Journal of Computational Physics **176,** 231–275 (2002)
doi:10.1006/jcph.2001.6970, available online at http://www.idealibrary.com on IDEAL®

A Cartesian Grid Method for Solving the Two-Dimensional Streamfunction-Vorticity Equations in Irregular Regions

Donna Calhoun[1]

Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, New York 10012-1185
E-mail: calhoun@cims.nyu.edu

Received January 18, 2001; revised October 10, 2001

We describe a method for solving the two-dimensional Navier–Stokes equations in irregular physical domains. Our method is based on an underlying uniform Cartesian grid and second-order finite-difference/finite-volume discretizations of the streamfunction-vorticity equations. Geometry representing stationary solid obstacles in the flow domain is embedded in the Cartesian grid and special discretizations near the embedded boundary ensure the accuracy of the solution in the cut cells. Along the embedded boundary, we determine a distribution of vorticity sources needed to impose the no-slip flow conditions. This distribution appears as a right-hand-side term in the discretized fluid equations, and so we can use fast solvers to solve the linear systems that arise. To handle the advective terms, we use the high-resolution algorithms in CLAWPACK. We show that our Stokes solver is second-order accurate for steady state solutions and that our full Navier–Stokes solver is between first- and second-order accurate and reproduces results from well-studied benchmark problems in viscous fluid flow. Finally, we demonstrate the robustness of our code on flow in a complex domain.

Key Words: incompressible flow; streamfunction-vorticity finite-volume; finite-difference; Cartesian grid; embedded boundary; computational fluid dynamics.

1. INTRODUCTION

In this paper, we describe a method for solving the two-dimensional incompressible Navier–Stokes equations in irregular physical domains. Our method is based on an underlying uniform Cartesian grid and second-order finite-difference/finite-volume discretizations

[1] The author was generously supported by the Applied Mathematical Sciences Program of the U.S. Department of Energy under Contract DE-FGO288ER25053 and DE-FG03-96ER25292, NASA Grant NAG5-7583, and NSF Grants DMS-9505021 and DMS-96226645.

0021-9991/02 $35.00

of the streamfunction-vorticity equations. Geometry representing stationary solid obstacles in the flow domain is embedded in the Cartesian grid and special discretizations near the embedded boundary ensure the accuracy of the solution in the cut cells.

The use of Cartesian meshes for solving problems with complex geometry has become quite widespread in the past decade. A key reason for this is that the cost associated with generating the Cartesian mesh is negligible compared with that required by body-fitted or unstructured finite-element meshes. This computational savings becomes especially pronounced when solving problems with moving geometry. In these problems body-fitted or finite-element meshes must be regenerated at each time step, whereas the Cartesian grid remains fixed relative to the moving geometry. Another reason is that one can often make use of fast solvers when discretizations are based on the standard stencils on an underlying Cartesian mesh.

Despite the popularity of Cartesian grid methods, however, relatively few such methods exist for viscous incompressible flow problems in irregular geometry. One approach that is commonly used in the bio-fluids' community is Peskin's immersed boundary method [39]. In this method, singular forces distributed along an elastic membrane immersed in a fluid are represented discretely with model delta functions. These delta functions spread the force exerted by the elastic membrane to neighboring grid nodes of the Cartesian mesh. Because these forces only appear as source terms in the fluid equations, fast solvers can be used for the numerical solution of these equations. This method has been used in a variety of problems, including modeling the flow of blood in the heart [38–43], aquatic locomotion [17], blood clotting [18–20], and wave motion in the cochlea [5, 6]. The method in its original form is limited to first order, although Cortez and Minion [11] have developed a higher order immersed boundary method known as the blob projection method.

Recently, Ye and coworkers describe a Cartesian grid algorithm based on the projection method [53]. Their algorithm uses a finite-volume discretization of the momentum equations and a compact interpolation scheme near the embedded boundary to achieve second-order accuracy. Their method is formally second-order accurate, even at the embedded boundaries, but they must resort to iterative methods with appropriate preconditioners to solve the resulting large, sparse nonsymmetric linear systems that arise from their treatment of the boundary.

Other Cartesian grid methods for incompressible flow include the projection method for the Euler equations described by Almgren and coauthors [1], volume-of-fluid methods for multiphase flows described by Puckett and coworkers [44], and work by Unverdi and Tryggvason [49]. The work by Unverdi and Tryggvason is an example of the immersed boundary method applied to multiphase flows and has led to some very impressive simulations of complex multiphase flow. Johansen and Colella have developed fully conservative, finite-volume methods for solving elliptic and parabolic equations on Cartesian grids with embedded boundaries [22]. In [23], Johansen extends these ideas to problems involving viscous fluid flow.

Our algorithm is designed in the spirit of the immersed boundary method. Along the embedded boundary, we determine a distribution of sources needed to impose the no-normal and no-tangential flow conditions. This distribution appears as a right-hand-side term in the discretized fluid equations and so we can use fast solvers to solve the linear systems that arise. Unlike the immersed boundary method, however, we do not attempt to model discrete delta functions directly. Instead, we incorporate jump conditions into the discretization of the equations, thereby obtaining boundary terms at grid cells near the

embedded boundary. These boundary terms are derived in a way that guarantees formally a second-order convergence rate.

Our approach is also related to an integral equation approach in that we determine the correct distribution of sources needed to impose boundary conditions by solving a small, dense linear system at each time step. Then in a manner described by Mayo [32, 33], we spread the singular sources to the grid and solve the elliptic and parabolic equations using fast Poisson solvers. In this sense, our approach is also related to the capacitance matrix method, described by Proskurowski and Widlund [45] and Buzbee and coworkers [8].

To handle the advective terms, we use the high-resolution algorithms in CLAWPACK [25], a software package based on the wave propagation algorithms of LeVeque [26]. Irregular boundary cells are handled using capacity form differencing, and the small cell problem is handled by modifying the capacity function in a manner described in [10].

The structure of the paper is as follows. In Section 1.1, we discuss the streamfunction-vorticity formulation of the Navier–Stokes equations and our fractional step approach to solving them. In Section 2, we introduce some basic notation that we use throughout the paper. In Section 2.1, we describe our finite-difference Stokes solver, and then in Section 2.2, we discuss our finite-volume scheme for handling the advective terms. Finally in Section 3, we look at several test problems which establish the accuracy of our method, as well as show that we reproduce results from some well-studied benchmark problems in viscous fluid flow. Finally, we demonstrate that our algorithm can be used to solve problems in a complex domain.

1.1. The Streamfunction-Vorticity Formulation

In this paper, we solve the incompressible Navier–Stokes equations in streamfunction-vorticity formulation. In a two-dimensional, M_{bodies}-ply connected domain with inclusions Ω_j, $j = 1, \ldots M_{bodies}$, the streamfunction-vorticity equations can be written as

$$\begin{aligned} \omega_t + (\vec{u} \cdot \nabla)\omega = \nu \nabla^2 \omega, \quad & \nabla^2 \psi = -\omega, \\ & u = \psi_y, \\ & v = -\psi_x \end{aligned} \tag{1}$$

subject to

$$\left.\begin{aligned} \psi &= \bar{\psi}_j \\ \psi_n &= 0 \\ \textstyle\int_{\partial\Omega_j} \omega_n \, ds &= 0 \end{aligned}\right\} \quad \text{on } \partial\Omega_j,\ j = 1, \ldots M_{bodies}, \tag{2}$$

where $\vec{u}(x, y, t) \equiv (u(x, y, t), v(x, y, t))$ is the velocity vector whose components are horizontal and vertical fluid velocities, respectively, $\psi(x, y, t)$ is the streamfunction, and $\omega(x, y, t) = v_x - u_y$ is the scalar vorticity. The kinematic viscosity ν is assumed to be constant. On the solid boundaries, the streamfunction ψ must satisfy two boundary conditions: one of Dirichlet type corresponding to a no-normal flow condition and one of Neumann type corresponding to a no-tangential flow condition. No boundary conditions are explicitly given for ω.

In multiply connected domains, we generally do not know the constant value $\bar{\psi}_j$ that the streamfunction takes on the boundary of inclusion Ω_j. In our formulation, we use an

integral condition on the flux of vorticity at the boundaries to close the system with respect to these unknowns. This condition ensures that the pressure remains single valued. To see this, we consider an equation for the tangential momentum given in terms of normal and tangential velocity components (u_τ, u_n) as

$$(u_\tau)_t + \left(u_\tau \frac{\partial}{\partial \tau} + u_n \frac{\partial}{\partial n}\right) u_\tau = -\frac{\partial p}{\partial \tau} + \nu \frac{\partial \omega}{\partial n}, \tag{3}$$

where we have used the divergence condition $u_x + v_y = 0$ and the definition of the vorticity $\omega = v_x - u_y$ to get the term $\nu \partial \omega / \partial n$. At the boundary of solid, stationary inclusion Ω_j, the velocity $u_\tau = u_n = 0$, and so the above equation reduces to

$$\frac{\partial p}{\partial \tau} = \nu \frac{\partial \omega}{\partial n}. \tag{4}$$

Integrating this along the boundary $\partial \Omega_j$, we obtain the expression

$$p(s) = p_0 + \nu \int_{s_0}^{s} \frac{\partial \omega}{\partial n}\, ds \tag{5}$$

for the boundary pressure $p(s)$, where the constant $p_0 \equiv p(s_0)$ is some reference pressure on the boundary. Since the pressure is a single-valued function, our integral expression in (1) follows immediately.

In Fig. 1, we show a sketch of a typical region in which we solve the equations given in (1).

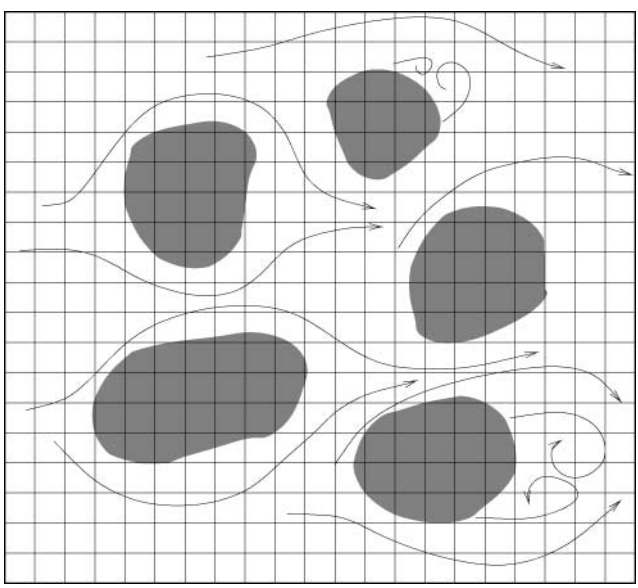

FIG. 1. Sketch of typical domain in which we solve the streamfunction-vorticity equations. The shaded regions represent stationary, rigid bodies, and the unshaded regions are the fluid domain. At the boundaries between solid and liquid regions, we must impose no-slip conditions. The irregular domain is embedded in a uniform Cartesian mesh.

1.2. Temporal Discretization

To discretize the equations given in (1) in time, we use a fractional step approach and split the advective terms from the elliptic and parabolic terms. In what follows, we use Δt for the time step and the superscript n to indicate a solution at time level $t_n \equiv n\Delta t$. Subscript n is reserved for differentiation with respect to a normal direction. Here, we discretize in time only and leave the details of the spatial discretizations for the following sections.

The basic method we describe is a hybrid finite-volume/finite-difference method. At the beginning of each time step, we assume that we have solutions ω^n, u^n, and v^n. The first step in our fractional step scheme is the advective step, given by

$$\frac{\omega^* - \omega^n}{\Delta t} = -(\vec{u}^n \cdot \nabla)\omega^n, \tag{6}$$

which we solve to obtain an intermediate solution ω^*. In this step, we track cell averages of vorticity and use the high-resolution wave propagation algorithms in the CLAWPACK [25] to update the vorticity field.

We follow this step with a Stokes flow step given by

$$\left.\begin{aligned} \frac{\omega^{n+1} - \omega^*}{\Delta t} &= \nu\nabla^2\omega^{n+1}, \\ \nabla^2\psi^{n+1} &= -\omega^{n+1}, \\ \psi^{n+1} &= \bar{\psi}_j^{n+1} \\ \psi_n^{n+1} &= 0 \\ \int_{\partial\Omega_j} \omega_n^{n+1}\, ds &= 0 \end{aligned}\right\} \quad \text{on } \partial\Omega_j,\ j = 1, \ldots M_{bodies}. \tag{7}$$

Here, we treat the vorticity and streamfunction as cell-centered pointwise values and discretize the equations using the immersed interface method, a finite-difference method for solving elliptic and parabolic equations in embedded regions. Since we are mainly interested in the spatial accuracy of our scheme, we illustrate our method for the backward Euler discretization of the diffusion equation, but our method can be easily adapted to other discretizations.

Finally, velocities at the new time level are computed from ψ^{n+1}:

$$u^{n+1} = \psi_y^{n+1}, \quad v^{n+1} = -\psi_x^{n+1}, \tag{8}$$

where the velocities are cell edge-averaged quantities.

2. THE NUMERICAL METHOD

The region in which we model the fluid flow is called the *flow domain* and the region occupied by the solid objects is the *no-flow* domain. The rectangular region consisting of both the flow domain and the no-flow domain is the computational domain and is denoted by $\mathbf{R}$. We establish a uniform Cartesian mesh on $\mathbf{R}$ with grid spacing Δx and Δy. For convenience, we assume that $\Delta x = \Delta y \equiv h$. Cell centers are labeled as points (x_i, y_j), $i = 1, \ldots N_x$,

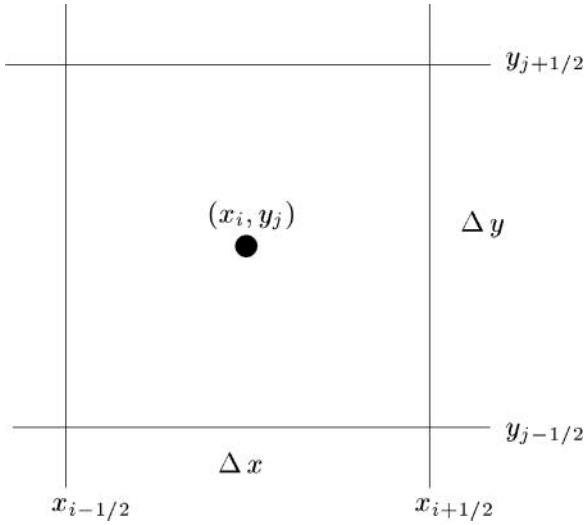

FIG. 2. Labeling scheme used for the Cartesian grid.

$j = 1, \ldots N_y$ where $x_i = (i - 1/2)\Delta x$ and $y_j = (j - 1/2)\Delta y$. Grid line intersections (or grid nodes) are labeled using fractional indices. For example, the grid node in the lower left corner of the cell centered at (x_i, y_j) is labeled $(x_{i-1/2}, y_{j-1/2})$. Figure 2 illustrates the labeling scheme.

For notational convenience, we describe our algorithm for a single embedded object Ω. The boundary of Ω is given by $\partial\Omega \equiv \Gamma$ and is parameterized using functions $(X(s), Y(s))$, where $s \in [0, S]$ and $X(0) = X(S)$ and $Y(0) = Y(S)$. The value ψ of on Γ is given by $\bar{\psi}$ and is in general unknown.

When we make reference to jumps in a solution across the boundary Γ, we assume that the flow domain is the positive region, denoted here as $\Omega+$, and the no-flow domain is the negative region, denoted using $\Omega-$. Using this convention, a jump in a quantity q across an interface is defined as $[\![q]\!] \equiv q^+(\alpha) - q^-(\alpha)$, where $q^+(\alpha)$ is the limiting value of q as we approach a boundary point α from the flow domain, $q^-(\alpha)$ is the limiting value as we approach the boundary from the no-flow domain, and α is a point on the interface.

Using the above notation, we illustrate our discretization scheme in Fig. 3.

2.1. Solving the Stokes Equations

In this section, we describe how we use the immersed interface method, introduced by LeVeque and Li [28], to discretize the Stokes equations given in (7). We save the discussion of the advective step for Section 2.2.

The coupled system of equations given in (7) can be rewritten as

$$\begin{aligned} \nabla^2\omega + \lambda^*\omega &= \lambda^*\omega^n, \\ \nabla^2\psi &= -\omega, \end{aligned} \tag{9}$$

where $\lambda^* = -1/\nu\Delta t$ and the boundary conditions on ψ are those given in (7). From here on, we drop the superscript $n + 1$ on quantities at time level t_{n+1}.

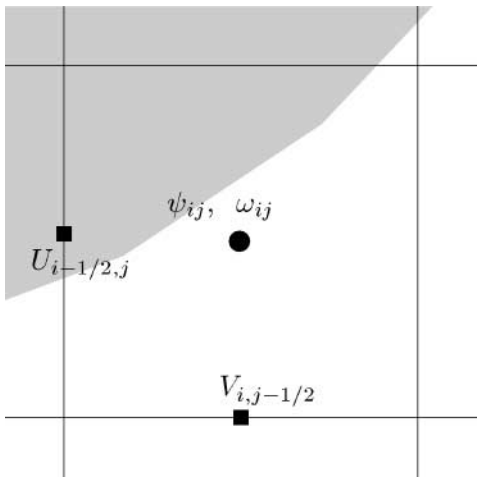

FIG. 3. Location of vorticity, streamfunction, and velocity values. Values are all cell centered, even in irregular cells. Velocities are edge-averaged values. The shaded region is the no-flow domain.

To handle both the parabolic and elliptic equations above, we describe our method in terms of the general elliptic equation given by

$$\nabla^2\phi + \lambda\phi = f, \tag{10}$$

where $\lambda \leq 0$.

A key feature of our algorithm is the discretization of the equation in (10). To discretize this equation, we use the standard 5-point stencil, modified by a correction term near the boundary. In general, the discrete equation will have the form

$$\frac{\phi_{i,j+1} + \phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} - (4 - \lambda h^2)\phi_{i,j}}{h^2} + D^{\lambda}[\phi](x_i, y_j) = f_{ij}, \tag{11}$$

where the correction term $D^{\lambda}[\phi](x_i, y_j)$ is nonzero only near the boundary. This correction term, which depends on λ, is computed using the immersed interface method [9, 28, 52] and is used to impose both the boundary conditions for the streamfunction and the singular sources for the vorticity equation. Because of the way in which it is constructed, this correction term leads to a scheme which is first order near the boundary and second order away from the boundary. This is sufficient for second-order accuracy globally. Boundary conditions on the rectangular computational domain are imposed in the standard way. We solve this system of equations by treating the correction term as a source term and using a fast Poisson solver.

2.1.1. *Determining Correction Term $D^{\lambda}[\phi](x_i, y_j)$*

We now describe in some detail how $D^{\lambda}[\phi](x_i, y_j)$ is computed. We define *irregular* points as those grid points (x_i, y_j) at which the standard 5-point stencil straddles the interface. All other grid points are called *regular* points. Associated with each grid point (regular or irregular) is an *interface* point, denoted in Cartesian coordinates as $(X(\mathcal{P}_\Gamma(x_i, y_j)), Y(\mathcal{P}_\Gamma(x_i, y_j)))$, where the function $s = \mathcal{P}_\Gamma(x, y)$ associates a point $(X(s), Y(s))$ on Γ with $(x, y) \in R$. For example, $\mathcal{P}_\Gamma(x, y)$ may be defined so that $(X(s), Y(s))$ is the closest point

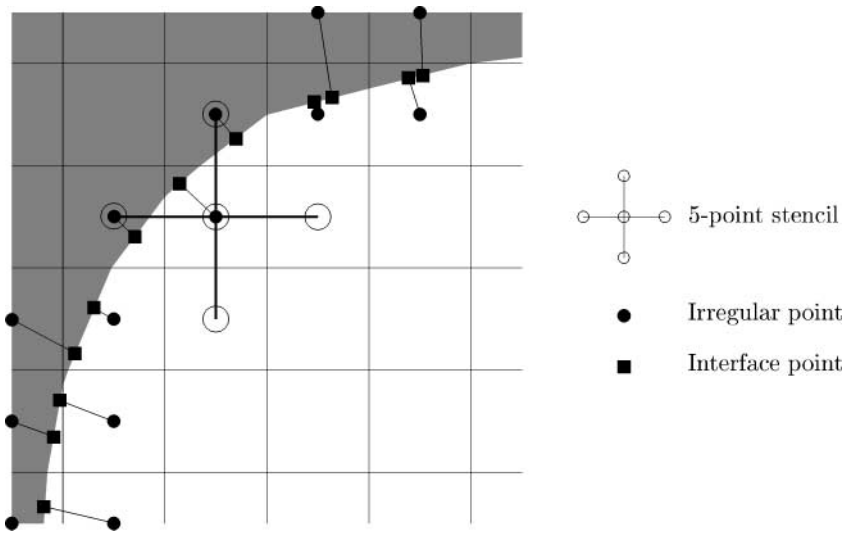

FIG. 4. Dark circles are *irregular* grid points, defined as those points at which a standard 5-point stencil (shown by empty circles) straddles the boundary of the flow domain. Irregular points can be either in the flow domain or in the no-flow domain. The black squares associated with each irregular point are the *interface* points assigned to each irregular grid point.

to (x, y) on the interface. We only require that $\mathcal{P}_\Gamma(x, y)$ be defined in such a way that the distance between an irregular point and its associated interface point is $O(h)$. For regular points, the definition of $\mathcal{P}_\Gamma(x, y)$ turns out to be irrelevant, since the correction term $D[\phi](x_i, y_j)$ is zero for those points. In Fig. 4, we show a typical arrangement of irregular points, relative to an embedded interface.

In general, the correction term has the form

$$D^\lambda[\![\phi]\!](x_i, y_j) \equiv d_{ij}^{\lambda,1}[\![\phi]\!] + d_{ij}^{\lambda,2}[\![\phi_n]\!] + d_{ij}^{\lambda,3}[\![\phi_s]\!] + d_{ij}^{\lambda,4}[\![\phi_{ss}]\!] + d_{ij}^{\lambda,5}[\![(\phi_n)_s]\!] + d_{ij}^{\lambda,6}[\![\nabla^2\phi + \lambda\phi]\!], \tag{12}$$

where $[\![\cdot]\!] = (\cdot)^+ - (\cdot)^-$ denotes a jump in the indicated quantity at the point $(X(s_{ij}), Y(s_{ij}))$, where $s_{ij} \equiv \mathcal{P}_\Gamma(x_i, y_j)$, on the interface. The derivative $d(\cdot)/ds$ is differentiation in the direction tangent to the boundary and $d(\cdot)/dn$ is differentiation in the normal direction. The entries in the coefficient vector $d_{ij}^\lambda \in \mathbb{R}^{6\times 1}$ depend on λ, the location of the point $(X(s_{ij}), Y(s_{ij}))$ relative to the grid, and the parameterization of the curve at s_{ij}. The procedure for computing these coefficients is given in the Appendix, Section A.1.2.1. Also in the Appendix, in Section A.1.1, we illustrate on a one-dimensional problem how the correction term $D^\lambda[\phi](x_i, y_j)$ is derived. The reader may want to look at Section A.1.1 now.

To compute $D^\lambda[\omega]$ and $D^\lambda[\psi]$, we must have the necessary jump conditions in ω and ψ. The jump conditions for vorticity are given in terms of the unknowns w and v and are

$$\begin{aligned} [\![\omega]\!] &= w, & [\![\omega_n]\!] &= v, & [\![\nabla^2\omega + \lambda^*\omega]\!] &= \lambda^*[\![\omega^n]\!], \\ [\![\omega_s]\!] &= w_s, & [\![(\omega_n)_s]\!] &= v_s & &= \lambda^* w^n, \\ [\![\omega_{ss}]\!] &= w_{ss}, \end{aligned} \tag{13}$$

where we have assumed that w and v are differentiable functions of the interface parameter s. To attach a physical meaning to these jump quantities, we make the assumption that the

vorticity inside Ω is identically zero. With this assumption, then, vorticity jumps at the boundary are just the boundary values of the vorticity.

To determine jumps in ψ and ψ_n, we use physical conditions to make convenient choices of values ψ in the no-flow domain. First, we want ψ to be constant inside our object so that we get zero velocities there when we difference ψ using the definitions of u and v given in (1). This gives us $\psi_x^- = \psi_y^- = 0$, or $\psi_n^- = 0$ at solid boundaries. Using this with the no-slip condition $\psi_n^+ = 0$ gives us the jump condition $[\![\psi_n]\!] = 0$. Second, we would like ψ to be continuous across the boundary so we do not introduce any unphysical singularities into the velocity field at the boundary. This gives us $\psi^+ = \psi^-$ at the boundary, or $[\![\psi]\!] = 0$. To get the last jump used in (12), we use the differential equation for ψ. In summary, all the necessary jumps are given by

$$\begin{aligned} [\![\psi]\!] &= 0, \quad [\![\psi_n]\!] = 0, \quad [\![\nabla^2\psi]\!] = -[\![\omega]\!], \\ [\![\psi_s]\!] &= 0, \quad [\![(\psi_n)_s]\!] = 0 \qquad\quad = -w, \\ [\![\psi_{ss}]\!] &= 0, \end{aligned} \tag{14}$$

where $w = [\![\omega]\!]$ is the only unknown.

Mayo [32, 33] first introduced the idea of incorporating jumps into the discretization of the Laplacian. In her approach, Mayo computes the solution to an integral equation for the distribution of sources needed to impose given boundary conditions on an irregular domain. She then uses the density to evaluate the integral to obtain the solution to the original Laplace equation at the irregular points defined above. These values can be obtained quite accurately, even though they are near the interface where the kernel of the integral operator becomes unbounded. To obtain the solution at all mesh points on a two-dimensional Cartesian mesh in which the boundary is embedded, she solves

$$\nabla_h^2 u = \nabla_h^2 u_{irr}, \tag{15}$$

where ∇_h^2 is the discrete 5-point Laplacian, and u_{irr} is equal to the solution (computed from the integral equation) at irregular points and zero everywhere else. This right-hand-side term is used to correct for the fact that in general, the solution as computed from the integral equation will not be smooth across the boundary. This approach has several advantages: (i) the integral equation can be solved very fast using a fast multipole method [36], (ii) fast Poisson solvers can be used to obtain the solution at a large number of mesh points, thus avoiding the need to evaluate the integral at all desired mesh points, and (iii) it is particularly well suited to exterior problems. The idea also extends very naturally to two- and three-dimensional Poisson equations and the biharmonic equation [34–36].

In the immersed interface method, the method described here, we do not solve an integral equation to obtain a density function, but rather solve for jumps in the solution directly. This method was first described by LeVeque and Li [28] for solving elliptic problems with discontinuous coefficients or singular sources along an interface. Later, these ideas were extended to boundary value problems in [9, 51, 52]. The present work most closely follows that of Yang [52].

2.1.2. *Determining Unknown Jumps $[\![\omega]\!] = w$ and $[\![\omega_n]\!] = v$*

We now describe how we can set up a linear system to solve for the unknown jumps $[\![\omega]\!] = w$ and $[\![\omega_n]\!] = v$. To do this, we choose a set of equally spaced *control* points along the

interface at which to solve for unknown jump conditions and then interpolate from these points to all the interface points when necessary. We denote such a set of control points $(X(s_k), Y(s_k)), k = 1, \ldots M$ and impose the given boundary conditions at these points only. In general, the number of control points will be much smaller than the number of irregular points.

In the following few definitions, we use the following notational conventions. For $\mathbf{v} \in \mathbb{R}^{M_1}$, we define

$$(\mathbf{v})_k \equiv \mathbf{v}^T e_k, \tag{16}$$

where e_k is the k^{th} column of the $M \times M$ identity matrix. Similarly, for $G \in \mathbb{R}^{N_x N_y \times 1}$, we define

$$(G)_{ij} \equiv G^T e_{p_{ij}}, \tag{17}$$

where $e_{p_{ij}}$ is the p_{ij}^{th} column in an $N_x N_y \times N_x N_y$ identity matrix. The subscript $p_{ij} \in Z$ takes the (i, j) index of an entry in an $N_x \times N_y$ grid and converts it to a unique index in an $N_x N_y \times 1$ vector. For example, we could have $p_{ij} \equiv (j-1)N_x + i$.

DEFINITION. Let $g(s)$ be a smooth function on Γ, and let $\mathbf{g} \in \mathbb{R}^{M \times 1}$ be the vector of values $\{g(s_k)\}_{k=1}^M$ defined on the control points s_k on Γ. The discrete operators $\mathcal{I}_{ij} \in \mathbb{R}^{1 \times M}$, $\mathcal{I}_k \in \mathbb{R}^{1 \times M}$, $\Delta_s \in \mathbb{R}^{M \times M}$, and $\Delta_{ss} \in \mathbb{R}^{M \times M}$ are defined in terms of their action on $\mathbf{g}$ as

$$\mathcal{I}_{ij}\mathbf{g} \approx g(s_{ij}), \quad \mathcal{I}_k\mathbf{g} = g(s_k), \quad (\Delta_s\mathbf{g})_k \approx g_s(s_k), \quad (\Delta_{ss}\mathbf{g})_k \approx g_{ss}(s_k), \tag{18}$$

where $s_{ij} \equiv \mathcal{P}_\Gamma(x_i, y_j)$. The operator $\mathcal{I}_{ij}$ interpolates between control points to interface points $(X(s_{ij}), Y(s_{ij}))$. The second operator $\mathcal{I}_k$ selects a value from the vector $\mathbf{g}$; it is simply the k^{th} row in the $M \times M$ identity matrix. The third and fourth operators map the vector $\mathbf{g}$ to a second vector whose entries are the derivatives of $\mathbf{g}$. By composing these operators, we can obtain first and second derivatives at interface points using $\mathcal{I}_{ij}\Delta_s\mathbf{g} \approx g_s(s_{ij})$ and $\mathcal{I}_{ij}\Delta_{ss}\mathbf{g} \approx g_{ss}(s_{ij})$.

All of the above operators assume an underlying interpolant through the points $\{g_k\}_{k=1}^M$. Some possible interpolants are the cubic spline and trigonometric polynomials. Since we are presently only handling closed boundaries (e.g., $X(s)$ and $Y(s)$ are periodic functions), we use trigonometric polynomials. Interpolation and differentiation is straightforward and at least when we need $\mathcal{I}_k\Delta_s\mathbf{g}$ and $\mathcal{I}_k\Delta_{ss}\mathbf{g}$, we can make use of the fast Fourier transform. Yang [52] used a standard 3-point stencil to obtain derivatives at control points (which in her case were also the interface points). Le Veque and Li [27] use cubic splines for interpolation and differentiation. Both of these methods yield reliable results as well.

DEFINITION. The matrices D_w^λ, D_v^λ, and D_r^λ, all in $\mathbb{R}^{N_x N_y \times M}$, are defined in terms of their actions on a vector $\mathbf{g} \in \mathbb{R}^{M \times 1}$ and are given by

$$\begin{aligned}
\left(D_w^\lambda \mathbf{g}\right)_{ij} &\equiv \mathcal{I}_{ij}\left\{d_{ij}^{\lambda,1} I_M + d_{ij}^{\lambda,3}\Delta_s + d_{ij}^{\lambda,4}\Delta_{ss}\right\}\mathbf{g}, \\
\left(D_v^\lambda \mathbf{g}\right)_{ij} &\equiv \mathcal{I}_{ij}\left\{d_{ij}^{\lambda,2} I_M + d_{ij}^{\lambda,5}\Delta_s\right\}\mathbf{g}, \\
\left(D_r^\lambda \mathbf{g}\right)_{ij} &\equiv \mathcal{I}_{ij}\left\{d_{ij}^{\lambda,6} I_M\right\}\mathbf{g},
\end{aligned} \tag{19}$$

where the coefficients $d_{ij}^{\lambda,p}$, $p = 1, \ldots 6$ are those in (12), and the matrix $I_M \in \mathbb{R}^{M \times M}$ is the $M \times M$ identity matrix. We use the superscript λ on these matrices to indicate that they depend on λ (through the coefficients $d_{ij}^{\lambda,p}$); the subscripts w, v, and r, however, do

not indicate a dependence on the jumps w, v, or r, but merely serve to indicate how these matrices are applied to the different types of jump conditions.

These matrices act to spread the function $g(s)$, which is defined only along the interface to the neighboring grid points. If we define

$$\begin{aligned} g(s) &\equiv [\![\phi(X(s), Y(s))]\!], \\ g_n(s) &\equiv [\![\phi_n(X(s), Y(s))]\!], \\ g_\Delta(s) &\equiv [\![\nabla^2\phi(X(s), Y(s)) + \lambda\phi(X(s), Y(s))]\!] \end{aligned} \tag{20}$$

and corresponding vectors $\mathbf{g}$, $\mathbf{g}_n$, and $\mathbf{g}_\Delta$, we can relate the matrices D_w^λ, D_v^λ, and D_r^λ to the correction term $D^\lambda[\phi]$ as follows:

$$D^\lambda[\phi](x_i, y_j) \approx \left(D_w^\lambda \mathbf{g} + D_v^\lambda \mathbf{g}_n + D_r^\lambda \mathbf{g}_\Delta\right) ij. \tag{21}$$

This is an approximation rather than an exact equality because the quantities g_s, g_{ss}, and $g(s_{ij})$ are only approximated (using (18)) rather than determined exactly.

For our present purposes, we use the spreading matrices to spread the jumps $w(s)$ and $v(s)$ defined in (13) to the grid. To do so, it is convenient to define vectors $\mathbf{w}$, $\mathbf{v}$, and $\mathbf{r}$.

DEFINITION. The vectors $\mathbf{w}$, $\mathbf{v}$, and $\mathbf{r}$, all in $\mathbb{R}^{M\times 1}$, are defined element-wise as

$$\begin{aligned} w_k &= [\![\omega(X(s_k), Y(s_k))]\!], \\ v_k &= [\![\omega_n(X(s_k), Y(s_k))]\!], \\ r_k &= \lambda^* [\![\omega^n(X(s_k), Y(s_k))]\!] \\ &= \lambda^* w_k^n. \end{aligned} \tag{22}$$

We use the matrices $D_w^{\lambda^*}$, $D_v^{\lambda^*}$, and $D_r^{\lambda^*}$, which are those defined above, with $\lambda = \lambda^* \equiv -1/\nu\Delta t$.

For the elliptic equation for the streamfunction, we need to compute a correction term $D^0[\psi]$, which is defined as in (12), with $\lambda = 0$. Since the only nonzero jumps in ψ are in $\nabla^2\psi$, we can simplify this expression, and we get

$$\begin{aligned} D^0[\psi](x_i, y_j) &= d_{ij}^{0,6} [\![\nabla^2\psi]\!] \\ &= d_{ij}^{0,6} [\![-\omega]\!] \\ &= -d_{ij}^{0,6} \mathcal{I}_{ij} \mathbf{w}. \end{aligned} \tag{23}$$

The only spreading matrix we need for the streamfunction is then given by D_r^0, which is just matrix D_r^λ with $\lambda = 0$.

Using this notation we can now write out a discrete system of equations to solve for the Stokes equations. This linear system is given by

$$\begin{aligned} A^{\lambda^*}\omega + D_w^{\lambda^*}\mathbf{w} + D_v^{\lambda^*}\mathbf{v} + D_r^{\lambda^*}\mathbf{w}^n &= \lambda^*\omega^n, \\ A^0\psi - D_r^0\mathbf{w} &= -\omega. \end{aligned} \tag{24}$$

The matrix $A^\lambda \in \mathbb{R}^{N_x N_y \times N_x N_y}$ is the matrix corresponding to the operator $(\nabla^2 + \lambda)$ and ψ, ω, $\omega^n \in \mathbb{R}^{N_x N_y \times 1}$ are vectors of approximations to the grid values $\psi(x_i, y_j)$ and $\omega(x_i, y_j)$. These equations are the immersed interface discretization of the equations given in (9).

We cannot yet solve (24) for ψ and ω because we still have the $2M$ unknowns in vectors $\mathbf{w}$ and $\mathbf{v}$. Furthermore, we have not incorporated any boundary conditions into the system of equations. To close the system of equations with respect to these $2M$ unknowns, we discretize each of the two boundary conditions on ψ given in (2) at the M control points along the interface. These discretized boundary conditions have the form

$$\begin{aligned}\sum_{m=-1}^{1}\sum_{\ell=-1}^{1}\alpha_k^{m,\ell}\psi_{i_k+m,j_k+\ell}+C[\psi](x_{i_k},y_{j_k})&=\bar{\psi},\\ \sum_{m=-1}^{1}\sum_{\ell=-1}^{1}\gamma_k^{m,\ell}\psi_{i_k+m,j_k+\ell}+C_n[\psi](x_{i_k},y_{j_k})&=0,\end{aligned}\tag{25}$$

where (x_{i_k}, y_{j_k}) is a grid point in the flow domain within a distance h of the control point $(X(s_k), Y(s_k))$. The weights $\alpha_k^{\ell,m}$ and $\gamma_k^{\ell,m}$ are given in Section A.1.3. The correction terms $C[\psi](x, y)$ and $C_n[\psi](x, y)$ are nonzero only if at least one of the nine points used in the stencil is in the no-flow domain. Since we are interpolating to a point on the boundary (the control point), at least one of the nine points will likely be in the no-flow domain, and so the correction terms $C[\psi]$ and $C_n[\psi]$ will usually be nonzero.

Just as in (11), these correction terms can be written in terms of jumps and are given by

$$\begin{aligned}C[\psi](x_{i_k},y_{jk})&=c_k^1[\![\psi]\!]+c_k^2[\![\psi_n]\!]+c_k^3[\![\psi_s]\!]+c_k^4[\![\psi_{ss}]\!]+c_k^5[\![(\psi_n)_s]\!]+c_k^6[\![\nabla^2\psi]\!],\\ C_n[\psi](x_{i_k},y_{jk})&=c_{n,k}^1[\![\psi]\!]+c_{n,k}^2[\![\psi_n]\!]+c_{n,k}^3[\![\psi_s]\!]+c_{n,k}^4[\![\psi_{ss}]\!]\\ &\quad+c_{n,k}^5[\![(\psi_n)_s]\!]+c_{n,k}^6[\![\nabla^2\psi]\!],\end{aligned}\tag{26}$$

where the jumps are evaluated at the control point $(X(s_k), Y(s_k))$. Using the jumps given in (14), we can simplify these expressions and write

$$\begin{aligned}C[\psi](x_{i_k},y_{j_k})&=-c_k^6w_k,\\ C_n[\psi](x_{i_k},y_{j_k})&=-c_{n,k}^6w_k.\end{aligned}\tag{27}$$

These coefficients are given in Section A.1.3.

To write these boundary conditions in matrix form, we introduce two definitions.

DEFINITION. The matrices I and I_n are both in $\mathbb{R}^{M\times N_xN_y}$ and are given by

$$(I\psi)_k=\sum_{m=-1}^{1}\sum_{\ell=-1}^{1}\alpha_k^{\ell,m}\psi_{i_k+\ell,j-k+m},\quad (I_n\psi)_k=\sum_{m=-1}^{1}\sum_{\ell=-1}^{1}\gamma_k^{\ell,m}\psi_{i_k+\ell,j-k+m}.\tag{28}$$

DEFINITION. The matrices I_r and $I_{n,r}$ are both in $\mathbb{R}^{M\times M}$ and are given by

$$(I_r\mathbf{w})_k=c_k^6w_k,\quad (I_{r,n}\mathbf{w})_k=c_{n,k}^6w_k.\tag{29}$$

We can now write the equations in (25) as

$$\begin{aligned}I\psi-I_r\mathbf{w}&=\bar{\psi},\\ I_n\psi-I_{n,r}\mathbf{w}&=0.\end{aligned}\tag{30}$$

Finally, to close the system with respect to the unknown $\bar{\psi}$, we discretize the integral condition in (1) by making use of the fact that $\mathbf{v}\equiv\omega_{\mathbf{n}}^+$, since it is assumed that $\omega_n^-\equiv 0$.

To approximate the integral on a closed boundary, we take advantage of the fact that the control points are equally spaced along the interface and approximate the integral condition using a trapazoidal rule as

$$\int_\Gamma \frac{\partial \omega}{\partial n}\, ds \approx \sum_{k=1}^{M} v_k \Delta s = \xi^T \mathbf{v}, \tag{31}$$

where $\xi \in \mathbb{R}^{M\times 1}$ and $\xi_k = 1, k = 1, \ldots M$. Of course, this may be replaced with other quadrature rules if the points are not equally spaced, or if the boundary is not a closed curve. For a recent discussion of high-order quadrature rules that use essentially equally spaced points, see [2].

In summary, we now have the following set of discrete equations (and their continuous counterparts) to solve for $\mathbf{w}$, $\mathbf{v}$, and $\bar{\psi}$.

$$\begin{aligned}
\omega_t = \nu\nabla^2\omega &\rightarrow A^{\lambda^*}\omega + D_w^{\lambda^*}\mathbf{w} + D_v^{\lambda^*}\mathbf{v} + D_r^{\lambda^*}\mathbf{w}^n = \lambda^*\omega^{\mathbf{n}},\\
\nabla^2\psi = -\omega &\rightarrow A^0\psi - D_r^0\mathbf{w} = -\omega,\\
\psi = \bar{\psi} &\rightarrow I\psi - I_r\mathbf{w} = \bar{\psi},\\
\psi_n = 0 &\rightarrow I_n\psi - I_{n,r}\mathbf{w} = 0,\\
\int_\Gamma \omega_n\, ds = 0 &\rightarrow \xi^T\mathbf{v} = 0.
\end{aligned} \tag{32}$$

Eliminating ψ and ω from the above set of equations, we obtain a $(2M+1)\times(2M+1)$ dense linear system for $\mathbf{w}$, $\mathbf{v}$, and $\bar{\psi}$ which we solve at each time step. This linear system is given by

$$\begin{pmatrix} IG^w - I_r & IG^v & -1\\ I_nG^w - I_{n,r} & I_nG^v & 0\\ 0 & \xi^T & 0 \end{pmatrix}\begin{pmatrix}\mathbf{w}\\ \mathbf{v}\\ \bar{\psi}\end{pmatrix} = \begin{pmatrix} I(A^0)^{-1}(A^{\lambda^*})^{-1}\left(\lambda^*\omega^n - D_r^{\lambda^*}\mathbf{w}^n\right)\\ I^n(A^0)^{-1}(A^{\lambda^*})^{-1}\left(\lambda^*\omega^n - D_r^{\lambda^*}\mathbf{w}^n\right)\\ 0\end{pmatrix}, \tag{33}$$

where

$$\begin{aligned} G^w &\equiv (A^0)^{-1}(A^{\lambda^*})^{-1}D_w^{\lambda^*},\\ G^v &\equiv (A^0)^{-1}(A^{\lambda^*})^{-1}D_v^{\lambda^*}. \end{aligned} \tag{34}$$

The k^{th} column in the arrays G^w and G^v, both in $\mathbb{R}^{N_xN_y\times M}$, can be viewed as the response in ψ to a unit jump in either ω or ω_n at control point s_k. For convenience, we write the matrix system in (33) as $SJ = R$, where $S \in \mathbb{R}^{(2M+1)\times(2M+1)}$, and $J, R \in \mathbb{R}^{(2M+1)\times 1}$.

To solve the complete Stokes equations, we form the matrix S once, factor it, and use it at each successive time step to determine the correction term needed to solve for the streamfunction and vorticity. To form the matrix, we set up a matrix–vector multiply whose arguments are the unknowns $\mathbf{w}$, $\mathbf{v}$, and $\bar{\psi}$. We call this matrix–vector multiply $(2M+1)$ times, each time passing in a column of the $(2M+1)\times(2M+1)$ identity matrix. The result of the matrix–vector multiply is a column in the matrix S. We then factor this matrix into its LU factors and store the factorization. At each time step, we only need to form the right-hand-side R, backsolve the system $LUJ = R$ to obtain the jumps $\mathbf{w}$, $\mathbf{v}$, and unknown $\bar{\psi}$, and then solve for ω and ψ using (24). The algorithm for solving the Stokes equations is summarized in Fig. 5.

Solving the Stokes equations

Form the matrix S in (33) and factor it into its LU decomposition. Set $\mathbf{w}^0$, the jumps at time level t_0, and ω^0, the vorticity field at t_0, to zero.

For time level t_n, $n = 0, 1, \ldots$

1. Form the right-hand-side R of (33) by solving

$$A^{\lambda^*}\omega_I = \lambda^*\omega^n - D_r^{\lambda^*}\mathbf{w}^n,$$
$$A^0\psi_I = -\omega_I$$

for ω_I and ψ_I. Impose homogeneous conditions on the computational domain. The right-hand-side vector R in (33) is given by $R = -[I\psi_I,\ I_n\psi_I,\ 0]^T$.

2. Backsolve the system $LUJ = R$ to get $\mathbf{w}$, $\mathbf{v}$, and $\overline{\psi}$.

3. Solve equations

$$A^{\lambda^*}\omega = \lambda^*\omega^n - D_w^{\lambda^*}\mathbf{w} - D_v^{\lambda^*}\mathbf{v} - D_r^{\lambda^*}\mathbf{r},$$
$$A^0\psi = -\omega + D_r^0\mathbf{w}$$

to get ω and ψ at the current time step.

4. Solutions at time level t_{n-1} are then given by $\mathbf{w}^{n+1} = \mathbf{w}$, $\omega^{n+1} = \omega$, and $\psi^{n+1} = \psi$.

FIG. 5. Time stepping algorithm for solving the Stokes equations.

One key advantage of our approach over, for example, the approach taken by Ye and coworkers [53] is that our approach can make use of fast Poisson solvers to solve the elliptic and diffusion equations and does not rely on preconditioners to obtain optimal efficiency.

2.2. Computing the Velocities from the Streamfunction ψ

The vorticity transport equation given in (6) is discretized in a finite-volume setting using the high-resolution algorithms available in CLAWPACK [25]. The details of the algorithm we use for handling the partial cells cut by the embedded geometry are described in [10]. Here, we only briefly describe how we compute the velocities from the streamfunction ψ.

The average velocities $U_{i\pm1/2,\,j}$ and $V_{i,\,j\pm1/2}$ along each edge are easily computed by differencing a streamfunction ψ at the corners. We compute the average horizontal velocity across the left edge of each mesh cell as

$$\begin{aligned} U_{i-1/2,\,j} &= \frac{1}{\Delta y}\int_{y_{j-1/2}}^{y_{j+1/2}} \psi_y(x,\,y)\,dy \\ &= \frac{1}{\Delta y}\left[\psi\big(x_{i-1/2},\,y_{j+1/2}\big) - \psi\big(x_{i-1/2},\,y_{j-1/2}\big)\right] \end{aligned} \tag{35}$$

and the average vertical velocity along the bottom edge of each mesh cell as

$$\begin{aligned} V_{i,j-1/2} &= -\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \psi_x(x, y)\, dx \\ &= -\frac{1}{\Delta x}\left[\psi\big(x_{i+1/2}, y_{j-1/2}\big) - \psi\big(x_{i-1/2}, y_{j-1/2}\big)\right]. \end{aligned} \tag{36}$$

Since ψ is assumed constant inside the bodies and continuous at the boundaries, the average velocity over an edge that is entirely inside the body is identically zero. This is exactly the result we would obtain for velocities computed from the above formulas. At an edge which cuts the boundary, we also have that $U_{i\pm1/2,j}$ and $V_{i,j\pm1/2}$ are averaged velocities over the entire edge. This can be deduced by breaking up the integral above into two pieces; one inside the body and one outside the body.

The corner values of the streamfunction needed to compute these velocities are obtained using an interpolation procedure described in Section A.1.2.3. These velocities are then passed to the advection algorithm in CLAWPACK and used to compute fluxes needed to update the vorticity cell averages.

We can maintain stability only if we take time steps which do not violate the CFL condition given by

$$\max_{i,j}\left(\left|\frac{U_{i\pm1/2}\Delta t}{\kappa_{ij}\Delta x}\right|, \left|\frac{V_{i\pm1/2}\Delta t}{\kappa_{ij}\Delta y}\right|\right) \le 1, \tag{37}$$

where κ_{ij}, $0 \le \kappa_{ij} \le 1$, is the fraction of a mesh cell (x_i, y_j) in the flow domain. We refer to this fraction as the *capacity* of the mesh cell. From (37), it is clear that arbitrarily small cells (i.e., cells for which $\kappa_{ij} \ll 1$) can place severe restrictions on the size of the time step needed to maintain stability. To avoid this severe time restriction, we increase the capacity of very small triangular cells to ensure that their volume is at least $O(h)$. This approach is described in [10].

2.3. Solving the Full Streamfunction-Vorticity Equations

We now have all the components we need to solve the full streamfunction-vorticity equations. In order to put everything together, however, we must explain the manner in which we alternate between the cell-averaged value obtained at the end of the advection step and the pointwise value we obtained at the end of the finite-difference Stokes flow step.

To describe our method, we consider three steps in the fractional step scheme: an advection step, followed by a diffusion step, followed by another advection step. These three steps are as follows:

$$\begin{aligned} &\text{Step 1,} \quad \omega^* = \omega^n - \Delta t(\vec{u}\cdot\nabla)\omega^n; \\ &\text{Step 2,} \quad \nabla^2\omega^{n+1} + \lambda^*\omega^{n+1} = \lambda^*\omega^*; \\ &\text{Step 3,} \quad \omega^{**} = \omega^{n+1} - \Delta t(\vec{u}\cdot\nabla)\omega^{n+1}. \end{aligned} \tag{38}$$

The question at hand is how to interpret ω^* in the second step, and how to interpret ω^{n+1} in the third step. In the advection step, the discrete value ω_{ij} is a cell average, approximating the average value of ω over the cell centered at (x_i, y_j). In the diffusion step, this value is a pointwise value, approximating $\omega(x_i, y_j)$.

We explain our approach in each of the three types of cells: full cells, empty cells, and partial cells. Here, as in (37), the *capacity* κ_{ij} of a mesh cell is the fraction of the cell in the flow domain.

- **Full cells** ($\kappa_{ij} = 1$) **and empty cells** ($\kappa_{ij} = 0$): In these cells, a cell-averaged value of ω will agree with a pointwise value to $O(h^2)$. For this reason, we use values ω^*_{ij} obtained at the end of step 1 directly as a right-hand-side value in step 2. Also, after the diffusion step, we can use ω^{n+1} directly as the right-hand-side in step 3.
- **Partial cells** ($0 < \kappa_{ij} < 1$): There are two types of partial cells to consider: those cells in which the cell center is not covered by the body, and those in which the cell center is covered by the body. In those cells in which the cell center is in the flow domain, we allow ω^* to be used directly as a right-hand-side value in step 2, and let ω^{n+1} be used as the right-hand-side value in step 3. In these cells, the cell average and the pointwise value at the centroid of the partial cell agree to $O(h)$.

In a cell centered at (x_i, y_j) whose center is covered by the body, the value of ω^* will in general be nonzero, since $\kappa_{ij} > 0$ in that cell. In these cells, we set the pointwise value which is used as the right-hand-side in step 2 to zero. This zero pointwise value is the only value that is consistent with the immersed interface method discretization of the diffusion equation; inside the body, the vorticity is zero, since we have adopted the convention that inside the body, we have $\omega^- = 0$.

In going from step 2 to step 3, we must recover a nonzero value of ω^{n+1}. We do this by extending smoothly the vorticity distribution to the cell center of the partial cell. The extended pointwise value at the cell center (x_i, y_j) is given to third order by

$$\begin{aligned}\tilde{\omega}(x_i, y_j) &= e^1_{ij}[\![\omega]\!] + e^2_{ij}[\![\omega_n]\!] + e^3_{ij}[\![\omega_s]\!] + e^4_{ij}[\![\omega_{ss}]\!] + e^5_{ij}[\![(\omega_n)_s]\!] + e^6_{ij}[\![\nabla^2\omega + \lambda^*\omega]\!] \\ &= e^1_{ij} w + e^2_{ij} v + e^3_{ij} w_s + e^4_{ij} w_{ss} + e^5_{ij} v_s + e^6_{ij}\lambda^* w^n. \end{aligned} \tag{39}$$

Solving the streamfunction-vorticity equations

Form the matrix S in (33) and factor it into its LU decomposition. Take one step of the Stokes flow algorithm described in Fig. 5, using $\mathbf{w^0} = \mathbf{0}$ and $\omega^0 = 0$. Use the resulting streamfunction to obtain initial velocities U^1 and V^1, using (35) and (36). Use the resulting vorticity field ω^1 to initialize flow below.

For time level t_n, $n = 1, 2, \ldots$

1. Extend solution ω^n to partial center-covered cells using (39).
2. Advect vorticity field ω^n using the finite-volume advection scheme and capacity form differencing in CLAWPACK to get ω^*. Impose no-flow conditions on all solid boundaries.
3. Set all values of ω^* in center-covered cells to zero.
4. Take one step of the Stokes flow algorithm described in Fig. 5. Use ω^* as the vorticity field at time level t_n and jumps $\mathbf{w^n}$ from the previous time level. Resulting values ψ, ω, $\mathbf{w}$, and $\mathbf{v}$ are values at time level t_{n+1}.
5. Interpolate values of ψ^{n+1} to grid nodes using (A.28).
6. Compute velocities U^{n+1} and V^{n+1} using (35) and (36).

FIG. 6. Algorithm for solving the streamfunction-vorticity equations.

The jumps are all evaluated at $(X(s_{i'j'}), Y(s_{i'j'}))$, where $s_{i'j'} = \mathcal{P}_\Gamma(x_{i'}, y_{j'})$ for some point $(x_{i'}, y_{j'})$ in the flow domain for which $|i - i'| \le 1$ and $|j - j'| \le 1$. The coefficients e_{ij}^p, $p = 1, \ldots 6$ are computed using the formulas given in the Appendix, Section A.1.2.2. This value agrees with a cell-averaged value of ω to $O(h)$.

In summary, our proposed algorithm for solving the full Navier-Stokes equations is given in Fig. 6.

3. NUMERICAL RESULTS

In this section, we validate our code by comparing our results to those obtained in other benchmark studies. In the first example, we evaluate the accuracy of the Stokes solver code by comparing it to the exact solution of Wannier [50]. In the second example, we compute low Reynolds number flows in an infinite array of cylinders, compute forces on the cylinder, and estimate a numerical rate of convergence for the proposed Navier–Stokes solver. In the third example, we show that we can reproduce the well-known van Karman vortex street, as well as flow patterns for steady flow past a cylinder. Finally we demonstrate that the algorithm works well for problems involving more complicated geometry.

In the first three examples, we compute drag around a circular obstacle. In all three examples, the drag is computed using the formula

$$F_x = r\nu\rho \int_0^{2\pi} \left(-\omega(\theta) + r \frac{\partial \omega(\theta)}{\partial n} \right) \sin(\theta)\, d\theta, \tag{40}$$

where r is the radius of the circular obstacle. To evaluate (40) from data computed at each time step, we discretize the integral using the trapezoidal rule and use jump conditions obtained at the control points s_k, $k = 1, \ldots M$. The discrete formula is given by

$$F_x = \frac{2\pi r \nu \rho}{M} \sum_{k=1}^{M} (-\mathrm{w}_k + r\mathrm{v}_k) \sin(\theta_k), \tag{41}$$

where θ_k is the angle that an outward-directed vector normal to Γ at the interface point $(X(s_k), Y(s_k))$ makes with the horizontal. In all of our computations, we set $\rho = 1$.

Software packages used. To solve the elliptic and parabolic equations arising in the Stokes solver, we use the fast Poisson solver HSTCRT available in the FISHPACK library [47]. This particular routine is ideally suited to our purposes since it returns the solution to the elliptic equation given in (10) at cell centers, but allows us to impose boundary conditions on the computational domain at cell edges.

To solve the advection equation, we use CLAWPACK [25], a software package which implements the fully conservative high-resolution wave propagation algorithms described by LeVeque [26]. This requires a capacity function κ_{ij}, edge velocities $U_{i\pm1/2,j}$, $V_{i,j\pm1/2}$, and a flux limiter choice, among other options. Boundary conditions on the computational domain are imposed by setting values in a layer of ghost cells bordering the computational domain. The use of a high-resolution algorithm for incompressible flow has several advantages. First, the algorithms numerically conserve the advected quantities. While our scheme is not conservative in the cut cells, we conserve vorticity in all full cells by using CLAWPACK. Second, the use of flux limiters improves resolution near the boundary, where sharp jumps in vorticity occur, and minimizes numerical diffusion in the flow regions away from the

boundaries. In all of the examples involving advection, we use the monotonized-central difference limiter, available as an option in CLAWPACK.

Choosing a time step. A current time step Δt_{curr} is chosen so that three competing objectives are satisfied:

1. Time step Δt_{curr} should not violate the CFL condition given in (37) required for stability of the advection scheme.
2. Time step Δt_{curr} should not be too small or we will take more time steps than necessary, thereby slowing down the progress of the simulation and introducing unnecessary numerical diffusion into the solution.
3. Finally, we do not want to change the value of Δt_{curr} too often, since whenever Δt_{curr} changes, we must re-form the matrix S in (33). This is a time-consuming process, and so we want to do it as infrequently as possible.

With these constraints in mind, we choose a time step Δt_{new} from a time step Δt_{cfl} and the current time step Δt_{curr} according to the following strategy. Let

$$W_{max} = \max_{i,j} \left\{ \left| \frac{U_{i\pm 1/2,j}}{\Delta x\, \kappa_{ij}} \right|, \left| \frac{V_{i,j\pm 1/2}}{\Delta y\, \kappa_{ij}} \right| \right\}, \tag{42}$$

where κ_{ij} is the modified capacity function described in [10]. Assume that Δt_{cfl} satisfies

$$0 < \alpha \le W_{max}\, \Delta t_{cfl} < \beta \le 1 \tag{43}$$

for parameters α and β. Then we choose Δt_{new} such that

$$\Delta t_{new} = \begin{cases} \Delta t_{curr} & \text{if } \Delta t_{curr} \in [\alpha \Delta t_{cfl},\, \Delta t_{cfl}/\beta] \\ \Delta t_{cfl} & \text{otherwise.} \end{cases} \tag{44}$$

The new Δt_{curr} is set to Δt_{new}. We typically choose $W_{max}\, \Delta t_{cfl} = 0.9$, and $\alpha = 0.5$ and $\beta = 0.95$. It is easy to check that $W_{max}\, \Delta t_{new} < 1$ so that Δt_{new} satisfies the CFL constraint.

3.1. Stokes Flow in an Annular Region

In this example, we solve the steady state Stokes equations for the infinite cylindrical bearing and compare our computed solution to the analytic one given by Wannier [50]. In this problem, one or both cylinders are allowed to rotate, thereby inducing a flow in the region between the cylinders. Our flow domain is the annular region shown in Fig. 7. The inner ring is centered at (0, 2.75) and has a radius $r_1 = 1$ and the outer ring is centered at (0, 3.25) and has radius $r_2 = 2$. The eccentricity of the bearing (i.e., the distance between the centers of the two circles) is given by $e = 0.5$.

The equations we solve to steady state are the Stokes equations given by

$$\begin{aligned} \omega_t &= \nu \nabla^2 \omega, \\ \nabla^2 \psi &= -\omega, \end{aligned} \tag{45}$$

subject to the boundary conditions $\psi = \psi_n = 0$ on the outer cylinder, and $\psi_n = -1$ on the inner cylinder. The non-zero value on the inner cylinder corresponds to a counterclockwise

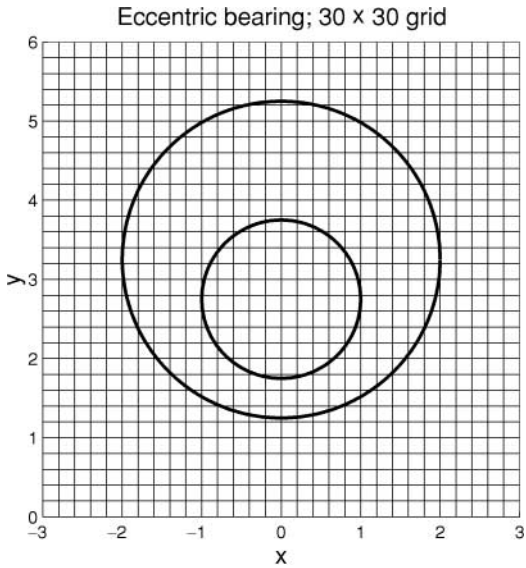

FIG. 7. Sketch of the domain for the problem of Stokes flow in an annular region.

rotation speed of 1. The value of ψ on the inner cylinder is an unknown. The exact solution for the streamfunction in this flow is given in [50] and is

$$\psi_W^*(x, y) = A \log\left(\frac{x^2 + (s+y)^2}{x^2 + (s-y)^2}\right) + By\frac{s+y}{x^2 + (s+y)^2} + Cy\frac{s-y}{x^2 + (s-y)^2} + Dy + E(x^2 + y^2 + s^2) + Fy \log\left(\frac{x^2 + (s+y)^2}{x^2 + (s-y)^2}\right), \tag{46}$$

where s is given by

$$s^2 = \frac{1}{4e^2}(r_2 - r_1 - e)(r_2 - r_1 + e)(r_2 + r_1 + e)(r_2 + r_1 - e) \tag{47}$$

or $s = \sqrt{105}/4$ for the present geometry. The coefficients $A, B, C, D, E,$ and F are given in the Appendix, Section A.2. We set

$$\psi_W(x, y) \equiv \psi_W^*(x, y) - \psi_{outer}^*, \tag{48}$$

where ψ_{outer}^* is the values of ψ_W^* on the outer ring. The shifted solution ψ_W is the one we compare our numerical solution against.

We use the immersed interface method, as described in Section 2.1, to solve Eq. (45). In Fig. 8, we show the results on a 120×120 grid.

To determine the order of accuracy of our solver, we solve the equations on several different grids and compute the norm of the error in the streamfunction as

$$\|\psi_W - \psi\|_p = \|e\|_p = \left(\frac{1}{\text{Area}(R)} \sum_{i,j} |e_{ij}|^p \Delta x \Delta y\right)^{1/p}, \quad 1 \le p < \infty, \tag{49}$$

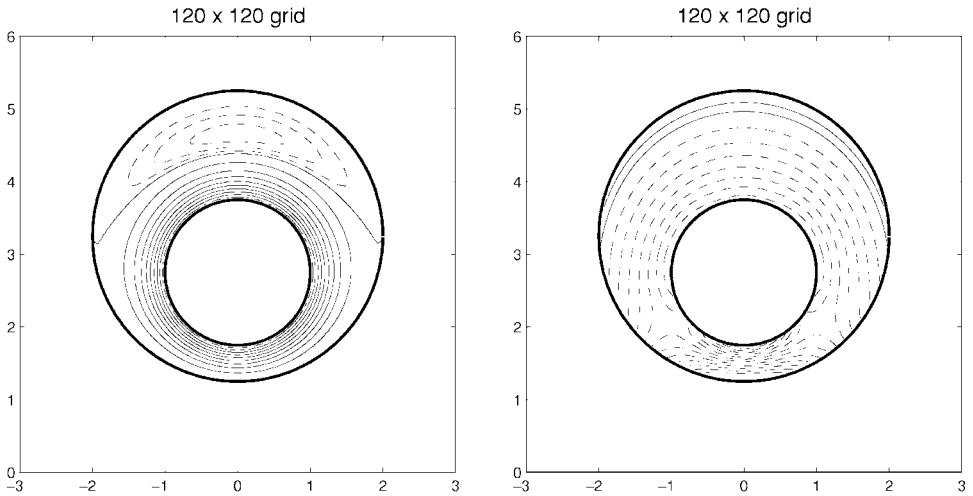

FIG. 8. Streamlines (left) and vorticity contours (right) of Wannier flow between two eccentric bearings. The solution shown here was computed on a 120×120 grid.

where e_{ij} is the error in the computed solution ψ_{ij} at grid point (x_i, y_j) and is computed as

$$|e_{ij}| \equiv |\psi_{ij} - \psi_W(x_i, y_j)|. \tag{50}$$

The quantity Area(R) is the area of the computational domain. The ∞-norm is defined in the standard way as

$$\|e\|_\infty = \max_{i,j} |e_{ij}|. \tag{51}$$

The numerical solution was computed on a series of grids with mesh sizes h_g for $g = 1, \ldots N_{grids}$, where $h_{g+1} < h_g$. We assume that the error in our computed solution takes the form

$$e_g \approx C h_g^p, \tag{52}$$

where p is the convergence rate of the scheme. Taking the log of this expression leads to the linear expression

$$\log(e_g) = p \log(h_g) + \log(C). \tag{53}$$

To determine the numerical convergence rate, we determine a least-squares fit to the data $(\log(h_g), \log(e_g))$ to obtain slope of the best-fit line p and the constant $\log(C)$.

In Fig. 9, we plot the errors, on a series of grids, in the solution ψ, the value $\bar{\psi}$ on the inner cylinder, and the drag coefficient as computed using (41). In the first plot, the errors in the solution converges at an estimated rate of 1.96, which is very nearly the expected rate of 2. In the second plot in Fig. 9, we show convergence results for the computed drag coefficient and flow rate. The exact value of the drag coefficient is given by

$$F_x = 8\pi \nu F, \tag{54}$$

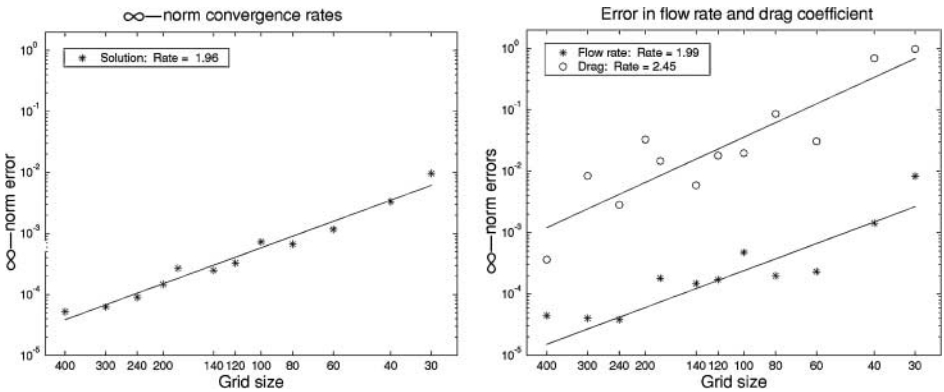

FIG. 9. Convergence rates for the ∞-norm error in the computed streamfunction (left), and flow rate and drag coefficient (right) for an eccentric bearing flow problem. The solid line is the best-fit line through the computed errors. The convergence rates for these errors (given by the slope of best-fit lines) are 1.96 for the solution, 2.45 for the drag, and 1.99 for the flow rate.

where F is the coefficient used in (46). We compute F_x using the formula given in (41) and compare our solution with the exact solution. We observe a convergence rate of 2.45. While this is considerably higher than we expect, we note that the errors in the drag are larger initially then for the solution, and so we would expect that if we went to finer and finer grids, we would eventually see an asymptotic convergence rate closer to 2. In the second plot in Fig. 9, we also plot the errors in the flow rate. In general, the flow rate is the difference between the two values of ψ on each ring of the annulus. In our case, since the outer ring has value 0, we have that the flow rate is equal to the value of the ψ on the inner ring. The errors shown in the plot are the same order of magnitude as the error in the solution and have a convergence rate of 1.99, as expected.

From this example, we conclude that our Stokes solver is performing quite well. Convergence rates for the solution and important quantities on the boundary are all very close to or larger than 2, as expected, and so we consider the Stokes solver a reliable component in the solution of the full streamfunction-vorticity equations.

3.2. Flow in a Biperiodic Array

In the following two examples, we compute the flow in an infinite periodic array of cylinders. In one case, we compute the force on a single cylinder at low Reynolds numbers ($\mathrm{Re} < 10$) and compare our results with asymptotic results obtained by various researchers. In a second example, we compute the solution at a higher Reynolds number ($\mathrm{Re} = 25$) on a series of grids and obtain an estimated convergence rate for our proposed Navier–Stokes solver. The purpose of these examples is to show that our code behaves quite reasonably for low Reynolds number flows and in particular for flow through the types of arrays that may appear in, for example, calculations in porous media and heat exchangers.

In both examples, we consider external flow in a unit square with periodic boundary conditions in both the horizontal and the vertical directions. To vary the Reynolds number for fixed cylinder size, we vary the viscosity parameter ν. In Fig. 10, we show an example of the current geometrical setup.

The boundary conditions on ω in the computational domain are taken to be periodic in both directions. For the streamfunction, we impose periodicity in the horizontal direction.

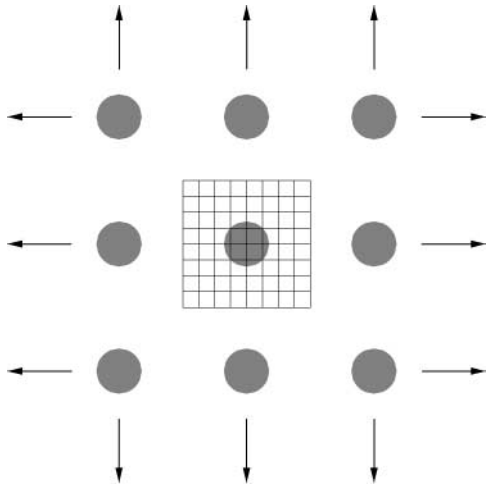

FIG. 10. The grid containing a single shaded circle is a sketch of the computational domain for the biperiodic array. Periodic boundary conditions are imposed on all four boundaries of the computational domain. Flow domain is the region exterior to the shaded circles.

In the vertical direction, we require that

$$\psi_{i,j} = \tilde{\psi}_{i,j} + U_0 y_j, \tag{55}$$

where U_0 is the desired average velocity in the horizontal direction and $\tilde{\psi}_{i,j}$ is the zero-mean solution returned from HSTCRT, with periodic boundary conditions imposed in both directions. In both examples, we advance the solution in time until a steady state is reached.

3.2.1. *Force in a Biperiodic Array*

For this problem, we compute the steady state fluid force (or drag) on the cylinder as a function of solid fraction, ϕ. The solid fraction is taken to be the area of the cylinder in the unit square. For all but the most dilute array ($\phi = 0.00625$), the computations were done on a unit square embedded with a 160×160 grid. For the most dilute array, we used a 320×320 grid. The number of control points was taken so that there were approximately five grid points between control points. In Fig. 11, we show the streamlines and vorticity contours for $\mathrm{Re} = \sqrt{5}$ and $\phi = 0.2$.

According to Mei and Auriault [37], the relationship between solid fraction, Reynolds number, and force in a flow past a cylinder with fore–aft symmetry can be expressed as

$$D \equiv \frac{F_x}{\rho \nu U_0} = k_0(\phi) + k_2(\phi)\mathrm{Re}^2, \quad \text{for } \mathrm{Re} \ll 1, \tag{56}$$

where the Reynolds number is based on the cylinder diameter and F_x is computed using (41). The functional relationships for $k_0(\phi)$ and $k_2(\phi)$ are given by Koch and Ladd [24].

To establish that our algorithm produces data consistent with this model, we use a least-squares fit to obtain the slope m^ϕ and intercept b^ϕ of the best-fit line through data (r_i^ϕ, F_i^ϕ), where F_i^ϕ is the force computed at Reynolds number $\mathrm{Re}^2 = r_i^\phi$ on a cylinder whose volume fraction is ϕ. The independent values r_i^ϕ are chosen so that $r_i^\phi \in [0, 10]$. We then compare the slopes and intercepts obtained for a variety of volume fractions with asymptotic expressions

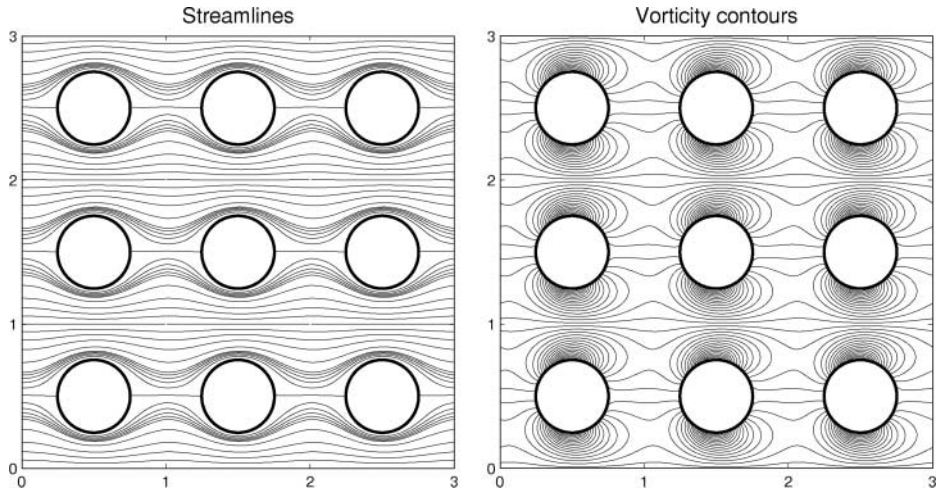

FIG. 11. Streamlines and vorticity contours for flow in a biperiodic domain. The solution was computed on one cylinder and plotted periodically. For this problem, we set Re $= \sqrt{5}$ and volume fraction $\phi = 0.2$. Streamline contour levels are $0 : 0.1 : 1$ and $0.45 : 0.01 : 0.55$. Vorticity contour levels are $-16 : 1 : 16$.

for $k_0(\phi)$ and $k_2(\phi)$ obtained by Sangani and Acrivos for dilute arrays [46] and Koch and Ladd for concentrated arrays [24]. The results are plotted in Fig. 12.

From the plots in Fig. 12, we see that, qualitatively, we get excellent agreement for both $k_0(\phi)$ and $k_2(\phi)$ with the asymptotic expressions in [46] and [24]. Quantitatively, the results for $k_0(\phi)$ appear to be much closer to the asymptotic expressions than do those for $k_2(\phi)$. We point out, however, that our agreement for $k_2(\phi)$ is no worse than that produced by Koch and Ladd [24] using a lattice Boltzman method.

We conclude from this numerical experiment that for low Reynolds number flows, our results are in excellent agreement with the asymptotic results.

3.2.2. *Numerical Convergence for Higher Reynolds Number Flows*

Using the geometric setting described above, we test the numerical convergence of our proposed scheme on a higher Reynolds number (Re $= 25$) flow. Since we do not have an

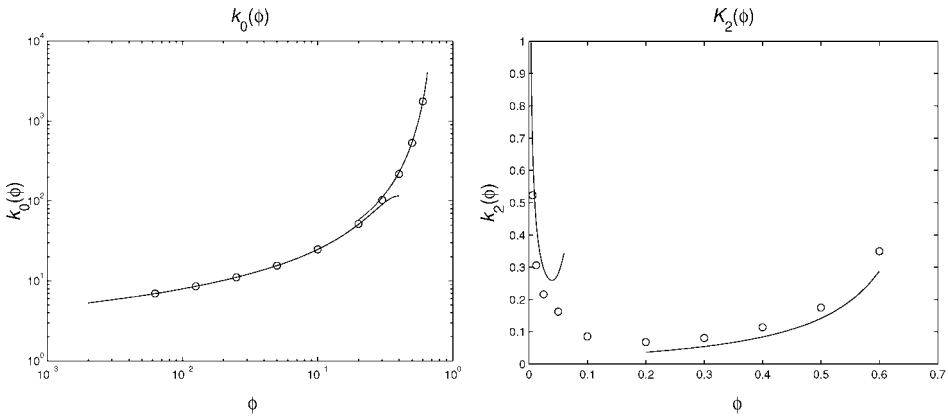

FIG. 12. Curve for intercept $k_0(\phi)$ (left) and slope $k_2(\phi)$ (right) of best-fit lines through forces computed from the present algorithm. The open circles are data computed from the present algorithm and solid lines are asymptotic expressions obtained from [24].

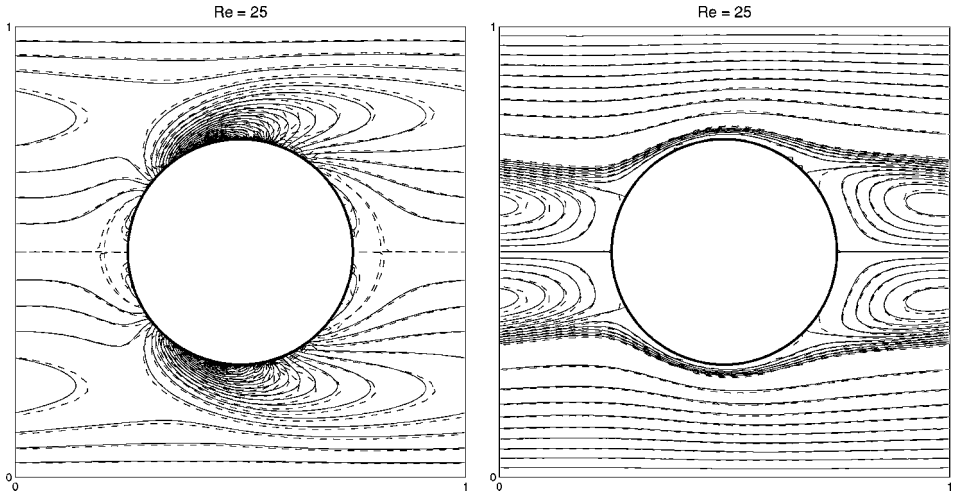

FIG. 13. Contours of vorticity (left) and streamlines (right) of flow in a biperiodic domain at Re $= 25$. Vorticity contours are $-50:2:50$, and streamline contours are $-1:0.05:1$ and $0.04:0.002:0.06$. The dashed line is the solution computed on a 50×50 grid, and the solid line is the solution computed on a 200×200 grid.

exact analytical solution to flow in a biperiodic array, we estimate an error in the solution computed on a particular grid by taking the difference between the solution on that grid and the solution computed on a grid with twice the resolution. Theoretically, this estimated error should have the same order of convergence as an error computed using an exact analytical solution. We then use these error estimates to compute an order of convergence, using the method described in Eqs. (52) and (53).

In Fig. 13, we show the vorticity and streamline contours for the flow computed for this problem. To visually assess the accuracy of our scheme, we plotted the contours computed on two different grids, a 50×50 grid and a 200×200 grid, on top of one another. The plots show very good agreement for the major features of the flow, including the recirculation zones at both the front and the back of the cylinder.

Next, we establish estimates for the numerical convergence rate of the velocities and streamfunction produced by the scheme. In Fig. 14, we plot the errors in these quantities and show a best-fit line through these errors. We estimate the 2-norm order of convergence to be

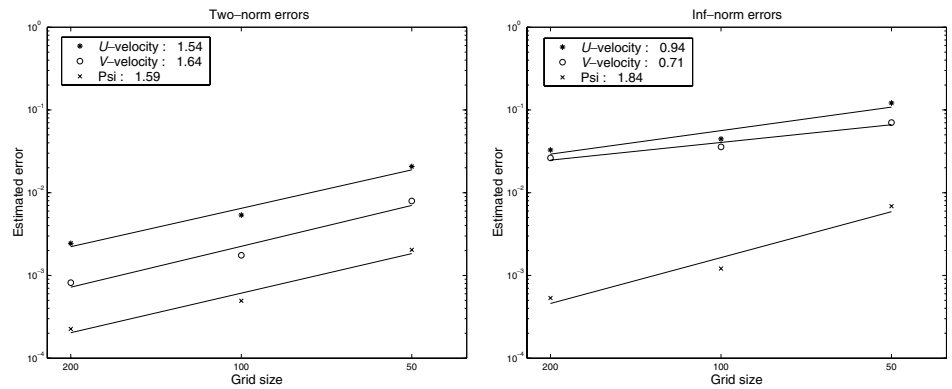

FIG. 14. Estimated errors (computed as the difference between the solutions computed on indicated grid (an $N \times N$ grid) and a finer grid (a $2N \times 2N$ grid) in the 2-norm (left) and the ∞-norm (right) in streamfunction ψ and velocity for the biperiodic flow shown in Fig. 13.

1.54 for the horizontal velocity, 1.64 for the vertical velocity, and 1.59 for the streamfunction. These convergence rates are clearly better than first order; the fact that they are not closer to second order may be due in part to the fact that high-resolution finite-volume schemes do not generally yield second-order results in the presence of sharp discontinuities. In our case, the vorticity has a sharp discontinuity at the boundary of the cylinder.

The ∞-norm order of convergence rates for the same quantities are 0.94, 0.71, and 1.84, respectively. The fact that the velocity convergence rates are lower in the ∞-norm than in the two norm is an indication that the largest errors are occurring on the boundary of our embedded cylinder. An improved scheme for handling the approximation of advective fluxes in the small cut cells may improve this convergence rate.

3.3. Flow Past a Cylinder

One classic, nontrivial two-dimensional problem for which there is a large pool of numerical and experimental results is the problem of calculating the flow around a circular cylinder. We now compare the results obtained from our algorithm with those results published in the literature.

In this problem, we have a circular cylinder embedded in an infinite domain. Flow in the far-field is assumed to be the uniform flow given by $u = U_\infty$ for $t > 0$. For our computations, we use a cylinder of radius $r = 0.5$, which is embedded in a 32×16 computational domain. The cylinder is located near the inflow boundary at (8, 8). The computational mesh is 640×320. To represent the circular cylinder and the solution on the cylinder, we use 16 control points. To vary the Reynolds number $\mathrm{Re} = 2rU_\infty/\nu$, we hold the velocity fixed at $U_\infty = 1$ and vary ν.

On the boundaries of the computational domain, we impose homogeneous conditions on ω and uniform flow conditions on the streamfunction. At the inflow boundary and the two horizontal boundaries, ω in the ghost cells is set to 0. At the outflow boundary, ω in the ghost cells is set equal to its neighboring value in the domain, i.e., $\omega_{641,j} = \omega_{640,j}$. For the diffusion equation, we set $\omega_n = 0$ on all four boundaries. For the streamfunction, we set $\psi(0, y) = U_\infty(y - 8)$ at the inflow boundary and $\psi(x, 0) = -4U_\infty$ and $\psi(x, 16) = 4U_\infty$ at the two horizontal boundaries. These conditions are designed to match the far-field velocity at the inflow boundary and two horizontal boundaries. At the outflow boundary, we set $\psi_n = 0$.

On the cylinder itself, we set $\psi = \bar{\psi} = 0$ for flow in the steady regimes ($\mathrm{Re} = 20$ and $\mathrm{Re} = 40$). For the unsteady regimes ($\mathrm{Re} = 50, 100, 200$), we allow the streamfunction to vary on the cylinder and impose the zero net-flux of vorticity condition given in (1).

Our numerical results are divided into the three sections: a steady flow regime ($\mathrm{Re} = 20$), a transitional flow regime ($\mathrm{Re} = 40, 50$), and an unsteady regime ($\mathrm{Re} = 100, 200$). For the steady flows corresponding to $\mathrm{Re} = 20$ and $\mathrm{Re} = 40$, we compute the length of the wake bubble, the separation angle, and the drag coefficient. For the unsteady flows, we compute the Strouhal number and report on the unsteady lift and drag in plots. We compare our results for these quantities with those found in the experimental and numerical studies of Tritton [48], Coutanceau and Bouard [12, 13], Dennis and Chang [14], and Fornberg [21]. In all of our plots, we plot the nondimensional quantities $\tilde{\omega} = \omega r/U_\infty$ and $\bar{\psi} = \psi/rU_\infty$.

3.4. Steady Flow: Re = 20

In Fig. 16, we have plotted the streamlines and vorticity contours computed for $\mathrm{Re} = 20$. We compared our contours with the same contours in [21] and found very good agreement. In Fig. 15, we show a close-up of the vorticity contours near the cylinder and see

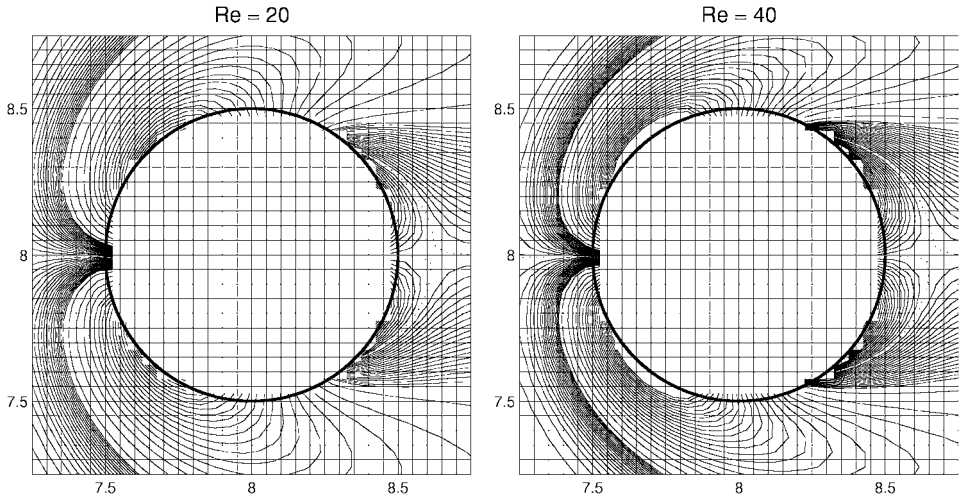

FIG. 15. Close-up of vorticity contours for Re = 20 (left) and Re = 40 (right). The two sets of contour levels used are $-4.6 : 0.2 : 4.6$ and $-0.5 : 0.025 : 0.5$. The dotted line is the zero contour level.

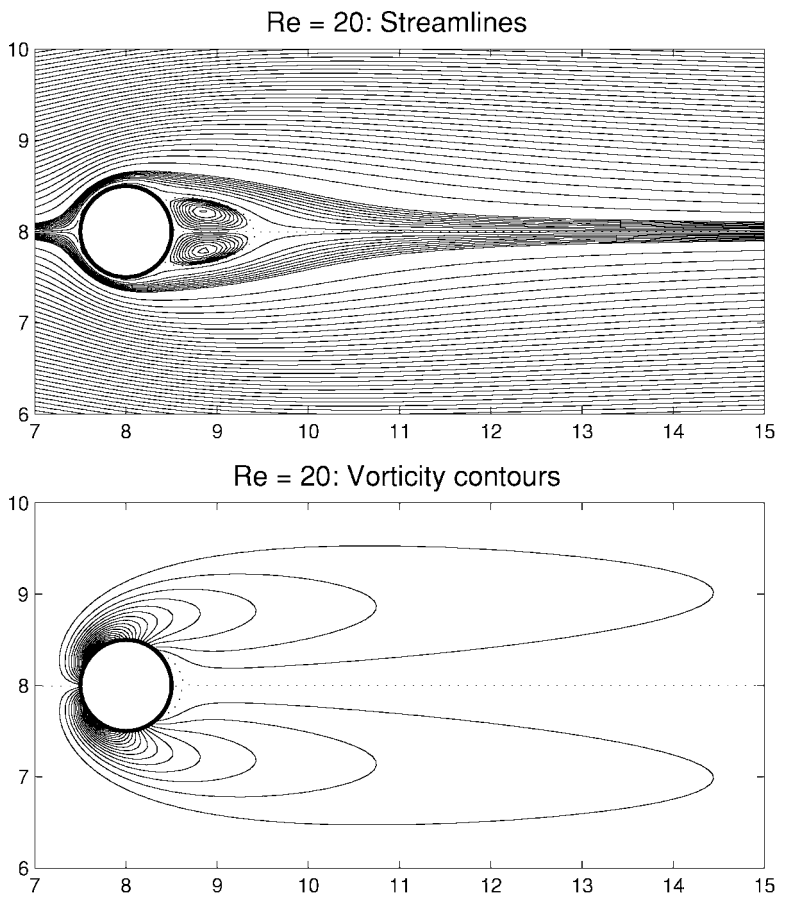

FIG. 16. Contours of vorticity and streamfunction for Re = 20. Contours levels $-5 : 0.2 : 5$ are used in both plots. Additional streamline contours are $-0.1 : 0.01 : 0.1$ and $-0.01 : 0.001 : 0.01$. Some of the streamline contours have been removed for clarity. The dotted line is the zero contour in both plots.

TABLE 1
Wake Bubble Length (L), Angle of Separation (θ), and Drag Coefficient (C_D) for Reynolds Numbers 20 and 40

	Re = 20			Re = 40		
	L	θ	C_D	L	θ	C_D
Tritton [48]	—	—	2.22	—	—	1.48
Coutanceau and Bouard [12]	0.73	42.3°	—	1.89	52.8°	—
Fornberg [21]	0.91	—	2.00	2.24	—	1.50
Dennis and Chang [14]	0.94	43.7°	2.05	2.35	53.8°	1.52
Present	0.91	45.5°	2.19	2.18	54.2°	1.62

Note. Values from first two research groups listed are results from experiments; second two are results of numerical computations.

that the recirculation zone behind the cylinder is very well captured in only a few grid cells.

In Table 1, we see that the length of the wake bubble and the separation angle agree quite well with that of Tritton, Coutanceau, and Fornberg. The drag coefficient $C_d = F_x/\rho U_\infty^2 r$ is somewhat high, but this is probably due to the fact that our computational domain is relatively small for this problem, given our computational boundary conditions for the far-field boundary.

3.5. Transition Regime: Re = 40 and 50

Experimentally, the transition to an unstable wake occurs between Re = 40 and Re = 50 [12]. To see how our code behaves in this transition regime, we computed the vorticity and streamlines for Reynolds numbers 40 and 50 to see if we can detect the onset of an instability. While we do not expect to be able to exactly determine the transition Reynolds number, we expect to detect the onset of an instability by Reynolds number 50.

In Fig. 17 and Fig. 18, we show the contours and streamlines for both Reynolds numbers. The measured wake bubble length, separation angle, and drag coefficient for Re = 40 are given in Table 1.

In Fig. 17 (Re = 40), we see that the two vortices in the streamline plot are perfectly aligned, indicating that the flow is stable. Furthermore, the zero contour in the vorticity plot is straight and lies exactly on the line dividing the top and bottom halves of the domain. For Re = 50, we see that the situation is quite different. The vortices in the streamline plot have begun to slide past one another, indicating the onset of vortex shedding, and the zero-contour level has begun to warp. From these two plots, we conclude that our algorithm is correctly capturing the transition from steady to unsteady flow. This behavior is in good agreement with results reported elsewhere.

3.5.1. *Unsteady Vortex Shedding: Re = 100 and 200*

In this regime, our algorithm reproduces the classic oscillatory wake behind the cylinder. Because our algorithm has slight asymmetries in its implementation details, we found that we do not need to artificially perturb the flow field to initiate the unsteady behavior.

In Fig. 19, we show vorticity contours for both Re = 100 and 200. The characteristic vortex shedding is clearly visible in both plots. In Fig. 20, we plot the time-dependent

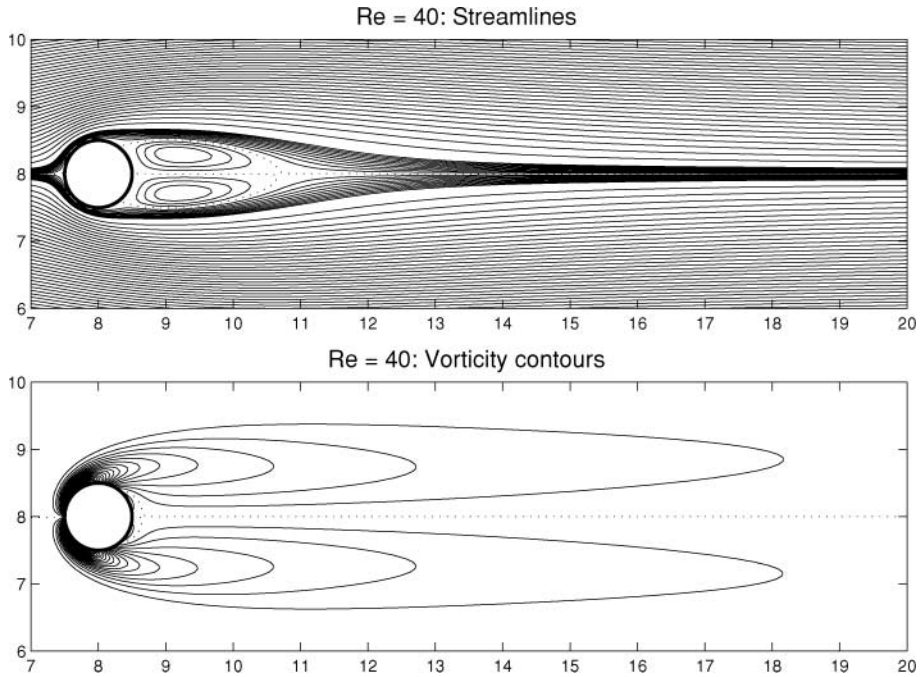

FIG. 17. Streamlines and vorticity contours for Re = 40. Contours levels shown for the streamlines are $-10:0.1:10$ and $-0.1:0.01:0.1$. Contour levels for the vorticity are $-5:0.2:5$. The zero contour level in each plot is shown with the dotted line.

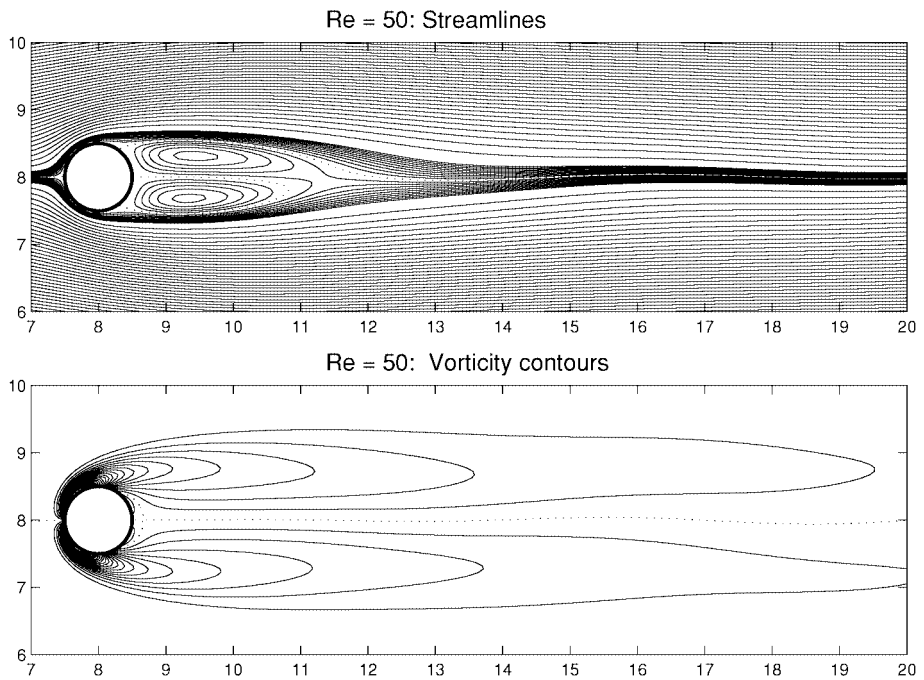

FIG. 18. Streamlines and vorticity contours for Re = 50. Contours levels shown are the same as those in Fig. 17.

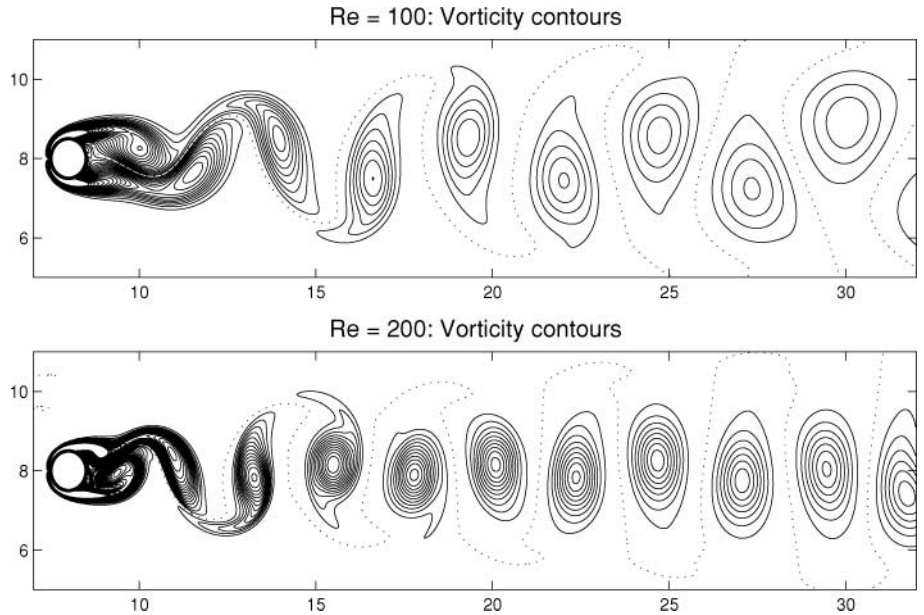

FIG. 19. Contours of vorticity for Re = 100 and 200. The contours shown in each plot are $-2:0.1:2$. The dotted line is the zero contour level.

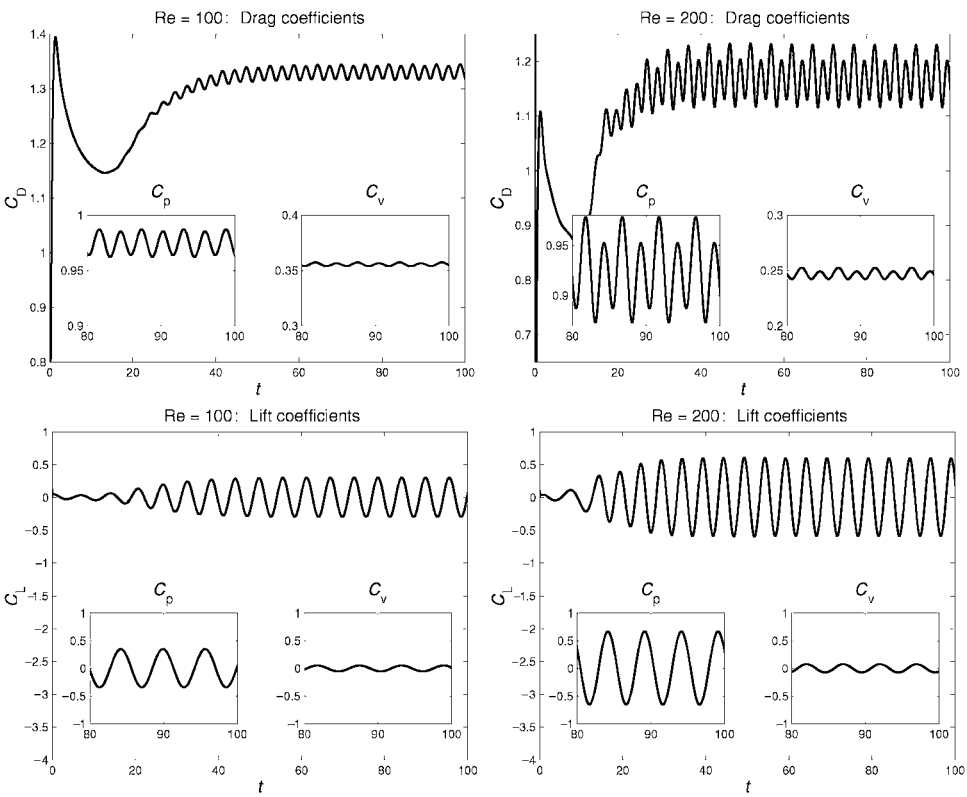

FIG. 20. Time-dependent lift and drag coefficients. (Top left) Drag, Re = 100; (Top right) drag, Re = 200; (bottom left) lift, Re = 100; (bottom right) lift, Re = 200. Inset graphs show pressure and viscous components of each coefficient.

TABLE 2
Drag Coefficients for Re = 100 and Re = 200

Drag (C_D)	Re = 100	Re = 200
Belov *et al*. [3]	—	1.19 ± 0.042
Braza *et al*. [7]	1.364 ± 0.015	1.40 ± 0.05
Liu *et al*. [30]	1.350 ± 0.012	1.31 ± 0.049
Present	1.330 ± 0.014	1.172 ± 0.058

Note. The values following the $\pm$ are the amplitude of the oscillations.

behavior of the lift and drag coefficients, C_D and C_L. The drag coefficient is given by $F_x/\rho U_\infty^2 r$ and the lift coefficient is $F_y/\rho U_\infty^2 r$, where

$$F_y = -r\rho\nu \int_0^{2\pi} (\omega(\theta) + r\omega_n(\theta)) \cos(\theta)\, d\theta. \tag{57}$$

The lift F_y is computed using jumps in a manner analogous to that for F_x. We also plot the pressure and viscous components of the lift and drag coefficients. These components are given by

$$\begin{aligned} C_{D_p} &= \frac{\nu r}{U_\infty^2} \int_0^{2\pi} \omega_n(\theta) \sin(\theta)\, d\theta, \qquad C_{D_v} = -\frac{\nu}{U_\infty^2} \int_0^{2\pi} \omega(\theta) \sin(\theta)\, d\theta, \\ C_{L_p} &= -\frac{\nu r}{U_\infty^2} \int_0^{2\pi} \omega_n(\theta) \cos(\theta)\, d\theta, \quad C_{L_v} = -\frac{\nu}{U_\infty^2} \int_0^{2\pi} \omega(\theta) \cos(\theta)\, d\theta \end{aligned} \tag{58}$$

and are also computed using jumps. These values are shown in insets in plots in Fig. 20 and reported in Table 2 and Table 3. Generally, our results are well within the range of results reported by other researchers.

For Re = 100, the total drag C_D computed using our algorithm was very close to that reported by Braza *et al*. [7]. Our value of $C_D = 1.330 \pm 0.014$ differs from that reported by them by only about 2%. The lift values do not compare quite so favorably; our value differed from theirs by about 20%.

The situation for Re = 200 is inconclusive, only because the values reported elsewhere vary by as much as 15% from each other. We compared our viscous drag values with those found in Belov *et al*. [3] and found very good agreement; our values differed from theirs by about 1.5%. Our computed drag coefficient was about 16% lower than that it Braza *et al*.

TABLE 3
Lift Coefficients for Re = 100 and Re = 200

Lift (C_L)	Re = 100	Re = 200
Belov [3]	—	± 0.64
Braza [7]	± 0.25	± 0.75
Liu [30]	± 0.339	± 0.69
Present	± 0.298	± 0.668

Note. The values following the $\pm$ are the amplitude of the oscillations.

and 10% lower than the results given by Liu *et al*. [30]. Our lift coefficients were about 3% higher than those in Belov *et al*. and about 3% lower than that of Liu *et al*. Our computed lift was about 10% lower than that computed by Braza *et al*. In general, our computed viscous drag coefficients were lower than those reported elsewhere, whereas our lift coefficients were in the mid range of values reported elsewhere.

The only concern we had with our results for the Re $= 200$ case is the apparent vertical asymmetry in the evolution of the drag coefficient. As the vortices shed, it appears that the value of the drag is different depending on whether we are shedding positive vorticity or negative vorticity. However, we computed the same flow on a finer grid and found that the the asymmetry disappeared. From this, we concluded that our algorithm was behaving properly in this regard.

To determine if we are getting the proper shedding frequency, we compute the nondimensional shedding frequency, or Strouhal number, given by $\mathrm{St} = 2fr/U_\infty$. To compute the dimensional frequency f we used the time evolution of the lift coefficient. We estimate a nondimensional Strouhal number of about 0.175 for Re $= 100$ and 0.202 for Re $= 200$. By comparison, Liu reports a value of $St = 0.164$ for Re $= 100$ and $St = 0.192$ for Re $= 200$. For Re $= 200$, Belov reports a value of $St = 0.193$.

3.6. Flow in a Complex Domain

In this problem, we demonstrate our algorithm on a more complex, multiply connected domain. We have a biperiodic domain with five rose-shaped obstacles embedded in it. In Fig. 21, we show the results of this example. The Reynolds number for this example is approximately 10, based on an average diameter of the roses. The average velocity in the horizontal direction is $U_0 = 1$ and is imposed using (55). The diffusion coefficient is set to $\nu = 0.1$. The computational domain is $[0, 4] \times [0, 4]$ with a 160×160 mesh. Each geometric shape is described using the functions

$$x_j(s) = \rho j(s) \cos(2\pi s/M) + c_j^1, \quad y_j(s) = \rho j(s) \sin(2\pi s/M) + c_j^2 \tag{59}$$

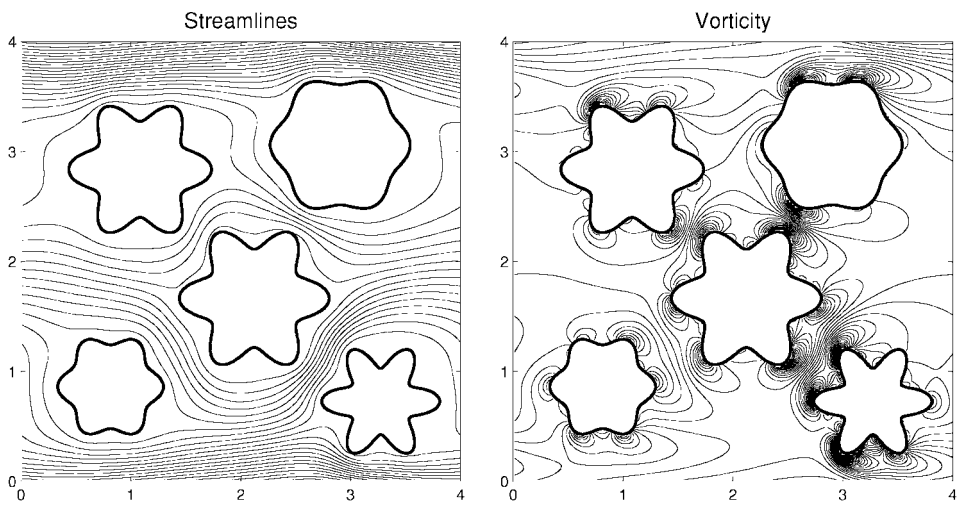

FIG. 21. Streamlines (left) and vorticity contours (right) for flow in complex domain.

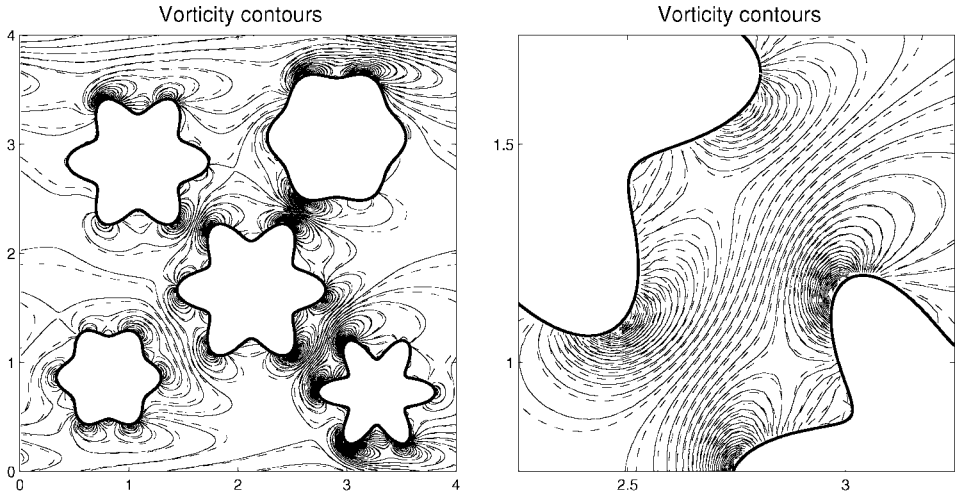

FIG. 22. Vorticity contours for flow in complex domain, computed on two different grids. The right plot shows a portion of the left plot. The solid line is the solution computed on a 160×160 grid; the dashed line is the solution computed on a 320×320 grid. Vorticity contours shown are $-65 : 2.5 : 100$.

for $s \in [0, 63]$, $j = 1, \ldots 5$. The functions $\rho_j(s)$ are given by

$$\rho_j(s) = a_j \cos(2\pi p_j s) + r_j, \tag{60}$$

where the constants a_j, p_j, r_j, c_j^1, and c_j^2 are given in the Appendix. To represent the interfaces, we use 64 control points on each rose shape.

In Fig. 21, we plot the results of this example. We also computed the streamfunction and vorticity on a finer grid and show the plots of vorticity of these two plots in Fig. 22. We see that even though the structure of the flow is quite complicated, we get good agreement between the two plots, even in the most complicated regions.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we have described a finite-difference/finite-volume algorithm for solving the streamfunction-vorticity equations in irregular geometry. We showed that our Stokes solver is second-order accurate and that the full solver produced results which were in good agreement with other experimental and numerical studies of benchmark viscous flow problems. With our algorithm, we met two objectives. First, we have developed a *direct* solver which makes use of existing fast Poisson solvers. The advantage of this approach is that the work per time step does not vary with Reynolds number. Second, we have shown how high-resolution algorithms for advection can be coupled with our Stokes solver. A key idea in our work is that we appeal directly to the notion of vorticity generation and determine singular sources of vorticity at solid boundaries, rather than attempt to impose boundary conditions on the vorticity.

In the future, we plan to extend the algorithm described here to moving geometry and problems involving immersed elastic boundaries. In order to handle moving geometry we will need to develop faster methods for determining the linear system that gives us the singular distribution of vorticity at boundaries. Forming the full matrix at each time step

will be prohibitively expensive, and so we propose two alternative approaches to forming the full matrix S in (33). One option is to take advantage of the fact that the linear system is expressed in terms of a matrix–vector multiply, and so we can solve the system iteratively, using, for example, GMRES. This could potentially be quite successful, since at each time step we have a good initial guess to the source distribution. Each matrix-vector multiply, however, will still require an application of the fast Poisson solver. Another approach is to replace our general elliptic equation in (10) with an equivalent integral equation. The integral equation leads to a very well-conditioned linear system and the corresponding matrix–vector multiply only requires $O(M)$ operations.

Because Cartesian grid methods are easily made adaptive, it may be interesting to consider how adaptive mesh refinement may be applied to our current algorithm. We have already shown in [10] that our capacity form differencing used to solve the transport equation is easily adapted using AMRCLAW [4]. To solve the elliptic and parabolic equations in an adaptive framework, we must replace our current fast Poisson solver with one that works in the adaptive setting. One obvious choice is a multigrid solver. Another choice is the multipole based solver developed by Ethridge and Greengard and described in [16].

To extend our approach to higher order, we would need to use a higher order Poisson solver (see [16] for example), and include higher order jumps and derivatives along the boundary. This will place higher demands on the smoothness of the boundaries, but for problems in which the higher order boundary derivatives are easily obtained, this approach should be quite straightforward to implement. Another advantage of our approach is that problems with piecewise constant coefficients (viscosity for example) are easily handled, as are variable and discontinuous coefficients. In fact, it was these sorts of problems that motivated the original immersed interface method [28].

Finally, there is the question of extending our code to three dimensions. One possibility is to solve the three-dimensional streamfunction-vorticity equations. These equations are not typically solved in three dimensions, because they involve solving for three components of ω and of the vector potential ψ. Additionally, there is a gauge freedom associated with the three-dimensional streamfunction-vorticity equations which must be resolved by imposing divergence conditions on ω and ψ, in addition to those on velocity. Also, determining the correct boundary conditions on the components of ψ is more complicated in three dimensions than in two. Nonetheless, Weinan and Liu [15] propose a 3d vorticity-vector potential formulation on nonstaggered grids that looks very promising. An alternative formulation is the projection methods of [1]. Because our methods for computing the singular sources and imposing boundary conditions on embedded boundaries are not specific to the equations being solved, we can apply our finite-difference scheme to any of these methods. For one example of how the immersed interface method can be applied to a primitive variable formulation of the Navier–Stokes equations, see [29].

APPENDIX

A.1. Derivation of Coefficients for the Immersed Interface Method

Here we provide formulas for the coefficients needed to construct the matrix S in (33) and for extending the solution ω to the no-flow domain and interpolating ψ to grid nodes. To motivate the derivations, we consider a one-dimensional example.

A.1.1. *A One-Dimensional Example*

Suppose we have two functions $u^+(x)$ and $u^-(x)$, both of which are analytic in an interval $R = [R_\ell, R_r]$ of the real line. Let the point $\alpha \in R$ divide the interval into a left region $\Omega^- = [R_\ell, \alpha]$ and a right region $\Omega^+ = (\alpha, R_r]$. We define a function $u(x)$ as

$$u(x) = \begin{cases} u^-(x) & \text{if } x \in \Omega^- \\ u^+(x) & \text{if } x \in \Omega^+. \end{cases} \tag{A.1}$$

The function $u(x)$ will in general be discontinuous at α and have discontinuous derivatives there. These discontinuities can be described in terms of jumps $[\![u]\!] \equiv u^+(\alpha) - u^-(\alpha)$, $[\![u_x]\!] \equiv u_x^+(\alpha) - u_x^-(\alpha)$, $[\![u_{xx}]\!] \equiv u_{xx}(\alpha) - u_{xx}^-(\alpha)$, and so forth. We would like to be able to use a standard 3-point stencil to approximate $u_{xx}(x)$ at a point x near the point α. This, of course, will yield an inaccurate result unless we account for the fact that $u(x)$ is discontinuous at α. But we could apply a 3-point stencil approximation to $u^+(x)$ or $u^-(x)$ and obtain accurate approximations to $u_{xx}(x)$. With this in mind, our goal is to recover u^+ and u^- from u. We do this by using jump conditions.

To recover $u^+(x)$ and $u^-(x)$, we use the fact that we can expand these functions about α and write their difference in terms of jumps. We define

$$C(x) \equiv u^+(x) - u^-(x) = [\![u]\!] + [\![u_x]\!](x-\alpha) + \frac{1}{2}[\![u_{xx}]\!](x-\alpha)^2 + O(|x-\alpha|^3), \tag{A.2}$$

where the jumps are evaluated at α. Using $C(x)$, we define a function $\tilde{u}$ as

$$\tilde{u}(x;\xi) = u(x+\xi) + \sigma(x;\xi)C(x+\xi), \tag{A.3}$$

where

$$\sigma(x;\xi) = \begin{cases} 1 & \text{if } x \in \Omega^+ \text{ and } x+\xi \in \Omega^- \\ -1 & \text{if } x \in \Omega^- \text{ and } x+\xi \in \Omega^+ \\ 0 & \text{otherwise.} \end{cases} \tag{A.4}$$

It is easy to show that

$$\tilde{u}(x;\xi) = \begin{cases} u^+(x+\xi) & \text{if } x \in \Omega^+ \\ u^-(x+\xi) & \text{if } x \in \Omega^-. \end{cases} \tag{A.5}$$

Now, instead of numerically differentiating $u(x)$, we difference $\tilde{u}(x)$. For example, we can approximate u_{xx} at a point x near α using a standard 3-point stencil. Doing so, we get

$$\begin{aligned} u_{xx}(x) &\approx \frac{\tilde{u}(x;h) - 2\tilde{u}(x;0) + \tilde{u}(x;-h)}{h^2} \\ &= \frac{u(x+h) + \sigma(x;h)C(x+h) - 2u(x) + u(x-h) + \sigma(x;-h)C(x-h)}{h^2} \\ &= \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \frac{\sigma(x;h)C(x+h) + \sigma(x;-h)C(x-h)}{h^2}. \end{aligned} \tag{A.6}$$

This final expression holds regardless of whether x is in Ω^+ or in Ω^-. The first term in the last equation is the usual 3-point stencil approximation to u_{xx}; the second term is a correction term that will be nonzero if $x - h < \alpha \leq x + h$. If $C(x)$ were truncated after three terms, then this correction term would be the one-dimensional analog (with $\lambda = 0$) to the correction term in (12), and the approximation to u_{xx} would be first-order accurate.

The formula in (A.6) is essentially that derived in [9]. Similar formulas can also be found in work by Liu and coworkers in [31], by Li and Lai, in [29], and by Wiegmann [51]. The goal of the present derivations is to make very general use of the idea of extending the solution smoothly beyond the interface. Here we use it not only to approximate a Laplacian but also for interpolation, extrapolation, and approximation of boundary conditions.

A.1.2. *The Immersed Interface Method in Two Dimensions*

The idea above extends to higher dimensions quite naturally. Below, we derive the two-dimensional analog of the function $\tilde{u}(x; \xi)$ and use it to determine extensions to the function ω, an interpolant for ψ, and correction terms for the discrete operator $\nabla^2 + \lambda$.

Throughout this discussion, we assume that boundaries are parameterized as $(X(s), Y(s))$ for some parameter $s \in [0, S_{max}]$. Again, we assume that we have only one boundary Γ parameterized as $\Gamma(s) = (X(s), Y(s))$. The boundary divides the computational domain into two regions, Ω^- and Ω^+, where we assume that the Ω^+ region is the flow domain. The projection $s = \mathcal{P}_\Gamma(x, y)$ associates with the point (x, y), a point $(X(s), Y(s))$ on the interface. For irregular grid points (x_i, y_j) it is assumed that the distance between $(X(\mathcal{P}_\Gamma(x_i, y_j)), Y(\mathcal{P}_\Gamma(x_i, y_j))$, and (x_i, y_j) is $O(h)$. Control point $(X(s_k), Y(s_k)$ is assumed to be near a grid point (x_{i_k}, y_{j_k}).

We begin by introducing several definitions.

DEFINITION. We define the function $\pi(\ell, m)$, for $\ell, m \in \{-1, 0, 1\}$ as

$$\pi(\ell, m) = 3m + \ell + 5. \tag{A.7}$$

This function is used to simply enumerate the grid values in a 3×3 box centered at (x_i, y_j). The lower left point (x_{i-1}, y_{j-1}) corresponds to $\ell = -1, m = -1$, or $\pi(-1, -1) = 1$. The upper right corner point (x_{i+1}, y_{j+1}) corresponds to $\ell = 1, m = 1$, or $\pi(1, 1) = 9$.

DEFINITION. The vector $S_h[u](x, y) \in \mathbb{R}^{9\times 1}$ is defined element-wise as

$$\text{Row}_{\pi(\ell,m)}(S_h[u](x, y)) = u(x + \ell h, y + mh). \tag{A.8}$$

The mapping is given in Fig. A1.

DEFINITION. The vectors $V_c[u](x, y)$ and $V_s^\lambda[u](s)$ consist of values of a function $u(x, y)$ and its derivatives and are defined as

$$V_c[u] = \{u, u_x, u_y, u_{xx}, u_{yy}, u_{xy}\}^T, \tag{A.9}$$

$$V_s^\lambda[u] = \{u, u_n, u_s, u_{ss}, (u_n)_s, \nabla^2 u + \lambda u\}^T, \tag{A.10}$$

where the entries in $V_s^\lambda[u]$ are evaluated at $(X(s), Y(s))$. The derivatives $d(\cdot)/ds$ and $d^2(\cdot)/ds^2$ in $V_s^\lambda[u]$ denote differentiation with respect to the interface parameter s; the derivative $d(\cdot)/dn$ is the normal derivative.

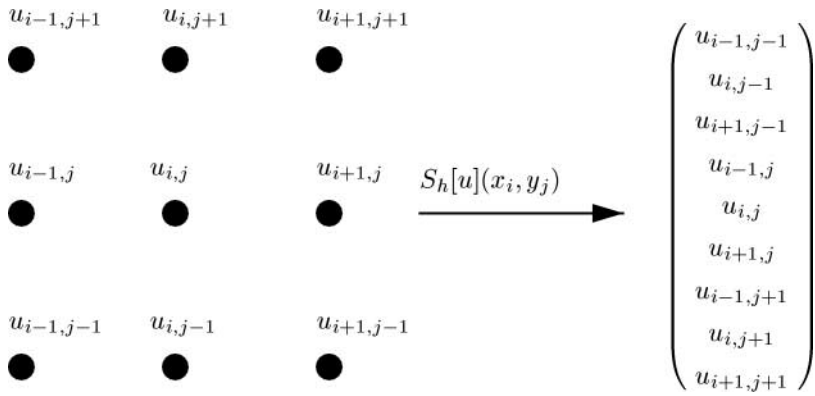

FIG. A1. Mapping $S_h[u]$ that takes grid values in a 3×3 stencil box to a 9×1 vector.

DEFINITION. The vectors $J_c[u](s)$ and $J_s^\lambda[u](s)$, both in $\mathbb{R}^{6\times 1}$, are vectors of the jumps in entries in $V_c[u](x, y)$ and $V_s^\lambda[u](s)$ and are defined as

$$J_c[u] = \{[\![u]\!], [\![u_x]\!], [\![u_y]\!], [\![u_{xx}]\!], [\![u_{yy}]\!], [\![u_{xy}]\!]\}^T, \tag{A.11}$$

$$J_s^\lambda[u] = \{[\![u]\!], [\![u_n]\!], [\![u_s]\!], [\![u_{ss}]\!], [\![(u_n)_s]\!], [\![\nabla^2 u + \lambda u]\!]\}^T, \tag{A.12}$$

where the jumps are evaluated at $(X(s), Y(s))$.

DEFINITION. The scalar $\sigma(x, y; \xi, \eta)$ is defined as

$$\sigma(x, y; \xi, \eta) \begin{cases} 1 & \text{if} (x, y) \in \Omega^+ \text{ and } (x + \xi, y + \eta) \in \Omega^- \\ -1 & \text{if} (x, y) \in \Omega^- \text{ and } (x + \xi, y + \eta) \in \Omega^+ \\ 0 & \text{otherwise.} \end{cases} \tag{A.13}$$

DEFINITION. The vector $P(\delta_1, \delta_2) \in \mathbb{R}^{1\times 6}$ is defined as

$$P(\delta_1, \delta_2) = \left[1, \delta_1, \delta_2, \frac{1}{2}\delta_1^2, \frac{1}{2}\delta_2^2, \delta_1\delta_2\right]. \tag{A.14}$$

DEFINITION. The matrix $\Phi_h'(x, y, s) \in \mathbb{R}^{9\times 6}$ is defined row-wise as

$$\text{Row}_{\pi(\ell,m)}(\Phi_h'(x, y, s)) = \sigma(x, y; \ell h, mh) P(x + \ell h - X(s), y + mh - Y(s)). \tag{A.15}$$

DEFINITION. The matrix $\Phi_h(x, y) \in \mathbb{R}^{9\times 6}$ is defined in terms of $\Phi_h'(x, y, s)$ as

$$\Phi_h(x, y) \equiv \Phi_h'(x, y, \mathcal{P}_\Gamma(x, y)). \tag{A.16}$$

DEFINITION. An extension $E_h[u](x, y) \in \mathbb{R}^{9\times 1}$ of the vector $S_h[u](x, y)$ is defined as

$$E_h[u](x, y) \equiv S_h[u](x, y) + \Phi_h(x, y) J_c[u](X(\mathcal{P}_\Gamma(x, y)), Y(\mathcal{P}_\Gamma(x, y))). \tag{A.17}$$

DEFINITION. The matrix $\Sigma^\lambda(s) \in \mathbb{R}^{6\times 6}$ is defined as

$$\Sigma^\lambda(s) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_s/r & -X_s/r & 0 & 0 & 0 \\ 0 & X_s & Y_s & 0 & 0 & 0 \\ 0 & X_{ss} & Y_{ss} & X_s^2 & Y_s^2 & 2X_sY_s \\ 0 & \sigma_{52} & \sigma_{53} & \sigma_{54} & \sigma_{55} & \sigma_{56} \\ \lambda & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \tag{A.18}$$

where

$$\begin{aligned} \sigma_{52} &= (rY_{ss} - Y_s r_s)/r^2, \\ \sigma_{53} &= -(rX_{ss} - X_s r_s)/r^2, \\ \sigma_{54} &= X_s Y_s/r, \\ \sigma_{55} &= -X_s Y_s/r, \\ \sigma_{56} &= \left(Y_s^2 - X_s^2\right)/r. \end{aligned}$$

The values X_s, Y_s, and higher derivatives are the derivatives of the parameterization $(X(s), Y(s))$ of Γ evaluated at the point s. Since we do not require an arclength parameterization, we introduce $r = \sqrt{X_s^2 + Y_s^2}$ to handle the scaling of the parameterization.

That completes the necessary definitions. We now make the following two remarks:

REMARK. The matrix Σ^λ is used to convert from Cartesian coordinates to a rotated coordinate system. Consequently, the following identities hold:

$$\begin{aligned} V_s^\lambda[u](s) &= \Sigma^\lambda(s) V_c[u](X(s), Y(s)), \\ J_s^\lambda[u](s) &= \Sigma^\lambda(s) J_c[u](s). \end{aligned} \tag{A.19}$$

Using the above, we define $E_h^\lambda[u](x, y)$ in terms of $J_s^\lambda[u](s)$ as

$$E_h^\lambda[u](x, y) \equiv S_h[u](x, y) + \Phi_h(x, y)\{\Sigma^\lambda(\mathcal{P}_\Gamma(x, y))\}^{-1} J_s^\lambda[u](\mathcal{P}_\Gamma(x, y)). \tag{A.20}$$

REMARK. The extension $E_h[u]$ is smooth in the sense that if we assume underlying smooth functions $u^+(x, y)$ and $u^-(x, y)$ (as we did in the one-dimensional example), we then have the approximation

$$E_h[u](x, y) = \begin{cases} S_h[u^+](x, y) + O(h^3) & (x, y) \in \Omega^+ \\ S_h[u^-](x, y) + O(h^3) & (x, y) \in \Omega^- \end{cases} \tag{A.21}$$

provided that the point $(X(\mathcal{P}_\Gamma(x; y)), Y(\mathcal{P}_\Gamma(x, y)))$ is sufficiently close to (x, y). The key here is that $E_h[u]$ can be used in second-order finite-difference approximations in the same manner that we used $\tilde{u}(x; \xi)$ in the one-dimensional example.

We now turn our attention to the manner in which we extend ω, interpolate ψ, and determine correction terms.

A.1.2.1. Determining the correction term $D^\lambda[\phi](x_i, y_j)$. The approximation to the operator $\nabla^2 + \lambda$ of a grid function $u(x_i, y_j)$ is defined as

$$\begin{aligned} \nabla^2 u(x_i, y_j) + \lambda u(x_i, y_j) &\approx G^\lambda E_h^\lambda[u](x_i, y_j) \\ &= G^\lambda S_h[u](x_i, y_j) + G^\lambda \Phi_h(x_i, y_j)(\Sigma^\lambda(s_{ij}))^{-1} J_s^\lambda[u](s_{ij}), \end{aligned} \tag{A.22}$$

where $s_{ij} \equiv \mathcal{P}_\Gamma(x_i, y_j)$ and G^λ is defined as

$$G^\lambda = \left[0, \frac{1}{h^2}, 0, \frac{1}{h^2}, -\frac{4-\lambda h^2}{h^2}, \frac{1}{h^2}, 0, \frac{1}{h^2}, 0\right] \tag{A.23}$$

for $\lambda \leq 0$. From this, it is clear that the correction term $D^\lambda[u](x_i, y_j)$ in (12) is given by

$$\begin{aligned} D^\lambda[u](x_i, y_j) &= \mathbf{d}^\lambda_{ij} J^\lambda_s[u](s_{ij}) \\ &= G^\lambda \Phi_h(x_i, y_j)(\Sigma^\lambda(s_{ij}))^{-1} J^\lambda_s[u](s_{ij}). \end{aligned} \tag{A.24}$$

The coefficients $\mathbf{d}^\lambda_{ij} \in \mathbb{R}^{1\times 6}$ in (12) are then given by

$$\mathbf{d}^\lambda_{ij} \equiv G^\lambda \Phi_h(x_i, y_j)(\Sigma^\lambda(s_{ij}))^{-1}. \tag{A.25}$$

For points (x_i, y_j) away from the boundary, $\Phi_h(x_i, y_j) \equiv 0$ and so the correction term is zero. The above is a first-order approximation to the operator $\nabla^2 + \lambda$.

A.1.2.2. Extending ω to the no-flow domain. From the above, we can easily define the extension of ω to partial center-covered cells. Suppose that the point (x_i, y_j) is the center of a partial center-covered cell. We find a point $(x_{i'}, y_{j'})$ in the flow domain such that $|i - i'| \leq 1$ and $|j - j'| \leq 1$. Then, we define the extension to ω as $\tilde{\omega}(x_{i'}, y_{j'}) \in \mathbb{R}^{9\times 1}$ as

$$\begin{aligned} \tilde{\omega}(x_{i'}, y_{j'}) &\equiv E^{\lambda^*}_h[\omega](x_{i'}, y_{j'}) \\ &= S_h[\omega](x_{i'}, y_{j'}) + \Phi_h(x_{i'}, y_{j'})(\Sigma^{\lambda^*}(s_{i'j'}))^{-1} J^{\lambda^*}_s[\omega](s_{i'j'}). \end{aligned} \tag{A.26}$$

The value $e^T_{\pi(i'-i, j'-j)}\tilde{\omega}(x_{i'}, y_{j'})$ is the *extension* of ω to the cell (x_i, y_j). In this case, the coefficients in (39) are given by

$$\mathbf{e}_{ij} = e^T_{\pi(i'-i, j'-j)} \Phi_h(x_{i'}, y_{j'})(\Sigma^{\lambda^*}(s_{i'j'}))^{-1}, \tag{A.27}$$

where $e_{\pi(\ell,m)}$ is column $\pi(\ell, m)$ of the 9×9 identity matrix.

A.1.2.3. Interpolating ψ. In order to compute velocities using (35) and (36), we must have the values of ψ at grid nodes. To obtain these values, we interpolated using cell-centered values of ψ. Typically, this interpolation will have the form

$$\psi(x_{i-1/2}, y_{j-1/2}) \approx \sum_{m=-1}^{1}\sum_{\ell=-1}^{1} w^{\ell,m}\psi_{i+\ell, j+m} + B[\psi](x_i, y_j). \tag{A.28}$$

With a few exceptions for nodes near the boundary of the computational domain, the weights $w^{\ell,m}$ do not vary from grid point to grid point. The correction term $B[\psi](x_i, y_j)$ is zero only for points near the solid boundary and has a form analogous to that given in (12). But because the only nonzero jump in ψ is $[\![\nabla^2\psi]\!]$, we have

$$\begin{aligned} B[\psi](x_i, y_j) &= b^6_{ij}[\![\nabla^2\psi]\!] \\ &= b^6_{ij}[\![-\omega]\!] \\ &= -b^6_{ij}\mathcal{I}_{ij}\mathbf{w}, \end{aligned} \tag{A.29}$$

where the jumps are evaluated at a point $s_{ij} \equiv \mathcal{P}_\Gamma(x_i, y_j)$.

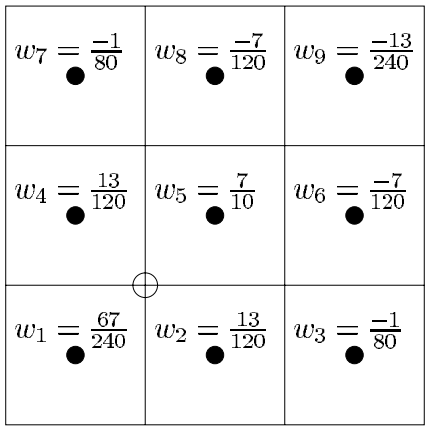

FIG. A2. Weights for interpolation coefficients W of ψ to grid nodes. The set of weights shown here will interpolate to the grid node given by the open circle. Interpolation weights for the other three grid nodes are formed by rotating these weights through the appropriate center axes of symmetry.

To compute the coefficient b_{ij}^6 above, we first choose a stencil center from a cell center neighboring the desired grid node. For all interior grid nodes, we can choose the point (x_i, y_j) as the stencil center for the point $(x_{i-1/2}, y_{j-1/2})$. Only near the boundary of the computational domain will we need to choose a different stencil center. We then approximate ψ at $(x_{i-1/2}, y_{j-1/2})$ using

$$\begin{aligned}\psi(x_{i-1/2}, y_{j-1/2}) &\approx W E_h^0[\psi](x_i, y_j) \\ &= W S_h[\psi](x_i, y_j) + W \Phi_h(x_i, y_j)(\Sigma^0(s_{ij}))^{-1} J_s^0[\psi](s_{ij}), \qquad \text{(A.30)}\end{aligned}$$

where $s_{ij} \equiv \mathcal{P}_\Gamma(x_i, y_j)$ and the coefficients $W \in \mathbb{R}^{1\times 9}$ are given in Fig. A2. The coefficients $w^{\ell,m}$ in (A.28) are exactly the entries of W and are given by

$$w^{\ell,m} = (W)_{\pi(\ell,m)}. \qquad \text{(A.31)}$$

The scalar b_{ij}^6 in (A.29) is given by

$$b_{ij}^6 = W \Phi_h(x_i, y_j)(\Sigma^0(s_{ij}))^{-1} e_6. \qquad \text{(A.32)}$$

With these coefficients, the formula given in (A.28) is accurate to $O(h^3)$. The three other sets of coefficients corresponding to the remaining three corners of the anchor box result from reflecting the coefficients in W around the appropriate axis of symmetry. For example, the coefficients for the point $(x_{i-1/2}, y_{j+1/2})$ result from swapping top and bottom rows of W (as the entries of W appear in the stencil in Fig. 2), and the coefficients for point $(x_{i+1/2}, y_{j-1/2})$ result from swapping left and right columns of W. Although it appears that (A.30) always requires a correction term, such terms are only used when the 9-point stencil intersects the interface; when it is completely on one side of the interface or another, $\Phi_h \equiv 0$.

A.1.3. *Discretizing the Boundary Conditions*

We now switch our emphasis from irregular points (x_i, y_j) and an associated interface point $(X(s_{ij}), Y(s_{ij}))$ to control points $(X(s_k), Y(s_k))$ and an associated grid point (x_{i_k}, y_{j_k}). This point (x_{i_k}, y_{j_k}) is a point in the flow domain and is chosen to be no more than a distance $O(h)$ away from the control point. Some care needs to be taken in choosing these grid points. In particular, one should avoid choosing the same grid point for multiple control points, as this will lead to a poorly conditioned matrix in (33). Note that we do not in general have $s_k = \mathcal{P}_\Gamma(x_{i_k}, y_{j_k})$, since s_k may not be in the range of $\mathcal{P}_\Gamma(x_i, y_j)$, $i = 1, \ldots N_x$, $j = 1, \ldots N_y$. For this reason, we use the definition $\Phi'_h(x, y, s)$ instead of $\Phi_h(x, y)$, since we do not want to assume the use of $\mathcal{P}_\Gamma(x, y)$. We then assume the use of Φ'_h in our definitions of extensions $E_h[u]$ below.

Just as we did above, our approach for determining the correct set of weights needed to interpolate ψ and ψ_n to control points $(X(s_k), Y(s_k))$ is to apply a set of weights used in the smooth case to a smooth extension $E_h[\psi]$ of ψ. Suppose that we have a set of weights $\alpha_k^{\ell,m}$ and $\gamma_k^{\ell,m}$ such that in the absence of any jumps at the interface, we would have

$$\begin{aligned} \psi(X(s_k), Y(s_k)) &= \sum_{\ell,m\in\{-1,0,1\}} \alpha_k^{\ell,m} \psi_{i_k+\ell, j_k+m} + O(h^3), \\ \psi_n(X(s_k), Y(s_k)) &= \sum_{\ell,m\in\{-1,0,1\}} \gamma_k^{\ell,m} \psi_{i_k+\ell, j_k+m} + O(h^2). \end{aligned} \tag{A.33}$$

Then we could apply these weights to the extension $E_h[\psi](x_{i_k}, y_{j_k})$ and get

$$\begin{aligned} \psi(X(s_k), Y(s_k)) &= \tilde{\alpha}_k E_h[\psi]\big(x_{i_k}, y_{j_k}\big) + O(h^3) \\ &\approx \tilde{\alpha}_k S_h[\psi]\big(x_{i_k}, y_{j_k}\big) + \tilde{\alpha}_k \Phi'_h\big(x_{i_k}, y_{j_k}, s_k\big) J_c[\psi](s_k) \\ &\approx \tilde{\alpha}_k S_h[\psi]\big(x_{i_k}, y_{j_k}\big) + \tilde{\alpha}_k \Phi'_h\big(x_{i_k}, y_{j_k}, s_k\big) (\Sigma^0(s_k))^{-1} J_s^0[\psi](s_k), \end{aligned} \tag{A.34}$$

where $\tilde{\alpha} \in \mathbb{R}^{1\times 9}$, and $(\tilde{\alpha}_k)_{\pi(\ell,m)} = \alpha_k^{\ell,m}$. The same approach can be used to approximate ψ_n at control point s_k, using weights $\tilde{\gamma}_k \in \mathbb{R}^{1\times 9}$.

It is fairly straightforward to compute the coefficients $\alpha_k^{\ell,m}$ and $\gamma_k^{\ell,m}$, and we do so now. We begin with a few definitions.

DEFINITION. The matrix $\Lambda(\delta_1, \delta_2) \in \mathbb{R}^{6\times 6}$ is defined as

$$\Lambda(\delta_1, \delta_2) \equiv \begin{pmatrix} 1 & \delta_1 & \delta_2 & \frac{1}{2}\delta_1^2 & \frac{1}{2}\delta_2^2 & \delta_1\delta_2 \\ 0 & 1 & 0 & \delta_1 & 0 & \delta_2 \\ 0 & 0 & 1 & 0 & \delta_2 & \delta_1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{A.35}$$

We use the abbreviation $\Lambda_k \equiv \Lambda(x_{i_k} - X(s_k), y_{j_k} - Y(s_k))$.

DEFINITION. The matrix $\Lambda^h \in \mathbb{R}^{9\times 6}$ is defined in a row-wise fashion as

$$\mathrm{Row}_{\pi(\ell,m)}(\Lambda^h) = \left[1, \ell h, mh, \frac{1}{2}(\ell h)^2, \frac{1}{2}(mh)^2, (\ell h)(mh)\right]. \tag{A.36}$$

With the above definitions, we can write

$$S_h[u]\big(x_{i_k}, y_{j_k}\big) = \Lambda^h \Lambda_k V_c[u](X(s_k), Y(s_k)) + O(h^3) \tag{A.37}$$

for a function u which we assume to be smooth, even at the interface Γ.

These equations represent nine equations in the six unknowns in the vector $V_c[u](X(s_k), Y(s_k))$. To reduce the number of equations in this system to six, we introduce the matrix $Z \in \mathbb{R}^{6\times 9}$ with the following two properties:

1. The product $Z\Lambda^h$ should be nonsingular.
2. The product $Z\xi$, for any vector $\xi \in \mathbb{R}^{9\times 1}$, should have entries that are $O(h^p)$, where $p \geq 0$.

One choice for Z is $(\Lambda^h)^T$. This leads to a least-squares solution to the vector $V_c[\psi](X(s_k), Y(s_k))$ in (A.37). Another option is to construct a Z which simply chooses six points of the 9-point stencil. This was used in the original immersed interface method. The choice we use here is

$$Z = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}. \tag{A.38}$$

This choice results in a second-order approximation of the mixed derivative u_{xy}.

Multiplying both sides of (A.37) by a particular choice of Z gives us the six equations

$$ZS_h[u]\big(x_{i_k}, y_{j_k}\big) = Z\Lambda^h \Lambda_k V_c[\psi](X(s_k), Y(s_k)) + O(h^3). \tag{A.39}$$

Solving for $V_c[u](X(s_k), Y(s_k))$ gives us

$$V_c[u](X(s_k), Y(s_k)) \approx (Z\Lambda^h \Lambda_k)^{-1} ZS_h[u]\big(x_{i_k}, y_{j_k}\big). \tag{A.40}$$

But since boundary conditions are in terms of normal derivatives, we multiply both sides of the above equation by $\Sigma^0(s_k)$ to get

$$V_s^0[u](s_k) \approx \Sigma^0(s_k)(Z\Lambda^h \Lambda_k)^{-1} ZS_h[u]\big(x_{i_k}, y_{j_k}\big). \tag{A.41}$$

We have

$$\begin{aligned} \psi(X(s_k), Y(s_k)) &= e_1^T V_s^0[\psi](s_k), \\ \psi_n(X(s_k), Y(s_k)) &= e_2^T V_s^0[\psi](s_k), \end{aligned} \tag{A.42}$$

and so the α's and γ's given in (25) and used at the introduction of this section are given by

$$\begin{aligned} \alpha_k^{\ell,m} &= e_1^T \Sigma^0(s_k)(Z\Lambda^h \Lambda_k)^{-1} Ze_{\pi(\ell,m)}, \\ \gamma_k^{\ell,m} &= e_2^T \Sigma^0(s_k)(Z\Lambda^h \Lambda_k)^{-1} Ze_{\pi(\ell,m)}, \end{aligned} \tag{A.43}$$

where $e_{\pi(\ell,m)}$ is a column in the 9×9 identity matrix.

Using (A.33), we have that the coefficients given in (27) are

$$\begin{aligned} c_k^6 &= e_1^T \Sigma^0(s_k)(Z\Lambda^h\Lambda_k)^{-1} Z\Phi_h'\big(x_{i_k}, y_{j_k}, s_k\big)\big(\Sigma^0(s_k)\big)^{-1} e_6, \\ c_{n,k}^6 &= e_2^T \Sigma^0(s_k)(Z\Lambda^h\Lambda_k)^{-1} Z\Phi_h'\big(x_{i_k}, y_{j_k}, s_k\big)\big(\Sigma^0(s_k)\big)^{-1} e_6. \end{aligned} \tag{A.44}$$

A.2. Wannier Flow Solution

The coefficients used in Eq. (46) are given in terms of the radii r_1 and r_2 of the inner ring and outer ring of the annulus, respectively, the distance e between the centers of the two rings, the velocities v_1 and v_2 of the two rings, and the variables d_1, d_2, and s, given as

$$\begin{aligned} s^2 &= \frac{1}{4e^2}(r_2 - r_1 - e)(r_2 - r_1 + e)(r_2 + r_1 + e)(r_2 + r_1 - e), \\ d_1 &= \frac{1}{2e}\big(r_2^2 - r_1^2\big) - \frac{1}{2}e, \quad d_2 = \frac{1}{2e}\big(r_2^2 - r_1^2\big) + \frac{1}{2}e. \end{aligned} \tag{A.45}$$

Using these, we get the coefficients A, B, C, D, E, and F as

$$\begin{aligned} A &= -\frac{1}{2}(d_1 d_2 - s^2)A_1, \\ A_1 &= \frac{2\big(d_2^2 - d_1^2\big)(r_1 v_1 + r_2 v_2)}{\big(r_2^2 + r_1^2\big)\big[\big(a_2^2 + a_1^2\big)\log\{(d_1 + s)(d_2 - s)/(d_1 - s)(d_2 + s)\} - 4se\big]} \\ &\quad + \frac{r_1^2 r_2^2\big(r_1^{-1} v_1 - r_2^{-1} v_2\big)}{s\big(r_1^2 + r_2^2\big)(d_2 - d_1)}, \\ B &= (d_1 + s)(d_2 + s)A_1, \\ C &= (d_1 - s)(d_2 - s)A_1, \\ D &= \frac{d_1 \log\{(d_2 + s)/(d_2 - s)\} - d_2 \log\{(d_1 + s)/(d_1 - s)\}(r_1 v_1 + r_2 v_2)}{\big(r_2^2 + r_1^2\big)\big[\big(a_2^2 + a_1^2\big)\log\{(d_1 + s)(d_2 - s)/(d_1 - s)(d_2 + s)\} - 4se\big]} \\ &\quad - \frac{2s\big[\big(r_2^2 - r_1^2\big)/\big(r_2^2 + r_1^2\big)\big](r_1 v_1 + r_2 v_2)}{\big(r_2^2 + r_1^2\big)\big[\big(a_2^2 + a_1^2\big)\log\{(d_1 + s)(d_2 - s)/(d_1 - s)(d_2 + s)\} - 4se\big]} \\ &\quad - \frac{r_1^2 r_2^2\big(r_1^{-1} v_1 - r_2^{-1} v_2\big)}{\big(r_1^2 + r_2^2\big)e}, \\ E &= \frac{1/2 \log\{(d_1 + s)(d_2 - s)/(d_1 - s)(d_s + s)\}(r_1 v_1 + r_2 v_2)}{\big(r_2^2 + r_1^2\big)\big[\big(a_2^2 + a_1^2\big)\log\{(d_1 + s)(d_2 - s)/(d_1 - s)(d_2 + s)\} - 4se\big]}, \\ F &= \frac{e(r_1 v_1 + r_2 v_2)}{\big(r_2^2 + r_1^2\big)\big[\big(a_2^2 + a_1^2\big)\log\{(d_1 + s)(d_2 - s)/(d_1 - s)(d_2 + s)\} - 4se\big]}. \end{aligned}$$

A.3. Constants for Complex Domain

The constants a_j, p_j, and r_j for the numerical example in the complex domain are given in Table A1.

TABLE A1
Constants Needed to Form Roses Given in Final Numerical Example

j	a_j	p_j	r_j	c_j^1	c_j^2
1	0.1057719321071	6	0.5438120613565	1.0837988826816	2.8379888268156
2	0.0446357373219	6	0.5922357710579	2.9050279329609	3.0614525139665
3	0.1174317320178	6	0.5644987493081	2.1229050279330	1.6648044692738
4	0.0502106914425	6	0.4307556381671	0.8156424581006	0.8603351955307
5	0.1173766231097	6	0.4190526007633	3.2737430167598	0.7262569832402

ACKNOWLEDGMENTS

The author thanks for their support the Courant Institute, where much of this paper was written, and the Applied Mathematical Sciences Program of the U.S. Department of Energy. The author also received generous support from the NSF and the DOE while a student at the University of Washington. The author also acknowledges the numerous helpful suggestions provided by Marsha Berger, of the Courant Institute, and Randall J. LeVeque, of the University of Washington.

REFERENCES

1. A. S. Almgren, J. B. Bell, P. Colella, and T. Marthaler, *A Cartesian Grid Projection Method for the Incompressible Euler Equations in Complex Geometries*, Lawrence Livermore National Lab Report UCRL-JC-118091 (Livermore, CA, 1994).
2. B. Alpert, Hybrid Gauss-trapazoidal quadrature rules, *SIAM J. Sci. Comput.* **20**, 1551 (1999).
3. A. Belov, L. Martinelli, and A. Jameson, A new implicit algorithm with multigrid for unsteady incompressible flow calculations. *AIAA 95-0049* **January**, 9 (1995).
4. M. J. Berger and R. J. LeVeque, *AMRCLAW Software*, available at `http://www.amath.washington.edu/~claw/`
5. R. P. Beyer, *A Computational Model of the Cochlea Using the Immersed Boundary Method*, Ph.D. thesis (University of Washington, 1989).
6. R. P. Beyer, A computational model of the cochlea using the immersed boundary method, *J. Comput. Phys.* **98**, 145 (1992).
7. M. Braza, P. Chassaing, and H. Ha Minh, Numerical study and physical analysis of the pressure and velocity fields in the near wake of a circular cylinder, *J. Fluid Mech.* **165**, 79 (1986).
8. B. L. Buzbee, F. W. Dorr, J. A. George, and G. H. Golub, The direct solution of the discrete Poisson equation on irregular grids, *SIAM J. Numer. Anal.* **8**, 722 (1971).
9. D. Calhoun, *A Cartesian Grid Method for Solving the Streamfunction Vorticity Equations in Irregular Geometries*, Ph.D. thesis (University of Washington, 1999), available at `ftp://www.amath.washington.edu/ftp/pub/calhoun/thesis/thesis.ps.gz`.
10. D. Calhoun and R. J. LeVeque, A Cartesian grid finite-volume method for the advection diffusion equation in irregular geometries, *J. Comput. Phys.* **157**, 143 (1999).
11. R. Cortez and M. Minion, The blob projection method for immersed boundary problems, *J. Comput. Phys.* **161**(2), 428 (2000).
12. M. Coutanceau and R. Bouard, Experimental determination of the main features of the viscous flow in the wake of a circular cylinder in uniform translation. Part 1. Steady flow, *J. Fluid Mech.* **79**, 231 (1977).
13. M. Coutanceau and R. Bouard, Experimental determination of the main features of the viscous flow in the wake of a circular cylinder in uniform translation. Part 2. Unsteady flow, *J. Fluid Mech.* **79**, 257 (1977).
14. S. C. R. Dennis and G. Chang, Numerical solutions for steady flow past a circular cylinder at Reynolds number up to 100, *J. Fluid Mech.* **42**, 471 (1970).

15. E. Weinan and J. G. Liu, Finite difference methods for 3d viscous incompressible flows in the vorticity-vector potential formulation on non-staggered grids, *J. Comput. Phys.* **138**, 57 (1997).
16. F. Ethridge and L. Greengard, *A New Fast-Multipole Accelerated Poisson Solver in Two Dimensions*, Technical Report (Courant Mathematics and Computing Laboratory, 2000), Courant Institute of Mathematical Science, New York University.
17. L. Fauci and C. S. Peskin, A computational model of aquatic animal locomotion, *J. Comput. Phys.* **77**, 85 (1988).
18. A. L. Fogelson, A mathematical model and numerical method for studying platelet adhesion and aggregation during blood clotting, *J. Comput. Phys.* **56**, 111 (1984).
19. A. L. Fogelson, Mathematical and computational aspects of blood clotting, in *Proceedings of the 11th IMACS World Congress on System Simulation and Scientific Computation*, edited by B. Wahlstrom (North Holland, Amsterdam, 1985), Vol. 3, pp. 5–8.
20. A. L. Fogelson and C. S. Peskin, A fast numerical method for solving the three-dimensional Stokes equations in the presence of suspended particles, *J. Comput. Phys.* **79**, 50 (1988).
21. B. Fornberg, A numerical study of steady viscous flow past a circular cylinder, *J. Fluid Mech.* **98**, 819 (1980).
22. H. Johansen and P. Colella, A Cartesian grid embedded boundary method for Poisson's equation on irregular domains, *J. Comput. Phys.* **147**, 60 (1998).
23. H. S. Johansen, *Cartsian Grid Embedded Boundary Finite difference Methods for Elliptic and Parabolic Partial Differential Equations on Irregular Domains*, Ph.D. thesis (Dept. Mech. Eng., University of California, Berkeley, 1997).
24. D. L. Koch and A. J. C. Ladd, Moderate Reynolds number flows through periodic and random arrays of aligned cylinders, *J. Fluid Mech.* **349**, 31 (1997).
25. R. J. LeVeque, *CLAWPACK Software* available at `http://www.amath.washington.edu/~claw/`.
26. R. J. LeVeque, Wave propagation algorithms for multidimensional hyperbolic systems, *J. Comput. Phys.* **131**, 327 (1997).
27. R. J. LeVeque and Z. Li, Immersed interface methods for Stokes flow with elastic boundaries or surface tension, *SIAM J. Sci. Comput.* **18**(3), 709 (1997).
28. R. J. LeVeque and Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, *SIAM J. Numer. Anal.* **31**, 1019 (1994).
29. Z. Li and M.-C. Lai, The immersed interface method for the Navier–Stokes equations with singular forces, *J. Comput. Phys.* **171**(2), 822 (2001).
30. C. Liu, X. Zheng, and C. H. Sung, Preconditioned multigrid methods for unsteady incompressible flows, *J. Comput. Phys.* **139**, 35 (1998).
31. X. D. Liu, R. P. Fedkiw, and M. Kang, A boundary condition capturing method for Poisson's equation on irregular domains, *J. Comput. Phys.* **160**(1), 151 (2000).
32. A. Mayo, The fast solution of Poisson's and the biharmonic equations on irregular regions, *SIAM J. Numer. Anal.* **21**, 285 (1984).
33. A. Mayo, Fast high order accurate solution of Laplace's equation on irregular regions, *SIAM J. Sci. Stat. Comput.* **6**, 144 (1985).
34. A. Mayo, Rapid, high order accurate evaluation of volume integrals of potential theory, *J. Comput. Phys.* **100**, 236 (1992).
35. A. Mayo and A. Greenbaum, Fast parallel iterative solution of Poisson's and the biharmonic equations on irregular regions, *SIAM J. Sci. Stat. Comput.* **13**, 101 (1992).
36. A. McKenney, L. Greengard, and A. Mayo, A fast Poisson solver for complex geometries, *J. Comput. Phys.* **118**, 348 (1995).
37. C. C. Mei and J. L. Auriault, The effect of weak inertia on flow through a porous medium, *J. Fluid Mech.* **222**, 647 (1991).
38. C. S. Peskin, Numerical analysis of blood flow in the heart, *J. Comput. Phys.* **25**, 220 (1977).
39. C. S. Peskin, Lectures on mathematical aspects of physiology, *Lect. Appl. Math.* **19**, 69 (1981).
40. C. S. Peskin, Lectures on mathematical aspects of physiology, *Lect. Appl. Math.* **19**, 38 (1981).

41. C. S. Peskin and D. M. McQueen, Modeling prosthetic heart valves for numerical analysis of blood flow in the heart, *J. Comput. Phys.* **37**, 113 (1980).
42. C. S. Peskin and D. M. McQueen, A three-dimensional computational method for blood flow in the heart: (i) immersed elastic fibers in a viscous incompressible fluid, *J. Comput. Phys.* **81**, 372 (1989).
43. C. S. Peskin and D. M. McQueen, A three-dimensional computational method for blood flow in the heart: (ii) contractile fibers, *J. Comput. Phys.* **82**, 289 (1989).
44. J. E. Pilliod Jr. and E. G. Puckett, *Second-Order Volume-of-Fluid Algorithms for Tracking Material Interfaces*, Lawrence Berkeley National Laboratory Technical Report No. LBNL-40744 (1993).
45. W. Proskurowski and O. Widlund, On the numerical solution of Helmholtz's equation by the capacitance matrix method. *Math. Comp.* **30**, 433 (1976).
46. A. S. Sangani and A. Acrivos, Slow flow past periodic arrays of cylinders with application to heat transfer, *Int. J. Multiphase Flow* **8**(3), 193 (1982).
47. P. Swarztrauber, *FFTPACK: A package of Fortran Subprograms for the Fast Fourier Transform of Periodic and Other Symmetric Sequences* (1985), available at `http://www.netlib.org.`
48. D. J. Tritton, Experiments on the flow past a circular cylinder at low Reynolds numbers, *J. Fluid Mech.* **6**, 547 (1959).
49. S. O. Unverdi and G. Tryggvason, A front-tracking method for viscous, incompressible, multifluid Flows, *J. Comput. Phys.* **100**, 25 (1992).
50. G. H. Wannier, A contribution to the hydrodynamics of lubrication, *Q. Appl. Math.* **8**(1), 1 (1950).
51. A. Wiegmann, *The Explicit Jump Immersed Interface Method*, Ph.D. thesis (University of Washington, 1997).
52. Z. Yang, *A Cartesian Grid Method for Elliptic Boundary Value Problems in Irregular Regions*, Ph.D. thesis (University of Washington, 1996).
53. T. Ye, R. Mittal, H. S. Udaykumar, and W. Shyy, An accurate Cartesian grid method for viscous incompressible flows with complex immersed boundaries, *J. Comput. Phys.* **156**, 209 (1999).