= \sum \boldsymbol{m}_k \cdot \nabla B_\delta = \sum \boldsymbol{m}_k \cdot \nabla(\Delta\psi) \\
&= \sum \boldsymbol{m}_k \cdot \left(\frac{\partial}{\partial x}\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} \right), \frac{\partial}{\partial y}\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} \right) \right) \\
&= \sum m_1 \frac{\partial}{\partial x}\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} \right) + m_2 \frac{\partial}{\partial y}\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} \right) \\
&= \sum m_1 \frac{\partial^3\psi}{\partial x^3} + m_1 \frac{\partial^3\psi}{\partial y^2 \partial x} + m_2 \frac{\partial^3\psi}{\partial x^2 \partial y} + m_2 \frac{\partial^3\psi}{\partial y^3} \\
&= \sum m_1 \frac{\partial^3\psi}{\partial x^3} + m_2 \frac{\partial^3\psi}{\partial x^2 \partial y} + m_1 \frac{\partial^3\psi}{\partial y^2 \partial x} + m_2 \frac{\partial^3\psi}{\partial y^3} \\
&= \sum \frac{\partial^2}{\partial x^2}\left(m_1 \frac{\partial\psi}{\partial x} + m_2 \frac{\partial\psi}{\partial y} \right) + \frac{\partial^2}{\partial y^2}\left(m_1 \frac{\partial\psi}{\partial x} + m_2 \frac{\partial\psi}{\partial y} \right) \\
&= \sum \Delta \left(m_1 \frac{\partial\psi}{\partial x} + m_2 \frac{\partial\psi}{\partial y} \right) \\
&= \Delta \sum \boldsymbol{m}_k \cdot \nabla\psi
\end{aligned}
$$

Thus since $\Delta\phi = \Delta \sum \boldsymbol{m}_k \cdot \nabla\psi$ we have $\phi = \sum \boldsymbol{m}_k \cdot \nabla\psi$

9.1.4.2. *Derivatives for* $\nabla\phi$. If we recall the definitions we have already established

$$
\frac{\partial r}{\partial x} = \frac{x_1 - z_{k,1}}{r} = \hat{x}_1
$$

$$
\frac{\partial \psi}{\partial r} = \frac{1}{2\pi r} F(r)
$$

and use the folowing relationships

$$
\begin{aligned}
\frac{\partial^2\psi}{\partial r^2} &= \frac{rF'(r) - F(r)}{2\pi r^2} \\
\frac{\partial^2 r}{\partial x^2} &= \frac{(x_2 - z_{k,2})^2}{r^3} = \frac{\hat{x}_2^2}{r} \\
\hat{x}_1^2 + \hat{x}_2^2 &= 1
\end{aligned}
$$

then we can derive derivatives that make the components of $\nabla\phi$.

$$
\begin{aligned}
\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial x}\right) =& \frac{\partial^2 \psi}{\partial r^2}\left(\frac{\partial r}{\partial x}\right)^2 + \frac{\partial^2 r}{\partial x^2}\frac{\partial \psi}{\partial r} \\
=& \left(\frac{rF'(r) - F(r)}{2\pi r^2}\right)\hat{x}_1^2 + \frac{F(r)}{2\pi r}\frac{\hat{x}_2^2}{r} \\
=& \left(\frac{rF'(r) - F(r)}{2\pi r^2}\right)\hat{x}_1^2 + \frac{F(r)}{2\pi r}\frac{1-\hat{x}_1^2}{r} \\
=& \left(\frac{rF'(r) - 2F(r)}{2\pi r^2}\right)\hat{x}_1^2 + \frac{F(r)}{2\pi r^2}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial y}\right) =& \frac{\partial^2 \psi}{\partial r^2}\left(\frac{\partial r}{\partial x}\frac{\partial r}{\partial y}\right) + \frac{\partial^2 r}{\partial x \partial y}\frac{\partial \psi}{\partial r} \\
=& \left(\frac{rF'(r) - F(r)}{2\pi r^2}\right)\hat{x}_1\hat{x}_2 - \frac{F(r)}{2\pi r}\frac{\hat{x}_1\hat{x}_2}{r} \\
=& \left(\frac{rF'(r) - 2F(r)}{2\pi r^2}\right)\hat{x}_1\hat{x}_2 \\
\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial x}\right) =& \frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial r}\frac{\partial r}{\partial y}\right) \\
=& \left(\frac{rF'(r) - 2F(r)}{2\pi r^2}\right)\hat{x}_1\hat{x}_2
\end{aligned}
$$

$$\begin{aligned}\frac{\partial}{\partial y}\left(\frac{\partial\psi}{\partial r}\frac{\partial r}{\partial y}\right) =& \frac{\partial^2\psi}{\partial r^2}\left(\frac{\partial r}{\partial y}\right)^2 + \frac{\partial^2 r}{\partial y^2}\frac{\partial\psi}{\partial r} \\ &= \left(\frac{rF'(r)-F(r)}{2\pi r^2}\right)\hat{x}_2^2 + \frac{F(r)}{2\pi r}\frac{\hat{x}_2^1}{r} \\ &= \left(\frac{rF'(r)-F(r)}{2\pi r^2}\right)\hat{x}_2^2 + \frac{F(r)}{2\pi r}\frac{1-\hat{x}_2^2}{r} \\ &= \left(\frac{rF'(r)-2F(r)}{2\pi r^2}\right)\hat{x}_2^2 + \frac{F(r)}{2\pi r^2}\end{aligned}$$

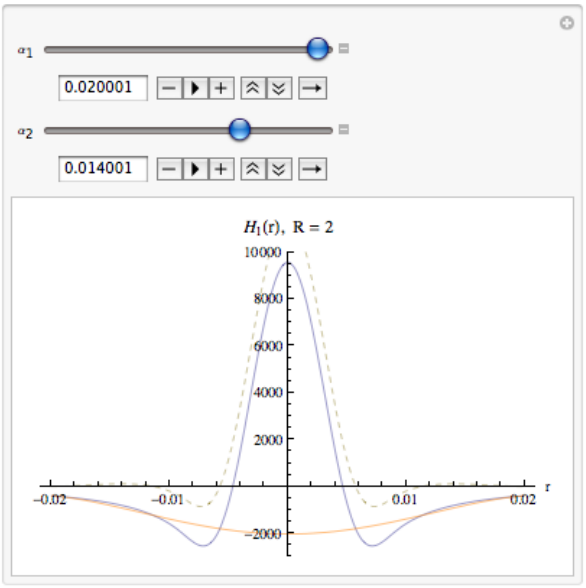
9.1.4.3. *Expanding* $\hat{\boldsymbol{x}}(\boldsymbol{m}\cdot\hat{\boldsymbol{x}})$.

$$\begin{aligned}\hat{\boldsymbol{x}}(\boldsymbol{m}\cdot\hat{\boldsymbol{x}}) &= (\hat{x}_1,\hat{x}_2)\left((m_1,m_2)\cdot(\hat{x}_1,\hat{x}_2)\right) \\ &= (\hat{x}_1,\hat{x}_2)(m_1\hat{x}_1+m_2\hat{x}_2) \\ &= \left(m_1\hat{x}_1^2+m_2\hat{x}_1\hat{x}_2\ ,\ m_1\hat{x}_1\hat{x}_2+m_2\hat{x}_2^2\right)\end{aligned}$$

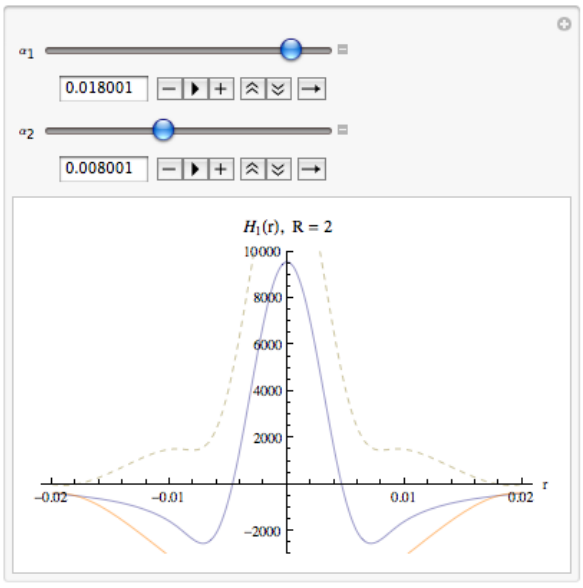
9.2. Additional splitting figures

9.2.1. Screenshots. Additional screenshots of the *Mathematica* program used to determine possible matching conditions for $H_1(r)$ and $H_2(r)$ are shown in Figures 9.1 - 9.4.

9.2.2. Graphs of Matching conditions. Here graphs of possible matching conditions are given according to the variables given in Tables 4.1 and 4.2.

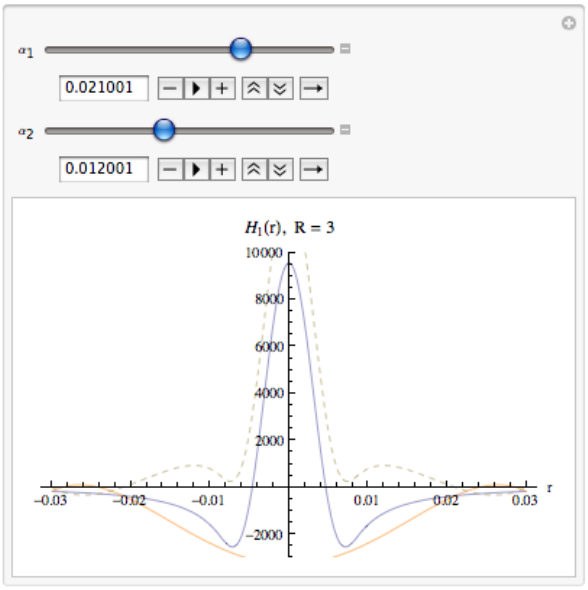


(A) $\alpha_1 = 2\delta$ and $\alpha_2 = 0.7(2\delta)$

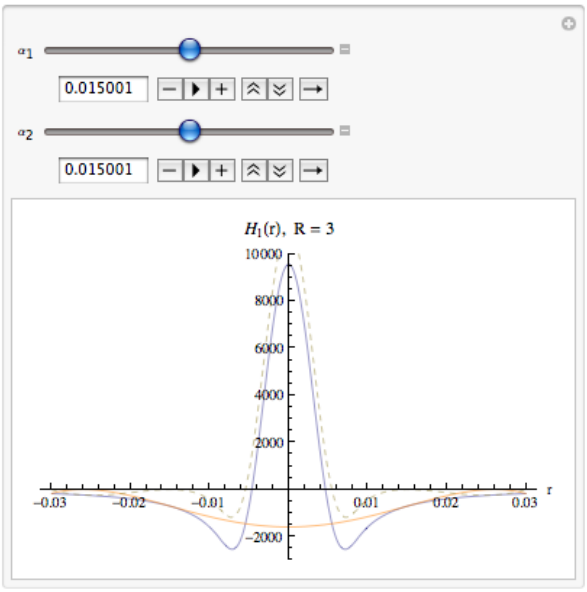


(B) $\alpha_1 = 0.9(2\delta)$ and $\alpha_2 = 0.4(2\delta)$

FIGURE 9.1. Screenshots of $H_1(r)$ with $\mathcal{R} = 2$ and labeled matching conditions

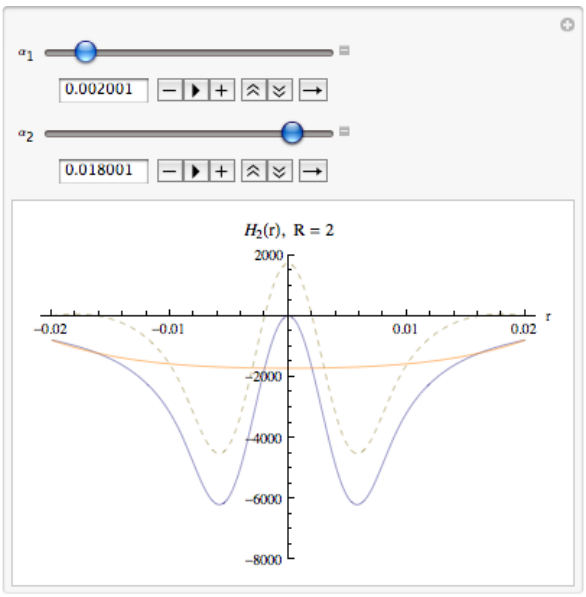


(A) $\alpha_1 = 0.7(3\delta)$ and $\alpha_2 = 0.4(3\delta)$

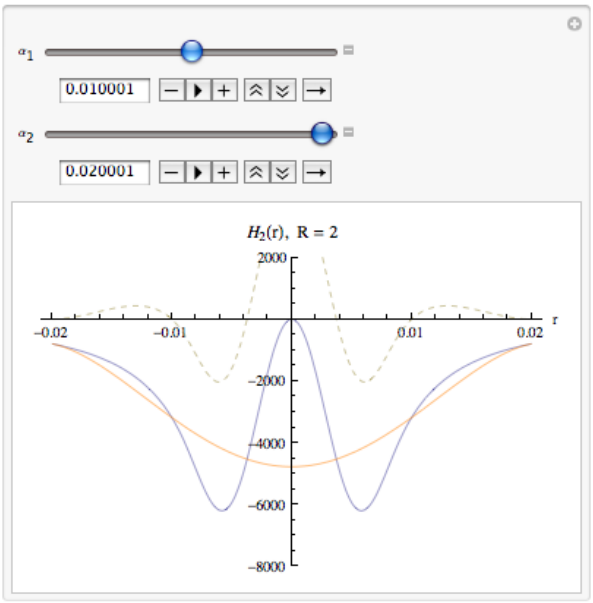


(B) $\alpha_1 = 0.5(3\delta)$ and $\alpha_2 = 0.5(3\delta)$

FIGURE 9.2. Screenshots of $H_1(r)$ with $\mathcal{R} = 3$ and labeled matching conditions

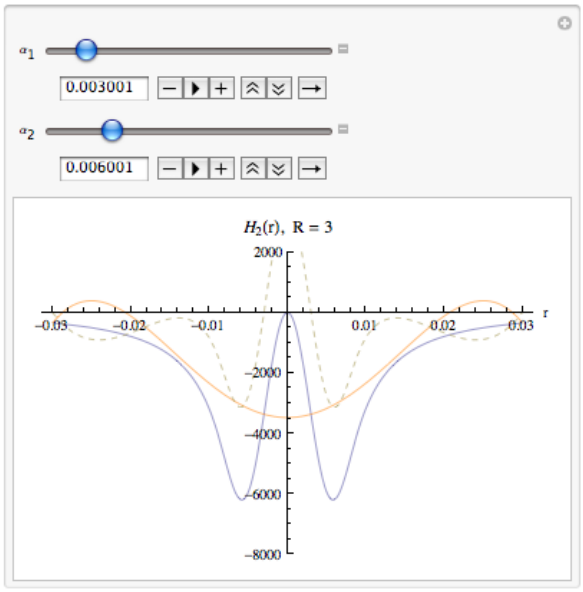


(A) $\alpha_1 = 0.1(2\delta)$ and $\alpha_2 = 0.9(2\delta)$

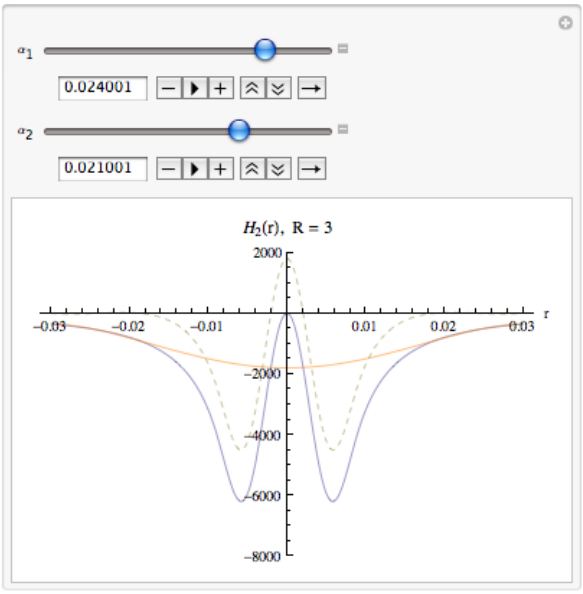


(B) $\alpha_1 = 0.5(2\delta)$ and $\alpha_2 = 2\delta$

FIGURE 9.3. Screenshots of $H_2(r)$ with $\mathcal{R} = 2$ and labeled matching conditions



(A) $\alpha_1 = 0.1(3\delta)$ and $\alpha_2 = 0.2(3\delta)$



(B) $\alpha_1 = 0.8(3\delta)$ and $\alpha_2 = 0.7(3\delta)$

FIGURE 9.4. Screenshots $H_2(r)$ with $R = 3$ and labeled matching conditions

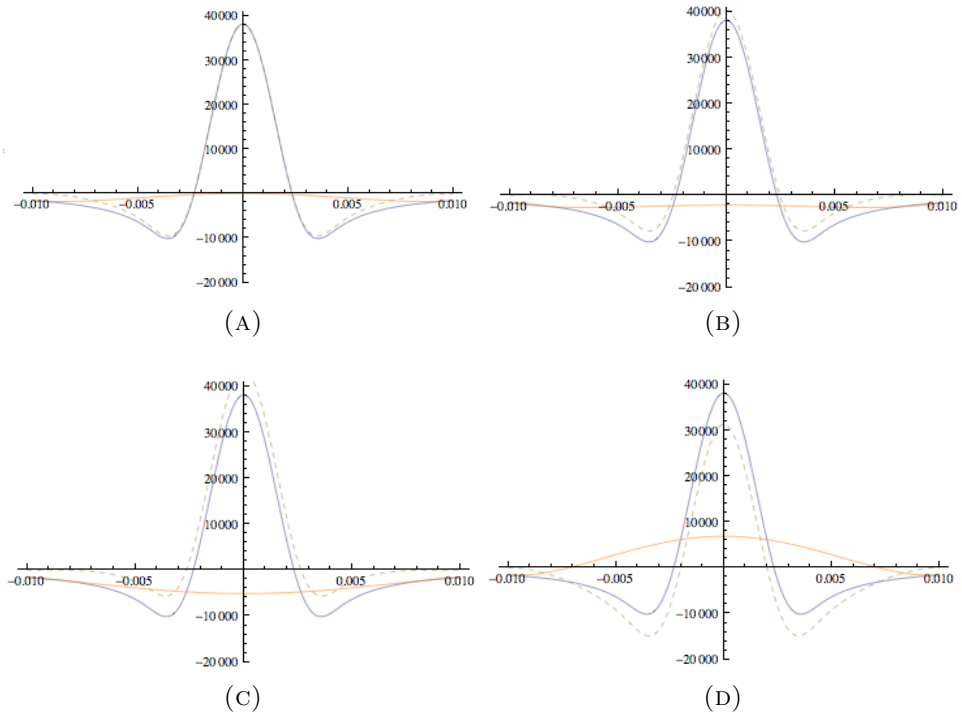


FIGURE 9.5. $H_1(r)$ with $\mathcal{R} = 2$ and matching conditions (v)-(viii)

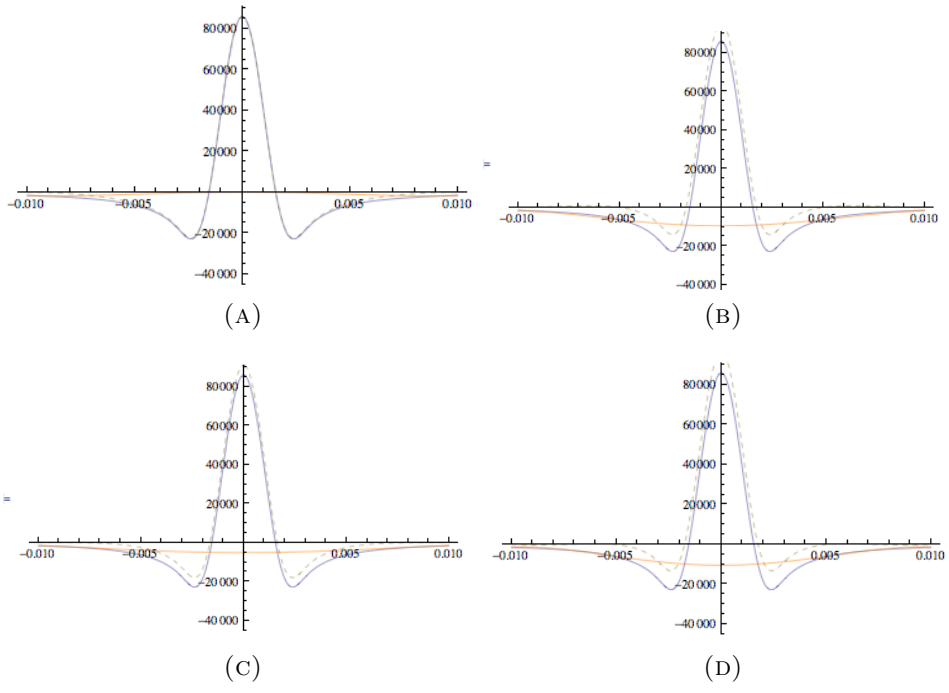


FIGURE 9.6. $H_1(r)$ with $\mathcal{R} = 3$ and matching conditions (ix)-(xii)

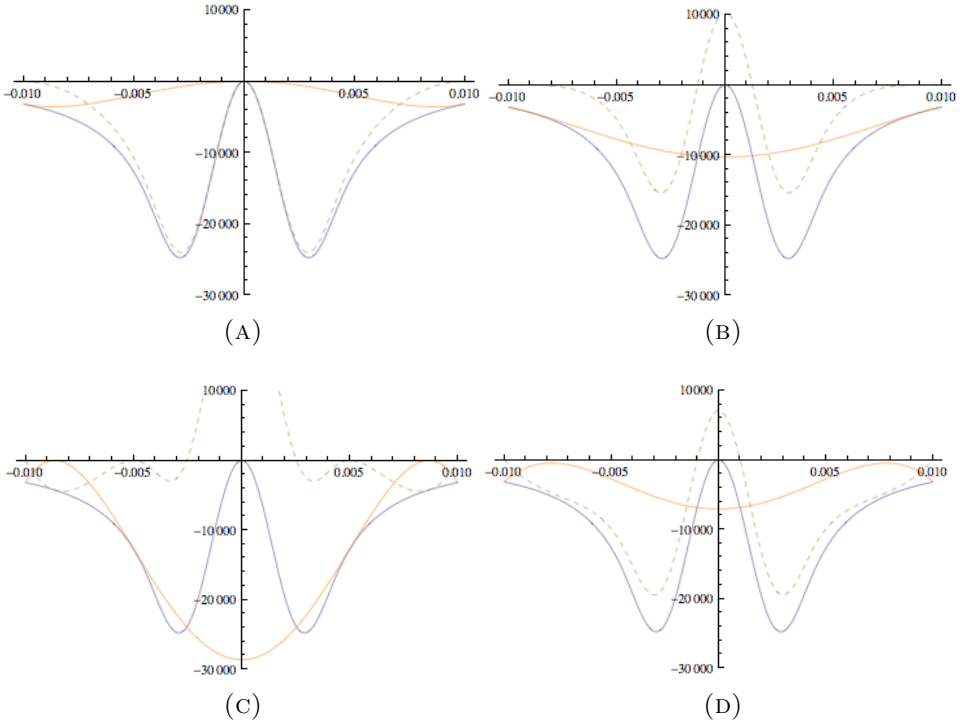


FIGURE 9.7. $H_2(r)$ with $\mathcal{R} = 2$ and matching conditions (v)-(viii)

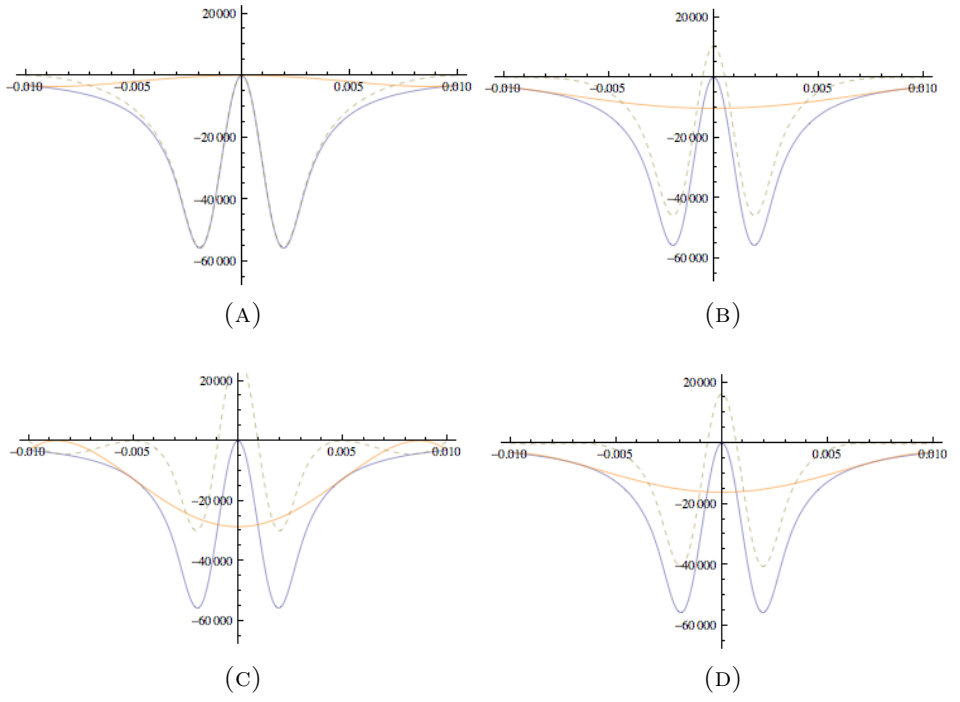


FIGURE 9.8. $H_2(r)$ with $\mathcal{R} = 3$ and matching conditions (ix)-(xii)

References

1. G. Akrivis, M. Crouzeix, and C. Makridakis, *Implicit-explicit multistep methods for quasilinear parabolic equations*, Numer Math (1999), no. 1747, 521–541.

2. A. S. Almgren, A. J. Aspden, J. B. Bell, and M. L. Minion, *On the use of higerh-order projection methods for incompressible turbulent flow.*

3. U. M. Ascher, *Implicit-explicit runge-kutta methods for time-dependent partial differential equations*, Appl Numer Math **25** (1997), no. 2-3, 151–167.

4. U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton, *Implicit-explicit methods for time-dependent partial differential equations*, SIAM J Numer Anal **32** (1995), no. 3, 797–823.

5. J. T. Beale and A. Majada, *High order accurate vortex methods with explicit velocity kernels*, J Comput Phys **58** (1985), 188.

6. K. Böhmer and H. J. Stetter (Eds.), *Defect correction methods, theory and applciations*, Springer-Verlag, Wein-New York, 1984.

7. A. Bourlioux, A. T. Layton, and M. L. Minion, *High-order multi-implicit spectral deferred correction methods for problems of reactive flow*, J Comput Phys **189** (2003), no. 2, 651 – 675.

8. E. L. Bouzarth and M. L. Minion, *A multirate time integrator for regularized stokeslets*, Journal of Computational Physics **229** (2010), no. 11, 4208 – 4224.

9. W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, SIAM, 2000.

10. D. L. Brown, R. Cortez, and M. L. Minion, *Accurate projection methods for the incompressible navier-stokes equations*, J Comput Phys **168** (2001), no. 2, 464 – 499.

11. J. C. Butcher, *The numerical analysis of ordinary differential equations: Runge-kutta and general linear methods*, Wiley-Interscience, New York, NY, USA, 1987.

12. T. F. Buttke, *Lagrangian numerical mehtods which preserve the hamiltonian structure of incompressible fluid flow*, 1993, pp. 39–57.

13. M P Calvo, J De Frutos, and J Novo, *Linearly implicit rungekutta methods for advectionreactiondiffusion equations*, Appl Numer Math **37** (2001), no. 4, 535–549.

14. M. P. Calvo and C. Palencia, *Avoiding the order reduction of runge-kutta methods for linear initial boundary value problems*, Math Comput **71** (2002), 1529–1543.

15. H .D. Ceniceros, J. E. Fisher, and A. M. Roma, *Efficient solutions to robust, semi-implicit discretizations of the immersed boundary method*, J of Comput Phys **228** (2009), no. 19, 7137 – 7158.

16. A. J. Chorin, *Numerical solution of the navier-stokes equations*, Math Comput **22** (1968), 745.

17. _____, *On the convergence of discrete approximations to the navier-stokes equations*, Math Comput **23** (1969), 341.

18. C. W. Clenshaw and A. R. Curtis, *A method for numerical integration on an automatic computer*, Numerische Mathematik **2** (1960), 197–205.

19. R. Cortez, *An impulse-based approximation of fluid motion due to boundary forces*, J Comput Phys **123** (1996), 341.

20. R. Cortez, N. Cowen, R. Dillon, and L. J. Fauci, *Simulation of swimming organisms: coupling internal mechanics with external fluid dynamics*, Comput Sci Eng **6** (2004), 38–45.

21. R. Cortez and M. L. Minion, *The blob projection method for immersed boundary problems*, J Comput Phys **161** (2000), no. 2, 428–453.

22. R. Cortez and D. A. Varela, *The dynamics of an elastic memebrane using the impulse method*, J Comput Phys **128** (1997), 224.

23. J. Crank and P. Nicolson, *A practical method for numerical evaluation of solutions of partial differential equations of the heat-conductive type*, Proc Cambridge Philos Sco **43** (1947), 50–67.

24. C. F. Curtiss and J. O. Hirschfelder, *Integration of stiff equations*, Proc Natl Acad Sci **38** (1952), 235–243.

25. K. Dekker and J. G. Verwer, *Stability of runge-kutta methods for stiff nonlienar differential equations*, North-Holland, 1984.

26. R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact newton methods*, SIAM J Num Anal **19** (1982), no. 2, pp. 400–408.

27. R. H. Dillon, L. J. Fauci, C. Omoto, and X. Yang, *Fluid dynamic models of flagellar and ciliary beating*, Ann NY Acad Sci **1101** (2007), 494–505.

28. A. Dutt, L. Greengard, and V. Rokhlin, *Spectral deferred correction methods for ordinary differential equations*, BIT Numerical Mathematics **40** (2000), 241–266.

29. M. Emmett and M. L. Minion, *Toward an efficient parallel in time method for partial differential equations*, submitted for publication.

30. C. Farhat and M. Chandesris, *Time-decomposed parallel time-integrators: theory and feasiblity studies for fluid, structure and fluid-structure applications.*, Internat J Numer Methods Engeg **58(9)** (2003), 1397–1434.

31. L. J. Fauci, *Interaction of oscillating silaments - a computational study*, J Comput Phys **86** (1990), 294313.

32. L. J. Fauci and C. S. Peskin, *A computational model of aquatic animal locomotion*, J Comput Phys **77** (1988), 85–108.

33. A. L. Fogelson, *A mathematical model and numerical methods for studying platelet adhesion and aggregation during blood clotting*, J Comput Phys **56** (1984), 111134.

34. A. L. Fogelson and R. D. Guy, *Immersed-boundary-type models for intravascular platelet aggregation*, Comp Meth Appl Mech Eng **197** (2008), 2087–2104.

35. Mayo Foundation for Meical Education and Research, http://www.mayoclinic.com/health/medical/IM03955.

36. L. Fox and E. T. Goodwin, *Some new methods for the numerical integration of ordinary differential equations*, Proc Cambridge Philos Soc **45** (1949), 373–388.

37. J. Frank, W. Hundsdorfer, and J. G. Verwer, *On the stability of implicit-explicit linear multistep methods*, Applied Numerical Mathematics **25** (1997), no. 2-3, 193–205.

38. C. W. Gear, *Numerical initial value problems in ordinary differential equations*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1971.

39. L. GREENGARD and V. ROKHLIN, *A FAST ALGORITHM FOR PARTICLE SIMULATIONS*, JOURNAL OF COMPUTATIONAL PHYSICS **73** (1987), no. 2, 325–348.

40. B. E. Griffith, *Immersed boundary method with finite element elasticity*, in review J Comput Phys.

41. B. E. Griffith, R. D. Hornung, D. M. McQueen, and C. S. Peskin, *An adaptive, formally second order accurate version of the immersed boundary method*, J Comput Phys **223** (2007), no. 1, 10–49.

42. B. E. Griffith, X. Lou, D. M. McQueen, and C. S. Peskin, *Simulating the fluid dynamics of natural and prosthetic heart valves using the immersed boundary method*, Int J App Mech **1** (2009), no. 1, 137–177.

43. J. L. Guermond, P. Minev, and J. Shen, *An overview of projection methods for incompressible flows*, Comput Meth Appl Mech Engrg **195** (2006), no. 44-47, 6011–6045.

44. E. Hairer, S. P. Nørsett, and G. Wanner, *Solving ordinary differential equations i, non-stiff problems*, Springer-Verlag, Berlin, 1993.

45. E. Hairer and G. Wanner, *Solving ordinary differential equations ii*, Springer-Verlag, Berlin, 1996.

46. _____, *Stiff differential euqations solved by radau methods*, J Comput App Math **111** (1999), 93–111.

47. Halvard, *File:thysanoptera-thripidae-sp.gif*, 2006, [Online; accessed 20-March-2012].

48. C. L. Hamlet, A. Santhnakrishnan, and L. A. Miller, *A numerical study of the effects of bell pulsation dynamics and oral arms on the exchange currents generated by the upside-down jellfish cassiopea xamachana*, J Exp Biol **214** (2011), 1911–1921.

49. M. Hopkins and L. J. Fauci, *A computational model of the collective fluid dynamics of motile microorganisms*, J Fluid Mech **455** (2002), 149174.

50. T. Y. Hou and Z. Shi, *An efficient semi-implicit immersed boundary method for the navier-stokes equations*, J Comput Phys **227** (2008), no. 20, 8968 – 8991.

51. J. Huang, J. Jia, and M. L. Minion, *Accelerating the convergence of spectral deferred correction methods*, J Comput Phys **214** (2006), no. 2, 633 – 656.

52. P. Hunter, *Biology is the new physics*, EMBO Rep **11** (2010), 350–352.

53. C. T. Kelly, *Iterative methods for linear and nonlinear equations*, SIAM, Philedelphia, 1995.

54. C. A. Kennedy and M. H. Carpenter, *Additive runge-kutta schemes for convection-diffusion-reaction equations*, Appl Numer Math **44** (2002), no. 1-2, 139–181.

55. Y. Kim and C. S. Peskin, *3-d parachute simulation by the immersed boundary method*, Comput Fluids **38 (6)** (2009), 1080–1090.

56. Ming-Chih L. and C. S. Peskin, *An immersed boundary method with formal second-order accuracy and reduced numerical viscosity*, J Comput Phys **160** (2000), no. 2, 705 – 719.

57. A. T. Layton and M. L. Minion, *Conservative multi-implicit spectral deferred correction methods for reacting gas dynamics*, Journal of Computational Physics **194** (2004), no. 2, 697 – 715.

58. D. V. Le, J. White, J. Peraire, K. M. Lim, and B. C. Khoo, *An implicit immersed boundary method for three-dimensional fluid-membrane interactions*, J Comput Phys **228** (2009), 8427–8445.

59. K. Leiderman, L. A. Miller, and A. L. Fogelson, *The effects of spatial inhomogeneities on flow through the endothelial surface layer*, J Theor Bio **252** (2008), no. 2, 313 – 325.

60. J-L. Lions, Y. Maday, and G. Turinici, *A parareal in time discretization of pde's*, C R Acad Sci Paris Serie I **332** (2001), 661–668.

61. A. A. Mayo and C. S. Peskin, *An implicit numerical method for fluid dynamics problems with immersed elastic boundaries*, Contemp Math **141** (1993), 261–277.

62. D. M. McQueen and C. S. Peskin, *Computer assisted design of pivoting-disc prosthetic mitral valves*, J Thorac Cardiovasc Surg **86** (1983), 126–135.

63. _____, *Computer assisted design of butterfly bileaflet valves for mitral position*, Scand J Thorac Cardiovasc Surg **19** (1985), 139–148.

64. _____, *A three-dimensional computational method for blood flow in the heart: (ii) contractile fibers*, J. Comput. Phys. **82** (1989), 289–297.

65. _____, *Heart simulation by an immersed boundary method with formal second order accuracy and reduced viscosity*, Mechanics for a New Millennium, Proceedings of the International Conference on Theoretical and Applied mechanics (ICTAM), Kluwer Academic Publishers, 2000.

66. D. M. McQueen, C. S. Peskin, and E. L. Yellin, *Fluid dynamics of the mitral valve: physiological aspects of a mathematical model*, Am J Physiol **242** (1982), H1095–H1110.

67. L. A. Miller and C. S. Peskin, *A computational fluid dynamics of 'clap and fling' in smallest of insects*, J Exp Biol **208** (2005), 195–212.

68. _____, *Flexible clap and fling in tiny insect flight*, J Exp Biol **212(19)** (2009), 3076–3090.

69. _____, *When vortices stick: and aerodynamic transition in tiny insects*, J Exp Biol **207** (2009), 3076–3090.

70. L. A. Miller, A. Santhanakrishnan, R. Ortiz, A. Keng, J. Cox, K. Leiderman, C. Hamlet, S. Jones, L. Fovargue, D. Fovargue, J. Prins, and M. L. Minion, *Three-dimensional viscous flow through arrays of cylinders with applications to the endothelial surface layer*, in review Bull of Math Bio.

71. M. Minion, *Semi-implicit spectral deferred correction method for ordinary differential equations*, Comm in Math Sci **1** (2003), no. 3, 471–500.

72. M. L. Minion, *A hybrid parareal spectral deferred corrections method*, Comm Appl Math and Comp Sci **5** (20010), no. 2, 265–301.

73. M. L. Minion and S. A. Williams, *Parareal and spectral deferred corrections*, AIP Conference Proceedings, vol. 1048, 2008, pp. 388–391.

74. Y. Mori and C. S. Peskin, *Implicit second-order immersed boundary methods with boundary mass*, Comp Meth in Appl Mech and Engrg **197** (2008), no. 2528, 2049 – 2067.

75. M. Nieuwdorp and et. al., *The endothelial glycocalyx: a potential barrier between health and vascular disease*, Curr Opin Lipidol **15** (2005), 507–511.

76. A. Nonaka, J. B. Bell, M. S. Day, C. Gilet, A. S. Almgren, and M. L. Minion, *A deferred correction coupling strategy for low mach number flow with complex chemistry*, submitted for publication.

77. M. Pemice and H. F. Walker, *Nistol: A newton iteratie solver for nonlinear system*, SIAM J Sci Comput **19** (1998), no. 1, 302–318.

78. V. Pereyra, *Iterated deferred correction for nonlinear boundary value problems*, Numer Math **11** (1968), 111–125.

79. C. S. Peskin, *Numerical analysis of blood flow in the heart*, J Comput Phys **25(3)** (1977), 220–252.

80. ———, *The immersed boundary method*, Acta Numer **11** (2002), 479517.

81. C. S. Peskin and D. M. McQueen, *A three-dimensional computational method for blood flow in the heart: (i) immersed elastic fibers in a viscous incompressible fluid*, J Comput Phys **81** (1989), 372405.

82. J. M. Stockie and B. R. Wetton, *Analysis of stiffness in the immersed boundary method and implications for time-stepping schemes*, J Comp Phys **154** (1999), no. 1, 41 – 64.

83. J. M. Stockie and B. T. R. Wetton, *Stability analysis for the immersed fiber problem*, SIAM J Appl Math **55** (1995), no. 6, 1577.

84. L. N. Trefethen, *Is Gauss quadrature better than Clenshaw-Curtis?*, SIAM Rev **50** (2008), no. 1, 67–87.

85. C. Tu and C. S. Peskin, *Stability and instability in the computation of flows with moving immersed boundaries: A comparison of three methods*, SIAM J Sci Statist Comput **13** (1992), no. 6, 1361–1376.

86. N. T. Wang and A. L. Fogelson, *Computational methods or continuum models of platelet aggregation*, J Comput Phys **151** (1999), 649675.

87. Z. Wang, J. Fan, and K. Cen, *Immersed boundary method for the simulation of 2D viscous flow based on vorticity-velocity formulations*, J Comput Phys (2009).

88. C. S. Peskin Y. Kim, M. Lai, *Numerical simulations of two-dimensional foam by the immersed boundary method*, J Comput Phys **229 (13)** (2010), 5194–5207.

89. P. E. Zadunaisky, *A method for the estimation of errors propagated in the numerical solution of a system of ordinary differential equations, the theory of orbits in the solar system and in stellar systems*, 1964.

90. _____, *On the estimation of errors propagated in the numerical integration of ordinary differential equations*, Numerische Mathematik **27** (1976), no. 1, 21–39.

91. L. Zhu and C. S. Peskin, *Simulation of a flexible flapping filament in a flowing soap film by the immersed boundary method*, J Comput Phys **179** (2002), 452468.