Efficient solutions to robust, semi-implicit discretizations of the Immersed Boundary Method

Hector D. Ceniceros

Department of Mathematics, University of California Santa Barbara , CA 93106

Jordan E. Fisher

Department of Mathematics, University of California Santa Barbara , CA 93106

Alexandre M. Roma

Departamento de Matemática Aplicada, Universidade de São Paulo, Caixa Postal 66281, CEP 05311-970, São Paulo-SP, Brasil.

Abstract

The Immersed Boundary Method is a versatile tool for the investigation of flow-structure interaction. In a large number of applications, the immersed boundaries or structures are very stiff and strong tangential forces on these interfaces induce a well-known, severe time-step restriction for explicit discretizations. This excessive stability constraint can be removed with fully implicit or suitable semi-implicit schemes but at a seemingly prohibitive computational cost. While more economical alternatives have been proposed for special cases, there is a practical need for a computationally efficient approach that can be applied more broadly. Recently, a robust semi-implicit discretization introduced by Peskin in the late 70's has received renewed attention. This discretization, in which the spreading and interpolation operators are lagged, leads to a linear system of equations for the interface configuration at the future time when the interfacial force is linear. However, this linear system is large and dense and thus it is challenging to streamline its solution. Moreover, while the same linear system or one of similar structure

Email addresses: hdc@math.ucsb.edu (Hector D. Ceniceros), www.math.ucsb.edu/~hdc (Hector D. Ceniceros), jordan@math.ucsb.edu (Jordan E. Fisher), roma@ime.usp.br (Alexandre M. Roma), www.ime.usp.br/~roma (Alexandre M. Roma)

could potentially be used in Newton-type iterations, nonlinear and highly stiff immersed structures pose additional challenges to iterative methods. In this work, we address these problems and propose cost-effective computational strategies for solving Peskin's lagged-operators type of discretization. We do this by first constructing a sufficiently accurate approximation to the system's matrix which can be expeditiously obtained by using a combination of pre-computed values and interpolation. The availability of a matrix allows for more efficient matrix-vector products and facilitates the design of effective iterative schemes. We then propose effective iterative methods to deal with both linear and nonlinear interfacial forces and simple or complex immersed structures with tethered or untethered points. One of these iterative approaches employs a splitting in which we first solve a linear problem for the interfacial force and then we use a nonlinear iteration to find the interface configuration corresponding to this force. We demonstrate that our approach is several orders of magnitude more efficient than the standard explicit method. In addition to considering the standard elliptical drop test case, we show both the robustness and efficacy of our proposed methodology with a challenging application of a 2D model of a heart valve.

Key words: semi-implicit method, Stokes flow, Navier-Stokes equations, heart valve, multigrid

---

1. Introduction

The Immersed Boundary (IB) Method introduced by Peskin [**?**] is a versatile tool for simulating flow-structure interaction for a wide range of applications. The IB Method employs a Lagrangian representation of the immersed structures and their interfacial forces and an Eulerian description of the flow variables (velocity and pressure). The Lagrangian description of the immersed boundaries provides a vast structure-building capability while the Eulerian flow description permits the use of efficient flow solvers. The power of the IB method lies in a seamless connection of the two descriptions by the use of two operations: *spreading* (of interfacial forces) and *interpolation* (of velocity at the immersed boundary), both achieved via mollified delta functions.

In a large number of applications, the immersed boundaries or structures are very stiff and strong tangential forces on these interfaces induce severe time-step restrictions for explicit discretization [**?** **?**]. Fully implicit dis-

cretizations and some suitable semi-implicit schemes remove this hindering constraint but seemingly at a cost that makes these options impractical [**?** **?**]. A very economical semi-implicit method has been proposed recently by Hou and Shi [**?** **?**]. This novel approach relies on an ingenious small scale decomposition and becomes explicit in Fourier space. While nearly computationally optimal, the method of Hou and Shi is applicable only to a simple interface, given as a closed curve. Thus, there is still a practical need for a robust, cost-effective approach that can be applied more broadly; one that can be used for immersed structures of complex geometry, composed of mix tethered and untethered links, which are required in many of the applications of the IB Method. The advancement of such an approach is the main focus of this work.

Our starting point is a semi-implicit scheme introduced Peskin [**?**] in the late 70's, in which the spreading and interpolation operators are lagged, i.e. evaluated at the current interfacial configuration rather than at the future one. Variations of this scheme were also considered by Tu and Peskin [**?**] and by Mayo and Peskin [**?**]. This lagged-operators discretization has recently received renewed attention. In particular, Newren, Fogelson, Guy, and Kirby proved that this scheme, in its first order or second order Crank-Nicolson form, is unconditionally stable when inertia is neglected and the interfacial force is linear and self-adjoint [**?**]. Numerical experiments in [**?**], as well as our own experiments, suggest this stability extends to the inertial case with nonlinear interfacial force. Thus, it is now established that Peskin's original semi-implicit discretization enjoys great stability properties and robustness, hence the relevant question is whether its solution can be computed at a reasonable cost. Recently, Mori and Peskin [**?**] took an important step to answer this question. They considered a variation of this scheme, with a linearized tension force discretization which leads to a linear system of equations for the interface configuration at the future time step. They also considered a fully implicit method solved iteratively where each of the iterates has the same structure as the linearized semi-implicit discretization. Mori and Peskin opted for Krylov subspace methods to solve the linear system to take advantage of the fact that matrix-vector products can be obtained via standard operations in the IB method and to avoid the construction of the system's dense matrix.

In this work we demonstrate that the solution to Peskin's operator-lagged discretization can be obtained much more efficiently by working directly with the matrix, and by using suitable linear and nonlinear iterative methods.

More precisely, we construct a sufficiently accurate approximation to the matrix which can be expeditiously obtained by using a combination of precomputed values and interpolation. The availability of a matrix allows for streamlined matrix-vector products and facilitates the design of effective iterative schemes. Most of the work to date on the investigation and removal of the numerical stiffness of the IB Method has focused on a simple test problem: a relaxing elliptical drop. While this test model contains the characteristic high stiffness of the IB approach in a simple interfacial geometry it does not showcase the additional complications that might arise when the immersed structure is composed of crossed links with both tethered and untethered points. Such complex structures are common in applications, starting with the origins of the method to investigate blood flow in the heart [**?**]. In this work, in addition to obtaining efficient methods for the standard elliptical drop test case, we also propose cost-effective iterative methods to deal with cases of complex immersed structures with tethered and untethered points. In our proposed approach we employ a splitting in which we first solve efficiently a linear problem for the interfacial force and then we use a fast-converging nonlinear iteration to find the interface configuration corresponding to this force. We develop this method in the context of a 2D model of a heart valve and demonstrate that our approach is several orders of magnitude more efficient than the standard explicit method.

The rest of the paper is organized as follows. In Section 2, we review the formulation of the IB method. Section 3 deals with the discretization with the focus on Peskin's semi-implicit scheme with lagged-opperators. We devote Section 5 to the construction of the approximation to the matrix and we obtain an estimate for that approximation. In Section 6, we consider the case of the relaxing elliptical drop with both linear and nonlinear forces. Section 7 is devoted to the heart valve problem and the splitting scheme.

2. The Immersed Boundary Method

To describe the method, in its simplest form, we consider a two-dimensional, incompressible, newtonian fluid occupying a domain $\Omega \subset \mathbb{R}^2$. Inside this domain we assume that there is an immersed, neutrally buoyant, elastic structure (also referred to as boundary or interface). This fluid-immersed interface is composed of a system Γ of elastic fibers whose position at any time t is represented in Lagrangian form by $\mathbf{X}(s,t)$, where $s \in B$ is a Lagrangian parameter. The interface Γ need not be closed or even continuous. The

governing equations are:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)=-\nabla p+\mu\nabla^2\mathbf{u}+\mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u}=0, \tag{2}$$

$$\frac{\partial \mathbf{X}}{\partial t}=\mathbf{u}(\mathbf{X},t), \tag{3}$$

where ρ and μ are the density and viscosity, respectively (both assumed to be constant). Here $\mathbf{u}(\mathbf{x},t)$ and $p(\mathbf{x},t)$ are the velocity field and the pressure, respectively, described in terms of the Eulerian, cartesian coordinate $\mathbf{x}$. The term $\mathbf{f}$ represents the singularly supported interfacial (tension) force of the immersed structure acting onto the fluid. The system (1)-(3) is supplemented with initial and boundary conditions. Throughout this work, we consider only periodic boundary conditions and Ω is a rectangular domain.

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

$$\frac{\partial \mathbf{X}}{\partial t}=\int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\mathbf{X}(s,t))d\mathbf{x}, \tag{4}$$

$$\mathbf{f}(\mathbf{x},t)=\int_B \mathbf{F}(\mathbf{X}(\cdot,\cdot)s,t)\delta(\mathbf{x}-\mathbf{X}(s,t))ds, \tag{5}$$

where δ denotes the (two-dimensional) Dirac delta distribution. In (5), $\mathbf{F}$ represents the elastic force density of Γ and is described in Lagrangian coordinates. For example, if the tangent direction $\mathbf{t}$ along the fibers varies smoothly and if the local elastic energy density is assumed to depend only on the tangential strain $|\frac{\partial \mathbf{X}}{\partial s}|$ then

$$\mathbf{F}(\mathbf{X},s,t)=\frac{\partial}{\partial s}\left(T\left(\left|\frac{\partial \mathbf{X}}{\partial s}\right|\right)\mathbf{t}\right). \tag{6}$$

Here $T(|\frac{\partial \mathbf{X}}{\partial s}|)$ is the fiber (interfacial) tension and $\mathbf{t}$ is the unit tangent,

$$\mathbf{t}=\frac{\frac{\partial \mathbf{X}}{\partial s}}{|\frac{\partial \mathbf{X}}{\partial s}|}. \tag{7}$$

Thus $\mathbf{F}$ is in general a nonlinear function of the interfacial configuration. We denote this relation by

$$\mathbf{F}=\mathcal{A}(\mathbf{X}). \tag{8}$$

3. Discretization

We consider uniform cartesian grids $\mathcal{G}_\Omega$ and $\mathcal{G}_B$ with grid size h and h_B to discretize Ω and B respectively and employ standard second order finite differences for the spatial derivatives. We write Peskin's original semi-implicit discretization in the form

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}+\mathbf{u}^n\cdot\mathbf{D}_h\mathbf{u}^n\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n+1}), \quad (9)$$

$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0, \quad (10)$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^n}{\Delta t}=\mathcal{S}_n^*\mathbf{u}^{n+1}, \quad (11)$$

where a superscript m denotes a numerical approximation taken at the time $m\Delta t$ and Δt is the time step. The spatial operators $\mathbf{D}_h$ and L_h are the standard second order approximations to the gradient and the laplacian, respectively, and $\mathcal{A}_{h_B}$ is a suitable discrete version of $\mathcal{A}$.

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

$$(\mathcal{S}_n G)(\mathbf{x})=\sum_{s\in\mathcal{G}_B} G(s)\delta_h(\mathbf{x}-\mathbf{X}^n(s))h_B, \quad (12)$$

$$(\mathcal{S}_n^* w)(s)=\sum_{\mathbf{x}\in\mathcal{G}_\Omega} w(\mathbf{x})\delta_h(\mathbf{x}-\mathbf{X}^n(s))h^2, \quad (13)$$

where $\delta_h(\mathbf{x})=d_h(x)d_h(y)$ and d_h is an approximation of the one-dimensinal delta. These operators are called lagged because the interface configuration $\mathbf{X}^n$ is used instead of the future configuration $\mathbf{X}^{n+1}$.

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

$$\delta_h(r)=\begin{cases}\frac{1}{4h}\left(1+\cos(\frac{\pi r}{2h})\right) & \text{if } |r|\le 2h\\ 0 & \text{otherwise.}\end{cases} \quad (14)$$

Let us rewrite (9) as

$$\mathbf{u}^{n+1}=-\frac{\Delta t}{\rho}\mathbf{D}_h p^{n+1}+\Delta t\nu L_h\mathbf{u}^{n+1}+\mathbf{a}^{n+1}, \quad (15)$$

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

$$\mathbf{a}^{n+1} = \frac{\Delta t}{\rho}\, \mathcal{S}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{u}^n - \Delta t\ \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n. \tag{16}$$

We can eliminate the pressure term in (15) using (10) by introducing a discrete projection P_h defined as

$$\mathbf{v} = P_h\mathbf{v} + \mathbf{D}_h\phi_v, \qquad \mathbf{D}_h \cdot P_h\mathbf{v} = 0, \qquad P_h\mathbf{D}_h\phi_v = 0. \tag{17}$$

for any smooth vector field $\mathbf{v}$ defined on the grid $\mathcal{G}_\Omega$. Applying P_h to (15), using (10) and that for periodic boundary conditions L_h and P_h commute we get

$$\mathbf{u}^{n+1} = \Delta t \nu L_h \mathbf{u}^{n+1} + P_h \mathbf{a}^{n+1}, \tag{18}$$

that is

$$\mathbf{u}^{n+1} = (I - \Delta t \nu L_h)^{-1} P_h \mathbf{a}^{n+1}. \tag{19}$$

Let us denote

$$\mathcal{L}_h = (I - \Delta t \nu L_h)^{-1} P_h. \tag{20}$$

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

Using this notation Peskin's semi-implicit method can be encoded as

$$\mathbf{u}^{n+1} = \mathcal{L}_h \mathbf{a}^{n+1}, \tag{21}$$

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathbf{u}^{n+1}, \tag{22}$$

where $\mathbf{a}^{n+1}$ is given by (16). Eliminating $\mathbf{u}^{n+1}$ in (22) we obtain a system of equations for the immersed boundary configuration $\mathbf{X}^{n+1}$:

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_b}(\mathbf{X}^{n+1}) + \mathbf{b}^n, \tag{23}$$

where

$$\mathcal{M}_n = \alpha\, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n, \tag{24}$$

with

$$\alpha = \frac{(\Delta t)^2}{\rho} \tag{25}$$

and

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h [\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n]. \tag{26}$$

We have thus reduced (9)-(11) to a single system of equations involving only the unknown $\mathbf{X}^{n+1}$. If the number of Lagrangian nodes is N_b then (23) represents a system of $2N_b$ equations for the $2N_b$ values $\mathbf{X}^{n+1} = (X^{n+1}, Y^{n+1})$. If we can solve this system then we can obtain $\mathbf{u}^{n+1}$ via (21).

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

$$(u, v)_\Omega = \sum_{\mathbf{x} \in \mathcal{G}_\Omega} u(\mathbf{x}) v(\mathbf{x}) h^2, \tag{27}$$

$$(\mathbf{u}, \mathbf{v})_\Omega = (u_1, v_1)_\Omega + (u_2, v_2)_\Omega, \tag{28}$$

and on B as

$$(F, G)_B = \sum_{\mathbf{x} \in \mathcal{G}_B} F(\mathbf{x}) G(\mathbf{x}) h_B, \tag{29}$$

$$(\mathbf{F}, \mathbf{G})_B = (F_1, G_1)_B + (F_2, G_2)_B, \tag{30}$$

then $\mathcal{S}_n$ and $\mathcal{S}_n^*$ as given in (12)-(13) are adjoints to each other and

$$(\mathbf{F}, \mathcal{M}_n \mathbf{F})_B = \alpha (\mathbf{F}, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n \mathbf{F})_B = \alpha (\mathcal{S}_n \mathbf{F}, \mathcal{L}_h \mathcal{S}_n \mathbf{F})_B \geq 0. \tag{31}$$

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

The focus of this paper is to propose efficient methods for solving (23) to be able to remove the severe numerical stiffness of the IB method in an

economical and robust fashion for a wide range of practical flow-structure situations. Of course, to be efficient the specific computational approach has to be dependent on the geometry of the immersed structure and the force operator $\mathcal{A}_{h_b}$. This is something that has been largely overlooked in the literature as the research has concentrated mostly on understanding and removing the stiffness on the simplest setting: a simple closed (elliptical) interface with a linear density force $\mathcal{A}_{h_b}$. Here, we consider both linear and nonlinear $\mathcal{A}_{h_b}$ with simple and complex immersed structure geometries to highlight the challenges in producing efficient solvers and to illustrate our proposed approaches.

4. Preliminary discussion of computational costs and efficiency

One of the most commonly used schemes with an explicit treatment of the immersed boundary is the so-called Forward Euler/Backward Euler (FE/BE) [**?**] in which the tension force is explicit (Forward Euler) and the viscous term is implicit (Backward Euler). That is,

$$\mathbf{u}^{n+1} = \mathcal{L}_h \left[\frac{\Delta t}{\rho} \mathcal{S}_n \mathcal{A}_{h_B}(\mathbf{X}^n) + \mathbf{u}^n - \Delta t \ \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n \right], \tag{32}$$

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathbf{u}^{n+1}. \tag{33}$$

The main cost of this scheme per time step is the fluid solver, i.e the operation involving $\mathcal{L}_h$. On the other hand any solution method for (23) requires the evaluation of $\mathbf{b}^n$, given by (26), and thus at least one fluid solver operation. Hence, it seems appropriate to measure the cost of iterative methods for (23) relative to that of one FE/BE time-step and we will refer to this unit as one FE/BE. Our goal is to present robust solution methods to (23) that have cost of just a few FE/BE's. Naturally, because the semi-implicit scheme allows for time steps several orders of magnitude larger than those permitted by FE/BE in a stiff problem, the extra computational work per time step will be more than compensated and a speed-up of several magnitudes could be achieved.

In the design of efficient iterative methods for (23) it is crucial to streamline the calculation of quantities of the form $\mathcal{M}_n\mathbf{F}$. These quantities can be computed in two ways. Given an interface-defined vector $\mathbf{F}$ we can apply in sequence the operations of spreading, fluid solver, and interpolation. Alternatively, we can construct a matrix representation of $\mathcal{M}_n$ and use matrix-vector

multiplication. Given that a direct computation of the matrix is prohibitively expensive, $Nb \times FE/BE$, all related work to date has avoided the latter approach. However, there are some advantages of having a matrix representation of $\mathcal{M}_n$ and, as we show in the next section, it is possible to construct a sufficiently accurate approximation to the matrix in only $O(FE/BE)$ operations.

One of the advantages of having a matrix representation of the operator $\mathcal{M}_n$ is that we can gain access to a wider range of iterative methods. Multigrid smoothers like Gauss-Seidel or S.O.R., which are inaccessible with the spreading-fluid solver-interpolation approach, could be easily implemented if the matrix is available. Also, multigrid-based methods are greatly simplified when we have a matrix representation of $\mathcal{M}_n$, which allows us to employ a straightforward algebraic multigrid as opposed to a geometric one. One of the more subtle issues with using a geometric multigrid is how to coarsen the Eulerian grid. Uniform coarsening would be ineffective for example in the case of our heart valve model because as we coarsen the valve geometry there remain points close together which would require a fine Eulerian mesh to resolve, see Fig. 3. If a geometric multigrid were to be employed we would need to implement adaptive coarsening.

A second advantage of having a matrix representation of $\mathcal{M}_n$ is that we can obtain direct solutions of *coarse* (small) linear systems of the form $M_n\mathbf{F} = \mathbf{Z}$ which could be useful in a number of situations; an instance of this will be discussed later in the context of the heart valve problem. But perhaps the main advantage is that of cost when computing quantities of the form $\mathcal{M}_n\mathbf{F}$. If this matrix-vector product is computed via the operator sequence spreading-fluid solver-interpolation the cost is about one FE/BE or $O(N^2 \log N) + O(N_b)$, when N^2 is the number of Eulerian nodes and the fluid solver is based on the Fast Fourier Transform (FFT). On the other hand, a matrix-vector multiplication involving $\mathcal{M}_n$ requires $O(N_b^2)$ operations. If $N_b \sim N$ as it is typical in 2D applications, then the second approach has lower computational complexity. While the elimination of the factor $\log N$ might seem like a modest gain, we find that in practice (for $N_b = 2N$) using the matrix representation to compute matrix-vector products is significantly faster than the spreading-fluid solver-interpolation approach, even at modest resolutions. Table 1 gives the CPU time in units of FE/BE (average CPU time of one FE/BE time-step for a given $N \times N$ Eulerian grid and $N_b = 2N$) for matrix-vector multiplications computed using a matrix representation of $\mathcal{M}_n$. For example, at $N_b = 1024$, the computation of a matrix-vector product

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

Table 1: Average CPU time in FE/BE units for a matrix-vector multiplication involving M_n for given $N_b = 2N$

with this approach is 1/71 FE/BE, i.e. 71 times faster than it would be using the spreading-fluid solver-interpolation option. If in addition, we take into the account that many matrix-vector products are needed in the course of an iterative method then the computational savings are significant. We note however that as the ratio N_b/N increases the savings get reduced and the computational advantage of using the matrix to obtain the product could be lost eventually. For example, in our application of the heart valve model where $N_b \sim 4N$, the speedup in using the matrix form drops to roughly 10 at moderate to fine resolutions. While not as dramatic as that in the $N_b = 2N$ case, it still leads to substantial savings in the overall algorithm. However, in a fully 3D application with a 2D immersed membrane we would have $N_b \sim N^2$ leading to a cost of $O(N^4)$ for a matrix-vector multiplication using the matrix while obtaining the same product via spreading-fluid solver-interpolation would be $O(N^3 \log N)$.

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

Let us consider again the linear fluid solver operator (20) with periodic boundary conditions. Utilizing the Fourier transform we obtain a representation formula of the form

$$\mathcal{L}_h \mathbf{f}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{G}_\Omega} G_h(\mathbf{x} - \mathbf{y}) \mathbf{f}(\mathbf{y}) h^2, \qquad \text{for } \mathbf{x} \in \mathcal{G}_\Omega, \tag{34}$$

where the Green function $G_h(\mathbf{x} - \mathbf{y})$ is a 2×2 matrix with vanishing zero Fourier mode.

Recall that $\mathcal{M}_n = \alpha \, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n$, thus its entries $(\mathcal{M}_n)_{ij}$ correspond to α times the velocity that is obtained by *interpolating* the values produced at a given interfacial node $\mathbf{X}_i$ by *spread* unit horizontal and vertical forces located at another immersed boundary node $\mathbf{X}_j$. For the continuum problem this velocity depends only on the difference $\mathbf{X}_j - \mathbf{X}_i$ but due to the spreading and interpolation operators this translation invariance is not exact at the

discrete level. However, as we prove later, we can still obtain sufficiently accurate approximations to $(\mathcal{M}_n)_{ij}$ by assuming translation invariance with the added benefit of a dramatic cost reduction. Specifically, we propose to approximate $(\mathcal{M}_n)_{ij}$ with values obtained by shifting both $\mathbf{X}_i$ and $\mathbf{X}_j$ an equal amount such that $\mathbf{X}_j$ lies exactly on an Eulerian node, which we may take as the origin. That is, we can fix the point (the origin) at which unit horizontal and vertical forces are applied (and spread) and then evaluate the effects everywhere else on the Eulerian grid by applying just *two* fluid solves: one each for a horizontal and vertical force at the origin. These Eulerian grid values can be pre-computed at the beginning of the simulation and then be used as a look-up table to obtain the corresponding values at any interfacial point $\mathbf{X}_i$ via standard interpolation. The generation of the one-time lookup table is only two FE/BE and the interpolation to generate the approximation of $\mathcal{M}_n$ is only $O(N_b^2)$.

We proceed now to detail the computation of $\mathcal{M}_n$. We write the configuration $\mathbf{X}$ of the discretized immersed structure as the $2Nb$-array

$$\mathbf{X} = \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} X_1 \\ \vdots \\ X_{N_B} \\ Y_1 \\ \vdots \\ Y_{N_B} \end{bmatrix}, \tag{35}$$

and similarly we write $\mathbf{F} = \mathcal{A}_{h_B}\mathbf{X} = (F, G)^T$. We seek then four $N_b \times N_b$ matrices A,B,C,D such that

$$\mathcal{M}_n\mathbf{F} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix}. \tag{36}$$

To calculate the entries of the sub-matrices composing $\mathcal{M}_n$, we consider horizontal and vertical, unit point forces $\mathbf{e}_1^j$ and $\mathbf{e}_2^j$ for $1 \leq j \leq N_B$, corresponding to each Lagrangian node. These are $2N_B$-arrays given by

$$(\mathbf{e}_1^j)_i = \begin{cases} 0 & \text{If } i \neq j, \\ 1 & \text{If } i = j \end{cases} \qquad 1 \leq i \leq 2N_B \tag{37}$$

and

$$(\mathbf{e}_2^j)_i = \begin{cases} 0 & \text{If } i \neq j + N_B, \\ 1 & \text{If } i = j + N_B \end{cases} \qquad 1 \leq i \leq 2N_B. \tag{38}$$

Then, we have for $1 \leq i \leq 2N_B$

$$(\mathcal{M}_n)_{ij} = (\mathcal{M}_n \mathbf{e}_1^j)_i \quad \text{for } 1 \leq j \leq N_B, \tag{39}$$

$$(\mathcal{M}_n)_{ij} = (\mathcal{M}_n \mathbf{e}_2^{j-N_B})_i \quad \text{for } N_B + 1 \leq j \leq 2N_B. \tag{40}$$

We define also

$$\mathbf{e}_1 = (1, 0) \tag{41}$$

$$\mathbf{e}_2 = (0, 1). \tag{42}$$

Now,

$$(\mathcal{M}_n \mathbf{e}_1^j)_i = \left(\alpha \, \mathcal{S}_n^* \mathcal{L}_h \sum_{\mathbf{X}_k \in \mathcal{G}_B} (\mathbf{e}_1^j)_k \delta_h(\mathbf{x} - \mathbf{X}_k) h_B \right)_i \tag{43}$$

$$= \left(\alpha h_B \, \mathcal{S}_n^* \mathcal{L}_h \left(\mathbf{e}_1 \delta_h(\mathbf{x} - \mathbf{X}_j) \right) \right)_i \tag{44}$$

$$= \alpha h_B \sum_{\mathbf{z} \in \mathcal{G}_\Omega} \mathcal{L}_h \left(\mathbf{e}_1 \delta_h(\mathbf{x} - \mathbf{X}_j) \right) \delta_h(\mathbf{z} - \mathbf{X}_i) h^2 \tag{45}$$

$$\approx \alpha h_B \sum_{\mathbf{z} \in \mathcal{G}_\Omega} \mathcal{L}_h \left(\mathbf{e}_1 \delta_h(\mathbf{x}) \right) \delta_h \left(\mathbf{z} - (\mathbf{X}_i - \mathbf{X}_j) \right) h^2 \tag{46}$$

The last approximation allows us to define a single, vector valued function

$$T(\mathbf{y}) = \alpha h_B \sum_{\mathbf{z} \in \mathcal{G}_\Omega} \mathcal{L}_h \left(\mathbf{e}_1^j \delta_h(\mathbf{x}) \right) \delta_h \left(\mathbf{z} - \mathbf{y} \right) h^2 \tag{47}$$

with respect to which we can compute the entries of the submatrices A and C. We have

$$\begin{bmatrix} (\mathcal{M}_n \mathbf{e}_1^j)_i \\ (\mathcal{M}_n \mathbf{e}_1^j)_{i+N_B} \end{bmatrix} \approx T(\mathbf{X}_i - \mathbf{X}_j). \tag{48}$$

Thus, the entries of the submatrices are given by

$$\begin{bmatrix} A_{ij} \\ C_{ij} \end{bmatrix} \approx T(\mathbf{X}_i - \mathbf{X}_j). \tag{49}$$

Similarly, we can obtain the submatrices B and D by considering the vertical point forces $\mathbf{e}_2^j$. Proceeding as before, we get

$$\begin{bmatrix} (\mathcal{M}_n \mathbf{e}_2^j)_i \\ (\mathcal{M}_n \mathbf{e}_2^j)_{i+N_B} \end{bmatrix} \approx U(\mathbf{X}_i - \mathbf{X}_j). \tag{50}$$

where

$$U(\mathbf{y}) = \alpha h_B \sum_{\mathbf{z}\in\mathcal{G}_\Omega} \mathcal{L}_h\left(\delta_h(\mathbf{x})\right)\delta_h\left(\mathbf{z}-\mathbf{y}\right)h^2. \tag{51}$$

Hence,

$$\begin{bmatrix} B_{ij} \\ D_{ij} \end{bmatrix} \approx U(\mathbf{X}_i - \mathbf{X}_j). \tag{52}$$

The domain of both T and U is Ω. To expedite calculation we precompute both functions on the discretized grid $\mathcal{G}_\Omega$. Any value of T or U may now be approximated via an interpolation between values on grid points. Each evaluation $T(\mathbf{x})$ or $U(\mathbf{x})$ now costs only $\mathcal{O}(1)$ hence in this manner we construct the entire matrix representation of $\mathcal{M}_n$ with $\mathcal{O}(N_B^2)$ cost.

5.1. Error in approximating the matrix representation of $\mathcal{M}_n$

Note though that on a square domain Ω we have via symmetry that $U(x,y) = T(y,x)$. For rectangular or otherwise irregular grids this symmetry does not hold, we do however have other symmetries on rectangular domains,

$$T(x,y) = T(-x,y), \quad T(x,y) = T(x,-y), \quad T(x,y) = T(-x,-y), \tag{53}$$
$$U(x,y) = U(-x,y), \quad U(x,y) = U(x,-y), \quad U(x,y) = U(-x,-y). \tag{54}$$
$$\tag{55}$$

This implies that equal force densities at two fiber points produce equal effects on each other, hence M_n is a symmetric matrix.

Suppose we were to use T to construct our entire matrix M_n. We would require $2N_b$ calls to T to construct M_n entirely which is prohibitively expensive if a fluid solve is invoked with every evaluation. The simplest method to counter this is to create a lookup table housing the values of T. It would be natural to simply precalculate all values of T and U on the Eulerian grid, letting $T_{i,j} = T(\mathbf{x}_{i,j}), U_{i,j} = U(\mathbf{x}_{i,j})$. However, fiber points may lie between grid points, thus we may lose some important dynamics by only defining T and U on $\mathbf{x}_{i,j}$. To counter this we may either create a lookup table on a finer grid or we can use linear interpolation between grid points. The latter is the

strategy we used for our simulations. We approximate

$$T(x,y) \approx a_1 a_2 T_{i,j} + a_1(1-a_2)T_{i,j+1} + (1-a_1)a_2 T_{i+1,j} + (1-a_1)(1-a_2)T_{i+1,j+1}, \tag{56}$$

$$a_1 = \frac{x - x_{i,j}}{x_{i+1,j} - x_{i,j}}, \tag{57}$$

$$a_2 = \frac{y - y_{i,j}}{y_{i,j+1} - y_{i,j}}, \tag{58}$$

where i, j are the largest integers such that $x > x_{i,j}$ and $y > y_{i,j}$.

Calculating M_n with a lookup table costs $\mathcal{O}(N_b^2)$. When N_b is particularly large, however, this cost can overshadow the cost of fluid solves. While we haven't taken any additional means of reducing the cost of computing M_n in our simulations we note briefly some ways one may. Calculating M_n with direct lookup rather than using interpolation is substantially faster but with undesirably low accuracy. A simple compromise is to use linear interpolation when calculating $T(x,y)$ if $(x^2+y^2)^{1/2}$ is small and direct lookup otherwise. This is cheap, with only $\mathcal{O}(N_b)$ interpolations needed, because most fiber points are distant from each other, as well as accurate, because T decays rapidly away from the origin. Alternatively we could employ an adaptive mesh on which to calculate T, taking a dense grid around the origin where most of the structure of T lies. This would allow us to use direct lookup for all entries of M_n without a loss in accuracy.

A further reduction in cost can be had by only computing part of M_n. Consider the case where A_{h_b} is linear. If we modify our definition of $\mathbf{b}^n$ to be

$$\mathbf{b}^n = \Delta t \mathcal{S}_n^* \mathcal{L}_h(\Delta t \mathcal{S}_n A_{h_b} \mathbf{X}^n + \rho \mathbf{u}^n - \Delta t \rho \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n) \tag{59}$$

then we can find $\mathbf{X}^{n+1}$ by solving for the change

$$\mathbf{X}^{n+1} - \mathbf{X}^n = M_n A_{h_b}(\mathbf{X}^{n+1} - \mathbf{X}^n) + \mathbf{b}^n. \tag{60}$$

Here, however, the cumulative influence of effects between distant fiber points is quite small since we have already factored in their aggregate effect through $\mathbf{b}^n$, thus we may choose to ignore them, using only a banded component of M_n to capture the pertinent influences. This can lead to speed up in calculating M_n as well as matrix-vector multiplication involving M_n if sparse data structures are used. This trick may prove particularly useful for problems with very large Ω, where large distances between immersed fibers drastically

diminish their influence on each other's stability. For instance, in our heart valve problem much of the horizontal boundary modeled with tethered nodes has little impact on the stability of the fiber comprising the valve. The corresponding entries in M_n relating these two parts of our immersed fibers could be held at zero if we utilized (125) and (126).

We have yet to mention the cost of computing the matrix of M_n. Indeed, if one is not careful this cost can swamp any advantages gleaned. We use a simple method to insure the cost remains nominal. The cost our simulations incur by calculating the matrix of M_n is listed in table **??** in the row dubbed Matrix construction. We see that while the cost of constructing M_n is much larger than multiplication involving M_n the cost is nonetheless smaller than the cost of a FE/BE timestep when $N_b = 2N$.

5.2. Error introduced

Our use of the lookup tables for T and U introduces error into our simulations in two ways. First, the linear interpolation used between values in the table, second, in the error introduced via (112). Linear interpolation introduces error of $\mathcal{O}(h^2)$, assuming our lookup table sits on the same Eulerian grid as Ω_h. The second error is more difficult to account for and we make no aims of quantifying it analytically. We instead rely on numerical experiments.

We work within the context of the ellipse relaxation problem with $N_b = 2N$. Assume that A is the exact matrix representation of M_n and B is the approximate matrix representation of M_n arrived at via the methods given above. We calculate both matrices and then compute the maximum norm between their difference. Two log-log graphs of the norm with respect to N are given in figures 7 and 8. In figure 7 we have fixed $\Delta t = h$ and we see that the error is $\mathcal{O}(h^2)$. In figure 8 we have fixed $\Delta t = .01h$ and our error worsens to $\mathcal{O}(h^{1.5})$.

It's clear that a shift is occurring where at first we have $\Delta t^p \gg h^q$ when $\Delta t = h$ and changing to $\Delta t^p \ll h^q$ when $\Delta t = .01h$. This suggests that $p = 1$. We are thus maintaining first order accuracy in time, as would be expected, however we lose spatial accuracy due to the transitions between discrete and distributional delta functions, leading to $q = .5$. Nonetheless the overall error, even after linear interpolation, is still at worst $\mathcal{O}(h^{1.5})$ which is smaller than the $\mathcal{O}(h)$ error of the overarching method. Thus we can be confident that our approximations to M_n are not introducing any unwarranted error.

6. Application: Ellipse Relaxation

Consider a closed, continuous membrane $\mathbf{X}(s,t)$ with a force distribution given by $\mathcal{A}(\mathbf{X}) = \sigma \mathbf{X}_{ss}$, where σ is a (large) constant. We take our domain as $\Omega = [0,1] \times [0,1]$ with periodic boundary conditions and we fix $\mu = \rho = 1$. Figure 1 shows the initial and final configurations of the interface. We discretize $\mathcal{A}(\mathbf{X})$ as

$$\left(\mathcal{A}_{h_b}\mathbf{X}\right)_i = \frac{1}{h_b^2}\left(\mathbf{X}_{i+1} - 2\mathbf{X}_i + \mathbf{X}_{i-1}\right), \tag{61}$$

where we have omitted the parentheses in the discrete force operator $\mathcal{A}_{h_B}$ to emphasize that it is linear. Thus, the semi-implicit discretization produces the linear system

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_b} \mathbf{X}^{n+1} + \mathbf{b}^n \tag{62}$$

to be solved at each time step. Equivalently, we have

$$(I - \mathcal{M}_n \mathcal{A}_{h_b})\mathbf{X}^{n+1} = \mathbf{b}^n, \tag{63}$$

Figure 1: Initial configuration of fiber in bold and final rest configuration in dotted line.

where I is the $2N_b \times 2N_b$ identity matrix. The matrix $I - \mathcal{M}_n \mathcal{A}_{h_b}$ is symmetric and non-singular. Indeed, the argument in [**?**] for $I - S_n^* P_h S_n \mathcal{A}_{h_b}$ can be applied to this case as well, for suppose

$$(I - \mathcal{M}_n \mathcal{A}_{h_b})\mathbf{X} = 0, \tag{64}$$

then multiplying both sides of (64) by $\mathcal{A}_{h_b}$ and taking the inner product with $\mathbf{X}$ we get

$$\begin{aligned} 0 = -(\mathbf{X}, \mathcal{A}_{h_b}\mathbf{X})_{I_B} + (\mathbf{X}, \mathcal{A}_{h_b}\mathcal{M}_n\mathcal{A}_{h_b}\mathbf{X})_{I_B} = \\ -(\mathbf{X}, \mathcal{A}_{h_b}\mathbf{X})_{I_B} + (\mathcal{A}_{h_b}\mathbf{X}, \mathcal{M}_n\mathcal{A}_{h_b}\mathbf{X})_{I_B}. \end{aligned} \tag{65}$$

Both terms in the left hand side of (65) are non-negative. Therefore, it follows that each has to be equal to zero and consequently

$$(\mathbf{X}, \mathcal{A}_{h_b}\mathbf{X})_{I_B} = 0. \tag{66}$$

This implies that $\mathcal{A}_{h_b}\mathbf{X} = 0$ [**?**] and hence from (64) we obtain that $\mathbf{X} = 0$. Thus (63) has a unique solution.

If the full matrix M_n is available (we will detail how to expedite its construction) it is easy to construct simple iterative schemes to solve (63). For example, denoting the diagonal component of M_n by M_n' and letting $M_n'' = M_n - M_n'$ we can write an iteration of weighted Jacobi type:

$$\mathbf{X}^{n+1,k+1} = (1-a)\mathbf{X}^{n+1,k} + a(I - M_n' A_{h_b})^{-1}(\mathbf{b}^n + M_n'' A_{h_b}\mathbf{X}^{n+1,k}). \tag{67}$$

When underrelaxed, $0 < a < 1$, this iteration converges well, typically requiring on the order of 10 iterations to obtain adequate residuals (on the order of the truncation error) and to maintain stability. Along the same lines, one can also consider a weighted Gauss-Seidel method of the form:

$$\begin{aligned} \mathbf{X}_i^{n+1,k+1} &= (1-a)\mathbf{X}^{n+1,k} \\ &\quad + a\left(\frac{1}{B_{ii}}\left(\mathbf{b}_i^n - \sum_{j<i} B_{ij}\mathbf{X}_j^{n+1,k+1} - \sum_{j>i} B_{ij}\mathbf{X}_j^{n+1,k}\right)\right) \end{aligned} \tag{68}$$

for $i = 1, 2, \ldots, 2N_B$, where $B = (I - \mathcal{M}_n\mathcal{A}_{h_b})$ and $0 < a < 1$. While (68) appears to converge slower than (67) in the numerical experiments it does, however, perform well as a smoother.

The most efficient and robust approach we found to solve (63) is a simple algebraic multigrid. Suppose for simplicity that $N_b = 2^m$ for some natural number m, we can then form a collection of Lagrangian grids of size $2^{m-1}, 2^m, \ldots, 2, 1$, our original grid being the first level and coarser grids coming afterward. For a given level l of the multigrid hierarchy the prolongation operator $\mathcal{P}_l$ takes a fiber $\mathbf{X}_{l+1}$ at level $l+1$ and adds a node between each pair of consecutive nodes equidistant between each. As a matrix, $\mathcal{P}_l$ has dimensions $2 \cdot 2^{l+1} \times 2 \cdot 2^l$ and has the form

$$\mathcal{P}_l = \begin{pmatrix} 1 & 0 & 0 & \\ .5 & .5 & 0 & \ldots \\ 0 & 1 & 0 & \\ 0 & .5 & .5 & \\ & \vdots & & \ddots \end{pmatrix}. \tag{69}$$

Restriction operators are taken to be the transposes of prolongation operators. Calling $\mathcal{K}_1 = (I - \mathcal{M}_n \mathcal{A}_{h_b})$ we define recursively $\mathcal{K}_l = \mathcal{P}_{l-1}^T \mathcal{K}_{l-1} \mathcal{P}_{l-1}$. The Gauss-Seidel iteration (68) provides a good smoother for the multigrid. All together, a single restriction, prolongation, and smoothing step look like

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

$$\mathcal{K}_l \mathbf{X}_l = \mathbf{b}_l \tag{70}$$

  calculate the residual and restrict it to the next lowest level

$$\tilde{\mathbf{r}}_{l+1} = \mathcal{P}_l^T \mathbf{b}_l - \mathcal{K}_l \mathbf{X}_l \tag{71}$$

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

$$\mathcal{K}_{l+1} \mathbf{X}_{l+1} = \tilde{\mathbf{r}}_{l+1} \tag{72}$$

- Correct our initial guess

$$\mathbf{X}_l \leftarrow \mathbf{X}_l + \mathcal{P}_l \mathbf{X}_{l+1} \tag{73}$$

- Smooth the high frequency errors in $\mathbf{X}_l$ via an underrelaxed Gauss-Seidel or (67). Both iterators perform comparably as smoothers.

The number of iterations needed depends more on the desired degree of accuracy than a requirement for stability. Typically a single iteration of a Full Multigrid Step with one V-cycle per level is sufficient to maintain stability. Computationally, it appears to be preferable to use the approximate solution $\mathbf{X}^{n+1}$ we obtain from (63) as the updated fiber position rather than applying (11) directly. This is because small errors in the solution to (63) are amplified when $\mathcal{A}_{h_b}$ acts on it.

We know turn to present the numerical results. A simple and popular scheme for the IB method is the FE/BE scheme. We use this scheme as a reference for the cost and efficiency of our proposed approaches. The methods discussed have five primary costs. First, when constructing the implicit system (**??**) we must calculate $\mathbf{b}^n$ via a fluid solve, a cost nearly equivalent to an FE/BE timestep cost. Second, if a matrix representation of M_n is sought, the cost of constructing the matrix can exceed that of the cost of a FE/BE step for N_b relatively large compared to N. Third and foremost is the cost of initializing the multigrid solver, calculating the coarse representations of M_n on the various levels of our grid. Fourth is the cost of our iterations to solve the linear system. This cost is dominated by the cost of inverting M_n or $(I - M_n A_{h_b})$ if A_{h_b} is linear. For the nonlinear system the additional cost of inverting A_{h_b} is negligible. The final cost is in computing $\mathbf{u}^{n+1}$ once $\mathbf{X}^{n+1}$ is known, involving another fluid solve. We will see that the sum of these costs results in a single timestep costing between 5 and 20 times that of an FE/BE timestep, depending on the size of N_b compared to N; however, the stability restraint on Δt for FE/BE will be orders of magnitude smaller than for our implicit methods, ultimately leaving us with methods substantially cheaper.

We detail first our simulations for the ellipse relaxation, with elasticity constant $\sigma = 10^5$ and fluid parameters $\rho = \mu = 1$. The initial configuration of the fiber is an ellipse centered at (.5,.5) with semi-major axis length .3 and semi-minor axis length .2, as in figure 1. As the simulation is run the ellipse attempts to stabilize to a circular configuration. The velocity of the fiber can reach roughly 500 units before slowing. Taking a standard length to be the geometric mean of our radii, .245, we calculate the Reynold's number of our fluid to be approximately 100.

For explicit simulations we will use an FE/BE scheme with Δt as large as allowable while maintaining stability given N, which we take to be approximately $\Delta t = .00025h$. For our implicit scheme we make use of a linear multigrid with stopping criteria of $\|\mathbf{r}\|_\infty < 30\Delta t$, where $\mathbf{r}$ is the residual

$\mathbf{b}^n - (I - M_n A_{h_b})\mathbf{X}^{n+1,k}$ at the kth iterate and $\|\cdot\|_\infty$ is the sup norm. Here Δt is taken to be close to the CFL condition, which is given approximately by $\Delta t = .02h$. The CFL restraint actually depends on $\|\mathbf{u}^n\|_\infty$ as well, but we approximate $\|\mathbf{u}^n\|_\infty$ at a conservative value when arriving at a formula for Δt.

The results of multiple simulations with varying timesteps for the explicit and implicit methods are given in table 2 and table 3. The rows in the table correspond to identical simulation runs with different N. We take $N_b = 2N$. The column marked average gives the average CPU time of a single timestep and following columns give the total CPU time up to the given simulation time T.

We see that the implicit scheme costs roughly 4 times as much per timestep, however, because Δt can be taken up to the CFL restraint the scheme is roughly 20 times faster than the explicit scheme in terms of total CPU time. This is a fairly modest gain. Increasing σ does not give our implicit scheme an edge because larger σ produces larger $\|\mathbf{u}^n\|_\infty$ and hence a more restrictive CFL restraint.

Methods have been proposed to remove the CFL restraint. Alternatively if our simulations use Stoke's flow then we can take arbitrarily sized timesteps with our implicit scheme while maintaining stability whereas the FE/BE method would remain mired in the same restrictive timesteps. We compare the two methods with Stoke's flow in tables 4 and 5.

We note that caution must be used when taking large timesteps with the implicit discretizations used here. The larger the timestep taken the greater is the artificial permeability introduced into the fiber.

7. Application; Heart Valve

We turn now to a more challenging application of the IB method. We consider a rigid valve immersed in blood flowing through an artery. The valve is indirectly restricted in motion by two hinges but is allowed to rotate. The valve, artery walls and hinges will all be modeled as immersed springs; all dynamics will be given gratis by the IB method once we provide the initial setup.

We take $\Omega = [0, 2] \times [0, 1]$ as our domain, again with periodic boundary conditions, which we discretize as a $2N \times N$ uniform gird. For a detail of the geometry of the problem see figure 2. Note that the top and bottom of the artery walls include cushions in the shape of two hills. We simulate

a horizontal flow from left to right through the artery by adding a forcing vector $\mathbf{f}_{j,k} = (v_{flow}, 0)$ to the heterogeneous force in (**??**), where in general v_{flow} may be time dependent. This changes our explicit term $\mathbf{b}^n$ in our implicit system to

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h(\rho \mathbf{u}^n - \Delta t \rho \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n + \Delta t \mathbf{f}). \tag{74}$$

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

$$\mathbf{f}_i = -k_i(\mathbf{X}_i - \bar{\mathbf{X}}_i) \tag{75}$$

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

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

given by

$$\mathbf{f}_i = \sum_{j=1}^{N_b} -k_{i,j} \frac{\mathbf{X}_i - \mathbf{X}_j}{|\mathbf{X}_i - \mathbf{X}_j|}(|\mathbf{X}_i - \mathbf{X}_j| - L_{i,j}). \tag{76}$$

Note that the force operator for our ellipse given by (61) is exactly the force given by (76) for a loop of fiber points connected by springs with suitable spring constants and zero resting length. For our valve though we require non-zero resting lengths in order to preserve the structure. Additionally, in order to maintain the rigidity of a solid body the spring constants must be taken very large, roughly $\mathcal{O}(10^9 h^{-2})$. Similarly for the tether points representing the artery walls the spring constants must be very large to portray the tautness of the biological fibers. Because the current acts to bend the valve as it is penned between its hinges larger values of v_{flow} require larger spring constants to maintain the structure. In Roma's original investigation of the problem the spring constants required to maintain rigidity led to restraints on the timestep on the order of 10^{-7} seconds, with the time scales of interest roughly on the order of 1 second. Removing this stiffness here however is not so direct as with the simpler ellipse relaxation problem. Our problem is still given by the equation

$$\mathbf{X}^{n+1} = M_n A_{h_b} \mathbf{X}^{n+1} + \mathbf{b}^n. \tag{77}$$

Figure 3: Top: A fine grid approximation to the valve, showing only linkage. Bottom: 'x's mark the prolongation of a coarse valve marked in 'o'.

Obviously though a linear multigrid can't directly accommodate our system. As noted we are making use of nonzero resting lengths for the springs comprising the valve and this leads to nonlinearities in our forcing function A_{h_b}. When attempting to solve (77) one can approximate A_{h_b} with various linear operators and hope to capture most of the pertinent dynamics, however our efforts in this direction proved fruitless. Newton iterations of (77) converge quite well, shifting the problem to a linear problem which might appear to resemble the linear system for the ellipse relaxation problem. If we let $J(\mathbf{X})$ represent the Jacobian of A_{h_b} at $\mathbf{X}$, then the Newton iteration is given by

$$(I - M_n J(\mathbf{X}^{n+1,k}))(\mathbf{X}^{n+1,k+1} - \mathbf{X}^{n+1,k}) = \mathbf{b}^n - (I - M_n A_{h_b})\mathbf{X}^{n+1,k}. \quad (78)$$

This does indeed resemble (**??**) with the linear J replacing A_{h_b}. Unfortunately multiplication against J destroys the positive-definiteness of M_n. To see this it suffices to show that J has both positive and negative eigenvalues. We consider a simple case with four immersed boundary points forming a square with side lengths $\sqrt{2}$ and with links connecting the perimeter of the square. We vary the resting length l of the connecting springs and compute the eigenvalues of the resulting forcing function's Jacobian. The results can be seen in figure 4. We see that as the resting length approaches and surpasses the side length of the square the Jacobian loses its seminegative-definiteness. This is the basic observation for most arbitrary structures. For our valve we take the resting length to be the starting length of the links comprising the valve, we thus immediately lose seminegative-definiteness once we perturb the valve structure.

Because of this the conjugate gradient method no longer converges. Biconjugate gradient method on the other hand fairs better, converging but not in an acceptable number of iterations. Biconjugate gradient may take upward of 100 iterations to converge satisfactorily. A major obstacle in accelerating convergence is that the natural preconditioner $I - M_n' J$ does not capture the dynamics of our system well. Indeed a Jacobi iteration analogous to (67) has very poor convergence, requiring strong underrelaxation and thousands of iterations. Without adequate smoothers we can't hope for an efficient linear multigrid to solve our Newton iteration. Smoothers for the nonlinear system (77) were likewise difficult to uncover, seemingly ruling out a nonlinear multigrid.

Note that for simpler geometries similar to those in the ellipse relaxation problem one may expect a direct generalization of the previous methods used

in the linear example to extend, even with nonlinear forcing. For instance different forcing functions for other geometries may yield Jacobians such that $M_n J$ is positive definite, in which case a linear multigrid may be used to solve the Newton iterations (78). Alternatively other nonlinear problems may possess suitable smoothers to allow for a nonlinear multigrid to directly solve (77). Nonetheless for the present problem we must change directions.

7.1. Solving the implicit system

We consider first the simplest possible iterator for (77), the fixed point iteration given by

$$\mathbf{X}^{n+1,k+1} = M_n A_{h_b} \mathbf{X}^{n+1,k} + \mathbf{b}^n. \tag{79}$$

The first thing we must observe about this iteration is that it fails spectacularly, in fact if our initial guess is $\mathbf{X}^{n+1,0} = \mathbf{X}^n$ then $\mathbf{X}^{n+1,1}$ is the explicit

Figure 4: Eigenvalues for the Jacobian of a fiber forcing function for four points linked as a square with varying resting length.

update provided by a FE/BE scheme which we know to be unstable for reasonable Δt. Intuitively, additional iterations of (79) can only exacerbate the instability: small errors in $\mathbf{X}^{n+1,k}$ are amplified enormously by A_{h_b} resulting in large errors in $\mathbf{X}^{n+1,k+1}$. This is precisely why taking our solution of (77) as our updated $\mathbf{X}^{n+1}$ rather than using (**??**) provides more stability, as using (**??**) is equivalent to pushing $\mathbf{X}^{n+1}$ through a fixed point iteration. Any errors present would be amplified.

The eigenvalues of J are at least on the order of the spring constants comprising our valve. It should seem natural to take advantage of this by reversing the fixed point iteration (79). Consider the alternative fixed point iteration

$$\begin{cases} M_n \mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n, \\ A_{h_b} \mathbf{X}^{n+1,k+1} = \mathbf{F}^{k+1}. \end{cases} \tag{80}$$

First we solve for $\mathbf{F}^{k+1}$. This is the force distribution that we would need to exist on the fiber at time t^n in order that after the current timestep the fiber would be updated through (**??**) to be located at our current guess $\mathbf{X}^{n+1,k}$. The linear system we must solve only involves the linear positive definite operator M_n, thus we can efficiently solve it using a multigrid. Second we determine what configuration $\mathbf{X}^{n+1,k+1}$ gives rise to this force through A_{h_b}. In many problems there is a unique $\mathbf{X}^{n+1,k+1}$ such that $A_{h_b}\mathbf{X}^{n+1,k+1} = \mathbf{F}^{k+1}$. For our current problem this is not the case.

To see this note that our valve is a neutrally buoyant object, detached from any fixed points, and the internal forces generated by perturbations in the valve's structure are unaffected by translation. Consider $\mathbf{e}_1$, a vector of length $2N_b$ equal to zero in every component except those corresponding to x-components of a node in our valve and similarly $\mathbf{e}_2$ for y-components. $\mathbf{e}_1$ and $\mathbf{e}_2$ are then horizontal and vertical translation vectors, $\mathbf{X} + \mathbf{e}_1$ representing the same configuration as $\mathbf{X}$ with the valve shifted right by one unit. Shifting has no influence on the force density,

$$A_{h_b}(\mathbf{X} + \mathbf{e}_1) = A_{h_b}(\mathbf{X} + \mathbf{e}_2) = A_{h_b}\mathbf{X}, \tag{81}$$

thus A_{h_b} is not injective and we may not have a unique solution to (80). Indeed, A_{h_b} is not surjective either so (80) may have no solution at all. This follows physically from conservation of momentum and angular momentum: A_{h_b} can't directly generate forces that move or rotate the valve. Translation and rotation can only be introduced via the fluid.

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

$$\mathbf{x}_c = \left(\frac{\langle \mathbf{X}^n, \mathbf{e}_1 \rangle}{\langle \mathbf{e}_1, \mathbf{e}_1 \rangle}, \frac{\langle \mathbf{X}^n, \mathbf{e}_2 \rangle}{\langle \mathbf{e}_2, \mathbf{e}_2 \rangle} \right). \tag{82}$$

We define also

$$\mathbf{e}_3(\mathbf{X}) = \frac{\partial}{\partial \theta} R_\theta(\mathbf{X}) = (-\tilde{Y}, \tilde{X}), \tag{83}$$

where $(\tilde{X}, \tilde{Y}) = \mathbf{X} - \mathbf{x}_c$ are the components of $\mathbf{X}$ in the coordinate system where $\mathbf{x}_c$ is the origin. $\mathbf{e}_3$ is the Jacobian of R_θ and is a column vector of length $2N_b$.

$\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ together represent three vectors outside the image of A_{h_b} associated with translation and rotation. We have then that there exist solutions to (80) only when $\mathbf{F}^{k+1}$ does not act to translate or rotate the valve. That is, a vector $\mathbf{F}$ lies in the image of A_{h_b} only when

$$\langle \mathbf{F}, \mathbf{e}_j(\mathbf{X}^{n+1,k}) \rangle = 0 \text{ for } j = 1, 2, 3. \tag{84}$$

To see this mathematically consider the simplest case. Suppose we have only two immersed fiber points

$$\mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix} \tag{85}$$

and there is a single link between them. Then the force they generate is

$$\mathbf{F} = T \begin{pmatrix} x_2 - x_1 \\ x_1 - x_2 \\ y_2 - y_1 \\ y_1 - y_2 \end{pmatrix} \tag{86}$$

for some tension scalar T. The rotation vector about the origin is simply $\mathbf{e}_3 = (-y_1, -y_2, x_1, x_2)^T$. We have then that

$$\langle \mathbf{F}, \mathbf{e}_3 \rangle = T(-y_1, -y_2, x_1, x_2) \begin{pmatrix} x_2 - x_1 \\ x_1 - x_2 \\ y_2 - y_1 \\ y_1 - y_2 \end{pmatrix} \tag{87}$$

$$= T\left(y_1(x_1 - x_2) + y_2(x_2 - x_1) + x_1(y_2 - y_1) + x_2(y_1 - y_2) \right) = 0 \tag{88}$$

A similar calculation shows that $\langle \mathbf{e}_1, \mathbf{F} \rangle = \langle \mathbf{e}_2, \mathbf{F} \rangle = 0$. For our more complicated valve we simply have a summation of such forces, thus our forcing function must satisfy (84) where $\mathbf{F} = A_{h_b}\mathbf{X}$ for any configuration $\mathbf{X}$. In general though most arbitrary vectors $\mathbf{F}$ won't satisfy (84) and we won't be able to find a solution to (80). To remedy this we must factor out those components sitting outside the image of A_{h_b}. This is easier to do with the linearized A_{h_b}, where we may simply project onto the vector space free of the problematic vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$. This suggests the modified Newton iteration to approximately solve for $\mathbf{X}^{n+1,k+1}$ in (80),

$$\begin{cases} J(\mathbf{X}^{n+1,k+1,l})(\mathbf{X}^{n+1,k+1,l+1} - \mathbf{X}^{n+1,k+1,l}) = P(\mathbf{F}^{k+1} - A_{h_b}\mathbf{X}^{n+1,k+1,l}), \\ \mathbf{X}^{n+1,k+1,0} = \mathbf{X}^{n+1,k} \end{cases} \tag{89}$$

where

$$P(\mathbf{F}) = \mathbf{F} - \sum_{i=1}^{3} \mathbf{e}_i(\mathbf{X}^{n+1,k})\langle \mathbf{F}, \mathbf{e}_i(\mathbf{X}^{n+1,k}) \rangle \tag{90}$$

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

Linear systems of the form $J\mathbf{X} = P\mathbf{F}$ are, strictly speaking, overdetermined by three degrees of freedom. To rectify this we simply isolate three degrees of freedom in $\mathbf{X}$ and fix them. These variables must be associated with the valve but are otherwise arbitrary. We could, for instance, fix the x and y component of a single node in the valve as well as the x component of an additional node. Once fixed we may proceed to solve for $\mathbf{X}$ such that $J\mathbf{X} = P\mathbf{F}$ is satisfied at all other free points. What is important, however, is that iterations of (89) in such a manner will converge to an $\mathbf{X}$ that satisfies $A_{h_b}\mathbf{X} = P\mathbf{F}$ at all points, even those we fixed. In this sense we arrive at an updated configuration $\mathbf{X}^{n+1,k+1}$ which satisfies (80) up to components outside the image of A_{h_b}.

Iterations of (80) typically provide square convergence. The limit need not be, and usually isn't, a solution to (77). The limit is, however, a stable update for our configuration $\mathbf{X}$. As an aside, it is important to note that this iteration converges precisely because the standard fixed point iteration

(79) diverges. As we modify the parameters of our simulation though, for instance if we were to drastically decrease the spring constants, then (79) may converge whereas (80) would diverge. There may, therefore, be instances with mixed spring constants, both large and small, where neither iteration converges. For our case with large spring constants everywhere there is no problem, (79) converges rapidly. Indeed increasing the spring constants aids in the convergence.

Of course this iteration, as it is currently formulated, does not allow for translation or rotation of our valve. Thus, while it may provide stable updates it does not lead to physically acceptable simulations. We must somehow reintroduce translation and rotation in our simulation. The simplest method to do this is to use the location and angle of the valve configuration obtained from an explicit update. To accomplish this we may simply take our initial guess $\mathbf{X}^{n+1,0}$ for iterations of (89) to be

$$\mathbf{X}^{n+1,0} = R_\theta \mathbf{X}^n + \mathbf{e}_1\langle \tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{e}_1\rangle + \mathbf{e}_2\langle \tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{e}_2\rangle \tag{91}$$

where

$$\theta = \langle \tilde{\mathbf{X}} - \mathbf{X}^n - \mathbf{e}_1\langle \tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{e}_1\rangle - \mathbf{e}_2\langle \tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{e}_2\rangle, \mathbf{e}_3(\mathbf{X}^n)\rangle \tag{92}$$

and

$$\tilde{\mathbf{X}} = \mathbf{X}^n + M_n A_{h_b} \mathbf{X}^n \tag{93}$$

is the FE/BE update prediction. Here we are taking the FE/BE prediction for the following timestep $\tilde{\mathbf{X}}$ and extracting its location and angle. We then take our current configuration $\mathbf{X}^n$ and modify the location and angle of the valve to match that in $\tilde{\mathbf{X}}$. Afterward we may proceed to apply iterations of (89) and since this doesn't introduce translation or rotation we will maintain the location and angle we extracted from the FE/BE prediction.

Other predictors may be employed as well. For instance we may rediscretize our problem on a coarser grid, solve for the updated timestep, prolong the solution to our original fine grid and extract its gross properties. In fact solving (80) via iterations of (89) is a good nonlinear smoother, opening the possibility for a nonlinear multigrid. Other predictors may involve higher order explicit update mechanisms for computing $\tilde{\mathbf{X}}$ or treating the valve as a true rigid body and predicting its motion through rigid body dynamics.

7.2. Solving the implicit system exactly

Many of these explicit predictions behave quite satisfactorily and in conjunction with (80) provide both stability and accurate solutions to (77). In

many applications this is sufficient. In fact if we perform iterations of (89) after using the initial guess (91) we obtain an approximate solution to (77) with error on the order of the accuracy of our discretization.

In our particular simulation, however, extra care must be taken. The close proximity of the valve to its hinges can quickly lead to collisions if small errors in translation propagate throughout the simulation. This proximity can also seriously affect the behavior of an explicit prediction, leading to wild translation and rotation. We thus seek a means to reintroduce the three degrees of freedom we have neglected in applying (80).

Suppose that we have a solution $\mathbf{X}^{n+1}$ to our system (77). We must have then

$$\langle \mathbf{F}^{n+1}, \mathbf{e}_j(\mathbf{X}^{n+1}) \rangle = 0 \text{ for } j = 1, 2, 3. \tag{94}$$

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

$$M_n \mathbf{F}^{n+1} = \mathbf{X}^{n+1} - \mathbf{b}^n. \tag{95}$$

This is because if $\mathbf{X}^{n+1}$ solves (77) then $\mathbf{F}^{n+1} = A_{h_b}\mathbf{X}^{n+1}$ and A_{h_b} can't generate forces with nonzero components along $\mathbf{e}_i$, $i = 1, 2, 3$. (94) suggests that before every iteration of (80) we simply shift and rotate $\mathbf{X}^{n+1,k}$ such that $\langle \mathbf{F}^{k+1}, \mathbf{e}_j(\mathbf{X}^{n+1,k}) \rangle = 0$ for $j = 1, 2, 3$, where $M_n \mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n$ as before. This can be approximately accomplished by finding (a_1, a_2) and θ, a translation vector and an angle, such that

$$\langle M_n^{-1}(\mathbf{X}^{n+1,k} + a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \theta\mathbf{e}_3(\mathbf{X}^{n+1,k}) - \mathbf{b}^n), \mathbf{e}_j(\mathbf{X}^{n+1,k}) \rangle = 0 \text{ for } j = 1, 2, 3. \tag{96}$$

Or, equivalently,

$$\begin{pmatrix} \langle \mathbf{F}_{\mathbf{e}_1}, \mathbf{e}_1 \rangle & \langle \mathbf{F}_{\mathbf{e}_2}, \mathbf{e}_1 \rangle & \langle \mathbf{F}_{\mathbf{e}_3}, \mathbf{e}_1 \rangle \\ \langle \mathbf{F}_{\mathbf{e}_1}, \mathbf{e}_2 \rangle & \langle \mathbf{F}_{\mathbf{e}_2}, \mathbf{e}_2 \rangle & \langle \mathbf{F}_{\mathbf{e}_3}, \mathbf{e}_2 \rangle \\ \langle \mathbf{F}_{\mathbf{e}_1}, \mathbf{e}_3 \rangle & \langle \mathbf{F}_{\mathbf{e}_2}, \mathbf{e}_3 \rangle & \langle \mathbf{F}_{\mathbf{e}_3}, \mathbf{e}_3 \rangle \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \theta \end{pmatrix} = - \begin{pmatrix} \langle \mathbf{F}^{k+1}, \mathbf{e}_1 \rangle \\ \langle \mathbf{F}^{k+1}, \mathbf{e}_2 \rangle \\ \langle \mathbf{F}^{k+1}, \mathbf{e}_3 \rangle \end{pmatrix}, \tag{97}$$

where $M_n \mathbf{F}_{\mathbf{e}_j} = \mathbf{e}_j$ for $j = 1, 2, 3$. We then construct an updated configuration

$$\mathbf{X}^{n+1,k} \leftarrow R_\theta \mathbf{X}^{n+1,k} + a_1\mathbf{e}_1 + a_2\mathbf{e}_2 \tag{98}$$

after which we proceed to calculate $\mathbf{X}^{n+1,k+1}$ via (89) using an unmodified $\mathbf{F}^{k+1}$. In this manner we obtain an iteration which converges rapidly to a solution of (77). This iteration converges nearly as quickly as (80) does for a fixed valve with no rotation or translation. There is an additional cost per

iteration though. The predominant costs come from solving linear systems involving M_n, which we must now do four times, calculating $\mathbf{F}^{k+1}$ as well as $\mathbf{F}_{\mathbf{e}_j}$ for $j = 1, 2, 3$.

Luckily these costs aren't great in practice. Obviously $\mathbf{F}_{\mathbf{e}_1}$ and $\mathbf{F}_{\mathbf{e}_2}$ only need to be calculated once per timestep. For our purposes though we can get away with calculating $\mathbf{F}_{\mathbf{e}_3}$ just once per timestep as well, even though $\mathbf{e}_3$ depends on the current guess $\mathbf{X}^{n+1,k}$ which changes throughout the calculation of a single timestep. Doing so does not degrade the final residual beyond the limits of accuracy provided by our multigrid solver. Additionally $\mathbf{F}_{\mathbf{e}_j}$ for $j = 1, 2, 3$ will not change greatly from timestep to timestep, precipitating their use as good initial guesses in the following timesteps. Together we will see that the actual costs of our iterators are not the predominant cost of each timestep but are rather dominated by the cost of initializing the linear system and multigrid itself.

There is an additional subtlety that we wish to point out. The prolongation and restriction operators we use in our multigrid for solving M_n^{-1} are geometrically inspired, relying on the underlying structure of our configuration. However this fails to give adequate weight to the fluid interactions *between* surfaces, in particular the interaction between the ends of the valve and the cushions as well as the interaction between the middle of the valve and the hinges.

In our case the multigrid still converges but not sufficiently quickly at key points. In particular with only a few iterations our multigrid may fail to capture all the necessary dynamics between the valve and hinges to prevent collision and protrusion. Certainly a more robust algebraic multigrid algorithm for selecting prolongation and restriction operators may solve this. Keeping with the mentality of a geometric multigrid however we simply employ a quick yet effective alteration to our smoother.

We maintain our standard Gauss-Seidel iteration, however, in addition we also perform a direct solve via Gaussian elimination of a subset of our problem. Because we are only concerned with the high accuracy needed to prevent collision between the valve and hinge we isolate those points close to the hinge. See figure 5. Holding all other points outside this region fixed we then proceed to solve the resulting reduced system. Afterward we perform the standard Gauss-Seidel iteration to smooth the interface between the points inside and outside the solved region. Because the number of points we are solving for is relatively small this procedure is quite cheap and doesn't contribute greatly to the cost of the multigrid iteration.

As an aside, note that even with exact solutions to (77) taking Δt large may still produce collisions due to inaccuracies in the discretization itself, rather than any shortcomings in our methods to solve the implicit systems. Nonetheless in such cases the exact solution to the implicit system (77) will still behave much more satisfactorily than with explicit updates for the valve location and angle.

8. Results for the application to the heart valve

For our heart valve simulation the initial condition is the rest configuration of all links and tethers, with the valve resting between the two hinges and points straight up. The current force is $v_{flow} = 100$ and all spring constants are taken to be $10^9 l^{-2}$, where $l = h/2$ is the length between nodes in our valve. We have much slower velocities here than in the ellipse relaxation, with speeds reaching roughly 10 units. Taking a characteristic length to be the length of the heart valve we obtain a Reynold's number of roughly 5. The slower speeds lead to a CFL restraint orders of magnitude larger and this allows our implicit scheme to show its power, even though now $N_b \approx 4N$, leading to an implicit timestep costing roughly 20 times that of an explicit one.

Figure 5: Configuration of our valve system after some period of time. The small box contains the subsystem of fiber points we solve exactly.

For the FE/BE simulation Δt is again taken to be the maximum allowable, with $\Delta t = 30h\sigma^{-2}$. For our implicit scheme we fix $\Delta t = .0025$, well below the CFL restraint. The results for the explicit simulations are given in table 6 with the results from the implicit simulations following in table 7.

Here the implicit scheme fairs much better than in the linear case, being free of a restrictive CFL condition. The total CPU time is on the order of 1000 times less. Δt may be taken even larger, up to the CFL restraint, leading to even greater savings in CPU time, but this will typically increase leakage through the immersed boundary and eventually allow the valve to collide and pass through the hinges.

8.1. Time dependent v_{flow}

The purpose of v_{flow} is to impart a force on the fluid which leads to the valve opening and closing. If v_{flow} is held constant we can only open the valve. To then close the valve we must introduce a time dependent v_{flow}. We define

$$v_{flow}(t) = \begin{cases} 60000t(.1-t) & t < .1 \\ -60000(t-.1)(.2-t) & t \geq .1 \end{cases} . \tag{99}$$

Solving the resulting numerical problem is typically more difficult with explicit methods; however, for our implicit methods no changes to the code are required except to directly account for the changing values of v_{flow}. We demonstrate the time evolution of the system in figure 6.

9. The matrix form of M_n

The methods described in this paper rely heavily on the fluid solver $\mathcal{L}_h$ as it occurs in the definition of M_n. There are two ways we can calculate quantities involving it. We can solve the single timestep Stokes problem (**??**) every time we need to know a quantity of the form $M_n\mathbf{F}$ or we can construct a matrix representation of M_n and employ matrix-vector multiplication. In this section we touch on the trade offs between these two different approaches, as well as detailing a process for calculating the matrix of M_n.

9.1. Discussion

We first contrast the functional differences between the two methods as it pertains to arriving at and coding iterators. There is no avoiding the work of implementing code to calculate $\mathcal{L}_h$. Even if we choose to construct the matrix of M_n we still require $\mathcal{L}_h$ in order to calculate $\mathbf{b}^n$ and $\mathbf{u}^{n+1}$ once we

Table 2: Ellipse relaxation with Forward Euler/Backward Euler scheme for convective flows. The total CPU time up to time T is given, with the average CPU time per timestep given.

N	N_b	Δt	Average	$T = .0001$	$T = .0002$	$T = .0005$	$T = .001$	$T = .005$
128	256	$1.95 \cdot 10^{-6}$	.0286	1.047	2.030	5.233	11.789	73.352
256	512	$9.76 \cdot 10^{-7}$	.143	14.577	29.076	72.908	146.079	730.210
384	768	$6.51 \cdot 10^{-7}$	.340	52.251	104.701	261.465	522.469	2613.190
512	1024	$4.87 \cdot 10^{-7}$	.634	128.374	256.994	643.436	1288.150	6491.947

Table 3: Ellipse relaxation with implicit scheme solved via linear multigrid for convective flows. The total CPU time up to time T is given, with the average CPU time per timestep given.

N	N_b	Δt	Average	$T = .0001$	$T = .0002$	$T = .0005$	$T = .001$	$T = .005$
128	256	$1.17 \cdot 10^{-4}$	.101	.125	.250	.609	1.015	4.341
256	512	$7.81 \cdot 10^{-5}$	.548	1.25	1.844	4.360	7.796	35.061
384	768	$5.21 \cdot 10^{-5}$	1.239	2.797	5.484	13.579	26.034	118.984
512	1024	$3.91 \cdot 10^{-5}$	2.337	7.640	15.094	32.813	62.845	299.095

Table 4: Ellipse relaxation with Forward Euler/Backward Euler scheme for Stokes flow. The total CPU time up to time T is given, with the average CPU time per timestep given.

N	N_b	Δt	Average	$T = .001$	$T = .002$	$T = .005$	$T = .01$	$T = .05$
128	256	$1.95 \cdot 10^{-6}$	.0267	12.544	27.484	74.820	130.583	683.955
256	512	$9.76 \cdot 10^{-7}$	.1447	147.186	294.760	736.971	1473.703	7410.498
384	768	$6.51 \cdot 10^{-7}$	.3458	530.083	1060.674	2656.393	5311.060	26555.300*
512	1024	$4.87 \cdot 10^{-7}$	.7078	1465.628	2899.736	7266.940	14533.881*	72669.405*

Table 5: Ellipse relaxation with implicit scheme solved via linear multigrid for Stokes flow. The total CPU time up to time T is given, with the average CPU time per timestep given.

N	N_b	Δt	Average	$T = .001$	$T = .002$	$T = .005$	$T = .01$	$T = .05$
128	256	.001	0.093	0.312	0.515	1.046	1.936	9.342
256	512	.001	0.531	1.328	2.437	5.624	10.906	53.078
384	768	.001	1.239	3.078	5.562	12.953	25.281	123.859
512	1024	.001	2.320	5.312	9.937	23.812	46.921	232.046

Table 6: Heart valve simulation with Forward Euler/Backward Euler scheme. The total CPU time up to time T is given, with the average CPU time per timestep given. * denotes an extrapolated value.

N	N_b	Δt	Average	$T = .01$	$T = .02$	$T = .05$	$T = .1$
128	520	$1.15 \cdot 10^{-7}$	.02434	2110.491	4212.461	10508.954	21017.909*
256	1053	$2.89 \cdot 10^{-8}$	.1091	37687.166*	75374.332*	188435.832*	376871.664*

Figure 6: Vorticity field of our valve system through time. $N = 512, N_b = 2082, \Delta t = .0025, \sigma = 2.62 \cdot 10^{14}$. Frames shown at total simulation time $T = .0625, .1125, .175, .2$

know $\mathbf{X}^{n+1}$. It may therefor be more convenient to avoid the extra work of calculating the matrix of M_n and simply use the preexisting code for calculating $\mathcal{L}_h$ directly. This convenience is most apparent if we are using a simple iteration such as (67) to solve our implicit system (**??**), requiring us only to calculate the diagonal components of the matrix M_n. However, we note immediately that many iteration schemes become unaccessible, simplest among them the Gauss-Seidel iteration.

More complicated methods such as full multigrid iterations, while feasible, are greatly simplified when we have a matrix representation of M_n, which allows us to employ a straightforward algebraic multigrid as opposed to a geometric one. One of the more subtle issues with using a geometric multigrid is how to coarsen the Eulerian grid. A uniform coarsening won't do in the case of our heart valve because as we coarsen the valve geometry there remain points close together which would require a fine Eulerian mesh to resolve, see figure 3. If a geometric multigrid were to be employed we would need some manner of adaptive mesh on our coarsened grids.

Additionally having the matrix form of M_n allows for direct solutions of linear systems of the form $M_n\mathbf{F} = \mathbf{X}$. Direct solution methods are normally prohibitively expensive but are relatively cheap on coarse grids or reduced subsystem, which is where we found use for direct methods in our heart valve simulation. Having access to exact solutions also allows for comparison of approximate solutions via other methods with an ideal standard.

Perhaps more important though is the issue of cost. Solving (**??**) in two dimensions via discrete Fourier transforms requires $\mathcal{O}(N^2 \log N)$ operations whereas matrix-vector multiplication involving M_n requires $\mathcal{O}(N_b^2)$ operations. Typically $N_b \sim N$, so matrix-vector multiplication with M_n is $\mathcal{O}(N^2)$. While this is a strictly lower order of cost, in practice which method is faster depends on the relative size of N_b and N.

To contrast the costs numerically we use the simplest case, our ellipse relaxation problem for convective flows. The cost of computing $\mathcal{L}_h$ is essentially that of performing a single FE/BE timestep, we will thus use FE/BE as a stand in for $\mathcal{L}_h$. The FE/BE method is roughly independent of N_b, thus we fix N_b at some small value and consider the cost of a FE/BE timestep as N varies. Similarly working with the matrix form of M_n is completely independent of N, we thus fix N at some small value and consider the cost of computing $M_n\mathbf{F}$ as N_b varies. The results of this experiment can be seen in tables **??** and **??**, with the cost of computing $M_n\mathbf{F}$ listed in the Matrix-Vector multiplication row.

In many problems $N_b = 2N$. In this case matrix-vector multiplication is more than 50 times quicker than FE/BE as N reaches 512. Because the iterators detailed in this paper can require many applications of M_n this speedup can be quite significant to overall performance of the simulation; however, this advantage can be lost as the ratio N_b/N grows. In the case of our heart valve we have $N_b \approx 4N$ and the speedup in using a matrix form drops to roughly 10 times at fine resolutions. Worse yet, in a fully 3D application with a 2D immersed membrane we would have $N_b \sim N^2$ leading to a matrix-vector multiplication costing $\mathcal{O}(N^4)$ compared to a likely $\mathcal{O}(N^3 \log N)$ cost for solving (**??**) directly. For our present problems though, and especially for the problem of ellipse relaxation, our simulations enjoyed a substantial overall speedup working with M_n in matrix form.

We have yet to mention the cost of computing the matrix of M_n. Indeed, if one is not careful this cost can swamp any advantages gleaned. We use a simple method to insure the cost remains nominal. The cost our simulations incur by calculating the matrix of M_n is listed in table **??** in the row dubbed Matrix construction. We see that while the cost of constructing M_n is much larger than multiplication involving M_n the cost is nonetheless smaller than the cost of a FE/BE timestep when $N_b = 2N$.

9.2. Calculating the matrix of $\mathcal{M}_n$

Before proceeding to demonstrate how we calculate the matrix of $\mathcal{M}_n$ we first require the definition of a new operator. Consider the continuous problem

$$\begin{cases} \rho\tilde{\mathbf{u}}_t = -\nabla\tilde{p} + \mu\nabla^2\tilde{\mathbf{u}} + \mathbf{g} \\ \nabla \cdot \tilde{\mathbf{u}} = 0 \\ \tilde{\mathbf{u}}(\mathbf{x}, 0) = 0 \end{cases} \tag{100}$$

with periodic boundary conditions on Ω and where $\mathbf{g}$ is a vector field on Ω. We define the operator $\mathcal{L}_{\Delta t}$ given by

$$\mathcal{L}_{\Delta t}(\mathbf{g}) = \tilde{\mathbf{u}}(\Delta t). \tag{101}$$

Here $\tilde{\mathbf{u}}(\Delta t)$ refers to the velocity field at time Δt. This operator is intended to correspond to the discrete operator $\mathcal{L}_{h_b}$ with $\mathcal{L}_{h_b}(\Delta t \mathbf{g}_{i,j}) \approx \mathcal{L}_{\Delta t}(\mathbf{g})$. We will provide an estimate of the error by this approximation later.

We proceed now to demonstrate a method for calculating $\mathcal{M}_n$ efficiently. Recall that we are writing the configuration $\mathbf{X}$ of the discretized immersed

structure as the $2Nb$-array

$$\mathbf{X} = \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} X_1 \\ \vdots \\ X_{N_B} \\ Y_1 \\ \vdots \\ Y_{N_B} \end{bmatrix}, \tag{102}$$

and similarly we write $\mathbf{F} = \mathcal{A}_{h_B}\mathbf{X} = (F, G)^T$. We seek then four $N_b \times N_b$ matrices A,B,C,D such that

$$\mathcal{M}_n\mathbf{F} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix}. \tag{103}$$

We identify $\mathcal{M}_n$ with this $2N_b \times 2N_b$ block matrix. To calculate the entries of the sub-matrices composing $\mathcal{M}_n$ we consider horizontal and vertical, unit point forces $\mathbf{e}_1^j$ and $\mathbf{e}_2^j$ for $1 \leq j \leq N_B$, corresponding to each Lagrangian node. These are $2N_B$-arrays given by

$$(\mathbf{e}_1^j)_i = \begin{cases} 0 & \text{If } i \neq j, \\ 1 & \text{If } i = j \end{cases} \qquad 1 \leq i \leq 2N_B \tag{104}$$

and

$$(\mathbf{e}_2^j)_i = \begin{cases} 0 & \text{If } i \neq j + N_B, \\ 1 & \text{If } i = j + N_B \end{cases} \qquad 1 \leq i \leq 2N_B. \tag{105}$$

Since we have for $1 \leq i \leq 2N_B$

$$(\mathcal{M}_n)_{ij} = (\mathcal{M}_n\mathbf{e}_1^j)_i \quad \text{for } 1 \leq j \leq N_B, \tag{106}$$

$$(\mathcal{M}_n)_{ij} = (\mathcal{M}_n\mathbf{e}_2^j)_i \quad \text{for } N_B + 1 \leq j \leq 2N_B, \tag{107}$$

the entries $(\mathcal{M}_n)_j$ of the matrix can be viewed as the components of the velocity produced at a given interfacial node $\mathbf{X}_i$ by a unit horizontal or vertical force located at another boundary node $\mathbf{X}_j$. For the continuum problem this velocity depends only on the difference $\mathbf{X}_j - \mathbf{X}_i$ (the argument of the corresponding Green function). We would like to use this observation

to expedite the calculation of the matrix. To this end we approximate $(\mathcal{M}_n)_{i,j}$ as follows. For $1 \leq j \leq N_B$,

$$(\mathcal{M}_n \mathbf{e}_1^j)_i = \frac{1}{\rho}\Delta t(\mathcal{S}_n^* \mathcal{L}_h(\Delta t \mathcal{S}_n \mathbf{e}_1^j))_i \tag{108}$$

$$\approx \frac{1}{\rho}\Delta t \mathcal{S}^*(\mathcal{L}_{\Delta t}(\delta(\cdot - \mathbf{X}_j), 0))(s_i) \tag{109}$$

$$= \Delta t \int_\Omega (\mathcal{L}_{\Delta t}(\delta(\cdot - \mathbf{X}_k), 0))\, \delta(\mathbf{x} - \mathbf{X}_i) d\mathbf{x} \tag{110}$$

$$= \Delta t \int_\Omega (\mathcal{L}_{\Delta t}(\delta(\cdot\,), 0))\, \delta(\mathbf{x} - (\mathbf{X}_i - \mathbf{X}_k)) d\mathbf{x} \tag{111}$$

$$\approx \Delta t \sum_{j,k \in \Omega} (\mathcal{L}_h(\delta_h(\mathbf{x}_{a,b}), 0))\, \delta_h(\mathbf{x}_{j,k} - (\mathbf{X}_i - \mathbf{X}_k)) \tag{112}$$

where $(\delta_h(\mathbf{x}_{a,b}), 0)_{a,b} = (\delta_h(\mathbf{x}_{a,b}), 0)$ is a vector field on Ω_h. The strategy here is to approximate M_n with its continuous counterpart where we may make use of a judicious change of variables before rediscretizing. The first line expands the definition of M_n. The second line is the move to the continuous problem, where the discrete vector field $\mathcal{S}_n(\mathbf{F}^k)$ is replaced by the corresponding distribution $(\delta(\cdot - \mathbf{X}_k), 0)$. The third line is an expansion of the definition of $\mathcal{S}^*$. The fourth line is the result of a translational change of variable and the final line is simply a move back to discrete space. Here we can no longer write the discrete expression using the operator $\mathcal{S}_n^*$ because we are interpolating the velocity field at the origin rather than at any particular fiber point.

(112) shows that $(M_n \mathbf{F})_i$ depends approximately on the difference $\mathbf{X}_i^n - \mathbf{X}_k^n$. This allows us to define a single function

$$T(x, y) = \Delta t^2 \sum_{j,k \in \Omega} (\mathcal{L}_h \delta_h(\mathbf{x}_{a,b}))\, \delta_h(\mathbf{x}_{j,k} - (x, y)) \tag{113}$$

with respect to which we can compute the entries of the matrices A and C. We have

$$(M_n \mathbf{F}^k)_i \approx T(X_i - X_k, Y_i - Y_k). \tag{114}$$

Thus the entries of our matrices are given by

$$(A_{i,j}, C_{i,j}) \approx T(X_i - X_j, Y_i - Y_j). \tag{115}$$

For B and D we may construct a similar function. We start with a force distribution $\mathbf{G}^k$ analogous to $\mathbf{F}^k$ defined as

$$\mathbf{G}_j^k = \begin{cases} (0,1) & \text{if } j = k \\ (0,0) & \text{otherwise} \end{cases} \tag{116}$$

and proceed as before to construct a function U such that

$$(M_n \mathbf{G}^k)_i \approx U(X_i - X_k, Y_i - Y_k) \tag{117}$$

and hence

$$(B_{i,j}, D_{i,j}) \approx U(X_i - X_j, Y_i - Y_j). \tag{118}$$

Note though that on a square domain Ω we have via symmetry that $U(x,y) = T(y,x)$. For rectangular or otherwise irregular grids this symmetry does not hold, we do however have other symmetries on rectangular domains,

$$T(x,y) = T(-x,y), \quad T(x,y) = T(x,-y), \quad T(x,y) = T(-x,-y), \tag{119}$$

$$U(x,y) = U(-x,y), \quad U(x,y) = U(x,-y), \quad U(x,y) = U(-x,-y). \tag{120}$$

$$\tag{121}$$

This implies that equal force densities at two fiber points produce equal effects on each other, hence M_n is a symmetric matrix.

Suppose we were to use T to construct our entire matrix M_n. We would require $2N_b$ calls to T to construct M_n entirely which is prohibitively expensive if a fluid solve is invoked with every evaluation. The simplest method to counter this is to create a lookup table housing the values of T. It would be natural to simply precalculate all values of T and U on the Eulerian grid, letting $T_{i,j} = T(\mathbf{x}_{i,j}), U_{i,j} = U(\mathbf{x}_{i,j})$. However, fiber points may lie between grid points, thus we may lose some important dynamics by only defining T and U on $\mathbf{x}_{i,j}$. To counter this we may either create a lookup table on a finer grid or we can use linear interpolation between grid points. The latter is the strategy we used for our simulations. We approximate

$$T(x,y) \approx a_1 a_2 T_{i,j} + a_1(1-a_2)T_{i,j+1} + (1-a_1)a_2 T_{i+1,j} + (1-a_1)(1-a_2)T_{i+1,j+1}, \tag{122}$$

$$a_1 = \frac{x - x_{i,j}}{x_{i+1,j} - x_{i,j}}, \tag{123}$$

$$a_2 = \frac{y - y_{i,j}}{y_{i,j+1} - y_{i,j}}, \tag{124}$$

where i, j are the largest integers such that $x > x_{i,j}$ and $y > y_{i,j}$.

Calculating M_n with a lookup table costs $\mathcal{O}(N_b^2)$. When N_b is particularly large, however, this cost can overshadow the cost of fluid solves. While we haven't taken any additional means of reducing the cost of computing M_n in our simulations we note briefly some ways one may. Calculating M_n with direct lookup rather than using interpolation is substantially faster but with undesirably low accuracy. A simple compromise is to use linear interpolation when calculating $T(x, y)$ if $(x^2 + y^2)^{1/2}$ is small and direct lookup otherwise. This is cheap, with only $\mathcal{O}(N_b)$ interpolations needed, because most fiber points are distant from each other, as well as accurate, because T decays rapidly away from the origin. Alternatively we could employ an adaptive mesh on which to calculate T, taking a dense grid around the origin where most of the structure of T lies. This would allow us to use direct lookup for all entries of M_n without a loss in accuracy.

A further reduction in cost can be had by only computing part of M_n. Consider the case where A_{h_b} is linear. If we modify our definition of $\mathbf{b}^n$ to be

$$\mathbf{b}^n = \Delta t \mathcal{S}_n^* \mathcal{L}_h (\Delta t \mathcal{S}_n A_{h_b} \mathbf{X}^n + \rho \mathbf{u}^n - \Delta t \rho \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n) \tag{125}$$

then we can find $\mathbf{X}^{n+1}$ by solving for the change

$$\mathbf{X}^{n+1} - \mathbf{X}^n = M_n A_{h_b} (\mathbf{X}^{n+1} - \mathbf{X}^n) + \mathbf{b}^n. \tag{126}$$

Here, however, the cumulative influence of effects between distant fiber points is quite small since we have already factored in their aggregate effect through $\mathbf{b}^n$, thus we may choose to ignore them, using only a banded component of M_n to capture the pertinent influences. This can lead to speed up in calculating M_n as well as matrix-vector multiplication involving M_n if sparse data structures are used. This trick may prove particularly useful for problems with very large Ω, where large distances between immersed fibers drastically diminish their influence on each other's stability. For instance, in our heart valve problem much of the horizontal boundary modeled with tethered nodes has little impact on the stability of the fiber comprising the valve. The corresponding entries in M_n relating these two parts of our immersed fibers could be held at zero if we utilized (125) and (126).

9.3. Error introduced

Our use of the lookup tables for T and U introduces error into our simulations in two ways. First, the linear interpolation used between values in

the table, second, in the error introduced via (112). Linear interpolation introduces error of $\mathcal{O}(h^2)$, assuming our lookup table sits on the same Eulerian grid as Ω_h. The second error is more difficult to account for and we make no aims of quantifying it analytically. We instead rely on numerical experiments.

We work within the context of the ellipse relaxation problem with $N_b = 2N$. Assume that A is the exact matrix representation of M_n and B is the approximate matrix representation of M_n arrived at via the methods given above. We calculate both matrices and then compute the maximum norm between their difference. Two log-log graphs of the norm with respect to N are given in figures 7 and 8. In figure 7 we have fixed $\Delta t = h$ and we see that the error is $\mathcal{O}(h^2)$. In figure 8 we have fixed $\Delta t = .01h$ and our error worsens to $\mathcal{O}(h^{1.5})$.

It's clear that a shift is occurring where at first we have $\Delta t^p \gg h^q$ when $\Delta t = h$ and changing to $\Delta t^p \ll h^q$ when $\Delta t = .01h$. This suggests that $p = 1$. We are thus maintaining first order accuracy in time, as would be expected, however we lose spatial accuracy due to the transitions between discrete and distributional delta functions, leading to $q = .5$. Nonetheless the overall error, even after linear interpolation, is still at worst $\mathcal{O}(h^{1.5})$ which is smaller than the $\mathcal{O}(h)$ error of the overarching method. Thus we can be confident that our approximations to M_n are not introducing any unwarranted error.

10. Conclusion

Implicit methods for alleviating the stiffness of the IB method have been around for some time. Their implementation involves solving systems of equations typically thought to be prohibitively expensive. Our intent with this paper was not to detail a single silver bullet to solve the IB problem but rather to illuminate paths to solving these systems. We have provided avenues to this end for two particular applications of the IB method but there will likely remain challenges when applying the techniques detailed here to the incredibly varied body of applications for which the IB method is implemented.

A key benefit of our methods is that they have no dependence on the specifics of the fluid solver $\mathcal{L}_h$. If we use an adaptive mesh to achieve fluid solves in $\mathcal{O}(N_b)$ time then this will carry over to our implicit solvers, allowing us to take arbitrary time steps with the minimal order cost possible. This

is a particularly exciting prospect for 3D cases and we hope this paper will stimulate further investigations.

Figure 7: Log-log plot of $\|A - B\|_\infty$ with $\Delta t = h$

Figure 8: Log-log plot of $\|A - B\|_\infty$ with $\Delta t = .01h$