A Cartesian Grid Method for Solving the Streamfunction Vorticity Equations in Irregular Geometries

Donna Calhoun

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

1999

Program Authorized to Offer Degree: Department of Applied Mathematics

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Donna Calhoun

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:

Randall J. LeVeque

Reading Committee:

Loyce Adams

Anne Greenbaum

Randall J. LeVeque

Date:

University of Washington

Abstract

A Cartesian Grid Method for Solving the Streamfunction Vorticity Equations in Irregular Geometries

by Donna Calhoun

Chair of Supervisory Committee

Professor Randall J. LeVeque
Applied Mathematics

I present a new algorithm for solving the streamfunction-vorticity equations in irregular, multiply connected regions. To avoid mesh generation difficulties associated with unstructured, body fitted grids, I embed the irregular domain in a uniform Cartesian mesh. The governing partial differential equations are discretized using standard finite volume and finite difference methods away from the irregular boundary. Near the irregular boundary, special discretizations are used to impose boundary conditions. I solve the vorticity transport equation using the high-resolution algorithms in the CLAWPACK package and capacity form differencing to handle the irregular geometry. By modifying the capacity function for small cells cut by the boundary, I can avoid the small cell instability problem associated with explicit Cartesian grid methods and achieve stability for Courant numbers close to 1. To solve for the streamfunction, I use the Immersed Interface Method, a second order finite difference method developed by LeVeque and Li for solving PDEs in irregular regions. Velocities at cell edges, including those cut by the boundary, are obtained by differencing the streamfunction. I compute the flow around a circular cylinder at Reynolds number 20, 40, 50 and 100 and show that I get very good agreement with both experimental and computational results. Flow in other geometries is also shown.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to express my sincerest appreciation to my research advisor, Randy LeVeque, for his patience and support throughout my career as a graduate student. Somehow, he managed to simultaneously give me the freedom to explore my own ideas, all the while providing me with focus necessary to make progress towards a degree. With his financial support, I was able to attend many stimulating conferences and through his connections, I have met several outstanding researchers, many of whom I hope to work with in the future. Most importantly, Randy's clarity and insight into problems are inspiring. I look forward to future collaborations with him.

My committee members have been very helpful during my six years in the department. Whether it was through rigorous coursework or suggestions of interesting problems to look at, I am appreciative of all their help.

Since most of my work was computational, I never could have made the progress I did without the excellent computer facilities in the Applied Mathematics Department. Various department faculty members have allocated grant money to our computer facilities, and I am quite grateful to them. Our computer administrator, Don Bovee, however, is the one who keeps the entire system running smoothly. His help over the years has been invaluable to me. The Department is extremely lucky to have him.

The friendships I have made with other members of the Applied Mathematics Department have been rewarding and I hope long-lasting. I especially want to thank my officemates, past and present, for the much needed levity during the long days at the computer, Kristin Swanson for her help and support while we were writing our theses, Sarah McCord for her friendship throughout the time we have been students here, and Christiane Helzel, a visiting student, for her wonderfully warm sense of humor.

Finally, I would like to thank my friends outside of the department. Without them, I would not have been able to enjoy the natural beauty of the Northwest to the extent that I have. Their friendship in the outdoors has been invaluable to me.

My grandmother Helen Calhoun and my parents Ron and Nancy have been wonderfully supportive of my work. I cannot thank them enough.

This work was supported in part by NSF Grants DMS-9505021, DMS-96226645, and DOE Grant DE-FG03-96ER25292.

Chapter 1

INTRODUCTION

Many interesting and challenging physical problems involve viscous, incompressible fluid flow in geometrically complex regions. In this thesis, we describe a new algorithm for numerically computing such flows in irregular, multiply-connected domains. A typical example of domains we consider is shown in Figure 1.1. Our algorithm, which is based on standard finite difference and finite volume methods on a uniform Cartesian grid, solves the streamfunction-vorticity equations, a non-primitive variable formulation of the incompressible Navier-Stokes equations. If we assume that the flow is two-dimensional, the fluid density and viscosity are constant, the boundaries are fixed, and any source terms can be neglected, these equations can be written

$$\begin{aligned} \omega_t + (\vec{u} \cdot \nabla)\omega &= \nu \nabla^2 \omega \\ \nabla^2 \psi &= -\omega \\ u = \psi_x, \qquad v &= -\psi_y \end{aligned} \tag{1.1}$$

where $\psi(x,y,t)$ is the streamfunction, the components of $\vec{u} \equiv (u(x,y,t), v(x,y,t))$ are the horizontal and vertical velocities, respectively, of the flow field, and $\omega(x,y,t) = v_x(x,y,t) - u_y(x,y,t)$ is the scalar vorticity. The constant ν is the kinematic viscosity. On the boundary of rigid bodies, 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-slip condition. No boundary conditions are explicitly given for ω. Two attractive features of these equations are that they automatically satisfy the incompressibility condition $\nabla \cdot \vec{u} = u_x + v_y = 0$ and that streamlines of the flow are given by level curves of the function ψ. A third feature which we exploit in our numerical scheme is the fact that in two dimensions, the vorticity is a conserved quantity. These equations are easily derived by taking the curl of the Navier-Stokes equations in primitive (velocity-pressure) formulation and eliminating any dependence in the third (z) component.

1.1 A new algorithm for solving the incompressible Navier-Stokes equations

To solve the equations given in (1.1), we first embed the irregular region in a uniform Cartesian grid, as shown in Figure 1.1. The general numerical method is based on

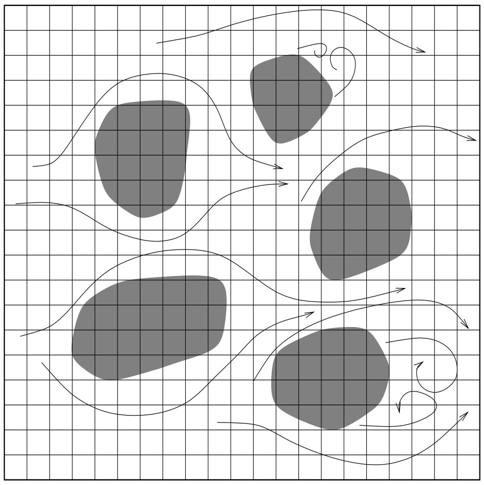


Figure 1.1: Sketch of typical domain in which streamfunction-vorticity equations are solved. The flow domain is the multiply-connected region exterior to all rigid bodies, shown here by shaded regions. At the boundaries of these embedded objects, we must impose no normal-flow and no-slip boundary conditions. The irregular domain is embedded in a uniform Cartesian grid.

standard finite volume and finite difference techniques away from the rigid boundaries of the domain. At all rigid boundaries, we must impose no normal-flow and no-slip boundary conditions. Since the geometry of the irregular domain does not generally align with the underlying grid, a key component of our work is to describe in detail how we accurately impose these boundary conditions along the no-slip boundary, which we approximate a by a single linear boundary segment in each partial grid cell, as seen in Figure 1.2.

The equations in (1.1) are discretized according to the diagram in Figure 1.2. The vorticity is defined at cell centers, the streamfunction at cell nodes and the vorticity at cell edges. We assume that the vorticity is a cell-averaged quantity, where the average is taken only over that portion of the cell which lies in the flow-domain. The velocity values are the average velocities over the entire cell edge. To solve the vorticity transport equation

$$\omega_t + (\vec{u} \cdot \nabla)\omega = \nu \nabla^2 \omega \tag{1.2}$$

we split the advection term from the diffusion term using a fractional step Godunov splitting. The advection equation arising from this splitting is then solved using the high resolution algorithms in CLAWPACK [33]. To minimize numerical oscillations associated with second order advection schemes, we apply flux limiters supplied by

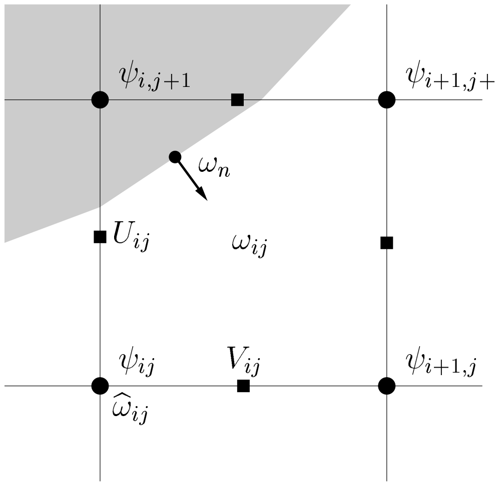


Figure 1.2: Discretization used in present algorithm for solving the streamfunction-vorticity equations.

CLAWPACK, and to handle the irregular geometry, we use capacity form differencing, also built into CLAWPACK. By modifying the capacity function in small triangular cells cut by the Cartesian grid, we avoid the small cell instability problem associated with explicit Cartesian grid methods and achieve stability for Courant numbers close to 1. This algorithm is described in Chapter 2. During the advection step, we impose a no-flux boundary condition on solid objects and appropriate inflow-outflow conditions on the computational domain.

The diffusion term in the transport equation is treated implicitly, also using a conservative finite volume update. In each partial cell, diffusive fluxes at partial cell edges are computed accurately using bilinear interpolation. Vorticity is diffused into the domain from solid boundaries and a no-flux condition is imposed on the computational domain. To determine the appropriate flux of vorticity needed to satisfy the no-slip boundary condition, we solve a Stokes flow problem at each time step. This problem is described by the coupled parabolic-elliptic system

$$\begin{aligned} \widehat{\omega}_t &= \nu \nabla^2 \widehat{\omega}, & \frac{\partial \widehat{\omega}}{\partial n} &= g \text{ on } \partial\Omega \\ \nabla^2 \psi &= -\widehat{\omega}, & & \\ \psi_n &= 0, \text{ on } \partial\Omega & \psi &= \text{constant on } \partial\Omega \end{aligned} \tag{1.3}$$

where both the streamfunction ψ and the vorticity $\widehat{\omega}$ are discretized at grid nodes, as shown in Figure 1.1. The boundary $\partial\Omega$ is the boundary of our irregular domain Ω. To obtain the right hand side for the discrete parabolic equation for $\widehat{\omega}$, we average values of the cell averaged ω, obtained from the advection step, onto grid nodes. The coupled parabolic-elliptic equations are discretized using the Immersed Interface Method, a second order finite difference method developed by LeVeque and Li [37, 38]. Once discretized, these equations can be set up as a linear system for the unknown flux g. This linear system, which depends on the time step Δt, is formed whenever the time step changes, and stored in factored form. The size of this matrix is quite small, and so storage is not a critical factor in deciding how best to solve the linear system. The linear system is solved with right hand sides computed at each time step. We use the flux g as the flux imposed during the diffusion step of the conservative fractional step scheme, and difference the streamfunction to obtain velocities at cell edges. The solution $\widehat{\omega}$ is discarded. In this manner, we satisfy the no-slip boundary condition at each time step. See Chapter 3 for details.

In summary, our method for solving the streamfunction-vorticity equations differs from other methods we are familiar with in a number of significant ways.

- Our irregular flow domain is embedded in a uniform Cartesian grid and the equations are then solved on that grid, using standard finite difference and finite volume methods, at least away from the boundary.
- The vorticity is represented as a cell-averaged conserved quantity, advected using a conservative high resolution algorithm. Flux limiters are applied to avoid numerical oscillations.
- We impose a flux condition on the vorticity at solid boundaries

The first item is significant because, with the exception of finite element methods, traditional methods for solving the streamfunction-vorticity equations require that the boundary be aligned with the computational grid. Most of these traditional methods impose Dirichlet conditions on vorticity at no-slip boundaries. However, such conditions do not adequately model the physical phenomenon of vorticity generation at solid boundaries, and so we have chosen instead to use a flux boundary condition on the vorticity. As we will see, this choice also fits conveniently into our finite volume algorithm. Finally, standard first order upwind methods for advection can lead to severe numerical diffusion and standard second order advection schemes lead to unphysical oscillations in the solution near discontinuities. Our use of high-resolution algorithms in CLAWPACK minimizes the effect of both of these undesirable behaviors of traditional finite difference advection schemes. This is especially important for solving the streamfunction-vorticity equations because of the singular nature of the vorticity distribution near rigid boundaries.

Other ways in which our algorithm differs from other more general Cartesian grid approaches to solving the Navier-Stokes equations is that we have overcome the small cell instability problem by modifying the capacity function. Also, we have adapted the Immersed Interface Method for use in solving the coupled Stokes flow problem, given in (1.3).

A Cartesian grid method that is most closely related to our approach for solving the viscous, incompressible Navier-Stokes equations is the Immersed Boundary Method now in popular use by researchers in computational biology [42]. Other finite difference and finite element methods on Cartesian grids with embedded boundaries have been developed for many different PDEs, for example [23], [25], [26], [39], [37], [36], [40], [41], [42], [48], [56], [58].

In the next section, we briefly describe some of the techniques that are in common use for solving viscous, incompressible flow in complex regions.

1.2 Other computational methods for solving viscous incompressible flow in geometrically complex regions

Cartesian grid methods for solving PDEs in irregular regions are a fairly recent development. Other methods with a longer history include the finite element method, boundary integral approaches, curvilinear grid transformations and vortex methods. Another more recent method is the lattice Boltzmann method. Below, we summarize the relative strengths and weaknesses of these methods as well as describe a few Cartesian grid methods commonly used in crystal growth applications and computational biology.

1.2.1 The Finite Element Method for engineering applications

In more traditional applications, such those that arise in solid mechanics and aeronautics, finite element methods (FEMs) on unstructured meshes are used almost exclusively. Because discrete spatial elements are aligned with the irregular boundary, important boundary effects can be well resolved. Moreover, if one is using an unstructured finite element mesh, the mesh can be easily refined in areas where details of the solution needs to be resolved. The accuracy of the finite element method can be improved by using higher order interpolating polynomials over the finite element, and in fact spectral accuracy can be achieved when one uses spectrally accurate Chebychev polynomials as the interpolating function over each element.

However, generating the meshes required by the FEM is as much an art as a science, and in fact has spawned an entire industry. Automatic grid generation is the topic of many books, journal articles and conferences, and while the automobile and aerospace industries are heavily invested in codes which use the FEM to solve complicated three dimensional flows around detailed computer models of vehicles, these codes are often proprietary and not generally available. Furthermore, to solve problems in moving domains, one must remesh every time the boundary moves. This can be computationally very expensive.

1.2.2 Simulating crystal growth

In the crystal growth community, researchers are interested in tracking the evolution of interfaces separating the liquid and solid phase of a pure substance or binary alloy. When the liquid is undercooled relative to the melting temperature of the solid, the interface grows unstably and dendritic shapes form, delineating a complicated region in which the PDEs governing the motion of the interface must be solved.

A common approach taken for solving these governing elliptic PDEs in the region exterior to the crystal is to use boundary integral or boundary element methods. These reduce the PDE for the field variable, usually temperature, to one involving data on the boundary of the crystal only. Typically, the velocity of the moving interface is related to the temperature gradient on the interface, and so one uses the boundary integral equation to obtain Neumann data, given Dirichlet data on the crystal surface. The advantage of the boundary integral approach is that arbitrarily complicated boundaries can be handled. Moreover, fast solvers, such as the fast Multipole method [28] can be used to dramatically improve computational costs of solving these PDEs. By reducing the PDE to the boundary, one never needs to worry about imposing boundary conditions on a fictitious computational domain. Exterior problems are solved as easily as interior problems.

The main drawback, however, is that boundary integral methods are typically used only in situations where a *quasi-steady-state* approximation applies [3, 10, 53] and are rarely used to solve a fully time-dependent problem. The mathematical theory behind boundary integral methods and related methods, known as boundary element methods, relies on sophisticated ideas from linear and functional analysis, and so applying the method to problems other than problems in potential theory requires some understanding of these areas of pure mathematics. Even in potential theory, standard techniques for handling the singular kernel are based on complex mathematical theory and must be implemented carefully.

Using boundary integral techniques, it is not straightforward to obtain the solution off of the interface, nor is it possible to use variable coefficients. It is interesting to note, however, that the first problem was recognized by Mayo [40]. Her approach to obtaining solutions off of the boundary is very similar to those ideas described by LeVeque and Li in their work on the Immersed Interface Method, which we describe in detail in Chapter 3. Also, the Immersed Interface Method can handle variable and discontinuous coefficients quite easily.

Researchers interested in handling fully time dependent problems have relied on *phase field* models. These models use what is called an *order parameter* ϕ to distinguish between the two phases of the material. The evolution of the order parameter is governed by an advection-diffusion equation, which must be solved along with the equation for the field variables. The phase field equations are usually solved on a uniform Cartesian mesh, and have been used quite successfully to model the evolution of crystals in both two and three spatial dimensions. However, in order to resolve the interface accurately and obtain the sharp interface limit, one must use a very fine mesh or adaptive mesh refinement near the interface [55].

1.2.3 Computational methods in biology : The Immersed Boundary Method

In biological applications, one method that is quite popular is the Immersed Boundary Method (IBM), developed by Charles Peskin [42]. This method, originally developed to model the flow of blood in the human heart, is particularly well suited for flows generated by the movement of flexible boundaries. In the case of the beating heart, the heart muscle relaxes and contracts, forcing the blood to flow through the heart. In another example, the IBM is used to model filament-like cilia distributed along the bodies of certain micro-organisms and which propel the micro-organism through the surrounding liquid. The Immersed Boundary Method has also been used to model bacterial swimming [21], biofilm in a porous medium [20] and the cochlea in the inner ear [7]. Recently, the IBM has been used as the basis for a software package called IBIS, which can be used to model a variety of biological phenomena similar to those just mentioned [22].

The flow in these applications is typically incompressible, viscous flow and so is governed by the full Navier-Stokes equations. Because these equations are challenging to solve in the simplest of geometries, the Immersed Boundary Method is especially attractive because one only needs to include the boundary effects as a source term. No special discretizations are needed near the interface, and in particular, one need not have detailed information about the geometry of the interface. The Navier-Stokes equations are discretized on a uniform Cartesian mesh and discretized delta functions whose strength is given by the force exerted by the flexible boundary, is included in the equations as a source term. These forces are then advected with the fluid and their location updated at each time step. Because the term only appears as a source term, the equations can be discretized on the uniform mesh using standard methods. Originally, the projection method was used to treat the incompressibility condition, although recently, other more direct methods have been adopted.

Because the Immersed Boundary Method relies on discrete delta functions, the accuracy of the method is limited to the accuracy of the approximations of these functions and their spacing along the boundary. Typically, this accuracy is only first order. LeVeque and Beyer [8] showed that in 1d, special delta functions could be used to obtain second order accuracy, although these results do not extend to 2d. An attempt to develop a second order Immersed Boundary Method was the basis for what is now known as the Immersed Interface Method [37].

Another drawback to the Immersed Boundary Method is that there is nothing to prevent fluid from passing between these singular forces, and hence through the irregular boundaries. This can be a problem in models in which the boundaries should be impermeable. Peskin and Printz, in [43], have developed methods for remedying this situation.

1.2.4 Curvilinear coordinate transformations

One technique that is often employed to handle geometry other than rectangular geometry is coordinate transformations and body fitted grids. Common transformations include cylindrical (in 2d) and spherical (in 3d). In aerodynamics, a popular transformation is the conformal map used to determine flow past a Joukowski aerofoil. To calculate flow past a circular cylinder and simulate the vortex shedding that leads to the well known von Karman street, researchers most often use the cylindrical coordinates with a logarithmic bunching of radial values near the surface of the cylinder. Variants of these methods include elliptic transformations, used to model flow around ellipses, and bipolar transformations, used to model flow past a pair of cylinders.

Such transformations have the obvious advantage that the grids conform to the irregular boundary. This is especially important when modeling thin boundary layers. Another advantage is that these curvilinear meshes are topologically equivalent to a Cartesian mesh, and so the equations of interest can be solved on the Cartesian mesh, taking into account additional geometrical terms, and then the solution transformed back to the curvilinear grid. The obvious disadvantage to these methods is that analytic transformations only exist for a limited number of geometries. To handle more complicated geometries, such as multiply-connected domains and irregular objects, one must resort to numerically computing the appropriate transformation, a task that may be quite challenging. Furthermore, the task of solving the PDE is made considerably more difficult by the presence of geometric terms appearing as a result of applying the transformation to the PDE.

1.2.5 Vortex methods

One popular class of methods for solving the incompressible Navier-Stokes equations in velocity-vorticity formulation is the class of vortex methods, popularized by the work of Chorin [14, 15]. See Gresho [29] for a review of work by other researchers in this field. The essential feature of a vortex method is the introduction of vorticity into the domain at solid boundaries in the form of vortex *sheets* or *blobs*. This vorticity serves to cancel out any slip velocity at the boundary. These sheets or blobs are then tracked in a Lagrangian manner as they follow the fluid flow. A Biot-Savart law is used to invert the relation $\nabla \times \vec{u} = \omega$ to get the velocity field in terms of the vorticity, and the vortex sheets or blobs participate in a random walk to simulate diffusion. From a physical point of view, they methods are appealing because they make explicit the intuitive idea that vorticity is generated at the boundary [15].

From a numerical point of view, these methods have the attractive feature that they are essentially explicit. No coupled system of equations is solved in order to determine the correct amount of vorticity to introduce at boundaries. Furthermore, they naturally handle complex geometry since there is no underlying grid. On the downside, the explicit nature of the scheme limits its usefulness to high Reynolds number flows. Diffusion dominated flows lead to a stiff system of equations which are better handled by implicit methods, such as the coupled streamfunction-vorticity equations we discuss in detail in Chapter 4 [6].

1.2.6 Lattice Boltzmann Method

A radically different approach to solving the fluid flow equations in complex geometries is taken by researchers using the lattice Boltzmann method (LBM). The lattice Boltzmann equations, based on microscopic models and mesoscopic kinetic equations, depart from the usual finite difference schemes in that they model macroscopic processes as the aggregate behavior of individual particles moving with velocities given by the lattice Boltzmann equations. The motion is determined so that at the macroscopic level, the particles model the desired behavior (advection,diffusion, etc). What makes the lattice Boltzmann method attractive for solving problems in complicated geometries is that non-reflecting (no-slip) boundary conditions are simple to apply. Particles simply bounce off of solid boundaries.

When tested on standard model problems in incompressible fluid dynamics, such as the driven cavity problem, or flow around a cylinder, the lattice Boltzmann method gives results that are in excellent agreement with numerical and experimental results found in the literature [12]. They are also applied to problems in porous media and shown to be quite robust [5]. For a detailed review of this method, see [12].

1.3 Outline of this thesis

As researchers using phase field models for crystal growth and the Immersed Boundary Method in biological applications have recognized, there are several advantages to using an underlying Cartesian grid on which to discretize the governing PDEs. First, since the underlying mesh is uniform, one can avoid the mesh generation difficulties associated with body-fitted finite difference grids or unstructured finite element meshes. Second, one can use a standard finite difference type discretization of the PDE away from the boundary of the irregular region. This may be especially important when including fluid flow in the problem.

The drawback to embedding the irregular flow domain into a Cartesian grid is that in general, the boundary of the complicated region or interface will not align with grid lines and so one must use special discretizations of the PDE near the interface in order to obtain algorithms which are both accurate and stable. In the following chapters, we demonstrate, in both the finite volume and finite difference setting, methods for handling these irregular cells cut by the grid. The main theme throughout this work will be solving the streamfunction-vorticity equations in complex regions.

In Chapter 2, we describe an algorithm for solving the advection-diffusion equation, given by

$$\kappa q_t + \nabla \cdot (\vec{u} q) = \nabla \cdot (D \nabla q) \tag{1.4}$$

where q is a density or concentration, $\vec{u}(x, y)$ is the specified velocity field, $D(x, y)$ is the diffusion coefficient, and $\kappa(x, y)$ is a capacity function. We obtain the velocity field by differencing a streamfunction, provided as data for the problem and show that even near irregular boundaries, this form of streamfunction differencing gives us an appropriate velocity. We also demonstrate how capacity form differencing, which is implemented in the CLAWPACK package, can be used to treat partial finite volume cells. We describe a stable algorithm for handling the small cells that are cut by the irregular boundary and a conservative scheme for handling the diffusion term. Several examples are given which show that the accuracy of the method is between 1.2 and 2, depending on the geometry and Peclet number. This chapter is excerpted from the article by Calhoun and LeVeque [11].

In Chapter 3, we describe in detail a method for solving constant coefficient elliptic and parabolic boundary value problems in embedded domains. We base our algorithm on LeVeque and Li's Immersed Interface Method, and on Yang's work in extending these ideas to boundary values problems [58]. Our approach differs in some minor ways from the approach taken by Yang, and is more similar to the Explicit Jump Immersed Interface Method, developed by Wiegmann [56, 57]. Like Wiegmann, we add jumps explicitly to the discretized PDE, but have introduced a general notation that works equally well in one, two and three dimensions. We outline our derivation which leads to general formulas for the discretization of the PDE and then verify the Immersed Interface Method on a variety of problems which result from simplifying the streamfunction-vorticity equations. In particular, we look at the potential flow problems for steady, irrotational flow

$$\nabla^2 \psi = 0 \tag{1.5}$$

in singly and multiplied-connected domains and verify our results with known solutions. In Chapter 3, we also look at solving the constant coefficient parabolic equation

$$u_t = \nu \nabla^2 u \tag{1.6}$$

subject to flux boundary conditions on an irregular boundary. For both the elliptic and parabolic problems, we confirm that the Immersed Interface Method gives us second order accuracy. Finally, we look at solving the Stoke's flow problem given in 1.3. To solve this problem, we must solve a coupled elliptic-parabolic problem. We compare our results for this problem with results found in the literature.

In Chapter 4, we describe in detail how we can couple the work of the first two chapters to solve the full streamfunction vorticity equations, given in (1.1). After surveying the wide variety of literature on the subject of how best to implement the no-slip boundary condition, we describe our method based on the Immersed Interface Method. In our method, the only additional work that is required to create vorticity at solid boundaries is the inversion of a matrix whose size depends on the number of boundary points used to represent the interface. We test our results on Reynolds numbers 20, 40, 50 and 100 and show that our results compare quite favorably with other results published in the literature. In particular, we demonstrate that for Reynolds number $Re < 40$, the wake behind the cylinder remains closed and stable, and that around $Re = 50$ we begin to see the vertical asymmetries in the vortex wake behind the cylinder and in the streamlines. At $Re = 100$, we demonstrate well developed vortex shedding and the von Karman vortex street. These results are consistent with other published results on this problem. For the subcritical Reynolds numbers, we compare the geometry of our wake with experimental results and show that the results are in good agreement. For the supercritical Reynolds numbers, we compare the frequency of shedding with those reported and also find that we are simulating the appropriate behavior.

As further verification of our code, we solve the streamfunction vorticity equations in spatially periodic, multiply connected-domains and compare our results computed on different grids. From this comparison, we conclude that even on the coarsest grids, our scheme is capturing the basic behavior of the flow. In one case, we are able to compare our results with those found in the literature, and show that we get good agreement.

Chapter 2

A CARTESIAN GRID FINITE-VOLUME METHOD FOR THE ADVECTION-DIFFUSION EQUATION IN IRREGULAR GEOMETRIES

The ultimate goal of this thesis is to describe a complete algorithm for solving the streamfunction-vorticity equations in irregular regions. We can split these equations up into three natural types of linear PDEs: a hyperbolic equation which governs the transport of vorticity, a parabolic equation which models the diffusion of vorticity, and an elliptic equation whose solution gives us the stream-lines of the flow. The goal of this chapter is to describe in detail how we solve the PDE governing the transport of vorticity. This equation, which is given by the advection-diffusion equation, is

$$\omega_t + (\vec{u} \cdot \nabla)\omega = \nu \nabla^2 \omega. \tag{2.1}$$

In 2d, the vorticity is a conserved quantity, and a conservative finite volume method is an appropriate scheme for this equation. In this chapter, we concentrate on solving advection-diffusion equations of this type and in particular, show how capacity form differencing, built into the high resolution algorithms in CLAWPACK, can be used to solve this equation in irregular boundaries.

This chapter is based on the paper by LeVeque and Calhoun [11].

2.1 *Introduction*

We consider the advection-diffusion equation

$$\kappa q_t + \nabla \cdot (\vec{u} q) = \nabla \cdot (D \nabla q) \tag{2.2}$$

in two space dimensions, where $q(x, y, t)$ is a density or concentration, $\vec{u}(x, y, t)$ is the specified velocity field, $D(x, y)$ is the diffusion coefficient, and $\kappa(x, y)$ is a capacity function. This function could represent, for example, heat capacity if q is the temperature, or porosity in a porous medium if q is the concentration of a transported solute. A fundamental feature of the numerical method developed here is the use of a discrete version of κ which also incorporates information about the fraction of a computational grid cell which lies in the physical domain. This is used in order to apply the method on Cartesian grids with embedded boundaries. Our goal is to solve (2.2) in geometrically complex regions with impermeable boundaries where the fluid flow must be tangent to the physical domain and the flux of q is specified (e.g. , no flux). Rather than attempting to determine a grid which conforms to the boundaries, we wish to use a uniform Cartesian grid in which the physical domain is embedded. The main advantages to using a Cartesian grid are that all difficulties associated with unstructured grid generation can be avoided, and standard finite volume or finite difference discretizations of the PDE on a uniform grid can be used, away from the boundaries at least.

Here we develop a finite volume method which is essentially second order accurate for smooth q in very general geometries and which can also handle steep gradients in q (even discontinuities in q if $D \equiv 0$). This is accomplished by using the high-resolution shock-capturing methods of CLAWPACK for the advection portion of the method, coupled with an implicit finite-volume discretization of the diffusion equation.

The principle contribution of this chapter is an algorithm with the following characteristics:

- Arbitrary geometry can be handled on a Cartesian grid, though we assume that the geometry is well resolved by a piecewise linear function on the underlying grid.

- After some preprocessing required to handle the irregular geometry, small irregular cells which are cut off by the boundary are updated by exactly the same formulas as regular cells away from the boundary.

- The advection portion of the algorithm is explicit and the time step is determined by the size of the regular grid cells, with no restriction caused by the small irregular cells.

- The accuracy and resolution in the irregular cells adjacent to the boundaries matches that obtained in the regular cells, and is essentially as good as would be obtained on similar problems with a grid conforming to the boundary.

- The algorithm is implemented using the CLAWPACK software and is freely available on the web, see
  `http://www.amath.washington.edu/~rjl/clawpack/cartesian`

By assuming that the geometry is well resolved on the grid, we assume in particular that each finite volume cell is a standard Cartesian grid cell which is possibly sliced by a linear approximation to the boundary, as shown in Figure 2.5, for example. Hence every cell is a simple polygon with at most five sides and the standard (i, j) labeling of grid cells can continue to be used. For geometry with more complications, for example a thin trailing edge which cuts a cell into two distinct pieces, a more complicated data structure would be required. The methods proposed here could still be used but could not be implemented as directly in CLAWPACK. See [18] for one discussion of such geometries in the context of elliptic equations.

The diffusion and advection algorithms will first be described and tested separately, and are then combined using a standard fractional step approach. The methods are described in two space dimensions, but all of the essential ideas can be extended directly to three space dimensions.

2.2 Notation and the use of a capacity function

We consider finite volume methods for the general conservation law

$$\kappa(x,y)q_t(x,y,t) + f(x,y,t,q)_x + g(x,y,t,q)_y = 0 \tag{2.3}$$

where f and g are the flux functions and κ is the capacity. In particular, we consider variable-coefficient linear equations including the *diffusion equation* in which

$$f = -D(x,y)\,q_x, \qquad g = -D(x,y)\,q_y \tag{2.4}$$

and the advection equation in which

$$f = u(x,y,t)\,q, \qquad g = v(x,y,t)\,q \tag{2.5}$$

and finally the advection diffusion equation which includes both advective and diffusive fluxes,

$$f = u(x,y,t)\,q - D(x,y)\,q_x, \qquad g = v(x,y,t)\,q - D(x,y)\,q_y. \tag{2.6}$$

Such equations arise from the integral form of the conservation law over an arbitrary region Ω

$$\frac{\partial}{\partial t}\int_\Omega \kappa(x,y)\; q(x,y,t)\; dx\, dy + \int_{\partial\Omega} \vec{n}\cdot\vec{f} ds = 0 \tag{2.7}$$

where $\vec{n}$ is the outward pointing normal at the boundary $\partial\Omega$ and $\vec{f} = (f,g)$ is the flux vector. The integral of κq is conserved up to fluxes through the boundary. The function κ represents the capacity of the medium as illustrated in the following two examples :

1. In heat conduction, if q is the temperature then (2.4) gives Fourier's Law of heat conduction for the flux of heat energy. The energy stored in the region Ω is $\int_\Omega \kappa q\; dx\, dy$ where κ is the heat capacity of the material and (2.7) arises from conservation of energy.

2. If q represents the concentration of a tracer in some fluid saturating a porous medium, and if $\kappa(x, y)$ is the porosity of the medium so that $\int_\Omega \kappa(x, y)\, dx\, dy$ is the volume of Ω which is available to the fluid, then $\int_\Omega \kappa q\, dx\, dy$ is the total mass of the tracer in the region Ω. If the fluid is flowing through the porous medium with the Darcy velocity (u, v) (defined below) then the conservation law (2.7) arises with fluxes (2.5). If the tracer also diffuses in the fluid then diffusive fluxes are also present.

Note that the integral $\int_\Omega \kappa q\, dx\, dy$ can also be interpreted as an integral of q over Ω with respect to the measure $d\mu = \kappa\, dx\, dy$. This interpretation may be useful in understanding our Cartesian grid finite volume integrals.

2.2.1 Incorporating the capacity function into finite volume methods on regular grids

A finite volume method is based on dividing the physical region into finite volumes (i.e. grid cells) and representing the numerical approximation by cell averages over those cells. The cell average is updated in each time step by approximations to the flux through each edge of the cell. We are concerned here with a Cartesian grid of the form shown in Figure 2.1. We start by considering a uniform grid with spacing Δx and Δy on a rectangular region, with no irregular geometry to handle.

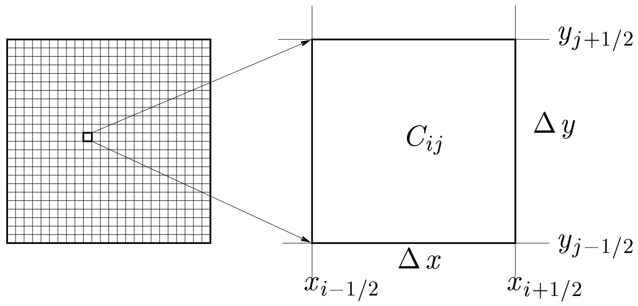


Figure 2.1: Cartesian grid with cell C_{ij} shown in enlargement

The grid is indexed by (i, j) and Q_{ij}^n represents the approximation to the average values of q on cell (i, j) at time t_n with respect to the measure $\kappa\, dx\, dy$:

$$Q_{ij}^n \approx \frac{\int_{C_{ij}} \kappa(x, y)\, q(x, y, t_n)\, dx\, dy}{\int_{C_{ij}} \kappa(x, y)\, dx\, dy}. \tag{2.8}$$

Using the fact that the area of cell C_{ij} is $\Delta x\ \Delta y$, we define the average value of κ over cell C_{ij} as

$$\kappa_{ij} = \frac{1}{\Delta x\ \Delta y} \int_{C_{ij}} \kappa(x, y)\, dx\, dy \tag{2.9}$$

Then we can write (2.8) as

$$Q_{ij}^n \approx \frac{1}{\kappa_{ij}\Delta x\ \Delta y}\int_{C_{ij}} \kappa q\ dx\, dy \tag{2.10}$$

Note that if $\kappa \equiv 1$, then $\kappa_{ij} \equiv 1$ and (2.10) reduces to the standard cell average.

Let $F_{i-1/2,j}$ and $G_{i,j-1/2}$ represent the numerical fluxes at the left edge and bottom edge of the cell, respectively. These fluxes will be measured in units of q flowing normal to the edge per unit time per unit length. Hence the total flux through the left edge over the time step Δt, for example, is given by $\Delta t\ \Delta y\ F_{i-1/2,j}$. In porous media problems these fluxes are computed using the *Darcy* velocity and in heat transfer problems, they are computed using an *effective conductivity.* We explain both of these briefly now.

Figure 2.2 shows a cartoon of flow in a porous medium where the shaded regions represent rock or sand. The fluid velocity may be large in the regions where fluid is flowing, but it is zero in solid regions where there is no fluid. Hence the average horizontal velocity across a vertical line segment such as the one sketched in the figure is less than the fluid velocity at any particular point in the fluid. The u-component of the *Darcy velocity* at a point is this average velocity taken over an "infinitesimal" vertical segment centered at the point, which is short relative to the scale at which this average fluid velocity varies, but still long relative to the structure of the porous medium. Hence the Darcy velocity is roughly equal to the average fluid velocity multiplied by the fraction of the infinitesimal vertical line which lies within the fluid.

At the edge of a finite volume grid cell (which is assumed to be large relative to the scale of the porous medium), it is clearly the Darcy velocity and corresponding flux which determines the flux into the grid cell.

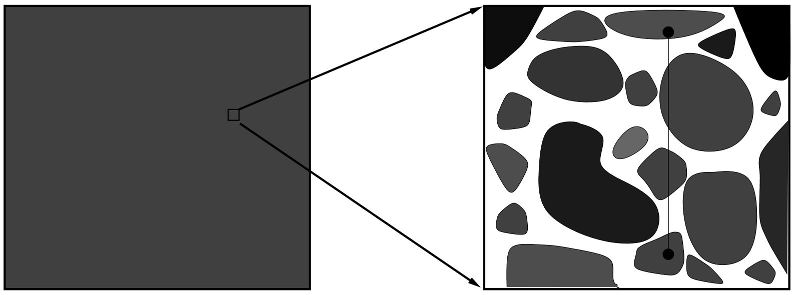


Figure 2.2: Example of porous medium. Quantities such as velocity are averaged over an infinitesimal line segment, such as the one shown.

The porosity of a porous medium at a point is defined analogously as the fraction of an "infinitesimal" region Ω about the point which is occupied by the fluid, where

Ω is again large relative to the porous structure. Our approach to handling Cartesian grids in complex geometries, which is described below, can be viewed as an extension of these ideas to the case where the solid is concentrated on some portion of the grid cell lying outside the fluid domain, rather than being distributed throughout the grid cell. As we show later, we can also define appropriate fluxes across edges which are only partially in the physical domain of interest.

In heat flow in composite materials, one can analogously define an *effective conductivity* as the average of the conductivity over an "infinitesmal" line segment which is short relative to the length over which the average conductivity varies, but long relative to any inhomogeneities in the composite material.

Using numerical fluxes based on Darcy velocities and effective conductivities, an explicit discretization of the conservation law (2.7) is given by

$$\frac{\kappa_{ij}\Delta x\,\Delta y\,Q_{ij}^{n+1} - \kappa_{ij}\Delta x\,\Delta y\,Q_{ij}^{n}}{\Delta t} + \Delta y\left[F_{i+1/2,j}^{n} - F_{i-1/2,j}^{n}\right] + \Delta x\left[G_{i,j+1/2}^{n} - G_{i,j-1/2}^{n}\right] = 0 \tag{2.11}$$

Rearranging terms, we can write an explicit formula for updating Q_{ij}^{n+1} as

$$Q_{ij}^{n+1} = Q_{ij}^{n} - \frac{\Delta t}{\kappa_{ij}\Delta x}\left[F_{i+1/2,j}^{n} - F_{i-1/2,j}^{n}\right] - \frac{\Delta t}{\kappa_{ij}\Delta y}\left[G_{i,j+1/2}^{n} - G_{i,j-1/2}^{n}\right] \tag{2.12}$$

Note that (2.12) is in conservation form in the sense that

$$\Delta x\ \Delta y\ \sum_{ij}\kappa_{ij}Q_{ij}^{n} \tag{2.13}$$

will be conserved up to fluxes through the boundary of the computational domain. This type of "capacity form" differencing is discussed in more detail in the context of hyperbolic conservation laws in [35]. In many problems $\kappa(x,y) \equiv 1$. However, even for these problems the idea of introducing a capacity function becomes useful in handling complex geometries, as described in the next section.

2.2.2 Using the capacity function to represent irregular regions

In addition to physical interpretations of heat capacity and porosity, we can use the capacity function κ to embed an *irregular flow domain*, such as the one shown in Figure 2.3, into a Cartesian computational domain. The natural extension of the interpretation of κ given in the preceding section is to define κ to be identically 0 everywhere outside of the irregular flow domain. This extension preserves the notion that the values of κ determine the *capacity* of the medium to hold the advected or diffused quantity q. Outside of the physical domain of interest, the medium has no capacity to hold q.

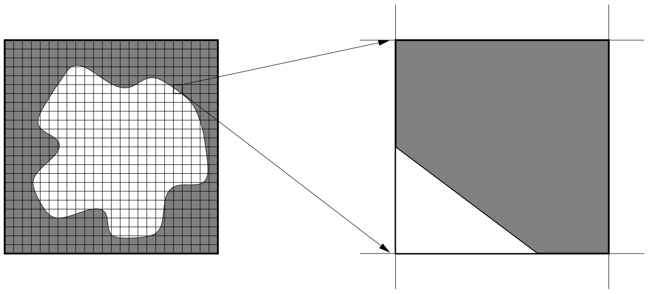


Figure 2.3: Irregular flow domain (unshaded region) embedded in a larger computational domain. Enlargement shows an irregular grid cell cut by a piecewise linear boundary. The capacity of the irregular cell is the unshaded fraction of the cell in the irregular flow domain.

We now view the computational domain as the full rectangular region in Figure 2.3, and refer to the original physical domain as the *flow domain* and to the portion which lies outside the original domain (the shaded region in Figure 2.3) as the *no-flow domain*.

With this extended interpretation in mind, we now consider an irregular grid cell such as the one magnified in Figure 2.3. We define κ_{ij} for this cell as in (2.9), after extending $\kappa(x, y)$ to be 0 in the no-flow domain. Note that $\kappa_{ij}\Delta x\Delta y$ gives the total capacity of the grid cell. In particular, in the simplest case where $\kappa(x, y) \equiv 1$ in the flow domain, $\kappa_{ij}\Delta x\Delta y$ is simply the area of the portion of this grid cell which lies in the flow domain (the unshaded portion).

To obtain a definition of the numerical fluxes $F_{i-1/2,j}$ and $G_{i,j-1/2}$ which is consistent with our previous use of these quantities in update formula (2.12), we first let $\hat{F}_{i-1/2,j}$ and $\hat{G}_{i,j-1/2}$ denote the average flux across that portion of each edge which is in the flow domain, as indicated in Figure 2.4. Then we let $h_{i-1/2,j}$ and $h_{i,j-1/2}$ be the lengths of these edge portions to the left and below the cell C_{ij} respectively. One or more of these may be zero. The total flux across each edge is then given by $h_{i-1/2,j}\,\hat{F}_{i-1/2,j}$ and $h_{i,j-1/2}\,\hat{G}_{i,j-1/2}$.

If we now define the fluxes

$$\begin{aligned} F_{i-1/2,j} &= \frac{h_{i-1/2,j}}{\Delta y}\hat{F}_{i-1/2,j} \\ G_{i,j-1/2} &= \frac{h_{i,j-1/2}}{\Delta x}\hat{G}_{i,j-1/2} \end{aligned} \tag{2.14}$$

we see that $\Delta y\, F_{i-1/2,j}$ and $\Delta x\, G_{i,j-1/2}$ are the total fluxes across the left edge and bottom edge, respectively, of cell C_{ij}. Hence, this definition of the fluxes on irregular cells is consistent with our definition on the regular cells.

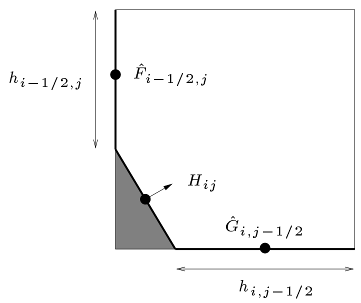


Figure 2.4: Fluxes at edges of irregular cells

To develop a general updating formula for irregular cells, we also introduce a flux H_{ij} across the boundary segment, as indicated in Figure 2.4. Let the length of this boundary segment be denoted by ℓ_{ij}. Then the general updating formula is

$$\begin{aligned} Q_{ij}^{n+1} = Q_{ij}^{n} - \frac{\Delta t}{\kappa_{ij}\Delta x}\left[F_{i+1/2,j} - F_{i-1/2,j}\right] - \frac{\Delta t}{\kappa_{ij}\Delta y}\left[G_{i,j+1/2} - G_{i,j-1/2}\right] \\ - \frac{\Delta t}{\kappa_{ij}\Delta x\,\Delta y}\left[\ell_{ij}H_{ij}\right]. \end{aligned} \tag{2.15}$$

For many problems $H_{ij} \equiv 0$. This is the correct boundary condition for tracer concentration in a flow bounded by a wall and also for heat conduction if the wall is insulated. If $H_{ij} = 0$, the update formula (2.15) reduces to exactly that given by (2.12).

For cells completely outside of the flow domain, we have $\kappa_{ij} = 0$ and the updating formula (2.12) appears to be invalid. However, if we note that fluxes $F_{i+1/2,j}$, $F_{i-1/2,j}$, $G_{i,j+1/2}$ and $G_{i,j-1/2}$ will always be zero for such cells, the earlier formula (2.11) will always be valid. For computational convenience, however, we want to apply the formula (2.12) or (2.15) everywhere. To do this, we can set κ_{ij} to some arbitrary but non-zero value for cells entirely in the no-flow region, since the fact that the fluxes in this region will be zero will ensure that the value of Q_{ij}^n remains zero.

A key feature of our approach is that we have eliminated irregular grid cells from the geometric structure of our problem, and instead take the viewpoint that some cells may have small capacity. This simplifies the structure of the computer programs and allows software packages such as CLAWPACK to be applied directly.

As we will later see, by shifting the small cell problem from one of cells with small volume to one of cells with small capacity, we also provide ourselves with a natural way to overcome the instability problems typically associated with the small cells. Although our capacity κ_{ij} is the same as the quantity Λ_{ij} introduced by Chern and Colella [13] in the context of front tracking, we use a different modification to the fluxes which is based more directly on wave propagation and which we believe gives better accuracy.

2.3 The diffusion equation

We now develop a finite volume method for the diffusion equation in the flow domain. The problem we wish to solve numerically is given by

$$\kappa q_t = \nabla \cdot (D(x,y)\,\nabla q) \qquad \text{in } \Omega \tag{2.16}$$

subject to Neumann boundary conditions on the boundary of Ω which we use to define flux values H_{ij} used in equation (2.15). As described in the previous section, we define κ to be identically 0 for (x,y) in the no-flow domain. The fluxes are given by equations (2.4).

As suggested in the previous discussion, it will be convenient to define numerical fluxes $F_{i-1/2,j}$ and $G_{i,j-1/2}$ so that the total flux across an edge of length Δx or Δy is given by $\Delta y\, F_{i-1/2,j}$ or $\Delta x\, G_{i,j-1/2}$. With this in mind, we approximate quantities q_x and q_y on regular cell edges by differencing across the edge and use these to approximate the fluxes as

$$F_{i-1/2,j} = -D_{i-1/2,j}\frac{Q_{ij} - Q_{i-1,j}}{\Delta x} \tag{2.17}$$

and

$$G_{i,j-1/2} = -D_{i,j-1/2}\frac{Q_{ij} - Q_{i,j-1}}{\Delta y}. \tag{2.18}$$

As before, these values are in units of q flowing normal to a cell edge per unit time per unit length.

On irregular cell edges, we approximate quantities $\hat{q}_x$ and $\hat{q}_y$ at the center of that portion of the edge which is in the flow domain by interpolating between values $Q_k, k = 1, \ldots 4$ in four neighboring cells. We interpret these values Q_k as being located at centers of mass of the regular and irregular cells. After we have located the centers of four cells surrounding the partial edge at which we wish to compute the flux, and we use the coordinates $(\xi_k, \eta_k), k = 1, \ldots 4$ of these four centers and corresponding values Q_k to determine the coefficients a, b, c, d of the bilinear function

$$\hat{q}(x,y) = ax + by + cxy + d. \tag{2.19}$$

We then use these coefficients to approximate $\hat{q}_x$ or $\hat{q}_y$ at the center (x_e, y_e) of a partial edge. For example, we have

$$\hat{q}_x(x_e, y_e) \approx a + cy_e. \tag{2.20}$$

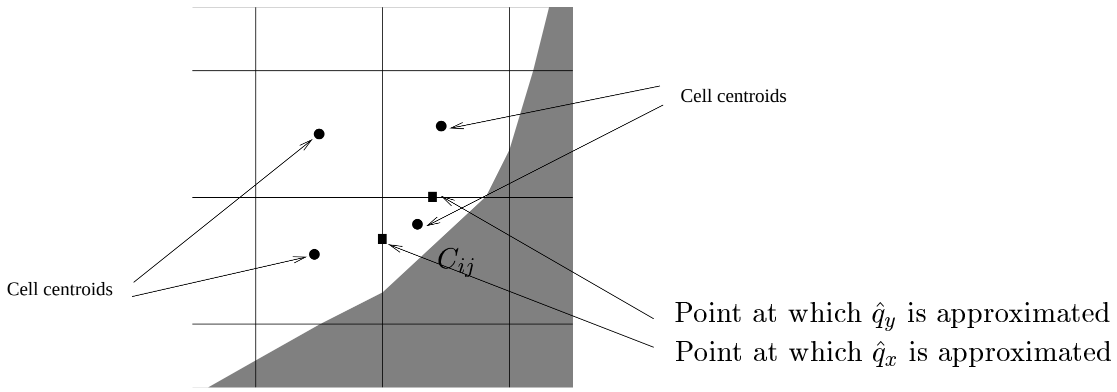


Figure 2.5: Typical set of points, located at center of mass of irregular cells, used to approximate $\hat{q}_x$ at edge $x_{i-1/2}$ and $\hat{q}_y$ at $y_{j+1/2}$. The fluxes $\hat{F}_{i-1/2,j}$ and $\hat{G}_{i,j+1/2}$ are computed using these approximations.

In Figure 2.5, the values Q_{ij}, $Q_{i-1,j}$, $Q_{i-1,j+1}$ and $Q_{i,j+1}$ are used to approximate values $\hat{q}_x$ at $x_{i-1/2}$ and $\hat{q}_y$ at $y_{j+1/2}$.
Other researchers have proposed methods for approximating diffusive fluxes at partial cell edges to second order accuracy [32]. Although our method is only formally first order at the cell edges, we have found that we obtain results which are still globally second order.

We use the approximations given by (2.19) to compute values $\hat{F}_{i-1/2,j}$ and $\hat{G}_{i,j+1/2}$ using

$$\begin{aligned} \hat{F}_{i-1/2,j} &= -\, D_{i-1/2,j}\, \hat{q}_x \\ \hat{G}_{i,j+1/2} &= -\, D_{i,j+1/2}\, \hat{q}_y \end{aligned} \tag{2.21}$$

The numerical fluxes $F_{i-1/2,j}$ and $G_{i,j+1/2}$ are then defined as in (2.14).

To discretize the heat equation given by (2.16), we use a standard Crank-Nicolson method at all grid points. First, we define an approximation to the integral

$$-\int_{\partial C_{ij}} \vec{f} \cdot \vec{n}\, ds. \tag{2.22}$$

in (2.7) as

$$W_{ij}^n = -\left\{ \Delta y \left(F_{i+1/2,j}^n - F_{i-1/2,j}^n \right) + \Delta x \left(G_{i,j+1/2}^n - G_{i,j-1/2}^n \right) \right\} \tag{2.23}$$

Then an implicit discretization of (2.7) is given by

$$\kappa_{ij} \Delta x\, \Delta y\, \frac{Q_{ij}^{n+1} - Q_{ij}^n}{\Delta t} = \frac{1}{2} \left(W_{ij}^n + W_{ij}^{n+1} \right) \tag{2.24}$$

which, upon rearrangement, is

$$Q_{ij}^{n+1} - \frac{\Delta t}{2\,\kappa_{ij}\,\Delta x\,\Delta y} W_{ij}^{n+1} = Q_{ij}^{n} + \frac{\Delta t}{2\,\kappa_{ij}\,\Delta x\,\Delta y} W_{ij}^{n}. \tag{2.25}$$

In the above, we assumed that there was no incoming flux from the boundary. If we assume that this flux, given again by H_{ij}, is non-zero and possibly time dependent, then our update formula looks like

$$Q_{ij}^{n+1} - \frac{\Delta t}{2\,\kappa_{ij}\,\Delta x\,\Delta y} W_{ij}^{n+1} = Q_{ij}^{n} + \frac{\Delta t}{2\,\kappa_{ij}\,\Delta x\,\Delta y} W_{ij}^{n} + \frac{\Delta t\,\ell_{ij}}{\kappa_{ij}\,\Delta x\,\Delta y} H_{ij}^{n+1/2} \tag{2.26}$$

where again ℓ_{ij} is the length of that portion of the boundary of the irregular flow domain which is in cell C_{ij}.

Because κ_{ij} varies from grid cell to grid cell, the matrix corresponding to the above system of linear equation is not symmetric. However, these equations are still easily solved using iterative method designed for such systems. The one which was chosen here and which seems to give quite good results is the stabilized bi-conjugate gradient method (BiCGSTAB) developed by Van der Vorst [19].

As for the stability of the numerical scheme described above, we have observed that in practice, the presence of small cells near the boundary does not place any unreasonable restrictions on the size of the time step we can take to maintain stability. This to be expected, since Crank-Nicolson is an unconditionally stable scheme.

2.4 The advection equation

The advection portion of the algorithm is implemented using the CLAWPACK software, based on the high-resolution wave-propagation algorithm described in [35]. The special case of advection is discussed in detail in [34] and will be described briefly below.

On the grid cell (i, j) we require the average normal velocity $U_{i-1/2,j}$ along the left edge of the cell and $V_{i,j-1/2}$ along the bottom edge. For simplicity in two dimensions we assume that an incompressible velocity field is specified by means of a stream function $\psi(x, y)$. This is not necessary for our algorithm and would not be appropriate in three dimensions, where a different approach would be needed to determine the proper Darcy velocities. The velocity is determined from the stream function by

$$u(x, y) = \psi_y(x, y), \qquad v(x, y) = -\psi_x(x, y).$$

Then the average velocity along each edge is easily computed by differencing the stream function at the corners:

$$\begin{aligned} U_{i-1/2,j} =& \frac{1}{\Delta y} \int_{y_{j-1/2}}^{y_{j+1/2}} \psi_y(x, y)\, dy \\ =& \frac{1}{\Delta y} [\psi(x_{i-1/2}, y_{j+1/2}) - \psi(x_{i-1/2}, y_{j-1/2})] \end{aligned} \tag{2.27}$$

at the left edge, and

$$\begin{aligned} V_{i,j-1/2} &= -\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \psi_x(x,y)\,dy \\ &= -\frac{1}{\Delta x}[\psi(x_{i+1/2},y_{j-1/2}) - \psi(x_{i-1/2},y_{j-1/2})] \end{aligned} \tag{2.28}$$

at the bottom edge. Note that this will automatically satisfy the discrete divergence-free condition

$$(U_{i+1/2,j} - U_{i-1/2,j})/\Delta x + (V_{i,j+1/2} - V_{i,j-1/2})/\Delta y = 0.$$

In the case of an irregular flow domain, ψ must be constant along any physical boundary and we assume that ψ is defined over the entire computational domain in such a way that ψ is identically constant in no-flow regions. Then the above formulas give zero average velocity along any cell edge that lies entirely in the no-flow domain. Moreover, the average velocity is computed correctly in cases where the edge is cut by the physical boundary, e.g. , at the left and bottom edges of the cell shown in Figure 2.3. This average velocity can be interpreted as analogous to the Darcy velocity described in Section 2.4. It is an effective velocity over the entire cell edge, rather than a pointwise approximation to the actual fluid velocity.

Using an upwind method, we can use these velocities to obtain the fluxes at cell edges. The discrete equations for the advection are then given by (2.12). The presence of κ_{ij} in the denominator suggests that a direct application of this set of explicit equations could lead to instability problems if κ_{ij} is very small. However, in [11], it is shown that it is possible to modify κ in such as way as to prevent such stability problems. We describe this modification here, although we refer the reader to our paper for complete details.

The change is accomplished by defining the capacity in all cells by a modified version of (2.9):

$$\hat{\kappa}_{ij} = \frac{r_{ij}}{\Delta x\ \Delta y}\int_{C_{ij}} \kappa(x,y)\ dx\,dy \tag{2.29}$$

in which we define

$$r_{ij} = \min\left(\frac{\Delta x}{h_{i-1/2,j}},\ \frac{\Delta x}{h_{i+1/2,j}},\ \frac{\Delta y}{h_{i,j-1/2}},\ \frac{\Delta y}{h_{i,j+1/2}}\right). \tag{2.30}$$

Note that $r_{ij} \geq 1$ and that $r_{ij} = 1$ if at least one of the cell edges is entirely in the flow domain, in which case $\hat{\kappa}_{ij}$ reduces to the usual definition of κ_{ij}. In the troublesome case of a small triangular cell, $r_{ij} > 1$ and the capacity is increased. This is equivalent to redefining the area of such cells based on a thin sliver rather than a small triangle, the sliver obtained by replacing the larger of the two lengths ($h_{i-1/2,j}$ or $h_{i,j+1/2}$, say) by the full size mesh width (Δy or Δx respectively). However, we do not actually modify the geometry of the grid at all; we simply increase the capacity of the small cell.

2.5 Fractional Step Methods for the Advection-Diffusion Equation

In the previous two sections, we have described in detail our algorithms for handling the advection and diffusion terms separately. These can be combined using standard fractional step methods to solve the coupled advection-diffusion equation. The advantage of using a fractional step approach is that we can still essentially treat the advection and diffusion operators separately. We only need to justify that the time accuracy of the solution obtained is acceptable.

In the fractional step approach, we first advect the tracer quantity Q^n to obtain an intermediate value Q^*. We then diffuse Q^* to obtain our approximate solution Q^{n+1}. The semi-discrete equations that we solve have the basic form

$$\begin{aligned} \frac{Q^* - Q^n}{\Delta t} &= -(\vec{u} \cdot \nabla) Q^n \\ \frac{Q^{n+1} - Q^*}{\Delta t} &= D\nabla^2 \left(\frac{Q^* + Q^{n+1}}{2} \right) \end{aligned} \tag{2.31}$$

where the right hand side of each discrete equation is discretized as described in the previous sections. This is sometimes called the *Godunov* splitting, and while it is formally only first order accurate, we have found in practice that we can obtain much better than first order accuracy. To see why this might be so, we compare the Godunov splitting to the *Strang* splitting method, a method which is formally second order accurate. We write down the Strang splitting, using the notation from above, as

$$\begin{aligned} \frac{Q^* - Q^n}{\Delta t/2} &= -(\vec{u} \cdot \nabla) Q^n \\ \frac{Q^{**} - Q^*}{\Delta t} &= D\nabla^2 \left(\frac{Q^* + Q^{**}}{2} \right) \\ \frac{Q^{n+1} - Q^{**}}{\Delta t/2} &= -(\vec{u} \cdot \nabla) Q^{**}. \end{aligned} \tag{2.32}$$

In the Strang splitting, the half time step of advection is followed immediately (in the next time step) by another half time step using the same operator. These two half steps can be combined into a single step of length Δt and in fact it is better to do so to reduce numerical diffusion and computational cost. Once this is done, the Strang splitting over many time steps is identical to the Godunov splitting except in the first and last time step, where a half step of advection is used rather than taking a full step only at the beginning. This change is typically negligible relative to other errors. Even though this error is formally first-order, it has a very small constant relative to the overall error. For this reason we use the simpler Godunov splitting, but this could easily be changed in the implementation.

2.6 *Numerical Results*

In the following numerical examples, we demonstrate the performance of our proposed algorithm. We seek to demonstrate that using our capacity form differencing, we can obtain results that have the right physical behavior near irregular boundaries and that converge to the correct solution as the computational grid is refined. In the first set of examples, we look at advection and diffusion of a plane wave in a channel and compare the results to the exact solution. In the second set of examples, we test the diffusion and solid body rotation in an annulus and compare our finite volume solution to a reference solution computed in polar coordinates. In the final set of examples, we look at flow through a field of irregular objects and compute washout curves for the advection of a passive tracer through the field.

In all sets of examples, we test the advection algorithm using velocities computed from a stream function $\psi(x, y)$ which has been determined either analytically or numerically. Before the actual computations using CLAWPACK are done, the stream function is computed at all grid nodes and the values are then differenced to compute values of u and v at all cell edges. These velocities then remain fixed for all time steps of the calculation.

To determine the numerical convergence rate of our solutions obtained in the following test problems, we compare our results to values of a reference solution, obtained either analytically or numerically. We assume that computed values Q_{ij} are located at the center of mass of regular and irregular cells. We then use this reference solution to compute the error e_{ij} in each cell. The weighted norm of this error over the entire computational grid is computed using

$$\begin{aligned} \| e \|_p &= \left(\frac{1}{\text{Area}(\Omega)} \sum_{i,j} |e_{ij}|^p \, \kappa_{ij} \Delta x \, \Delta y \right)^{1/p} \\ &= \left(\frac{\Delta x \Delta y}{\Delta x \, \Delta y \sum_{i,j} \kappa_{ij}} \sum_{i,j} |e_{ij}|^p \, \kappa_{ij} \right)^{1/p} \\ &= \left(\frac{1}{\sum_{i,j} \kappa_{ij}} \sum_{i,j} |e_{ij}|^p \, \kappa_{ij} \right)^{1/p}, \quad 1 \le p < \infty. \end{aligned} \tag{2.33}$$

The ∞-norm is defined in the standard way as

$$\| e \|_\infty = \max_{i,j} |e_{ij}|. \tag{2.34}$$

In all of the following mesh and contour plots, we only plot solution values in cells which are either completely or partially in the flow domain. Cells that lie completely in the no-flow domain are masked in all plots (using the `NaN` value in MATLAB).

2.6.1 Advection and diffusion of a plane wave in a channel

For this set of examples, we look at the basic behavior of our solution in the presence of a planar boundary which is not aligned with the computational grid. The physical domain is a channel with parallel sides, as shown in Figure 2.6. The channel is oriented at an arbitrary angle (approximately 0.0754 radians) to the grid and the flow is advected at speed U_0, with $0 \leq U_0 \leq 1$ in a direction parallel to the walls. We impose no-normal-flow boundary conditions on the channel walls.

The computational grid used here is $N \times M$, where N is the number of cells in the horizontal direction and M is the number of cells in the vertical direction. For the channel problem, we have chosen $N = 2M$. Convergence rates are reported in terms of N.

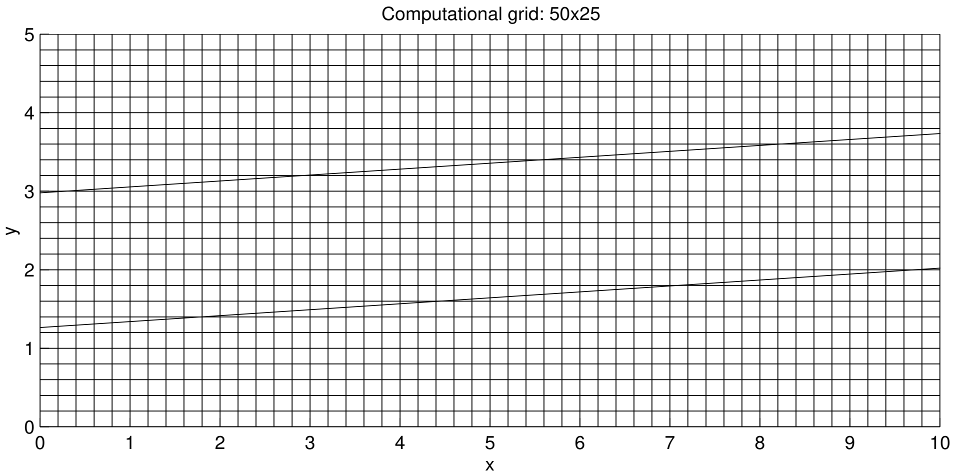


Figure 2.6: Coarse grid (50×25) used for flow in channel. The channel makes an angle of ≈ 0.0754 radians with bottom of computational domain and is ≈ 1.71 units wide.

The basic plane wave shape that we use to initialize the flow is given by

$$w(\xi) = 0.5 \left\{ \operatorname{erf}\left(\frac{0.75 - \xi}{\sqrt{4D}}\right) + \operatorname{erf}\left(\frac{0.75 + \xi}{\sqrt{4D}}\right) \right\} \tag{2.35}$$

where $D = 0.01$ is the diffusion coefficient and ξ is the streamwise distance from the center of the channel. For problems involving advection, we initialize our flow with this wave shifted towards the inflow end of the channel. For a given velocity U_0, we have that the solution for the pure advection equation is

$$q_{adv}(\xi, t) \equiv w(\xi - U_0(t - 3)). \tag{2.36}$$

The pure diffusion solution is

$$q_{diff}(\xi, t) = 0.5 \left\{ \operatorname{erf}\left(\frac{0.75 - \xi}{\sqrt{4D(1+t)}} \right) + \operatorname{erf}\left(\frac{0.75 + \xi}{\sqrt{4D(1+t)}} \right) \right\}. \tag{2.37}$$

The general advection-diffusion solution is

$$q_{adv\text{-}diff}(\xi, t) \equiv q_{diff}(\xi - U_0(t-3), t). \tag{2.38}$$

In Figure 2.7 we plot the function $q_{adv\text{-}diff}(\xi, t)$ at four different times for $U_0 = 1$ and $D = 0.01$.

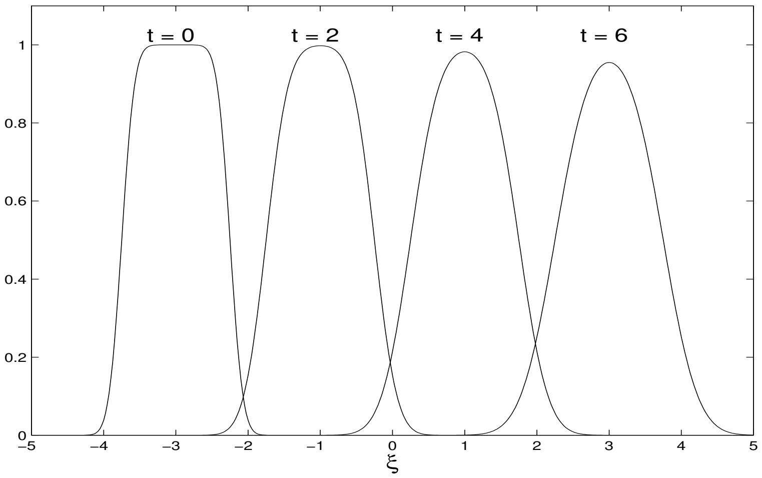


Figure 2.7: Plot of exact solution $q_{adv\text{-}diff}(\xi, t)$ to plane wave advection-diffusion in channel. The velocity is set to $U_0 = 1$ and the diffusion coefficient is set to $D = 0.01$. The Peclet number is 100.

Advection of a plane wave in the channel

To see how the advection portion of the algorithm behaves as the grid is refined, we computed the solution on three different grids. In Figure 2.8, we show the results of this series of computations. As expected, the contours in each plot in Figure 2.8 remain essentially normal to the wall.

The glitches in the plots are typically caused by inaccurate values in one or two very small cells cut by the boundary. To some extent this nonsmooth behavior is due to the use of MATLAB to plot contour lines. MATLAB takes the data as being at uniformly spaced points, which is really not correct in the small cells, where it is more

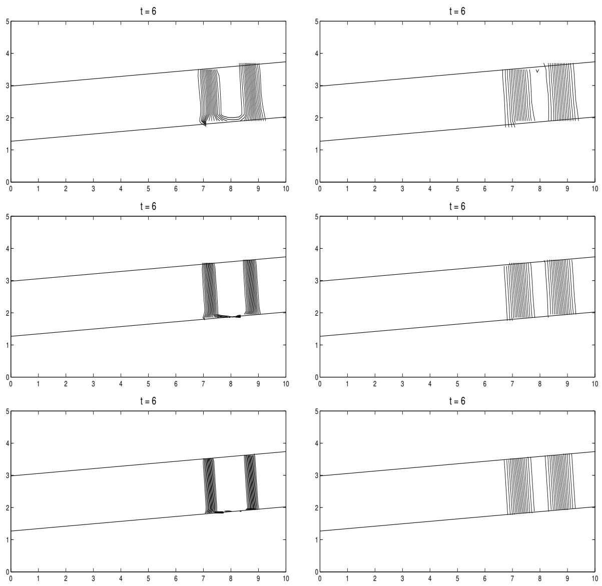


Figure 2.8: Advection and diffusion of plane wave in channel, computed on three different grids. The plots on the left show the results of pure advection and the plots on the right show the results of advection-diffusion. All plots are shown at time $t = 6$. From top to bottom the grid sizes are 50×25, 100×50 and 200×100.

properly viewed as located at centroids. Another reason may lie in the fact that small cells at the upper edge increase in size in the direction of the flow, whereas at the lower edge they decrease in size. We have not investigated this anomalous behavior in any detail, but do wish to point out that our scheme can produce slightly different results, depending on the orientation of the grid to the boundary, and on the order in which small cells appear as the flow passes through them.

While the algorithm is not perfect in these cells, it is notable that:

- The method remains stable and the incorrect values typically remain within the range of the nearby states.

- Such values only occur when a sharp front is passing by. Once it has passed the values rapidly settle down to the correct post-front state.

- These inaccuracies in the tiny cut cells typically have no impact on the solution in full cells nearby. (This is tested in a later example, see Figure 2.15.)

- As observed below, when this advection algorithm is coupled with our diffusion algorithm for advection-diffusion equation, this anomalous behavior almost com-

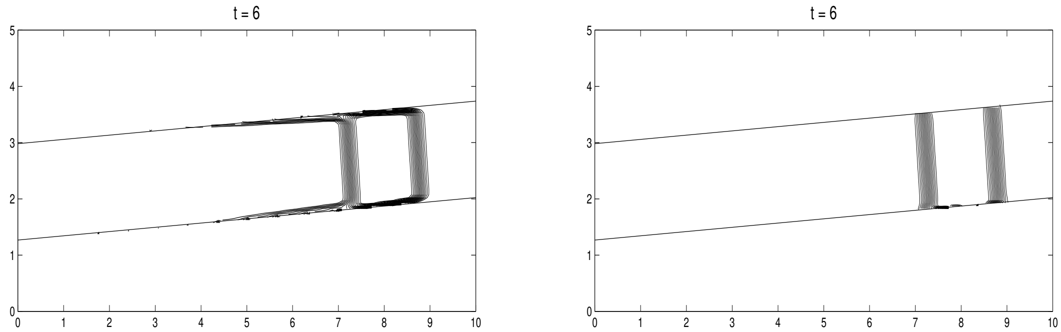


Figure 2.9: The solution in the first plot was computed without using a capacity function (i.e. $\kappa_{ij} \equiv 1$). In the second plot, the capacity function was used to take into account areas of small cells. The computational grid is 200×100.

pletely disappears as the implicit diffusion operator smooths out these spikes in the data.

Is the capacity function really necessary? As a final test of the advection in a channel, we demonstrate that by using "capacity form differencing" as described in the previous section, we can essentially eliminate smearing that would occur near the boundary if the numerical scheme did not take into account areas of small cells. For this test, we compute two solutions, one using the capacity function, modified as necessary to ensure stability, and the second without using the capacity function (i.e. setting $\kappa \equiv 1$ in all cells). In Figure 2.9, we see that when the capacity function is not used, the results are smeared out considerably. Using the Darcy velocities at the cell interfaces gives a stable method, but the velocity is smaller at edges of the cut cells than elsewhere leading to slower advection along the walls. The smaller velocity is offset by the small area of the cells when our capacity algorithm is used.

Diffusion in the channel

To test the diffusion portion of the code, we considered the diffusion of the initial pulse given in (2.35) and compare the results of our algorithm to the exact solution given in (2.37). While we don't show any plots of the diffusion alone, we can see in Figure 2.8 that when a diffusion term is present, there is no noticeable difference between the solution in small cells and that in neighboring full cells. In fact, the diffusion term tends to smooth out any spikes in the data produced by the advection algorithm. Figure 2.12 shows second order convergence in both the 1-norms and max-norms for the pure diffusion algorithm, and that the errors decrease quite smoothly as we refine the grid. As we note in our discussion of the convergence results in Section 2.6.1, this is not necessarily true when we include advection.

Performance of BiCGSTABin the finite volume diffusion algorithm. To solve the non-symmetric linear system of equations arising from the implicit discretization of the diffusion equation, we used the linear iterative method BiCGSTAB. This iterative method has several advantages over other methods for non-symmetric systems, and we have found its performance to be quite acceptable. Unlike GMRES, BiCGSTAB does not require additional storage for orthogonal vectors, and unlike the original BiCGSTAB, does not require a matrix *transpose* vector multiply. Furthermore, BiCGSTAB is straightforward to implement. One very clear presentation of the algorithm is given in [27].

In Figure 2.10, we show how BiCGSTAB performs as a function of grid size and of diffusion coefficient. In both cases, we measure the performance by looking at number of iterations BiCGSTAB needs to reduce the residual of the linear system to the desired level (10^{-8}, in this test case). In the first test case, we look at how varying the size of the linear system affects the number of iterations BiCGSTAB requires. In this test, we varied the grid size and held the diffusion coefficient constant at $D = 0.01$. We then determined the average number of iterations required for BiCGSTAB to advance the solution from t to $t + \Delta t$, for $\Delta t = 0.025$. This average was computed over the 20 time steps needed to advance the solution to $t = 0.5$.

For the second test case, we held the number of grid points (and hence the order of the linear system) fixed at $N = 200$ and varied the diffusion coefficient between 10^{-3} and 0.5. Again, we held the time step fixed at $\Delta t = 0.025$ and computed the average number of iterations over the first 20 times steps. In both test cases, we solved only the diffusion problem.

What is interesting to note from the two plots is that increasing the diffusion coefficient has a much greater impact on the number of iterations BiCGSTAB requires than does increasing the grid size. A grid of 400×200 at $D = 0.01$ required about 7 iterations, whereas, a grid of half that size, for a diffusion coefficient of $D = 0.05$ required 25 iterations. One explanation for this may lie in the fact that the underlying matrix that BiCGSTAB is inverting differs from a symmetric matrix in only $O(N)$ rows. As the matrix increases in size, the ratio of the number of rows perturbing the symmetry of the matrix versus the total number of rows behaves like $O(N^{-1})$. As the grid size is increased, for fixed diffusion coefficient, the matrix behaves more like a symmetric matrix.

On the other hand, increasing the diffusion coefficient affects the diagonal dominance of the underlying matrix, since at least away from the interface, the operator we are inverting is $(1 + \Delta t \, D\nabla^2)$. As we increase D, the matrix looks more like the discrete Laplacian, which requires many more iterations to invert than its parabolic counterpart.

These results are characteristic of the behavior of BiCGSTAB for all problem geometries investigated, and so are not reported for other test problems.

(a) (b)

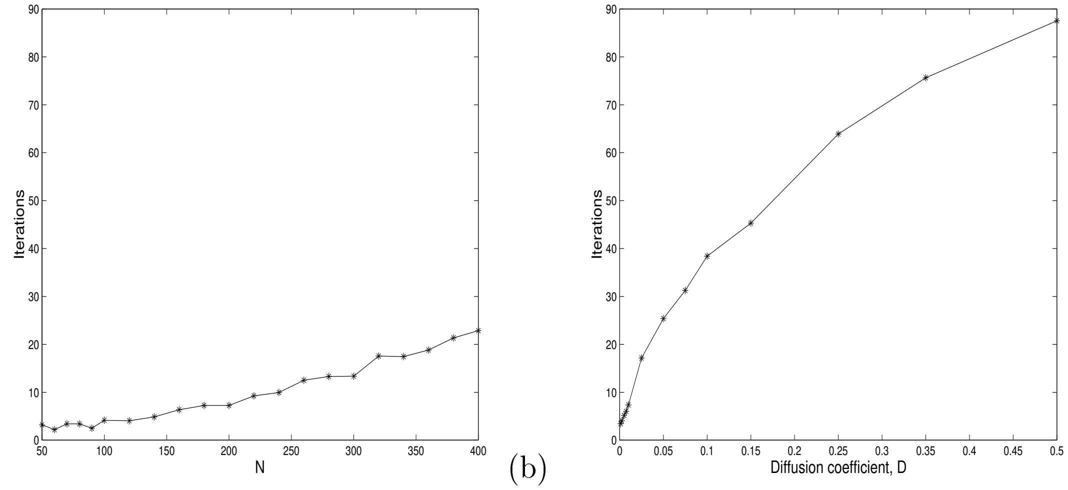


Figure 2.10: Performance of BiCGSTAB, as measured by the average number of iterations required to advance the solution from t to $t + \Delta t$, for $\Delta t = 0.025$. *(a)* Number of iterations versus grid size, for fixed diffusion coefficient, $D = 0.01$ *(b)* Number of iterations versus diffusion coefficient, for fixed grid of size 200×100. In both plots, the time step was fixed at $\Delta t = 0.025$.

Advection and diffusion in the channel

Finally, we combine the advection and diffusion schemes using the fractional step approach described in Section 2.5. Contour plots showing the results for this test problem are shown on the right side of Figure 2.8. Note that the contour lines are now perpendicular to both sides of the channel. The implicit diffusion algorithm smooths out the incorrect values in very small cells and corrects the anomalous behavior seen at the boundary in the pure advection case. Figure 2.11 shows a mesh plot of the results on the coarsest grid.

Convergence results for the channel for different Peclet numbers

In Figure 2.12, we show the convergence rate for the solution of the advection-diffusion equation in the channel for different values of the non-dimensional parameter $Pe = U_0\ell/D$, known as the Peclet number. Here, ℓ is the length scale of the problem, and for our purposes, is taken to be exactly 1. For $0 < Pe < \infty$, we varied Pe by varying the velocity and holding the diffusion coefficient fixed at $D = 0.01$. For $Pe = 0$, we carry out only the diffusion step of the fractional step scheme,using $D = 0.01$ and for $Pe = \infty$, we carry out only the advection step of the splitting scheme, using $U_0 = 1$.

For a given Peclet number, we compute a single convergence rate r for our algorithm by assuming that the truncation error in our discretization has the form

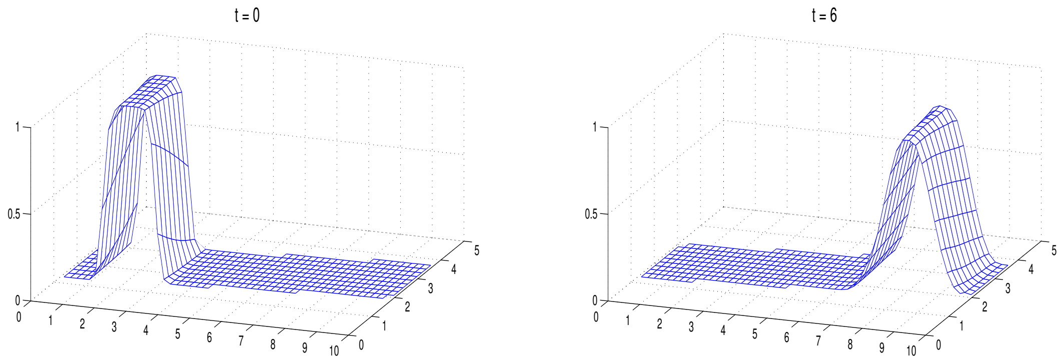


Figure 2.11: Advection-diffusion of a plane wave down the channel. Mesh plots of the initial data and solution at time $t = 6$ on the coarse grid, 50×25.

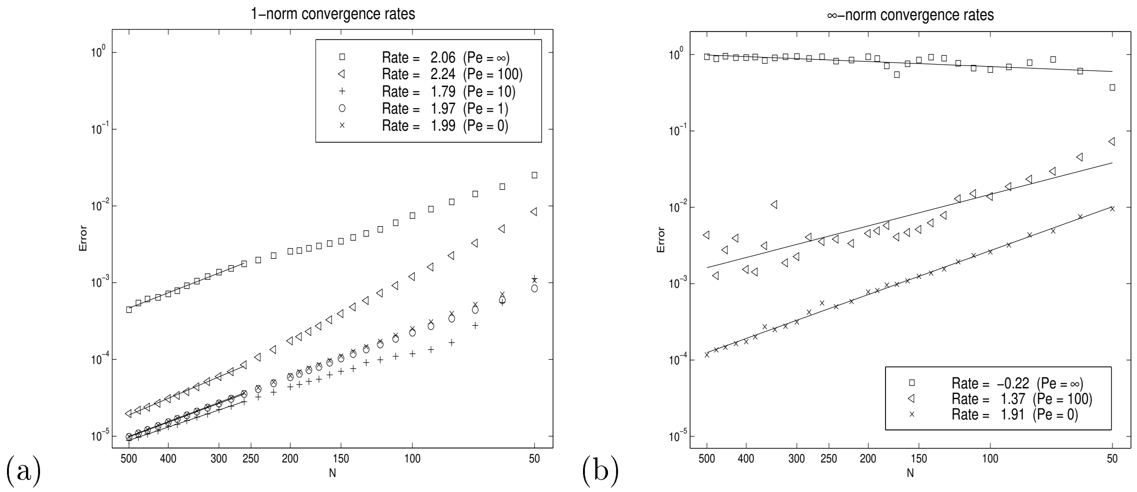


Figure 2.12: *(a)* 1-norm convergence rates and *(b)* ∞-norm convergence rates for advection-diffusion in channel, for varying Peclet numbers. Solid line is the best-fit line used to compute convergence rate for each case.

$\epsilon(h) = Ch^r$ for some constant C. By computing the error on several grids, we can then approximate the convergence rate by fitting a line through the log of the computed errors.

To compute the errors, we use as a reference solution the exact solution given in (2.38). The errors were all computed at time $t = 6$, roughly the time it takes for the wave moving at the largest velocity tested ($U_0 = 1$) to traverse the length of the channel. For problems computed at slower velocities, the wave did not travel as far, but in all cases which involved diffusion ($Pe < \infty$), the total effect of diffusion was the same. The errors, computed using the 1-norm and ∞-norm, along with the rates of convergence are shown in the $\log - \log$ plots in Figure 2.12.

We use so many different grid spacings because the exact manner in which the geometry intersects the Cartesian grid naturally has an impact on the accuracy on any particular grid. As a consequence, the error may not be smoothly varying as the grid spacing is decreased. From the plots, though, it is clear that the error (at least in the 1-norm) fluctuates about lines with slopes indicating second-order accuracy. The 1-norm errors are dominated by errors in the full cells, and so do in fact vary smoothly. The max-norm errors, however, are dominated by the errors in small cells and so show less smooth behavior.

Because the diffusion tends to smooth out any errors produced in small cells by the advection scheme, we expect that the lower the Peclet number (and, hence, the more dominant the diffusion term), the smaller the magnitude of the computed errors. We see in Figure 2.12 that this is in fact that case. The largest errors are produced by the pure advection ($Pe = \infty$) case, and the smallest errors are produced by the pure diffusion case. In the max-norm, we see that the pure advection may not even converge, but with even just a small amount of diffusion, we get convergence rates that approach second order as we decrease the Peclet number. In the second plot of Figure 2.12, we have plotted max-norm errors for 3 of the 5 Peclet numbers ($Pe = \infty$, $Pe = 100$, and $Pe = 0$. The convergence rates for the remaining two cases were 0.90 ($Pe = 10$) and 1.47 ($Pe = 1$). The magnitude of the errors were very close to those cases plotted and so to avoid cluttering the graph, these errors are not shown.

2.6.2 Advection and diffusion in an annulus

For this set of problems, we look at diffusion and solid body rotation inside an annulus composed of two concentric circles embedded in a square computational grid. The goal here is to test how well our advection-diffusion scheme works in more complicated geometry. The annulus is a convenient choice because we can compare our finite volume solution to a reference solution computed in polar coordinates.

Figure 2.13 illustrates, on a 30×30 grid, the geometry used for this test case. The radius of the inner circle of the annulus is $R_1 = 0.25$. The outer radius is $R_2 = 1.25$. The center of each circle is at $\approx (1.509, 1.521)$. The initial profile of our solution is a function of θ only and is given in terms of the wave profile similar to that used for the channel. This wave profile is

$$w(\theta) = 0.5\left\{\operatorname{erf}\left(\frac{\pi/6-\theta}{\sqrt{4D}}\right) + \operatorname{erf}\left(\frac{\pi/6+\theta}{\sqrt{4D}}\right)\right\}. \tag{2.39}$$

In the interior of the annulus, the solution was initialized using

$$q_0(\theta) \equiv w(\theta - \pi/2). \tag{2.40}$$

We imposed no-flow boundary conditions at the two boundaries of the annulus.

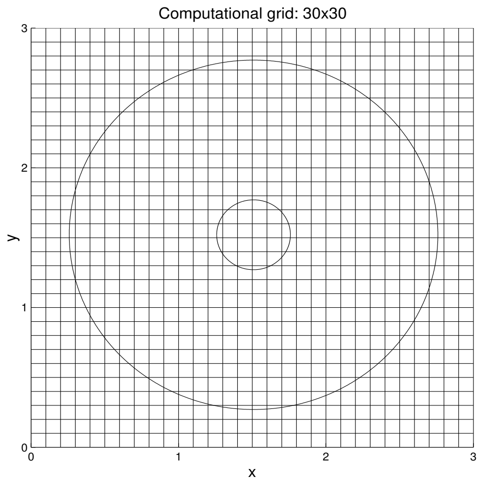


Figure 2.13: Geometry for annulus test problem solver. The inner radius of the annulus is $R_1 = 0.25$. The outer radius is $R_2 = 1.25$ and the center is at $\approx (1.509, 1.521)$.

Solid body rotation in an annulus

To test our proposed numerical scheme for the advection solver, we considered advection under solid body rotation. The velocities for solid body rotation can be determined from the stream function

$$\psi(x,y) = \begin{cases} 0 & r > R_2 \\ 0.2\,\pi\,(R_2^2 - r^2) & R_1 \le r \le R_2 \\ 0.2\,\pi\,(R_2^2 - R_1^2) & r < R_1 \end{cases} \tag{2.41}$$

where $r = ((x-1.509)^2 + (y-1.521)^2)^{1/2}$ is the distance of a particular point (x, y) to the center of the annulus. For this particular choice of ψ, the flow makes one complete revolution in 5 time units.

Figure 2.14 shows contour plots of the computed solution as it rotates around the annulus. The solution is shown at a sequence of times on two different grids. As in the channel flow case, the flow remains essentially parallel to both boundaries of the annulus, although there is evidence of numerical diffusion in the coarse grid solution.

As a second validation of the solid body rotation, we looked at the computed solution as a function of θ at different values of r, after one and two full rotations, and compared these values with the initial conditions. These comparisons are presented in Figure 2.15 for a 60×60 grid. In Figure 2.15(a), we plot the value of the solution in the partial cells cut by the outer boundary of the annulus versus θ. The solution stays bounded and stable in these cells, but some inaccuracy in the smallest cells is observed as the front propagates past. However, this loss of accuracy in the cut cells does not appear to affect the solution away from the boundary. In Figure 2.15(b), we show the solution a small distance inside the boundary, as interpolated from the full cell values to points (R_2^-, θ_i), where $R_2^- = R_2 - \sqrt{2}\Delta x$ is a radial value slightly inside the outer edge of the annulus, and the θ_i values are equally spaced values of θ in $[-\pi, \pi]$. Here the solution agrees quite well with the expected behavior.

Figure 2.15(c) shows the solution near the middle of the annulus, at $r = (R_1 + R_2)/2$. Figure 2.15(d) shows the solution just inside the inner boundary, at $R_1^+ = R_1 + \sqrt{2}\Delta x$, again interpolated from the full grid cells. Finally, Figure 2.15(e) shows the values in the partial cells cut by the inner boundary. Note that near the inner boundary the solution is poorly resolved because there are very few grid cells around this circle (see Figure 2.13). In all cases there is very little degradation of the solution between the first and second revolution.

Diffusion in an annulus

To test diffusion in the annulus, we allowed the initial data to diffuse over time (with no advection). The results we obtained agreed well with a reference solution computed by solving the diffusion equation in polar coordinates:

$$\frac{\partial q}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial q}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 q}{\partial \theta^2}, \qquad R_1 \le r \le R_2, \quad -\pi \le \theta \le \pi. \tag{2.42}$$

We obtained a spectrally accurate solution to (2.42) using a Fourier expansion in the azimuthal direction and a pseudospectral method based on Chebyshev polynomials in the radial direction. We used 256 points in θ and 32 points in the r-direction (where the solution varies more smoothly). This gives a highly accurate reference solution, which was then interpolated to the Cartesian grid using Fourier and Chebyshev interpolation.

Figure 2.16 shows contour plots comparing the computed and reference solution at four different times. Even near the boundaries of the annulus it is evident that the no-flux boundary condition is satisfied. To show how our computed solution behaves as the grid is refined, a fine grid solution and corresponding reference solution are also plotted in Figure 2.16.

Solid body rotation coupled with diffusion in an annulus

Contour plots showing the results of combining the diffusion and advection in an annulus are shown in Figure 2.17. Even the coarse grid solution agrees reasonably well with the reference solution. In Figure 2.18 we show a mesh plot of the same set of results at two different times.

Convergence results for the annulus for different Peclet numbers

In Figure 2.19, we show 1-norm and max-norm convergence results for the annulus test problem. As we did for the channel, we investigated how the convergence rates varied with Peclet number. For this test, however, we held the velocity fixed (except in the $Pe = 0$ case, where there is no rotation), and varied the diffusion coefficient. The Peclet numbers tested were $Pe = 0, 20, 100, 500, \infty$. For $0 < Pe < \infty$, we computed the diffusion coefficient for a given Peclet number as $D = U_0\ell/Pe$, where for this problem, $\ell = 1$ and U_0 was taken to be the tangential velocity on the outer boundary of the annulus. For the stream-function used, this velocity is given by $U_0 = 2\pi R_2/5 = 0.5\pi$. For $Pe = 0$, the diffusion coefficient was set to $D = 0.01$. For $Pe = \infty$, we used a diffusion coefficient of $D = 0.01$ only to initialize the flow, using (2.40).

Our choice of Peclet numbers differs from the channel case in that here, the total effect of the advection was kept fixed over the range of Peclet numbers chosen, whereas before, we held the effects of diffusion fixed. All errors were computed at $t = 5$, after the initial wave had made one complete revolution in the cases with advection. The errors vary more smoothly with decreasing grid size in this case than was observed in the channel. This may be because the annulus cuts through cells in a wider variety of random ways regardless of where it is located, while the channel leads to a periodically repeating pattern of cut cells, and hence may be more sensitive to its location, or the grid spacing. As seen in Figure 2.19, the errors for the annulus all lie very close to the best-fit line for both the 1-norm and the max-norm.

The rates of convergence for this test problem were not as close to second order as they were in the channel case, but we do see that the rates generally improve as the Peclet number decreases. We consider this to be a challenging test case because of the poor resolution of the rapidly-varying solution near the inner boundary, as seen in Figure 2.15(e), for example. Even on the finest grids there are relatively few grid points around the inner boundary.

2.6.3 Advection through a field of irregular objects

In this set of examples, we constructed a field of irregularly shaped inclusions and simulated the advection of a passive tracer through this field. One can imagine that this field represents, for example, rocks in a porous medium or the cross section of pipes in a heat exchanger. The purpose of this set of examples is threefold. First, we illustrate that our reliance on the stream-function for obtaining our velocity field does not limit us to objects for which we know a stream function analytically. Second, we show that our proposed advection-diffusion algorithm is not particularly sensitive to geometry chosen. And third, we demonstrate that our capacity-form differencing with the modified capacity function is effective at reducing smearing even when the stream-function is only known approximately. Figure 2.20 shows the field of irregular objects, embedded in the coarsest grid used for this example.

To compute the stream function in the multiply-connected domain, we solved an elliptic equation in the domain using the Immersed Interface Method [37] as the basis for an approach loosely based on the work of Yang [58]. This Cartesian grid method produces second-order accurate values of the streamfunction at all grid points, which were chosen to be the corners of our finite volume cells. We discuss this method in detail and show exactly how we obtain these streamlines in Chapter 3. Differencing these values gives the required Darcy velocities at cell edges. Figure 2.22 shows a contour plot of the stream-function used for this set of test problems, computed on a very fine grid.

We use initial data $q(x, y, 0) \equiv 0$ and a boundary condition $q(0, y, t) = 1$ up to time 20 (and zero thereafter), which corresponds to the injection of a tracer at the left (inflow) boundary over this time interval. Figure 2.22 shows plots of the solution at different times.

In order to compare the results more quantitatively as the grid is refined, we also computed "washout" curves $w(t)$ which measure the total mass passing through the right edge of the domain as a function of time. This is computed by

$$w(t_n) = \sum_{j=1,N} Q_{M,j} \Delta y \approx \int_0^5 q(10, y, t)\, dy \tag{2.43}$$

which sums the values of Q along the rightmost row of grid cells on an $M \times N$ grid. This gives the curves shown in Figure 2.23. The general behavior of flow through the field of irregular objects is preserved as the grid is refined. Even with the coarsest grid, where the fine-scale structure of the flow is poorly resolved in Figure 2.22, the washout curve shows quite good agreement. For many applications, e.g., groundwater flow simulations, this is of great importance.

2.6.4 Advection and diffusion through a field of irregular objects

In this test, we simulated a tracer advecting and diffusing through the same field of irregular objects. Figure 2.25 shows the results of this simulation at 3 different times. To see that the normal derivative of q is close to zero near the boundaries, as expected from the Neumann boundary conditions with diffusion, we also show a contour plot of this solution in Figure 2.24.

Washout curves are again shown in Figure 2.23. When diffusion is added, there is virtually no difference in the washout curves at different grid resolutions. Even the under-resolved grid of Figure 2.20 gives excellent results.

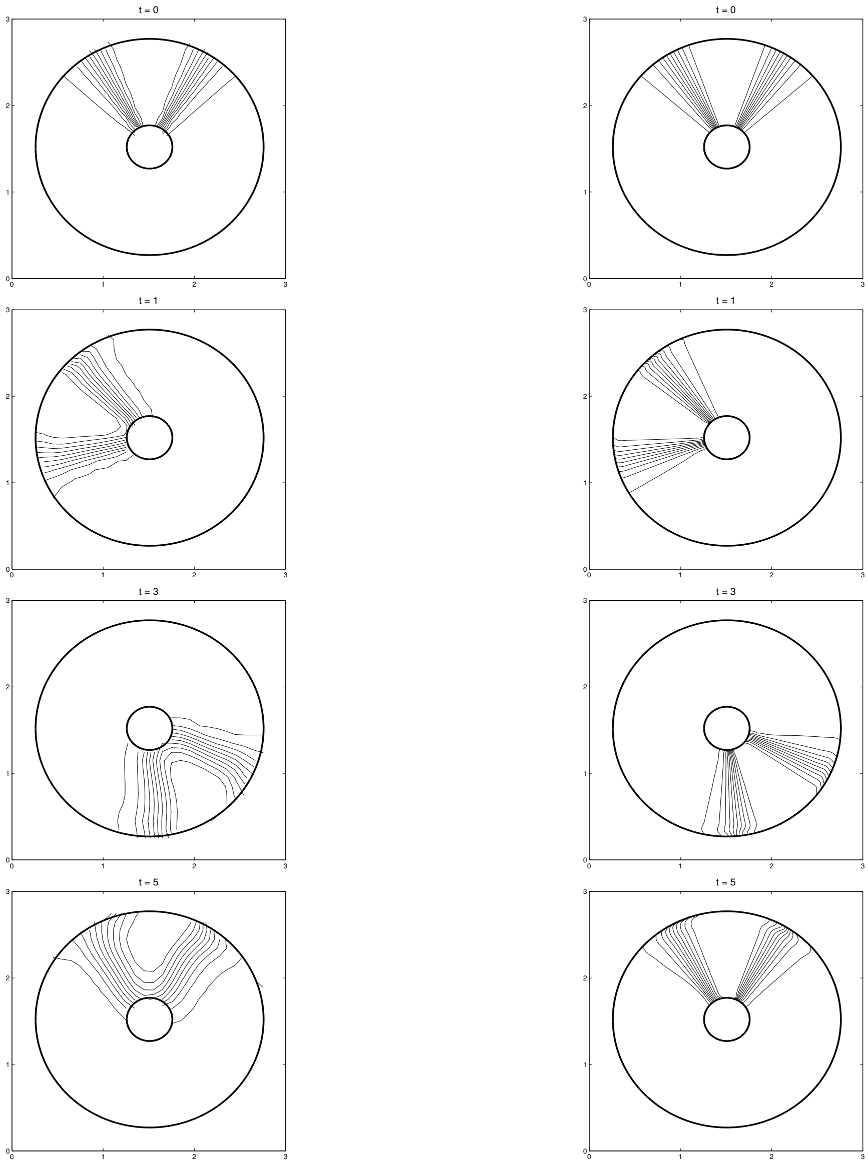


Figure 2.14: Contour plots on the coarse (30×30) and fine (150×150) grid showing results of solid body rotation at various times during one full rotation. Contour levels shown are $(0.1, 0.2, \ldots, 0.9.)$

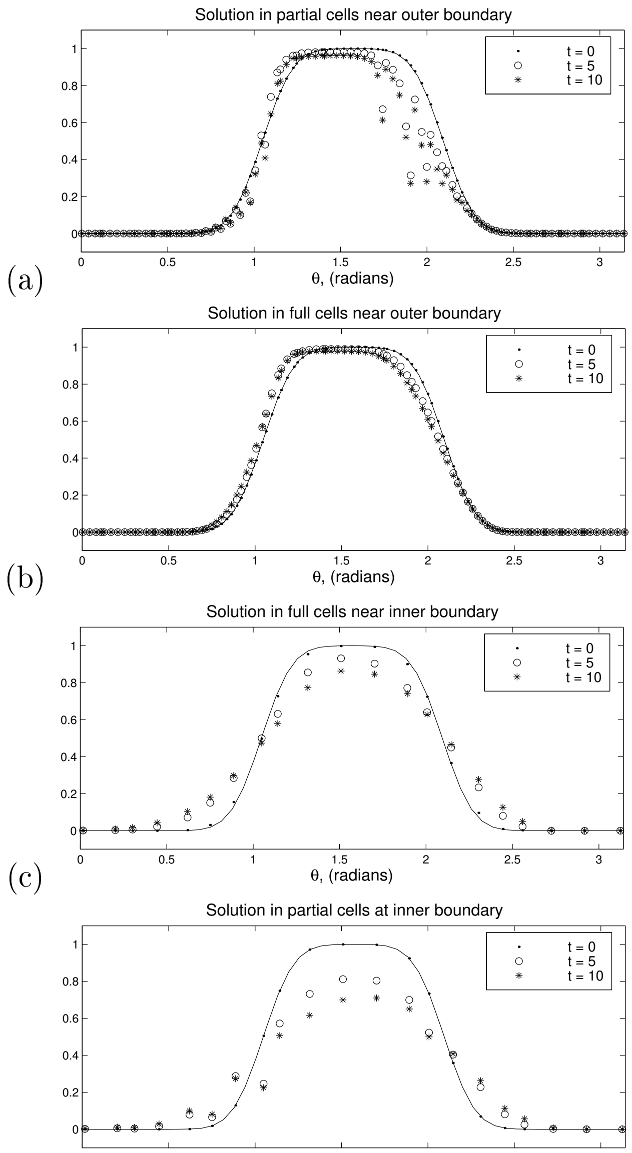


Figure 2.15: Computed solution to the pure advection equation in the annulus, shown as a function of θ for four different values of r. The numerical solution was computed on a 60×60 grid and is shown at two different times $t = 5$ (after one revolution) and $t = 10$ after two revolutions). The true solution is shown as a solid line. (a) At $r = R_2$, along the outer boundary. (b) At $r = R_2^-$, just inside the outer boundary. (c) At $r = R_1^+$, just inside the inner boundary. (d) At $r = R_1$, along the inner boundary.

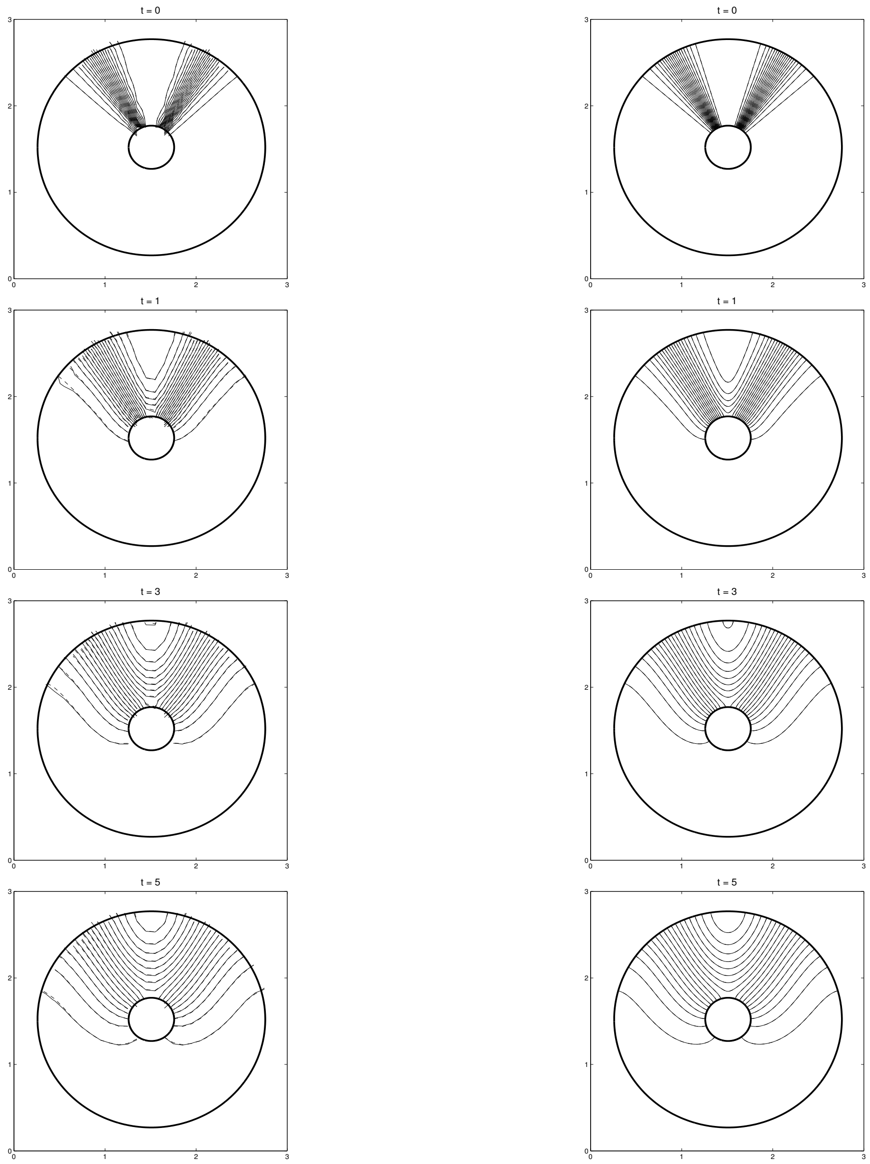


Figure 2.16: Contour plots on the coarse (30×30) and fine (150×150) grid showing results of diffusion in the annulus at various times. The solid lines represent results of the finite volume scheme, and dashed lines are the reference solution, interpolated to appropriate grid. Contour levels shown are $(0.01, 0.05, 0.1, 0.15, \ldots, 1.0.)$

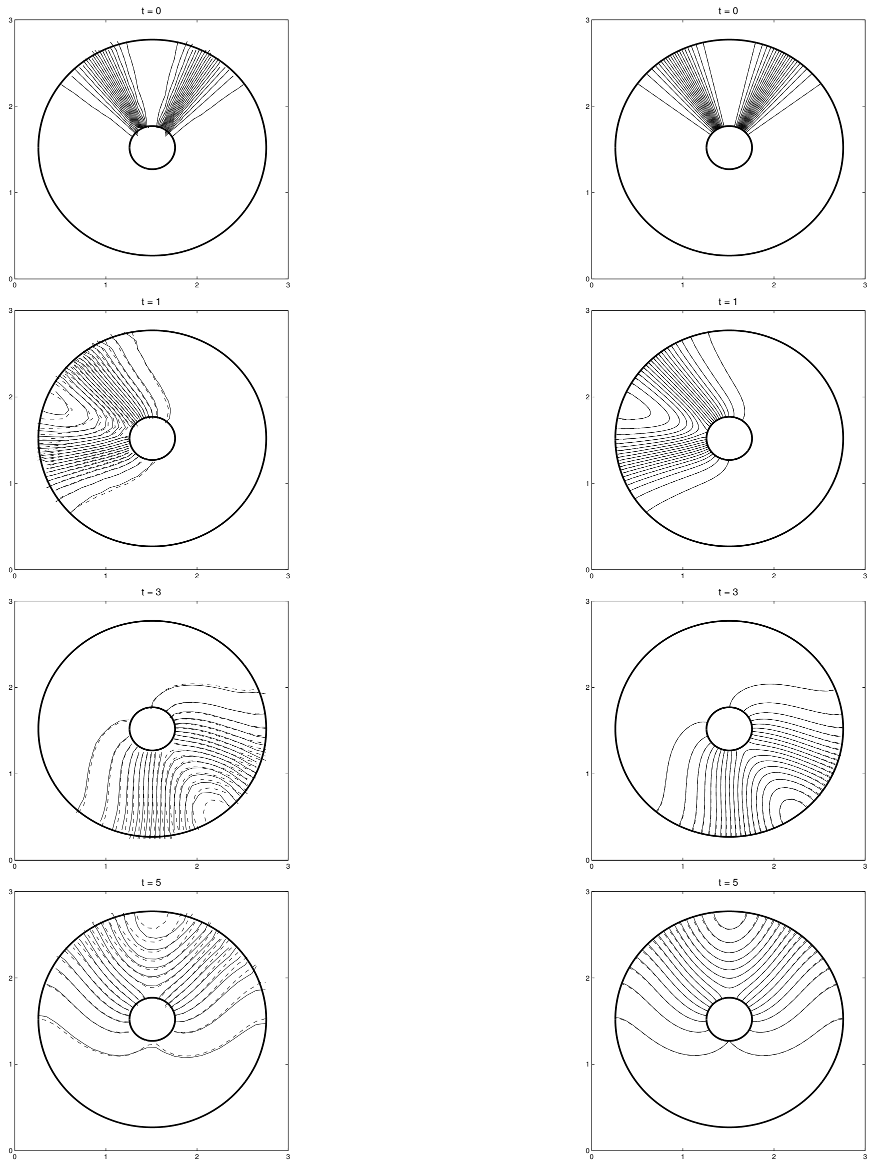


Figure 2.17: Contour plots on the coarse (30×30) and fine (150×150) grid showing results of solid body rotation coupled with diffusion at $Pe = 100$ in the annulus at various times over one full rotation. The solid lines represent the results of the finite volume scheme, and dashed lines are the reference solution, interpolated to the Cartesian grid. Contour levels shown are $(0.01, 0.05, 0.1, 0.15, \ldots, 1.0.)$

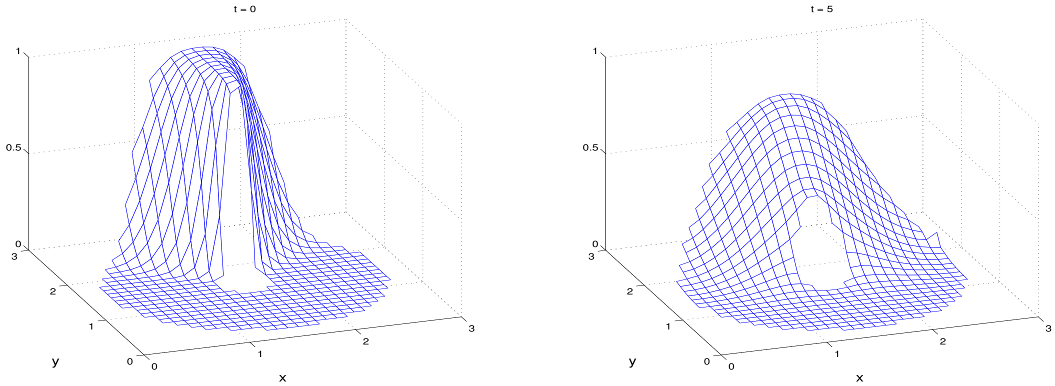


Figure 2.18: Mesh plots on coarse grid (30 × 30) for advection and diffusion in the annulus with $Pe = 100$. (a) Initial data. (b) Computed solution after one complete revolution.

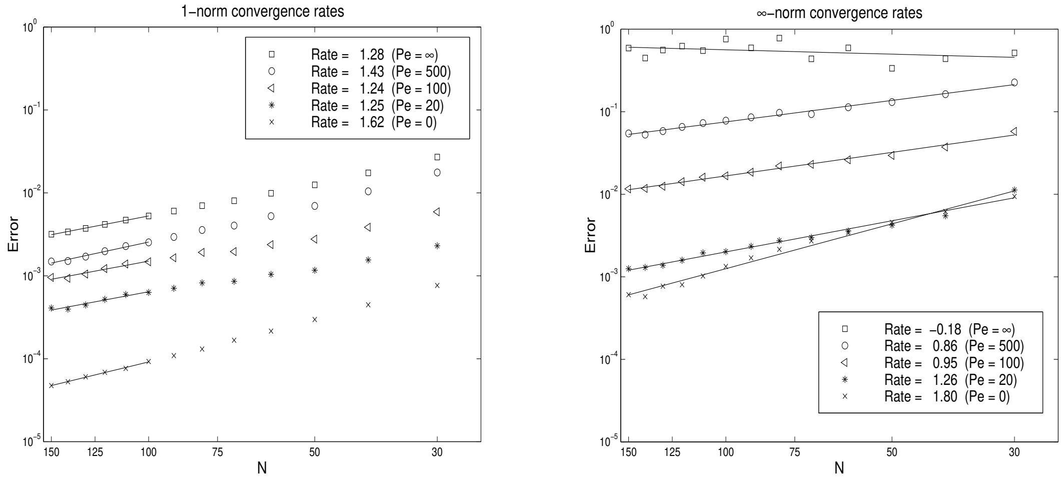


Figure 2.19: *(a)* 1-norm convergence rates and *(b)* ∞-norm convergence rates for advection-diffusion in the annulus, for varying Peclet numbers. Solid line is the best-fit line used to compute convergence rate for each case.

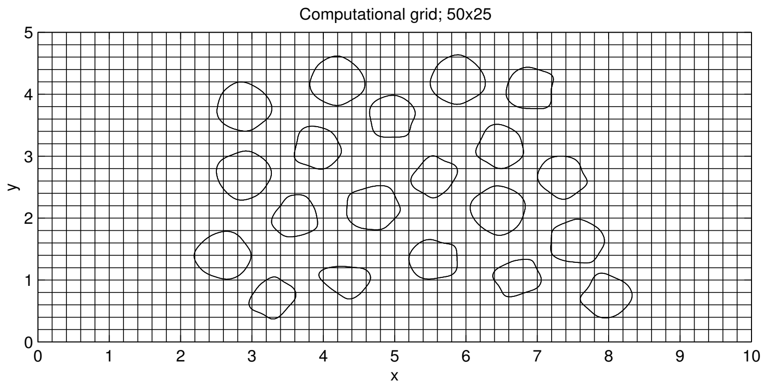


Figure 2.20: Coarse grid (50 × 25) with embedded objects.

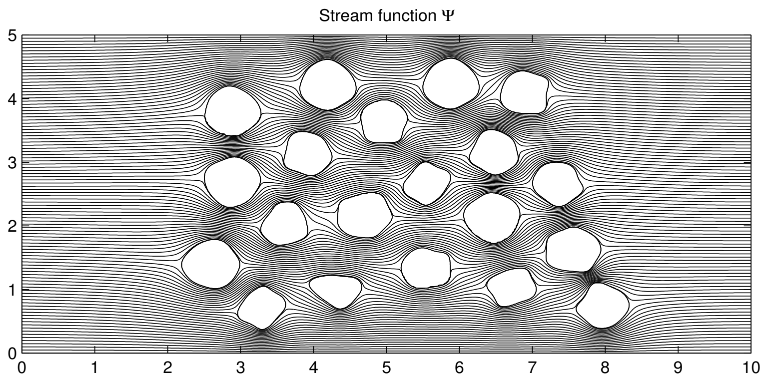


Figure 2.21: Stream function used for flow through irregular objects

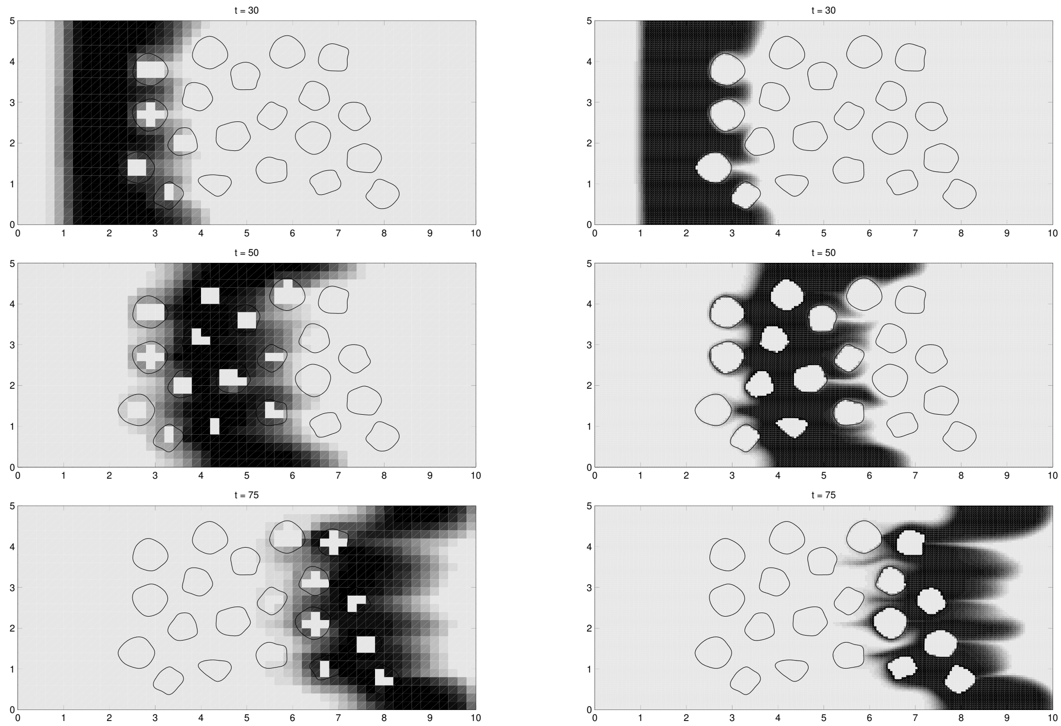


Figure 2.22: Advection of passive tracer through field of irregularly shaped objects. Coarse grid (left side) is 50×25 and fine grid (right side) is 200×100.

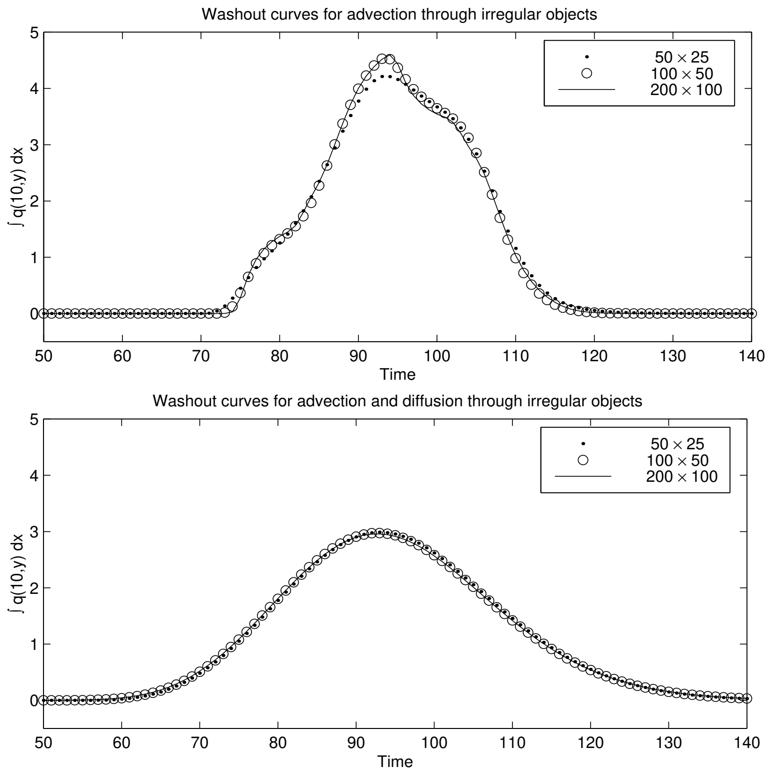


Figure 2.23: Washout curves computed on three different grids. The upper plot is for advection only. The lower plot is for the advection-diffusion equation.

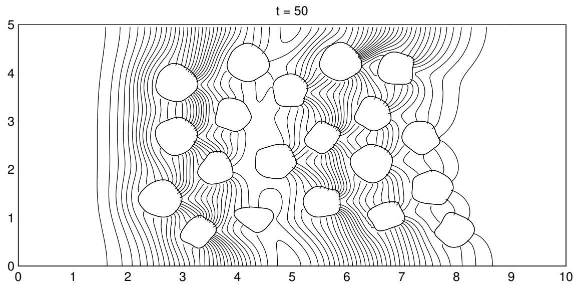


Figure 2.24: Contour plot of advection-diffusion results through field of irregular objects for the fine grid calculation at time $t = 50$. Note that the contour line are roughly perpendicular to the boundary of each object.

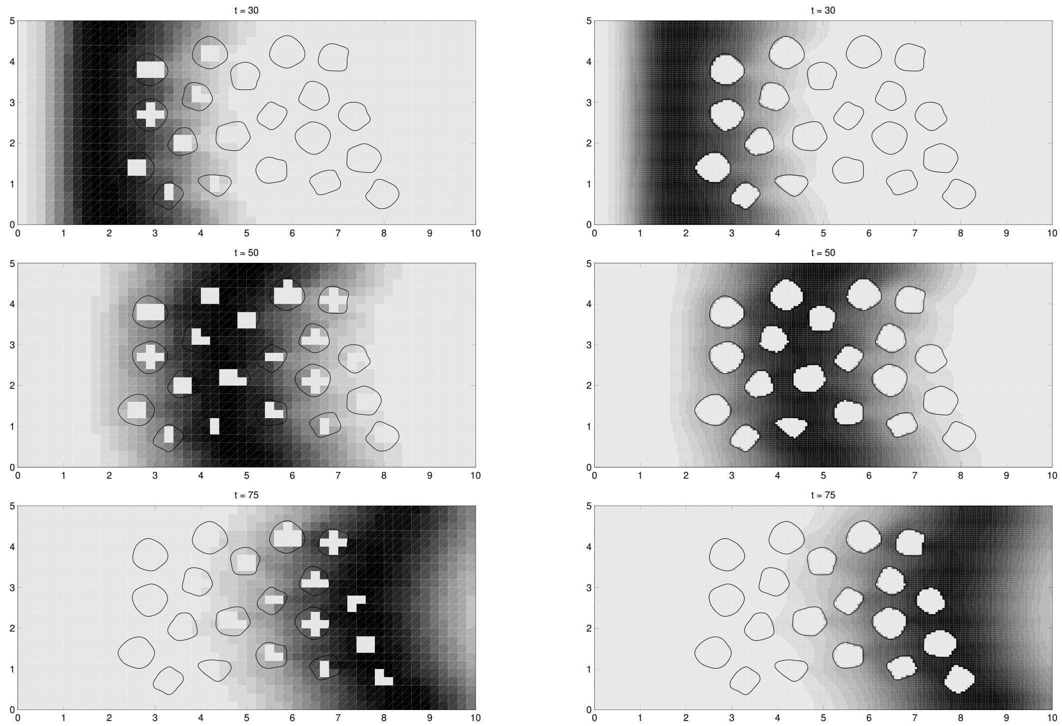


Figure 2.25: Advection and diffusion of passive tracer through field of irregularly shaped objects. Coarse grid (left side) is 50×25 and fine grid (right side) is 200×100.

Chapter 3

SOLVING ELLIPTIC AND PARABOLIC EQUATIONS IN IRREGULAR REGIONS

The first goal of this chapter is to describe in detail how we use the Immersed Interface method, developed by LeVeque and Li [36, 37] to solve constant coefficient elliptic and parabolic boundary value problems given by

$$\nabla^2 u = f, \qquad (x,y) \in \Omega \tag{3.1}$$

subject to Dirichlet conditions on $\partial\Omega$, and

$$u_t = \nu \nabla^2 u, \qquad (x,y) \in \Omega \tag{3.2}$$

subject to Neumann conditions on $\partial\Omega$. For this Chapter, our irregular multiply-connected flow domain is the region Ω, show in Figure 3.1. The flow domain Ω is the region exterior to all embedded objects.

The second goal is to combine the above equations to solve the equations describing Stokes flow in a multiply-connected domain. These equations in the exterior domain $\Omega = R \backslash \cup_{i=1}^{M} \Omega_j$ for a rectangular computational domain R are given by

$$\begin{aligned} \widehat{\omega}_t = \nu \nabla^2 \widehat{\omega}, \quad (x,y) \in \Omega \qquad & \frac{\partial \widehat{\omega}}{\partial n} = g, \quad (x,y) \in \partial\Omega \\ \nabla^2 \psi = -\widehat{\omega}, \quad (x,y) \in \Omega \qquad & \psi = \psi_j, \quad (x,y) \in \partial\Omega \\ \int_{\Omega_j} \frac{\partial \widehat{\omega}}{\partial n}\, ds = 0, \quad j = 1, M_{bodies} \qquad & \frac{\partial \psi}{\partial n} = 0, \quad (x,y) \in \partial\Omega \end{aligned} \tag{3.3}$$

where $\widehat{\omega}(x,y,t)$ is the vorticity of the flow and $\psi(x,y,t)$ the streamfunction whose level sets describe the streamlines of the flow. The conditions on the right are boundary conditions which hold only on the boundary of solid objects in the flow domain. Both the set of constants $\psi_j,\ j = 1, = hdotsM_{bodies}$ and g are unknowns and must be solved for as part of the solution process. We use $\widehat{\omega}$ instead of ω to distinguish the parabolic equation we solve here using the finite difference Immersed Interface Method from the parabolic equation we solve using the conservative finite volume method described in Chapter 2. As we will see in Chapter 4, we solve the above Stokes flow problem just to get the flux g needed to impose the no-slip boundary condition. This flux is imposed during the diffusion step of the finite-volume update.

To solve the elliptic, parabolic and Stokes flow equations, we use the Immersed Interface Method (IIM), a second order finite difference Cartesian grid method developed by LeVeque and Li [36, 37]. The original IIM was developed to solve PDEs with

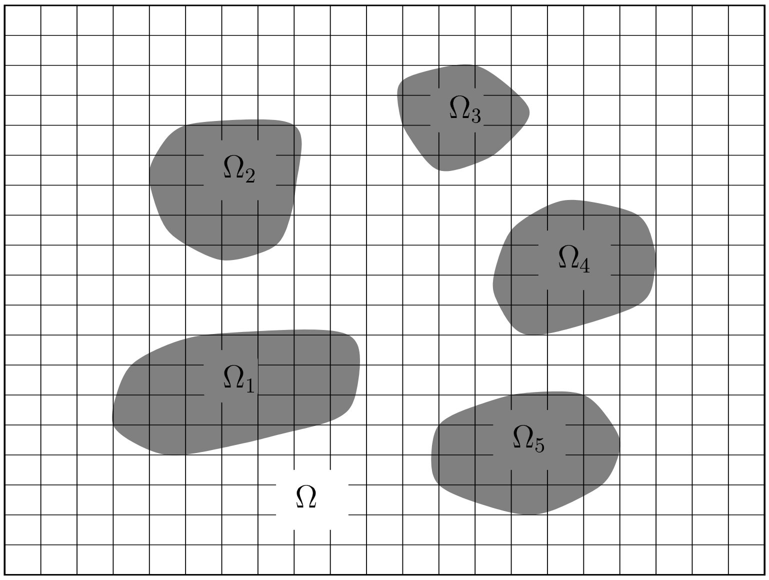


Figure 3.1: Typical domain in which elliptic and parabolic problems are solved. Shaded inclusions Ω_j are solid objects. Fluid domain is region Ω exterior to these objects.

singular source terms that arise physically in situations where, for example, forces are distributed only along a thin elastic membrane, latent heat is released from the solidifying interface separating the solid and liquid phases of a pure substance, or discontinuities in material properties exist along an interface. In the original method, these sources are expressed as jumps in the function or derivative at the interface, which in turn are incorporated into the finite difference discretization of the PDE.

In the original formulation, the jumps needed to correct finite difference stencils were assumed to be known from the physical problem. In the present situation, we don't have singular sources along an interface, per se, and moreover, are not given enough jump information at the boundary to make direct use of the IIM as it was originally formulated. In this chapter, we present an adaptation, loosely based on the work of Yang [58] and Wiegmann [56], of the IIM for use in solving boundary value problems. We show that one can approximate the elliptic or parabolic PDEs at Cartesian grid points near the boundary by including jump conditions into the finite difference stencils near these points. Some of the jump information is obtained by using given boundary value information but the remainder of the jump information must be computed as part of an auxiliary step in the solution process.

In the process of showing how we can use jump information to discretize elliptic and parabolic PDEs, we re-derive the IIM and introduce a notation which we believe should make the implementation details of the IIM much more straightforward and less error prone than previous descriptions. The reason that we take the trouble to re-derive the IIM is that despite its obvious applicability to a wide class of problems, the IIM has not been widely adopted. Part of this, of course, is due to the fact that the IIM, having first appeared in 1994, is still relatively new, and researchers with time and effort invested into one numerical methodology are reluctant to switch to another method. But another reason may be that at first glance, the Immersed Interface Method appears quite complicated. The original derivation of the necessary formulas involved tedious algebra and resulted in lengthy formulas which had to be translated into computer code. Fortunately, one can reformulate the problem so that most of the algebra is taken care of by inverting appropriate matrices rather than solving linear systems by hand. In this chapter we re-derive the IIM using this matrix formulation and present formulas which we believe are much more compact than those presented in [38, 58] and which will hopefully encourage a more widespread use of the IIM.

The first part of this chapter will be devoted to deriving the Immersed Interface Method for constant coefficient problems. Our discussion in this section applies to a wide variety of constant coefficient PDEs of the form

$$\nabla^2 u + \lambda u = f \tag{3.4}$$

although we will use as our model problem Poisson's equation $\nabla^2 u = f$. Later in this chapter, we describe specific modifications to solve parabolic equations, which we also put in the form given above.

In the last part of this chapter, we provide numerical results which show that the IIM is second order accurate and is quite robust for a variety of problems and irregular domains, including multiply-connected domains. In this section, we choose examples that arise in the context of solving the streamfunction vorticity equations. In particular, we solve the potential flow equations for the streamlines of incompressible, inviscid flow, the diffusion equation for vorticity, and finally, the Stokes flow equations, given in (3.3), for unsteady, viscous, incompressible flow.

3.1 Imposing boundary conditions using singular source terms

We begin our discussion with some observations about solving elliptic equations on a rectangular region. We suppose that we are given a two dimensional square region R on which we wish to solve the problem

$$\nabla^2 u = f, \qquad x \in R \tag{3.5}$$

subject to Dirichlet boundary conditions $u = g$ on the boundary ∂R of the region R. We establish a uniform Cartesian mesh on R with nodes corresponding to $x_{ij} \equiv (x_i, y_j), i, j = 1, \ldots N + 1$ where $x_i = (i-1)\Delta x$ and $y_j = (j-1)\Delta y$ for mesh widths Δx and Δy respectively. For convenience, we may make the assumption that $\Delta x =$

$\Delta y \equiv h$. At each grid point, we wish to solve for the function value $u_{ij} \approx u(x_i, y_j)$. To do this, we discretize the Poisson problem given in (3.5) at a grid point x_{ij} as

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2} = f_{ij} + B_{ij}, \qquad i,j = 2, \dots N \tag{3.6}$$

where the value of u_{ij} in these equations that correspond to grid points on the boundary of R ($i, j = 1$ or $N+1$) have been set to 0, and their inhomogenous counterpart supplied through the value B_{ij}. For example, the equation for the point $x_{2,j}$ relies on the boundary value $u_{1,j}$. We set this value to $g_{1,j}$ and move it to the other side. In this case, $B_{2,j} = -g_{1,j}/h^2$. Setting $u_{1,j} = 0$ is just a convenient way to use a general formula for all points, whether they are near a boundary or not. In Figure 3.2(a), the grid points corresponding to non-zero entries in the vector B are shown.

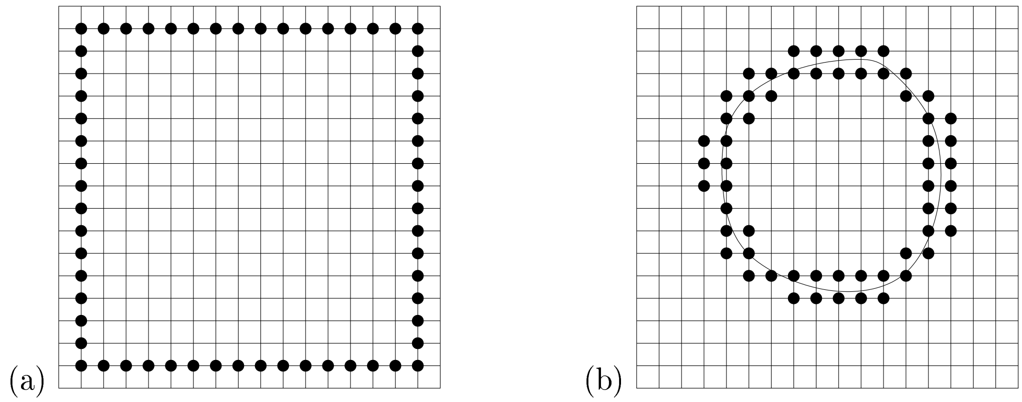


Figure 3.2: (a) Grid points corresponding to non-zero entries in the vector B, used to impose boundary conditions in rectangular problem. (b) Grid points corresponding to non-zero entries in the vector C, used to impose boundary conditions at irregular boundary.

These equations can be written as a linear system involving the matrix $A \in \mathbb{R}^{(N-1)^2 \times (N-1)^2}$ and vectors U, F and B all in $\mathbb{R}^{(N-1)^2}$. This system is

$$AU = F + B \tag{3.7}$$

where the matrix A corresponds to the discrete Laplacian, the vector F has entries corresponding to the known right hand side values f_{ij} and vector B involves the non-homogeneous terms B_{ij}. This system is easily solved using a Fast Poisson Solver, and the desired solution u_{ij} satisfying the given boundary conditions is obtained at each internal grid node $x_{ij}, i, j = 2, \dots N$.

To motivate the Immersed Interface Method, we now make a few observations about the vector B used above:

- Only those entries in B corresponding to grid points near the boundary are non-zero.
- For the Dirichlet problem, these non-zero entries are $O(1/h^2)$.
- If $g = 0$, the vector B is identically zero.

What these observations suggest is that B acts like a singular source term with support only near the boundary of our domain. The fact that for the Dirichlet problem, the entries are order $O(1/h^2)$ suggests that this boundary term is behaving like a dipole source. The third observation serves to point out that if our boundary conditions are homogenous, the strength of this dipole is zero.

The inhomogenous boundary term B is not commonly viewed as a singular source term. However, if we consider solving Poisson's equation using boundary integral techniques on an irregular domain, then the first step in solving the problem consists of determining the strengths of singular sources distributed along the boundary of this domain so that the boundary conditions are satisfied. After this singular source term is determined, the solution can be obtained at arbitrary points by evaluating the integral representation, which now involves a known source term, of the original Poisson problem.

The main idea behind the Immersed Interface Method as it is applied to boundary value problems is to determine a vector C, which will play the role of B used in the rectangular case and which can be used to impose boundary conditions on irregular regions. In a manner resembling what we did in the rectangular case described above, this vector C becomes an additional term on the right hand side of a linear system. An analog to the system of equations in (3.6) is the system

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2} = f_{ij} - C_{ij}, \qquad i, j = 2, \ldots N \tag{3.8}$$

where now C_{ij} is an approximation to a singular source term on a the irregular boundary and is only non-zero at the irregular grid points, shown in Figure 3.2(b). The corresponding linear system is given by $AU = F - C + B$, where B is still used to impose boundary conditions on the computational domain. The remainder of this chapter is devoted to deriving this vector C and verifying that numerically, we obtain second order accurate solutions to the PDE.

3.2 *Accurate approximation to derivatives along a curved boundary or interface*

In this section, we derive finite difference formulas for values and derivatives of a function $u(x, y)$ evaluated at an arbitrary point (α_x, α_y). We begin by deriving such formulas for derivatives in Cartesian directions, and later generalize these results to formulas for derivatives in directions normal and tangent to a given curve.

We begin by deriving finite difference stencils for approximations to derivatives in the vector $U_c(x_{ij}) \equiv [u(x_{ij}), u_x(x_{ij}), u_y(x_{ij}), u_{xx}(x_{ij}), u_{yy}(x_{ij}), u_{xy}(x_{ij})]^T$. For this discussion, and throughout the remainder of this thesis, we will use the following general definitions of our grid. First, the grid is assumed to be uniform Cartesian grid with mesh width spacing $\Delta x = \Delta y = h$. Grid values $(x_i, y_j) \equiv x_{ij}$ are given by $x_i = (i-1)\,\Delta x$ and $y_j = (j-1)\,\Delta y$. A discretization of the function $u(x,y)$ is given by $u_{ij} \equiv u(x_i, y_j) \equiv u(x_{ij})$.

Typically, we obtain finite difference formulas by expanding a set of values located at different grid points about the point at which the approximation is desired. We then obtain a linear system, which we invert to obtain stencil weights for the approximation. To illustrate this, we consider the problem of approximating $\nabla^2 u(x_{ij})$. To obtain second order accuracy, we expect to require six points in the neighborhood of the point x_{ij}. These six points make up a *stencil*, which we define by the set

$$S_{ij} \equiv \{(i+k, j+\ell) : (k,\ell) \in S\},$$

where the set S is a subset of the set $\mathbf{S}$ defined by

$$\mathbf{S} = \{(k,\ell) : k, \ell \in \{-1, 0, 1\}\}.$$

A typical example of the set S will be

$$S = \{(0,1), (-1,0), (0,0), (1,0), (0,-1), (1,1)\} \tag{3.9}$$

which correspond to the anchor point offsets for five points of the standard five point stencil plus an extra sixth point in the upper right corner of the stencil box. In Figure 3.3, we show this stencil and label each point with its corresponding offset value. The anchor point corresponds to offset $(0,0)$. Also in Figure 3.3, we show the three canonical stencils arrangements that lead to non-singular matrix systems. An additional nine stencils are obtained by rotating these three. A more detailed discussion of why these are the only three possible arrangements is given in [56].

For what follows, it will be convenient to impose an ordering on the elements of the stencil S_{ij} (or, equivalently, on the set S of offsets). We will call a stencil with such an ordering an *ordered stencil*. To identify the entries of the ordered stencil, we use the subscript (or where necessary, a superscript) p, where $p = 1, \ldots 6$. For example, we have that the p^{th} entry of S is $s_p = (k_p, \ell_p)$.

This ordering will automatically be inherited by all vectors and matrices based on the ordered stencil. For example, we can define a vector u_{ij} by evaluating a function $u(x,y)$ at each grid point in an ordered stencil. The p^{th} entry of this vector is then $u_p \equiv u(x_{i+k_p}, y_{j+\ell_p})$. Presently, we will see that rows of matrices used to derive finite difference approximations will also be defined in terms of an ordered stencil.

To obtain approximations to derivatives of u at points x_{ij}, we expand each of the values in U_{ij} in h about the value $u(x_{ij})$. This is just a Taylor series expansion and

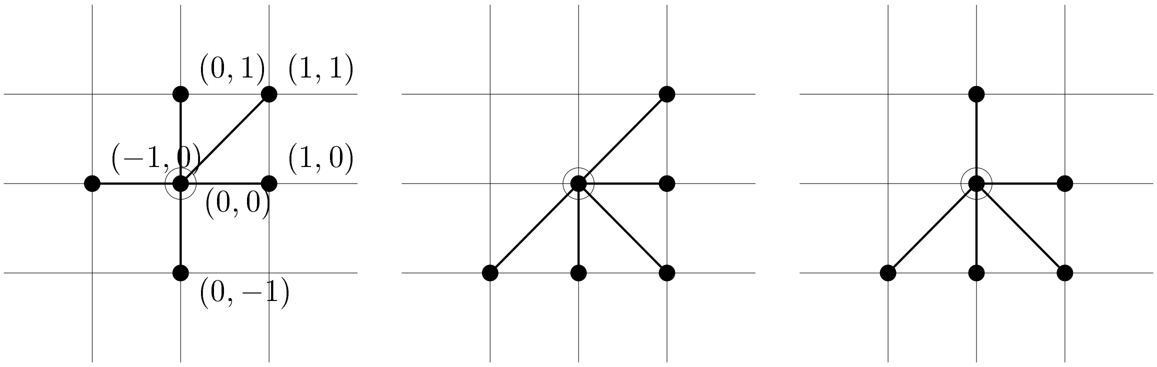


Figure 3.3: Possible six point stencils used to obtain second order approximations to function values and their derivatives. The values in parenthesis are the index offsets for each element of the stencil. The anchor point is labeled with the offset $(0,0)$.

is given by

$$\begin{aligned} u(x_{i+k,j+\ell}) = u(x_{ij}) + u_x(x_{ij})(kh) + u_y({}_{ij})(\ell h) + \frac{1}{2}u_{xx}(x_{ij})(kh)^2 \\ + \frac{1}{2}u_{yy}(x_{ij})(\ell h)^2 + u_{xy}({}_{ij})(kh)(\ell h) + O(h^3) \end{aligned} \tag{3.10}$$

for each $(k,\ell) \in S$. This leads to six equations which we can write as the following linear system:

$$U_{ij} = \Lambda^h \, U_c(x_{ij}) + \tau^1(h) \tag{3.11}$$

where the p^{th} row of Λ^h is given by

$$e_p^T(\Lambda^h) = \begin{pmatrix} 1 & k_p h & \ell_p h & \frac{1}{2}(k_p h)^2 & \frac{1}{2}(\ell_p h)^2 & (k_p h)(\ell_p h) \end{pmatrix}. \tag{3.12}$$

The vector U_c is given by

$$U_c(x_{ij}) \equiv \begin{pmatrix} u(x_{ij}) \\ u_x(x_{ij}) \\ u_y(x_{ij}) \\ u_{xx}(x_{ij}) \\ u_{yy}(x_{ij}) \\ u_{xy}(x_{ij}) \end{pmatrix} \tag{3.13}$$

and the quantity $\tau^1(h)$ in (3.11) is the 6×1 vector of truncation errors associated with each expansion. From the truncated Taylor series expansion, we know that each entry in $\tau^1(h)$ is $O(h^3)$. We use the vector of truncation errors merely to indicate the order of the error and do not specify the coefficients of the truncation errors. We could have, for a particular expansion, that $\tau^1(h) \equiv 0$. Also, we have, for different instances of this vector, only $\tau^1(h) \sim \tau^1(h)$, where we apply the $\sim$ operator in an element-wise fashion. In general, also, we have that $\tau^1(h) \pm \tau^1(h) \sim \tau^1(h)$, not $\tau^1(h) - \tau^1(h) = 0$ or $\tau^1(h) + \tau^1(h) = 2\tau^1(h)$.

By solving the above linear system, we obtain discretizations of each of the six derivatives in the vector $U_c(x_{ij})$ and can write

$$U_c(x_{ij}) = (\Lambda^h)^{-1} U_{ij} + (\Lambda^h)^{-1} \tau^1(h). \tag{3.14}$$

Using Maple, we can determine the order of magnitude for the entries in the vector $\tau^2(h) \equiv (\Lambda^h)^{-1}\tau^1(h)$. These are

$$\tau^2(h) = \begin{pmatrix} O(h^3) \\ O(h^2) \\ O(h^2) \\ O(h) \\ O(h) \\ O(h) \end{pmatrix}. \tag{3.15}$$

For example, the Laplacian is approximated by

$$\nabla^2 u(x_{ij}) = (e_4 + e_5)^T (\Lambda^h)^{-1} U_{ij} + (e_4 + e_5)^T (\Lambda^h)^{-1} \tau^1(h). \tag{3.16}$$

where e_p is the p^{th} column of the 6×6 identity matrix. As expected, the formulas for $\nabla^2 u$ is the usual five point stencil; some of the weights computed from the inversion of Λ^h are zero. Moreover, the coefficients of the 4^{th} and 5^{th} entries of the truncation error vector $\tau^2(h)$ are also zero in this case. Because of this fortuitous cancellation, which is due to the fact that our approximation is at the center of the stencil, we actually have

$$(e_4 + e_5)^T (\Lambda^h)^{-1} \tau^1(h) \sim O(h^2). \tag{3.17}$$

and our approximation to the Laplacian is actually second order accurate.

Had we chosen an expansion point other than the grid point x_{ij}, we would obtain six non-zero stencil weights and find that the approximations to the higher derivatives are only $O(h)$. To interpolate values in the vector U_c at arbitrary points (α_x, α_y) near the point x_{ij}, we expand each of the entries in the vector $U_c(x_{ij})$ about the point $u(\alpha_x, \alpha_y)$ to get the matrix system

$$U_c(x_{ij}) = \Lambda^\alpha\, U_c(\alpha) + \tau^2(h) \tag{3.18}$$

where the matrix Λ^α is given by

$$\Lambda^\alpha \equiv \begin{pmatrix} 1 & (x_i-\alpha_x) & (y_j-\alpha_y) & \frac{1}{2}(x_i-\alpha_x)^2 & \frac{1}{2}(y_j-\alpha_y)^2 & (x_i-\alpha_x)(y_j-\alpha_y) \\ 0 & 1 & 0 & (x_i-\alpha_x) & 0 & (y_j-\alpha_y) \\ 0 & 0 & 1 & 0 & (y_j-\alpha_y) & (x_i-\alpha_x) \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.19}$$

We can obtain this system directly by differentiating the Taylor series expansion for $u(x_{ij})$ expanded about a point (α_x, α_y). When we derive the expressions for the entries in Uc in this manner, we can also easily see that the resulting truncation error is given by $\tau^2(h)$.

Using this definition of $U_c(\alpha)$, we can derive expressions for derivatives in Cartesian directions at point (α_x, α_y) by first writing

$$\begin{aligned} U_{ij} &= \Lambda^h U_c(x_{ij}) + \tau^1(h) \\ &= \Lambda^h \Lambda^\alpha U_c(\alpha) + \Lambda^h \tau^2(h) + \tau^1(h) \end{aligned} \tag{3.20}$$

To obtain approximations to the entries of $U_c(\alpha)$, we write

$$\begin{aligned} U_c(\alpha) &= \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij} + \left(\Lambda^h \Lambda^\alpha\right)^{-1} \left(\Lambda^h \tau^2(h) + \tau^1(h)\right) \\ &= \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij} + \tau^2(h). \end{aligned} \tag{3.21}$$

We will justify shortly that the truncation error vector for this expression for $U_c(\cdot)$ is in fact $\tau^2(h)$.

One consequence of choosing a general point at which to approximate our derivatives is that we no longer get a second order accurate approximation to the higher derivatives. For example, we have that

$$\nabla^2 u(\alpha_x, \alpha_y) = (e_4 + e_5)^T \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij} + O(h) \tag{3.22}$$

since $(e_4 + e_5)^T \tau^2(h) \sim O(h)$. If $(\alpha_x, \alpha_y) = x_{ij}$, the Λ^α is the identity matrix, the coefficients in the truncation error vector $\tau^2(h)$ are identically 0 and (3.21) reduces to (3.14).

We can generalize this formula even further by not restricting ourselves to the (x, y) coordinate directions, but rather by considering derivatives in directions normal and tangential to some curve parameterized as $\sigma(x) \equiv (x(s), y(s))$. What we will be interested in are formulas for u_s, u_n, u_{ss} and $(u_n)_s$. We can obtain these in terms of elements of the array U_c by applying the chain rule and getting, for example

$$\begin{aligned} u_s &= x_s u_x + y_s u_y \\ u_n \equiv \nabla u \cdot \mathbf{n} &= \frac{1}{r}(y_s u_x - x_s u_y) \end{aligned} \tag{3.23}$$

where $r = \sqrt{x_s^2 + y_s^2}$. Note that if s is an arc-length parameterization of the curve σ then $r = 1$, but in general $r \neq 1$. Two more equations derived in this manner are

$$\begin{aligned} u_{ss} &= x_{ss}u_x + y_{ss}u_y + x_s^2 u_{xx} + y_s^2 u_{yy} + 2x_s y_s u_{xy} \\ (u_n)_s &= \frac{1}{r}\left(y_{ss}u_x - x_{ss}u_y + x_s y_s u_{xx} - x_s y_s u_{yy} + (y_s^2 - x_s^2)u_{xy}\right) \end{aligned} \tag{3.24}$$

Our goal is to express all the derivatives tangent and normal to the curve in terms of the six Cartesian derivatives $u, u_x, u_y, u_{xx}, u_{yy}, u_{xy}$. It may not be immediately obvious why we need to do this, but later, we will see that jumps normal and tangent to an irregular boundary are more readily available from given boundary conditions than are their Cartesian counterparts. With that in mind, we now seek two more equations (in addition to the four given above) so that we can form a system of equations expressing derivatives u_s, u_n, etc. in terms of the six Cartesian derivatives. One equation we haven't written down is just the identity $u(x, y) = u(x(s), y(s))$. A logical choice for the sixth equation is an expression for $(u_n)_n$. With a bit of work, we can write down a linearized expression for $(u_n)_n$ by assuming that our interface is actually parameterized in two directions and is given by $\overline{x}(s,\tau) = x(s) + y_s\tau$, $\overline{y}(s,\tau) = y(s) - x_s\tau$. If we consider u as a function of $(\overline{x}(s,\tau), \overline{y}(s,\tau))$, then we could derive the linearized expression

$$(u_n)_n \equiv (u_\tau)_\tau = u_{xx}(y_s)^2 + 2x_s y_s u_{xy} + u_{yy}(x_s)^2 \tag{3.25}$$

We now need to obtain an independent expressions for each of the quantities u, u_n, u_s, u_{ss}, and $(u_n)_s$; otherwise the equations derived above are only identities, and don't lead to meaningful expressions. To obtain these independent expressions for u and u_n at the boundary, we will use boundary conditions supplementing the PDE we wish to solve. The tangential derivatives of these quantities can be interpolated from these values. To get an independent expression for $(u_n)_n$, we need to involve the PDE itself. In general, the PDEs we will be interested in solving have the form:

$$\nabla^2 u + \lambda u = f \tag{3.26}$$

Using the five equations we have plus the expression for $(u_n)_n$ given in (3.25), it is possible to solve for all five Cartesian derivatives in terms of derivatives normal and tangent to the curve. We can then substitute the resulting expressions for each of the five Cartesian derivatives into the PDE in (3.26) and solve for $(u_n)_n$. The resulting expression would be a linear combination of the normal and tangential derivatives and would involve the coefficients c_i from the PDE and f. This expression could then serve as our independent expression for $(u_n)_n$. However, this expression is no doubt quite complicated (we haven't actually computed it, but have verified that such an expression can be obtained), and so we take a much more direct approach. Since we don't actually need the approximation $(u_n)_n$, but only need the second order information from the PDE it would provide, we opt to use the PDE directly as our sixth equation.

With the four equations given in (3.23)-(3.24), the identity $u(x, y) = u(x(s), y(s))$ and the PDE, we now have six equations involving the six Cartesian derivatives in the vector Uc. In matrix form, we write down these six equations as

$$\begin{pmatrix} u(\cdot) \\ u_n(\cdot) \\ u_s(\cdot) \\ u_{ss}(\cdot) \\ (u_n)_s(\cdot) \\ f(\cdot) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & y_s/r & -x_s/r & 0 & 0 & 0 \\ 0 & x_s & y_s & 0 & 0 & 0 \\ 0 & x_{ss} & y_{ss} & x_s^2 & y_s^2 & 2x_s y_s \\ 0 & y_{ss}/r & -x_{ss}/r & x_s y_s/r & -x_s y_s/r & (y_s^2 - x_s^2)/r \\ \lambda & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} u(\cdot) \\ u_x(\cdot) \\ u_y(\cdot) \\ u_{xx}(\cdot) \\ u_{yy}(\cdot) \\ u_{xy}(\cdot) \end{pmatrix} \tag{3.27}$$

which we write as

$$U_s = \Sigma U_c \tag{3.28}$$

where the vector U_s defined as

$$U_s \equiv [u,\ u_n,\ u_s,\ u_{ss},\ (u_n)_s,\ f]^T \tag{3.29}$$

and Σ is the matrix in (3.27). This equation is actually an identity, since there is no truncation error associated with any of the six equations and furthermore, it holds regardless of where U_c and U_s are evaluated, as long as they are evaluated at the same point (α_x, α_y) along the curve $\sigma(s) = (x(s), y(s))$.

With this formulation, we can now write down approximations to the entries in Us. These are given by

$$U_s(\alpha) = \Sigma U_c(\alpha) = \Sigma \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij} + \tau^2(h) \tag{3.30}$$

where we have made use of the sparsity pattern in Σ and the fact all non-zero entries of Σ are $O(1)$ to conclude that

$$\Sigma \tau^2(h) \sim \tau^2(h). \tag{3.31}$$

Using (3.30), we now have, for example, that

$$\begin{aligned} u(\alpha) &= e_1^T \Sigma \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij} + O(h^3) \\ u_n(\alpha) &= e_2^T \Sigma \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij} + O(h^2) \end{aligned} \tag{3.32}$$

As we describe shortly, these formulas could be used to discretize boundary conditions on an irregular boundary described by $\sigma(s) \equiv (x(s), y(s))$.

Before continuing on to solve PDEs in irregular regions, we briefly justify why the truncation error in (3.21) is $\tau^2(h)$. What we wish to show is that for $p = 1, \ldots 6$, we have

$$\left(\Lambda^h \Lambda^\alpha\right)^{-1} \left[\Lambda^h \tau^1(h) + \tau^1(h)\right] \sim \tau^2(h) \tag{3.33}$$

where, just to remind the reader,

$$\tau^2(h) \sim \begin{bmatrix} O(h^3) & O(h^2) & O(h^2) & O(h) & O(h) & O(h) \end{bmatrix}^T$$

and each entry in $\tau^1(h)$ is $O(h^3)$. We rewrite (3.33) as

$$\left(\Lambda^h \Lambda^\alpha\right)^{-1} \left[\Lambda^h \tau^2(h) + \tau^1(h)\right] = \left(\Lambda^\alpha\right)^{-1} \tau^2(h) + \left(\Lambda^h \Lambda^\alpha\right)^{-1} \tau^1(h)$$

Using the symbolic package Maple, we have shown that for each p, $p = 1, \ldots 6$

$$\begin{aligned}
e_1^T \left(\Lambda^h \Lambda^\alpha\right)^{-1} e_p &\sim O(1) \\
e_2^T \left(\Lambda^h \Lambda^\alpha\right)^{-1} e_p &\sim O(h^{-1}) \\
e_3^T \left(\Lambda^h \Lambda^\alpha\right)^{-1} e_p &\sim O(h^{-1}) \\
e_4^T \left(\Lambda^h \Lambda^\alpha\right)^{-1} e_p &\sim O(h^{-2}) \\
e_5^T \left(\Lambda^h \Lambda^\alpha\right)^{-1} e_p &\sim O(h^{-2}) \\
e_6^T \left(\Lambda^h \Lambda^\alpha\right)^{-1} e_p &\sim O(h^{-2}).
\end{aligned}$$

From this, we have that

$$\left(\Lambda^h \Lambda^\alpha\right)^{-1} \tau^1(h) \sim \tau^2(h).$$

Furthermore, by symmetry, we expect the two equations given by

$$U_c(x_{ij}) = \Lambda^\alpha U_c(\alpha) + \tau^2(h)$$

and

$$U_c(\alpha) = \left(\Lambda^\alpha\right)^{-1} \, U_c(x_{ij}) - \left(\Lambda^\alpha\right)^{-1} \, \tau^2(h)$$

to have the same truncation error. So we conclude that

$$\left(\Lambda^\alpha\right)^{-1} \, \tau^2(h) \sim \tau^2(h).$$

We can then write

$$\left(\Lambda^\alpha\right)^{-1} \tau^2(h) + \left(\Lambda^h \Lambda^\alpha\right)^{-1} \tau^1(h) \sim \tau^2(h).$$

This justifies the truncation errors given above.

We remind the reader that our final goal of this chapter is to explain a method which has been come to be known as the Immersed Interface Method, (IIM) and to show how it can be used to solve a variety of linear PDEs. But instead of progressing directly to the details of the IIM, we now take a slight detour and describe a more naive approach to solving boundary value problems in irregular regions. While we don't necessarily advocate this approach, and certainly view the IIM as superior to it, we nonetheless feel that because it is an obvious generalization of what we do in rectangular regions to solve BVPs, it should be described in some detail. Also, it makes nice use of the formulas derived thus far.

3.2.1 Simple approach to solving boundary value problems on irregular regions

With the above stencil formulas for derivatives along curves in two space, we can formulate a method for solving boundary value problems in an irregular region using an underlying Cartesian grid. Once again, the motivation for this approach will come from our understanding of how we solve BVPs problems on rectangular regions. For this discussion, then, we will assume that our physical region is given by the rectangular domain R, which, without loss of generality, is assumed to be square. Grid points in R are labeled $x_{ij}, i, j = 1, \ldots N+1$. As our canonical example, we will use Poisson's equation subject to mixed boundary conditions:

$$\begin{aligned} \nabla^2 u = f, &\quad \text{on } R \\ au + bu_n = c, &\quad \text{on } \partial R \end{aligned} \tag{3.34}$$

where, we assume that a, b and c are constant, and both a and b are non-zero. We use the slightly more complicated boundary conditions to illustrate in a compact way the manner in which both Dirichlet and Neumann conditions are handled.

In the rectangular region, which now includes the boundary points corresponding to $i = 1, N+1$ or $j = 1, N+1$, we discretize the Laplacian using our usual five point stencil. We obtain $(N+1)^2$ equations of the form

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2}, \quad i, j = 1, \ldots N+1 \tag{3.35}$$

This system by itself is underdetermined, as some of the equations involve values outside of R. Therefore, we introduce $4N$ additional equations by discretizing the boundary conditions at points along the boundary. These discrete equations each have the general form

$$au_{bd} + b\frac{u_{ex} - u_{in}}{2h} = c \tag{3.36}$$

where u_{bd} is the value at a grid point on the boundary, u_{in} is a value at a point just inside the domain and u_{ex} is the value at a point just outside of the domain. A typical arrangement is shown in Figure 3.4. To approximate u_n, we just use the usual second order accurate two point stencil. With these equations, we can now close the system in (3.35) and solve for the unknown values $u_{ij}, i, j = 1, \ldots N+1$. While one could accomplish this by solving the extended system involving all $(N+1)^2 + 4N$ equations, what is usually done at this point is to solve for u_{ex} in the discretization of the boundary condition (3.36) and substitute this value back into (3.35). Doing this reduces the system to one involving only the $(N+1)^2$ unknowns in R.

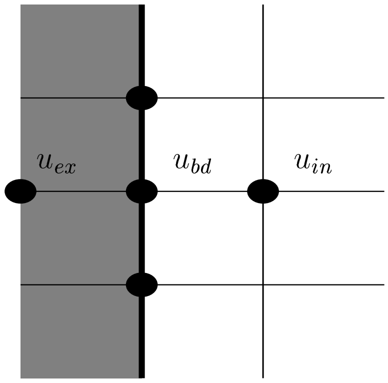


Figure 3.4: Stencil used to approximate boundary conditions in rectangular case.

To illustrate this, we choose a boundary point on the left edge, and let $u_{ex} = u_{0,j}$, $u_{in} = u_{2,j}$ and $u_{bd} = u_{1,j}$. Solving for $u_{0,j}$ in (3.36), we get

$$u_{0,j} = u_{2,j} + \frac{2h}{b}(c - au_{1,j}) \tag{3.37}$$

Substituting this expression into (3.35) gives us a modified equation for the grid point $x_{1,j}$

$$\frac{2u_{2,j} + u_{1,j+1} + u_{1,j-1} - (4 + 2ha/b)u_{1,j}}{h^2} = f - \frac{2c}{b} \tag{3.38}$$

Doing this with each boundary point gives us $4N$ modified equations, which with the remaining equations in the interior of our grid, gives us a total $(N+1)^2$ equations.

While solving BVPs on rectangular regions is considered quite standard, the rectangular case is in fact a very special case for the following reasons:

- Boundary conditions are imposed only at grid points x_{ij} of the computational domain.

- It is trivial to discretize the boundary conditions; the normal derivative u_n is discretized using the usual two point stencil, and u_{bd} requires no special grid discretization, since it is a value at a grid point.
- The discretized equations for the boundary conditions are uncoupled from each other and so it is easy to solve for the unknown values u_{ex}.

Furthermore, in the rectangular region, we have the additional advantage that we can use a fast Poisson solver to solve the resulting $(N+1)^2$ equations.

We now proceed to take advantage of the formulas derived in the earlier section to generalize the above approach to irregular regions. As we will see, however, most of the advantages listed above are lost in the irregular case.

Our physical domain is now assumed to be an irregular region Ω embedded into a rectangular computational domain R. For present purposes, we assume that Ω is the region exterior to a single embedded object. We still use Poisson's equation, subject to the same boundary conditions, as our model problem. Just as in the rectangular case, we discretize Poisson's equation at points inside of our physical domain and get the system of equations

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2} = f_{ij} \quad x_{ij} \in \Omega \tag{3.39}$$

Again, there will be values of u_{ij} used in the above equations that are not in our physical domain, and so we will need to introduce boundary conditions to close the system. If we assume that there are P values u_{ij} used in equations(3.39), but that correspond to grid points x_{ij} that lie outside of Ω, then we will need P additional discretized boundary equations. To obtain these P equations, we first locate P points $\alpha^\mu = (\alpha_x^\mu, \alpha_y^\mu), \mu = 1, \dots P$ on the boundary. Then to discretize the boundary conditions at these points, we use the formulas derived in the previous section. Using the formulas for boundary conditions given in (3.32), our discretization of the boundary conditions in (3.34) at P expansion points α^μ are

$$(ae_1^T + be_2^T)\Sigma \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij}^\mu = c, \quad \mu = 1, \dots P. \tag{3.40}$$

Here, the stencils S_{ij}^μ for the vectors U_{ij}^μ are chosen so that the only points which lie outside the domain are those which appear in (3.39).

Equations in (3.39)-(3.40) can now be solved for the unknowns u_{ij} in Ω. Solving this system, however, is considerably more complicated in the irregular case than it was in the rectangular case. The reasons for this are that

- The discretized equations for the boundary conditions are no longer uncoupled. In order to obtain the boundary values needed by equations in (3.39), we need to solve a $P \times P$ linear system.

- It is no longer straightforward to obtain the modified equations analogous to those in (3.38).

In general, we are not going to be able to write down this linear system explicitly. Instead, we need formulate a linear objective function of the form $C_{ex}U_{ex} + C_{in} = c$, where U_{ex} is a vector in $\mathbb{R}^P$ of unknown values u_{ij} corresponding to grid points outside of the domain Ω, and F involves the right hand side. This linear system can be written down in the form of a subroutine which describes how to apply C_{ex} to the vector U_{ex}. This subroutine is shown in Figure 3.5

Function Cex(Uex)

1. Solve the system of equations

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2} = f_{ij} \quad x_{ij} \in \Omega$$

   for u_{ij}, the solution at points x_{ij} in Ω. Use values in the vector U_{ex} as guesses to the boundary values u_{ij} corresponding to points outside of Ω.

2. The μ^{th} row of $C_{ex}U_{ex} + F$ is then given by

$$\mathrm{Row}_\mu(C_{ex}U_{ex} + F) \equiv (ae_1^T + be_2^T)\Sigma \left(\Lambda^h \Lambda^\alpha\right)^{-1} U_{ij}^\mu$$

   where U_{ij} is a $x \times 1$ vector of values u_{ij} in the stencil S_{ij} needed to approximate the boundary conditions $au + bu_n = c$ at point $\alpha_m u$ on the boundary.

END Function Cex

Figure 3.5: Subroutine used to determine boundary conditions on irregular region.

We can then use the subroutine as an objective function in an iterative method designed to solve the system $C_{ex}U_{ex} + F = c$. In practice, one should rearrange the terms in this equation to get a linear system of the form $C_{ex}U_{ex} = c - F$ and then solve the system using a linear iterative method for the boundary values U_{ex}. These boundary values are then used one more time to solve the system of equations in (3.39).

This approach was implemented and tested and shown to converge at a second order rate. We solve the linear system of equations using the linear iterative method BICGSTAB, and found that the matrix C_{ex} was fairly well conditioned for Dirichlet data, but poorly conditioned for Neumann problems. The code for this problem was implemented as part of a project for a seminar on the ZPL parallel programming language, and has now become part of suite of test programs that the ZPL group is using to develop a sparse array data type.

One potential advantage of the above scheme is that at most, the only interface information that we need is the normal vector at an arbitrary point along the interface. While the above formulas seem to rely on all the information in Σ, defined in (3.27), we could have easily derived an approximation to u_n by writing $u_n = \nabla u \cdot \mathbf{n}$, where $\mathbf{n}$ is a normal to the interface. Of course, if we only have Dirichlet data, then we don't even need the normal vector. The advantage of only requiring this information is that we can easily extend this naive approach to 3d. Another advantage, although not as significant, is that we only solve the equations on the region we are interested in. There is no need to even solve for u_{ij} outside the domain. As a result, the linear system we solve will actually have fewer than $(N+1)^2$ equations. Of course, the significance of this advantage will depend on how Ω is embedded in R.

The main drawbacks to the above approach is that (1), the size of the system C_{ex} is of order $1/h$ and so increases as we refine the grid. Second, the linear system given in Step 1 of the algorithm for computing the linear objective function is not symmetric and cannot be solved using a fast Poisson solver. Since solving a Poisson equation is often just a piece of a much larger computation, it is critical that this computation be as efficient as possible. Therefore, we now seek an approach which will allow us take advantage of fast solvers.

In what follows, we show that by incorporating jumps into the discretization of our PDE, we can obtain an linear system whose size is not directly related to the size of the underlying grid, and which can take advantage of a fast Poisson solver. The remainder of this chapter is devoted to the details of this approach, now known as the Immersed Interface Method.

3.3 The Immersed Interface Method

To arrive at the method for solving PDEs in irregular regions described above, we generalized the idea of solving for unknown boundary values and incorporating them into a discretization of the Laplacian. As we saw, in the irregular case, this meant that we needed to solve a non-trivial linear system to obtain these boundary values and then incorporate them into the equations involving the Laplacian. But this led to a system of modified equations that could not be solved using fast solvers.

As an alternative, we now describe how we can use jump information on the interface to impose boundary conditions. Recall that in the rectangular case, we obtain system of modified equations that takes the form

$$AU = f + B \tag{3.41}$$

were A is the discrete Laplacian, modified near boundary points, and B involves the inhomogeneous terms in the boundary conditions. To generalize this formulation of the BVP to irregular regions, we propose searching for a matrix B that gives us second order accuracy in the irregular case. Unlike in our naive approach, we make the additional assumption that our model equation, Poisson's equation, holds everywhere, not just inside of the domain of interest. We propose adding a correction term C_{ij} to the discrete equations to get

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2} + C_{ij} = f_{ij} + B_{ij}, \quad i,j = 1, \ldots N+1 \tag{3.42}$$

where now, C_{ij} is determined in such a way that the left hand side of the above discretization is a second order approximation to the Laplacian, even if values u_{ij} extend beyond our irregular physical domain. Just as in the rectangular case, where we only obtained modified equations near the interface, the C_{ij}'s will only take on non-zero values near the interface. The main advantage to this formulation is that once we have the values C_{ij}, we include them as source terms (analogous to the B in the rectangular case) on the right hand side of the above discretization, and so can use fast solvers to solve the resulting linear system of equations.

As we will show, the C_{ij}'s are approximations to a linear combination of the jumps in the solution and its derivatives at the interface. That is,

$$\begin{aligned} C_{ij} &= b_1[u] + b_2[u_x] + b_3[u_y] + b_4[u_{xx}] + b_5[u_{yy}] + b_6[u_{xy}] \\ &\equiv \mathcal{B}_c^T\,[U_c] \end{aligned} \tag{3.43}$$

where the b_p's are constants which depend upon the grid point x_{ij}, $\mathcal{B}_c$ is the vector $[b_1, b_2, ...b_6]^T$ and U_c is defined as in (3.13). Later, we will see that it will be convenient to describe C_{ij} in terms of $[U_s] = [[u], [u_n], [u_s], [u_{ss}], [(u_n)_s], f]$ rather than $[U_c]$, and so will also write C_{ij} as

$$\begin{aligned} C_{ij} &= b_1[u] + b_2[u_n] + b_3[u_s] + b_4[u_{ss}] + b_5[(u_n)_s] + b_6[f] + O(h^3) \\ &\equiv \mathcal{B}_s^T\,[U_s] + O(h^3) \end{aligned} \tag{3.44}$$

The entries of $\mathcal{B}_c$ and $\mathcal{B}_s$ are what we now need to determine.

The idea of using jump conditions to modify stencils near boundaries of irregular regions was first described by Mayo [40] and formalized by LeVeque and Li [36, 37]. In their work, they focused mainly on problems involving discontinuous coefficients. Later, Yang, in her PhD thesis described how these ideas could be used to solve boundary value problems in irregular regions [58]. Wiegmann, in his PhD thesis, improved on the stability of the method and developed the Explicit Jump Immersed Interface Method (EJIIM) [56,57]. Also in a PhD thesis, Zhang applied the Immersed Interface Method to the problem of solving the acoustic wave equations in heterogenous materials [59]. In the discontinuous coefficient case, the underlying matrix is no longer symmetric. Adams, in [1], developed a multigrid algorithm for solving the resulting non-symmetric system. The work described here is based loosely on the work of Yang and Wiegmann.

While the Immersed Interface Method was derived in detail in the above cited references, we re-derive the method here in a manner which we hope leads to a fairly algorithmic approach using the IIM. The minor differences between our approach and that taken by Li, and others are

- We do not rely on an arclength parameterization of the interface or boundary curve, but rather allow for general parameterizations. This makes it especially convenient to use high order methods to obtain interface information such as curvature and normals.
- We obtain formulas which make it obvious the role that various elements (jumps, proximity of a grid point to the interface, choice of stencils) play in the construction of the correction term C_{ij},
- The formulas presented lead to a straightforward approach to computer implementation,
- The resulting formulas make it clear how to extend the IIM methods to PDEs other than those described by Yang, Li, Wiegmann, etc.

Here, we only derive the formulas for boundary value problems with constant coefficients; however, we our derivation can be extended to case of variable, even discontinuous coefficients in a very straightforward manner.

We now proceed in three sections. First, we define the function $C(x, y)$, the correction term needed when differencing functions across boundaries. Then we use this corrected function to derive interpolation formulas for the derivatives in the vector U_s used in the last section. Finally, we use these interpolation formulas to discretize boundary conditions at irregular boundaries and illustrate how we can solve elliptic, parabolic and coupled systems of boundary value problems.

3.3.1 The correction term

We begin by supposing that we have two smooth functions u^+ and u^- each defined on the entire rectangular region R. For our purposes, it will be convenient to assume that these functions are in C^∞. However, in practice, infinite smoothness isn't required. Embedded in R is a subdomain Ω^+. The region $R\backslash\Omega^+$ is defined as Ω^-. The boundary between Ω^+ and Ω^- is Γ. For the remainder of this chapter, we will assume $\Omega \equiv \Omega^-$, the irregular region exterior to the single embedded object $\Omega_1 \equiv \Omega^+$.

Given functions u^+ and u^- defined over all of R, we define a function u as

$$u(x,y) = \begin{cases} u^+(x,y) & \text{if } (x,y) \in \Omega^+ \\ u^-(x,y) & \text{if } (x,y) \in \Omega^- \end{cases} . \tag{3.45}$$

In general, we expect that the function $u(x,y)$ will have discontinuities in its value and derivatives at the boundary Γ. We call these discontinuities *jumps* in $u(x,y)$ and define a few of them at a point $\alpha = (\alpha_x, \alpha_y)$ on the interface as

$$\begin{aligned} [u] &\equiv u^+(\alpha_x,\alpha_y) - u^-(\alpha_x,\alpha_y) \\ [u_x] &\equiv u_x^+(\alpha_x,\alpha_y) - u_x^-(\alpha_x,\alpha_y) \\ [u_y] &\equiv u_y^+(\alpha_x,\alpha_y) - u_y^-(\alpha_x,\alpha_y) \\ [u_{xx}] &\equiv u_{xx}^+(\alpha_x,\alpha_y) - u_{xx}^-(\alpha_x,\alpha_y) \\ [u_{yy}] &\equiv u_{yy}^+(\alpha_x,\alpha_y) - u_{yy}^-(\alpha_x,\alpha_y) \\ [u_{xy}] &\equiv u_{xy}^+(\alpha_x,\alpha_y) - u_{xy}^-(\alpha_x,\alpha_y) \end{aligned} \tag{3.46}$$

Since each of the functions u^+ and u^- are smooth (in C^∞, for our purposes), we can expand each of them in a Taylor series about a point $\alpha = (\alpha_x, \alpha_y)$ on the interface. Assuming that these series converge for all $(x,y) \in \Omega$ [1], for any choice of expansion point α, we can subtract the expansion for u^- from that for u^+ and define $C(x,y)$, the difference between these two functions as

$$\begin{aligned} C(x,y) &\equiv u^+(x,y) - u^-(x,y) \\ &= [u] + [u_x](x-\alpha_x) + [u_y](y-\alpha_y) + \frac{1}{2}[u_{xx}](x-\alpha_x)^2 + \\ &\quad \frac{1}{2}[u_{yy}](y-\alpha_y)^2 + [u_{xy}](x-\alpha_x)(y-\alpha_y) + O(h^3) \end{aligned} \tag{3.47}$$

We use this definition of the correction term $C(x,y)$ to write down the following identities for u^+ and u^- in terms of $u(x,y)$ and $C(x,y)$.

$$u^+(x,y) \equiv \begin{cases} u(x,y) + C(x,y) & (x,y) \in \Omega^- \\ u(x,y) & (x,y) \in \Omega^+ \end{cases} \tag{3.48}$$

[1] The requirement that the Taylor series converge independent of α for all $(x,y) \in \Omega$ is really only made for the sake of the present discussion; in practice we only retain a few of the terms in $C(x,y)$ and choose expansion points close enough to (x,y) to ensure convergence of the truncated series. This truncated approximation will of course depend on the choice of α and only approximates the difference $u^+ - u^-$.

Similarly, we have that

$$u^-(x,y) \equiv \begin{cases} u(x,y) & (x,y) \in \Omega^- \\ u(x,y) - C(x,y) & (x,y) \in \Omega^+ \end{cases} \tag{3.49}$$

The reason for writing these explicitly is that in practice, we only have the discontinuous function $u(x,y)$ and its jumps along Γ. However, we will want to be able to differentiate u at points near the interface using finite differences. Since we can't use finite differences across the interface on the discontinuous function u and hope to obtain any accuracy, we will want to recover the smooth function u^+ or u^- from u and differentiate that function instead. This function that we recover is given by

$$\widetilde{u}(x,y;\xi,\eta) \equiv \begin{cases} u^+(x+\xi,y+\eta) & (x,y) \in \Omega^+ \\ u^-(x+\xi,y+\eta) & (x,y) \in \Omega^- \end{cases} \tag{3.50}$$

and can be thought of simply as a smooth extension of the discontinuous function $u(x,y)$.
Using the definition for u^+ and u^-, we now have, for $x_{ij} \in \Omega^+$, that

$$\widetilde{u}(x,y;\xi,\eta) = \begin{cases} u(x+\xi,y+\eta) + C(x+\xi,y+\eta), & (x+\xi,y+\eta) \in \Omega^- \\ u(x+\xi,y+\eta), & (x+\xi,y+\eta) \in \Omega^+ \end{cases} \tag{3.51}$$

and for $(x,y) \in \Omega^-$ as

$$\widetilde{u}(x,y;\xi,\eta) = \begin{cases} u(x+\xi,y+\eta), & (x+\xi,y+\eta) \in \Omega^- \\ u(x+\xi,y+\eta) - C(x+\xi,y+\eta), & (x+\xi,y+\eta) \in \Omega^+ \end{cases} \tag{3.52}$$

In Figure 3.6, we show a 1d illustration of the function $\widetilde{u}$ as an extension of a discontinuous function u.

We can now use $\widetilde{u}$ to obtain the correction term C_{ij} for the discrete Laplacian. Since $\widetilde{u}$ is a smooth function and equal to either u^+ or u^-, depending on whether $x_{ij} \in \Omega^+$ or Ω^-, we can difference it using the standard five point stencil and obtain an approximation to the Laplacian that will now involve a correction term made up of corrected points in the stencil.

So far, we have assumed that $C(x,y)$ is exactly $u^+ - u^-$ and that $\widetilde{u}$ is exactly u^+ or u^-. No discrete approximations have been made. In practice, however, we only have jumps up to second derivatives, and so we will want to consider a truncated version of the correction term $C(x,y)$. In general, we define $C_i(x,y) \equiv C(x,y) + O(h^{i+1})$. For $i = 0, 1, 2$, we have for example,

$$\begin{aligned} C_0(x,y) &= [u] \\ C_1(x,y) &= [u] + [u_x](x-\alpha_x) + [u_y](y-\alpha_y) \\ C_2(x,y) &= [u] + [u_x](x-\alpha_x) + [u_y](y-\alpha_y) + \frac{1}{2}[u_{xx}](x-\alpha_x)^2 + \\ &\quad \frac{1}{2}[u_{yy}](y-\alpha_y)^2 + [u_{xy}](x-\alpha_x)(y-\alpha_y) \end{aligned} \tag{3.53}$$

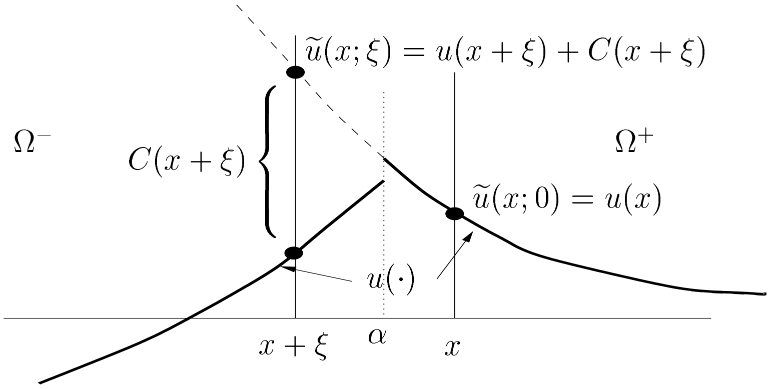


Figure 3.6: Diagram of 1d function $\widetilde{u}(x;\xi)$ shown as a smooth extension of the discontinuous function $u(\cdot)$. The correction term $C(x)$ is defined so that $\widetilde{u}(x_i;-\xi) = u(x+\xi) + C(x+\xi)$.

From the jump information that we can readily obtain from the PDE and boundary conditions, we have enough information to use the truncated function $C_2(x,y)$. It will also be convenient to introduce a functions $\widetilde{C}(x,y;\xi,\eta)$ and $\widetilde{C}_i(x,y;\xi,\eta)$, which we define as

$$
\begin{aligned}
\widetilde{C}(x,y;\xi,\eta) &\equiv C(x+\xi, y+\eta) \\
\widetilde{C}_i(x,y;\xi,\eta) &\equiv C_i(x+\xi, y+\eta)
\end{aligned}
\tag{3.54}
$$

Example

We are now ready to illustrate how these correction terms $C(x,y)$ or $C_2(x,y)$ can be incorporated into finite difference stencils to accurately approximate derivatives across interfaces. For this example, we choose a stencil S for the point x_{ij} as

$$
S = \{(0,0),(0,1),(-1,0),(1,0),(0,-1),(1,-1)\} \tag{3.55}
$$

We assume that for $p = 1,2,3$, $x_p \in \Omega^+$ and for $p = 4,5,6$, $x_p \in \Omega^-$. This stencil is shown in Figure 3.7. We also define $u_{ij} \equiv u(x_i,y_j)$ and $\widetilde{u}(x_{i+k}, y_{j+\ell}; kh, \ell h)$. Assume we are given a function $u(x,y)$. Assuming that $u_{ij} = u(x_i,y_j)$ exactly (no discrete approximations), we can approximate $\nabla^2 u(x_{ij}) \equiv u(x_1)$, $x_1 \in \Omega^+$ to second order

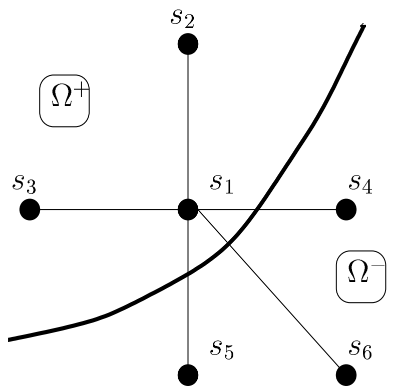


Figure 3.7: Stencil used in example illustrating how correction terms arise. Function values at stencil points in Ω^- will be corrected so that resulting function is smooth.

using

$$
\begin{aligned}
\nabla^2 u(x_{ij}) &= \frac{\widetilde{u}_{i+1,j} + \widetilde{u}_{i-1,j} + \widetilde{u}_{i,j-1} + \widetilde{u}_{i,j+1} - 4\widetilde{u}_{ij}}{h^2} + O(h^2) \\
&= \frac{u_{i+1,j} + \widetilde{C}(x_i, y_j; h, 0) + u_{i-1,j}}{h^2} + \\
&\qquad\qquad \frac{u_{i,j-1} + \widetilde{C}(x_i, y_j; 0, -h) + u_{i,j+1} - 4u_{ij}}{h^2} + O(h^2) \\
&= \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}}{h^2} \\
&\qquad\qquad + \frac{1}{h^2}\left(\widetilde{C}(x_i, y_j; h, 0) + C(x_i, y_j; 0, -h)\right) + O(h^2) \\
&= \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}}{h^2} \\
&\qquad\qquad + \frac{1}{h^2}\left(\widetilde{C_2}(x_i, y_j; h, 0) + \widetilde{C_2}(x_i, y_j; 0, -h)\right) + O(h)
\end{aligned} \tag{3.56}
$$

The correction term C_{ij} for this particular approximation located at the anchor point s_1 is

$$
C_{ij} \equiv \frac{1}{h^2}\left(\widetilde{C_2}(x_i, y_j; h, 0) + \widetilde{C_2}(x_i, y_j; 0, -h)\right) \tag{3.57}
$$

It is easy to verify that

$$\begin{aligned} C_{ij} &= \frac{1}{h^2}\left(\widetilde{C_2}(x_i, y_j; h, 0) + \widetilde{C_2}(x_i, y_j; 0, -h)\right) \\ &= \frac{2}{h^2}[u] + \frac{1}{h}[u_x] - \frac{1}{h}[u_y] + \frac{1}{2}[u_{xx}] + \frac{1}{2}[u_{yy}] \\ &= \left(\frac{2}{h^2}, \frac{1}{h}, \frac{-1}{h}, \frac{1}{2}, \frac{1}{2}, 0\right)^T [U_c] \end{aligned} \tag{3.58}$$

From this, we can easily see that the vector $\mathcal{B}_c$ introduced in (3.43) is given by

$$\mathcal{B}_c = \left(\frac{2}{h^2}, \frac{1}{h}, \frac{-1}{h}, \frac{1}{2}, \frac{1}{2}, 0\right) \tag{3.59}$$

In the above example, we showed how we could derive an expression for C_{ij} for a particular grid point near an interface. Now, we want to set up some notation and derive general formulas for this correction term. We still make the assumption that we have all the jump information we need. Later, we will describe how to set up a linear system to obtain jump information that is missing when we solve boundary value problems.

To compute general formulas for the correction term, we first need several definitions, which we list now.

Definition 1. *The region Ω_{ij} is defined as*

$$\Omega_{ij} = \begin{cases} \Omega^+ & x_{ij} \in \Omega^+ \\ \Omega^- & x_{ij} \in \Omega^- \end{cases}, \tag{3.60}$$

Definition 2. . *The variable $\sigma_{ij}^{k,\ell}$, which gives us the sign on the correction term, is defined by*

$$\sigma_{ij}^{k,\ell} = \begin{cases} 1 & \text{if } x_{i+k,j+\ell} \in \Omega^- \text{ and } x_{ij} \in \Omega^+ \\ -1 & \text{if } x_{i+k,j+\ell} \in \Omega^+ \text{ and } x_{ij} \in \Omega^- \\ 0 & \text{if } x_{i+k,j+\ell} \in \Omega_{ij} \end{cases} \tag{3.61}$$

where (k_p, ℓ_p) is the p^{th} entry of the six point stencil S used at point x_{ij}. Using this sign quantity, we could define $\widetilde{u}(x, y; \xi, \eta)$ in (3.51) and (3.52) at the p^{th} point in the stencil S_{ij} as

$$\widetilde{u}(x_i, y_j; k_p h, \ell_p h) = u(x_{i+k_p}, y_{j+\ell_p}) + \sigma_{ij}^{k_p,\ell_p}\widetilde{C_2}(x_i, y_j; k_p, \ell_p) + O(h^3) \tag{3.62}$$

Definition 3. *The vector* $\widetilde{P}(x, y; \xi, \eta)$ *is defined in terms of the function* $\widetilde{C_2}(x, y; \xi, \eta)$ *from (3.54) so that*

$$\widetilde{C_2}(x_i, y_j; \xi, \eta) \equiv \widetilde{P}(x_i, y_j; \xi, \eta)^T [U_c(\alpha)],$$

where

$$\widetilde{P}(x_i, y_j; \xi, \eta) \equiv \begin{pmatrix} 1 \\ (x_i + \xi - \alpha_x^{\mu(i,j)}) \\ (y_j + \eta - \alpha_y^{\mu(i,j)}) \\ (x_i + \xi - \alpha_x^{\mu(i,j)})^2/2 \\ (y_j + \eta - \alpha_y^{\mu(i,j)})^2/2 \\ (x_i + \xi - \alpha_x^{\mu(i,j)})(y_j + \eta - \alpha_y^{\mu(i,j)}). \end{pmatrix} \tag{3.63}$$

The vector $[U_c(\alpha)]$ *is the vector of jumps* $[U_c(\alpha)] \equiv U_c^+(\alpha) - U_c^-(\alpha)$*, where, as defined in (3.13),* $U_c(\cdot) \equiv [u(\cdot), u_x(\cdot), u_y(\cdot), u_{xx}(\cdot), u_{yy}(\cdot), u_{xy}(\cdot)]^T$*. In the following discussion, we may drop the explicit dependence on* α *and just write* $[U_c]$ *instead of* $[U_c(\alpha)]$*; however, it is always assumed that the entries of the jump vector* $[U_c]$ *are to be evaluated at a point* α *on the interface.*

Definition 4. *The matrix* $\Phi_{ij} \in \mathbb{R}^{6\times 6}$ *is defined as*

$$e_p^T(\Phi_{ij}) = \sigma_{ij}^{k_p,\ell_p} \widetilde{P}(x_i, y_j; k_p h, \ell_p h)^T. \tag{3.64}$$

The sign quantity σ *is used to get the correct sign on the correction term that appears in equations (3.51) and (3.52). Note that this sign will be zero for those rows corresponding to grid points in the stencil which lie on the same side of the interface as the anchor point.*

Definition 5. *The vector* $\widetilde{U}_{ij} \in \mathbb{R}^{6\times 1}$ *is defined so that*

$$e_p^T \widetilde{U}_{ij} \equiv \widetilde{u}(x_i, y_j; k_p h, \ell_p h) \tag{3.65}$$

where (k_p, ℓ_p) *is the* p^{th} *entry in an ordered stencil* S_{ij}*.*

With the definitions above, we can now write out the six equations corresponding to (3.62) as

$$\widetilde{U}_{ij} = U_{ij} + \Phi_{ij}[U_c] + \tau^1(h) \tag{3.66}$$

where, as before, the entries of $\tau^1(h)$ are each $O(h^3)$.

We now derive the correction term C_{ij} used to correct the Laplacian. Since $\widetilde{u}(x_i, y_j; \xi, \eta)$ is smooth, and has continuous first and second derivatives, we can expand it about a grid point x_{ij} not exactly on the boundary, and write, for example

$$\begin{aligned}\widetilde{u}(x_i, y_j; k_p h, \ell_p h) &= u(x_i, y_j) + u_x(x_i, y_j)(k_p h) + u_y(x_i, y_j)(\ell_p h) + \\ &\quad \frac{1}{2}u_{xx}(x_i, y_j)(k_p h)^2 + \frac{1}{2}u_{yy}(x_i, y_j)(\ell_p h)^2 + \\ &\quad u_{xy}(x_i, y_j)(k_p h)(\ell_p h) + O(h^3)\end{aligned} \tag{3.67}$$

We can use the derivatives of $u(x, y)$ here since we assume that x_{ij} is not exactly on the interface and so these derivatives are well defined. All six equations of the above can be written out as a linear system using Λ^h, defined in (3.12). This system is

$$\widetilde{U}_{ij} = \Lambda^h U_c(x_{ij}) + \tau^1(h) \tag{3.68}$$

Using (3.66), we can then write

$$\begin{aligned}U_c(x_{ij}) &= (\Lambda^h)^{-1}\widetilde{U}_{ij} + (\Lambda^h)^{-1}\tau^1(h) \\ &= (\Lambda^h)^{-1}\left[U_{ij} + \Phi_{ij}[U_c] + \tau^1(h)\right] + \tau^1(h) \\ &= (\Lambda^h)^{-1} U_{ij} + (\Lambda^h)^{-1}\Phi_{ij}[U_c] + \tau^2(h)\end{aligned} \tag{3.69}$$

where, recalling that $\tau^2(h) \sim [O(h^3), O(h^2), O(h^2), O(h), O(h), O(h)]$, we make use of the fact that

$$\begin{aligned}(\Lambda^h)^{-1}\tau^1(h) + \tau^1(h) &\sim \tau^2(h) + \tau^1(h) \\ &\sim \tau^2(h).\end{aligned} \tag{3.70}$$

As before, we can write down an approximation to the Laplacian as

$$\begin{aligned}\nabla^2 u(x_i, y_j) &= (e_4 + e_5)^T \left[(\Lambda^h)^{-1} U_{ij} + (\Lambda^h)^{-1}\Phi_{ij}[U_c]\right] + O(h) \\ &= (e_4 + e_5)^T(\Lambda^h)^{-1} U_{ij} + (e_4 + e_5)^T(\Lambda^h)^{-1}\Phi_{ij}[U_c] + O(h)\end{aligned} \tag{3.71}$$

This is just the usual five point stencil plus a correction term and can be written

$$\nabla^2 u(x_i, y_j) = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}}{h^2} + C_{ij} + O(h) \tag{3.72}$$

where now, the general formula for C_{ij} used is given by

$$C_{ij} \equiv (e_4 + e_5)^T(\Lambda^h)^{-1}\Phi_{ij}[U_c] \tag{3.73}$$

It is easy to verify that for the particular example given in 3.3.1, this general formula gives us

$$\begin{aligned} C_{ij} &= \frac{1}{h^2}\left(\widetilde{C_2}(x_i, y_j; h, 0) + \widetilde{C_2}(x_i, y_j; 0, -h)\right) \\ &= \frac{1}{h^2}\left(\sigma_{ij}^{1,0}\widetilde{P}(x_i, y_j; h, 0) + \sigma_{ij}^{0,-1}\widetilde{P}(x_i, y_j; 0, -h)^T\right)^T [U_c] \\ &= \left(\frac{2}{h^2}, \frac{1}{h}, \frac{-1}{h}, \frac{1}{2}, \frac{1}{2}, 0\right)^T [U_c] \\ &= \frac{2}{h^2}[u] + \frac{1}{h}[u_x] - \frac{1}{h}[u_y] + \frac{1}{2}[u_{xx}] + \frac{1}{2}[u_{yy}] \end{aligned} \tag{3.74}$$

as we verified in (3.58). The advantage of these formulas is that given a six point stencil near an interface, a nearby expansion point (α_x, α_y) on the interface, and information about which points in the stencil are in Ω^+ and which are in Ω^-, one can easily automate the computation of the coefficients $(e_4 + e_5)^T(\Lambda^h)^{-1}\Phi_{ij}$ of the jump vector $[U_c]$. First, one computes $\sigma_{ij}^{k,\ell}$ for each point (k, ℓ) in the stencil S_{ij}. Then one forms the rows in the matrix Φ_{ij} by multiplying $\sigma_{ij}^{k,\ell}$ by row vectors $\widetilde{P}(x_i, y_j; k_p h, \ell_p h)$. Finally, the matrix Λ^h is formed and inverted and multiplied by Φ_{ij}. The coefficients of $[U_c]$ are given by the sum of the 4^{th} and 5^{th} rows of the product $(\Lambda^h)^{-1}\Phi_{ij}$.

If we want to correct the general linear operator

$$L(u) \equiv \nabla^2 u + \lambda u \tag{3.75}$$

we use the general formula

$$C_{ij} \equiv (\lambda e_1 + e_4 + e_5)^T (\Lambda^h)^{-1}\Phi_{ij}[U_c] \tag{3.76}$$

Again, the coefficients of the jump vector $[U_c]$ are just linear combinations of the rows of the matrix $(\Lambda^h)^{-1}\Phi_{ij}$.

To summarize, we now discretize the PDE $\nabla^2 u = f$ in the domain $\Omega \equiv \Omega^-$. Using (3.72), we have

$$\frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}}{h^2} + C_{ij} = f_{ij} + B_{ij}, \qquad i, j = 2, \ldots N \tag{3.77}$$

where B_{ij} is used to impose boundary conditions on the boundary of the computational domain. Because of the way in which C_{ij} is constructed, it is automatically zero away from the interface (at regular points). If the point x_{ij} at which we discretize the PDE is more than a few mesh widths away from the interface, then the matrix Φ_{ij} is the zero matrix, since in this situation $x_{i+k_p, j+\ell_p} \in \Omega_{ij}$ for $p = 1, \ldots 6$ and, from (3.61), $\sigma_{ij}^{k_p,\ell_p} = 0, p = 1, \ldots 6$. There is no contribution from the jumps $[U_c]$ at the interface.

At this point, the equations given in (3.77) cannot be solved to get values u_{ij} in R, since we don't have all the jumps needed to form $[U_c]$, and hence the correction term C_{ij}. To obtain these unknown jumps, we need additional equations, which we obtain by discretizing the boundary conditions.

3.3.2 Discretizing boundary conditions using jumps

In the last section, we derived a formula for C_{ij}, the term needed to correct the usual five point stencil approximation to the Laplacian near a discontinuity. This term, which we showed is a linear combination of jumps in the vector $[U_c(\alpha)]$, is included as a source term in the discretization of the PDE at all irregular points.

As we pointed out, the set of equations given in, for example (3.77) is incomplete; we don't have all the jump information needed to form the vector $[U_c]$. We close this system of equations by discretizing the boundary conditions using the same jump information. The manner in which we do this is the topic of this section.

For the present discussion, we assume that the boundary conditions are of either Neumann or Dirichlet type and have the form

$$\begin{array}{ll} \text{(Dirichlet)} & u(\alpha) = c(\alpha) \\ \text{(Neumann)} & u_n(\alpha) = c(\alpha) \end{array} \tag{3.78}$$

where α is a point along the interface. We can easily handle more complicated mixed boundary conditions of the form $a(\alpha)u(\alpha) + b(\alpha)u_n(\alpha) = c(\alpha)$, but to keep the discussion simple, we don't consider this case here.

We now discuss briefly how we use jump information to obtain accurate discretizations of these boundary conditions at a point α on the interface. In a manner analogous to what was done in (3.68), we use equation (3.21) to expand the entries of $\widetilde{U}_{ij}$ in a Taylor series about the point α. In vector form, this equation is essentially that given in (3.20) and is

$$\widetilde{U}_{ij} = \left(\Lambda^h \Lambda^\alpha\right) U_c^{\pm}(\alpha) + \tau^2(h) \tag{3.79}$$

where now we have used the superscript $\pm$ to indicate that we are interpolating the function $u^+(x,y)$ if $x_{ij} \in \Omega^+$ or the function $u^-(x,y)$ if $x_{ij} \in \Omega^-$. Solving the above system for $U_c^{\pm}(\alpha)$ and using the definition of $\widetilde{U}_{ij}$ given in (3.66), we can derive interpolation formulas for the entries of $U_c^{\pm}(\alpha)$. These are

$$\begin{aligned} U_c^{\pm}(\alpha) &= (\Lambda^h\Lambda^\alpha)^{-1}\widetilde{U}_{ij} + \tau^2(h) \\ &= (\Lambda^h\Lambda^\alpha)^{-1}\left[U_{ij} + \Phi_{ij}[U_c] + \tau^1(h)\right] + \tau^2(h) \\ &= (\Lambda^h\Lambda^\alpha)^{-1} U_{ij} + (\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}[U_c] + \tau^2(h) \end{aligned} \tag{3.80}$$

To get derivatives in the normal and tangential directions so that we can discretize boundary conditions, we use the fact that $U_s \equiv \Sigma U_c$ (see (3.28)) and multiply both sides of the above expression for U_c by Σ. We also make the substitution $[U_c] = \Sigma^{-1}[U_s]$ and get

$$U_s^{\pm}(\alpha) \;=\; \Sigma(\Lambda^h\Lambda^\alpha)^{-1} U_{ij} + \Sigma(\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}\Sigma^{-1}[U_s] + \tau^2(h) \tag{3.81}$$

Approximations to $u(\alpha)$ and $u_n(\alpha)$, the quantities needed to approximate the Dirichlet and Neumann boundary conditions given in (3.78), are then

$$\begin{aligned} u^{\pm}(\alpha) &= e_1^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}U_{ij} + e_1^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}\Sigma^{-1}[U_s] + O(h^3) \\ u_n^{\pm}(\alpha) &= e_2^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}U_{ij} + e_2^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}\Sigma^{-1}[U_s] + O(h^2) \end{aligned} \tag{3.82}$$

These expressions differ from those in (3.32) only in that they involve non-zero jumps; if the function $u(x,y)$ and its derivatives are continuous across the interface, these two sets of interpolation formulas are identical.

When we discretized the Laplacian in (3.71), we assumed that we wanted to approximate $\nabla^2 u$ at a grid point x_{ij}. To form the correction term C_{ij}, we had to choose an expansion point α close to x_{ij} and form Φ_{ij} using this expansion point. We form this correction term at all irregular points, whether or not they are in our domain of interest. When we discretize boundary conditions, the situation is slightly different. Here, we assume that we have chosen M points $\alpha_k, k = 1, M$ along the interface at which we wish to impose boundary conditions. To discretize the boundary condition at α_k, we need to find a grid point x_{ij} near α_k which is in the domain Ω^+ or Ω^- in which we want to solve the PDE. With this grid point x_{ij} and α_k, we can then form the matrices Φ_{ij}, Σ, Λ^h and Λ^α needed to form the coefficients of the vectors U_{ij} and $[U_s]$.

3.3.3 A simplified notation

The formulas for the approximation to our model problem, Poisson's equation, and the two types of boundary conditions we are considering are

$$\begin{aligned} \nabla^2 u(x_i,y_j) &= (e_4+e_5)^T(\Lambda^h)^{-1}U_{ij} + (e_4+e_5)^T(\Lambda^h)^{-1}\Phi_{ij}\Sigma^{-1}[U_s] + O(h) \\ u^{\pm}(\alpha) &= e_1^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}U_{ij} + e_1^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}\Sigma^{-1}[U_s] + O(h^3) \\ u_n^{\pm}(\alpha) &= e_2^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}U_{ij} + e_2^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}\Sigma^{-1}[U_s] + O(h^2) \end{aligned}$$

where we have replaced $[U_c]$ in (3.72) by $\Sigma^{-1}[U_s]$. Our motivation for doing this will be evident shortly.

These formulas get rather unwieldy so we now introduce several definitions to simplify the notation.

Definition 6. *The three vectors* $\mathbf{w}$, $\mathbf{v}$ *and* $\mathbf{r}$ *all in* $\mathbb{R}^6$ *are defined as*

$$\begin{aligned} \mathbf{w} &\equiv [[u],\ 0,\ [u_s],\ [u_{ss}],\ 0,\ 0]^T \\ \mathbf{v} &\equiv [0,\ [u_n],\ 0,\ 0,\ [(u_n)_s],\ 0]^T \\ \mathbf{r} &\equiv [0,\ 0,\ 0,\ 0,\ 0,\ [f]]^T \end{aligned}$$

so that $[U_s] = \mathbf{w} + \mathbf{v} + \mathbf{r}$.

Definition 7. *The* coefficient vectors $\mathcal{A}$, $\mathcal{D}'_{ij}$, $\mathcal{D}_{ij}$ *and* $\mathcal{D}^H_{ij}$ *are given by*

$$\begin{aligned}
\mathcal{A} &\equiv (e_4 + e_5)^T(\Lambda^h)^{-1} \\
\mathcal{D}'_{ij} &\equiv [b_1,\ 0,\ b_3,\ b_4,\ 0,\ 0] \\
\mathcal{D}_{ij} &\equiv [0,\ b_2,\ 0,\ 0,\ b_5,\ 0] \\
\mathcal{D}^H_{ij} &\equiv [0,\ 0,\ 0,\ 0,\ 0,\ b_6]
\end{aligned}$$

where b_p *is the* p^{th} *entry of* $\mathcal{B}$*, given by*

$$\mathcal{B} \equiv (e_4 + e_5)^T(\Lambda^h)^{-1}\Phi_{ij}\Sigma^{-1}$$

This is exactly the vector $\mathcal{B}_s$ *introduced in (3.44). From now on, however, we drop the subscript* s *and assume that all jumps are in the normal and tangential directions.*

Definition 8. *The vectors* $\mathcal{I}_k$ *and* $\mathcal{I}_{n,k}$*, both in* $\mathbb{R}^6$ *are defined as*

$$\begin{aligned}
\mathcal{I}_k &\equiv e_1^T\Sigma\left(\Lambda^h\Lambda^\alpha\right)^{-1} \\
\mathcal{I}_{n,k} &\equiv e_2^T\Sigma\left(\Lambda^h\Lambda^\alpha\right)^{-1}.
\end{aligned}$$

Definition 9. *Vectors* $\mathcal{I}^w_k$*,* $\mathcal{I}^v_k$*, and* $\mathcal{I}^f_k$*, all in* $\mathbb{R}^6$ *are given by*

$$\begin{aligned}
\mathcal{I}^w_k &\equiv [b'_1,\ 0,\ b'_3,\ b'_4,\ 0,\ 0] \\
\mathcal{I}^v_k &\equiv [0,\ b'_2,\ 0,\ 0,\ b'_5,\ 0] \\
\mathcal{I}^f_k &\equiv [0,\ 0,\ 0,\ 0,\ 0,\ b'_6]
\end{aligned}$$

where $\mathcal{B}' \in \mathbb{R}^6$ *is given by*

$$\mathcal{B}' \equiv e_1^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}.\Sigma^{-1}$$

Vectors $\mathcal{I}^w_{n,k}$*,* $\mathcal{I}^v_{n,k}$ *and* $\mathcal{I}^f_{n,k}$ *are defined analogously, using* $\mathcal{B}''$ *in place of* $\mathcal{B}$*, where*

$$\mathcal{B}'' \equiv e_2^T\Sigma(\Lambda^h\Lambda^\alpha)^{-1}\Phi_{ij}\Sigma^{-1}.$$

Using these definitions, we can now write the approximation to the Laplacian as

$$\nabla^2 u(x_i, y_j) = \mathcal{A}\,U_{ij} + \mathcal{D}'_{ij}\mathbf{w}_{ij} + \mathcal{D}_{ij}\mathbf{v}_{ij} + \mathcal{D}^H_{ij}\mathbf{r}_{ij} + O(h). \tag{3.83}$$

Similarly, the discretization of the two types of boundary conditions at the k^{th} boundary point α_k can be written as

$$\begin{aligned}
u^\pm(\alpha_k) &= \mathcal{I}_k U_{ij} + \mathcal{I}^w_k\mathbf{w}_k + \mathcal{I}^v_k\mathbf{v}_k + \mathcal{I}^f_k\mathbf{r}_k + O(h^3) \\
u^\pm_n(\alpha_k) &= \mathcal{I}_{n,k}U_{ij} + \mathcal{I}^w_{n,k}\mathbf{w}_k + \mathcal{I}^v_{n,k}\mathbf{v}_k + \mathcal{I}^f_{n,k}\mathbf{r}_k + O(h^2)
\end{aligned} \tag{3.84}$$

The three terms in the discretization of the PDE can be viewed as the three singular terms

$$
\begin{aligned}
\mathcal{D}'_{ij}\mathbf{w} &\approx \int_\Gamma c'(s)\delta'(x_{ij} - \sigma(s))\,ds \\
\mathcal{D}_{ij}\mathbf{v} &\approx \int_\Gamma c(s)\delta(x_{ij} - \sigma(s))\,ds \\
\mathcal{D}^H_{ij}\mathbf{r} &\approx f^- + [f]\,\widetilde{\chi}_{\Omega^+}(x_{ij})
\end{aligned}
\tag{3.85}
$$

where $\delta'()$ is the dipole function, $\delta()$ is the Dirac delta function and $\widetilde{\chi}_{\Omega^+} + \chi_{\Omega^+}$ is a smoothed version of the characteristic function χ_{Ω^+}. The function $\sigma(s)$ is a parameterization of the interface Γ and the functions $c'(s)$ and $c(s)$ are the strengths of the dipole and delta functions respectively. The jump $[f]$ in the third equation is the jump in the right hand side of Poisson's equation, the model problem we have been using throughout this discussion.

While we do not attempt to verify rigorously that the approximations in (3.85) hold, we can argue that the terms on the left hand side capture the correct singular behavior suggested by these approximations. First, dipole singularities in the source term will lead to non-zero jumps in the solution of the PDE; similarly, delta function singularities lead to jumps in the derivative of the solution, and discontinuities in the right hand side f lead to a jump in the second derivative. So if the strengths of the singular functions on the right hand side of these approximations are non-zero, we expect non-zero values in the corresponding terms on the left hand side. Second, these discrete approximations are only non-zero at irregular grid points. From their definition, these irregular grid points are no more than one mesh width away from the interface and so as the mesh is refined, the region in which these approximations are non-zero goes to zero. This behavior models in a discrete sense the notion that the singular functions have support only on the boundary. Finally, our discrete approximations have the right singular behavior. We have that the $\mathcal{D}'_{ij}\mathbf{w} \sim O(1/h^2)$, $\mathcal{D}_{ij}\mathbf{v} \sim O(1/h)$ and $\mathcal{D}^H_{ij}\mathbf{r} \sim O(1)$. This singular behavior is consistent with the fact that the dipole is a higher order singularity than the delta function, which is more singular than the characteristic function.

3.3.4 Determining unknown jumps

With the notation introduced in the previous section, we are now ready to describe very generally how we can solve our model problem. We postpone a discussion of the numerical details until later. We assume that we want to solve the problem in the exterior region $\Omega = \Omega^-$, subject to Dirichlet conditions. The model problem is

$$
\begin{aligned}
\nabla^2 &= f, \qquad (x, y) \in \Omega^- \\
u &= g, \qquad (x.y) \in \Gamma
\end{aligned}
\tag{3.86}
$$

Using jumps $[U_c(\alpha_k)]$ at points along the interface, and defining $f_{ij} \equiv 0$ for $x_{ij} \in \Omega^+$, we write down the discrete equations as

$$\begin{aligned} \mathcal{A}U_{ij} + \mathcal{D}'_{ij}\mathbf{w}_{ij} + \mathcal{D}_{ij}\mathbf{v}_{ij} + \mathcal{D}^H_{ij}\mathbf{r}_{ij} &= f_{ij}, \quad x_{ij} \in R \\ \mathcal{I}_k U^k_{ij} + \mathcal{I}^w_k \mathbf{w}_k + \mathcal{I}^v_k \mathbf{v}_k + \mathcal{I}^f_k \mathbf{r}_k &= g_k, \quad k = 1, \dots M. \end{aligned} \tag{3.87}$$

To form the correction terms needed for each set of equations, we must have all the jump information necessary to form the vectors $\mathbf{w}$, $\mathbf{v}$ and $\mathbf{r}$. Some of this information can be obtained from the data for the given problem. If we assume that outside of our domain of interest, the solution is exactly 0, then we have that $[u] = u^+ - u^- = 0 - u^- = -g$. From this information, we can form the vector $\mathbf{w}$, using interpolation to obtain necessary tangential derivatives. We also know the right hand side of the PDE we are solving and so we have $[f] = f^+ - f^- = 0 - f^-$ needed to form the vector $\mathbf{r}$. What we don't have is the jump in the normal derivative $[u_n]$ for the vector $\mathbf{v}$.

To obtain this unknown jump information, we must solve the system of equations given in (3.87). This system is linear and can be written in the general form $CV + F = g$, where V is a vector in $\mathbb{R}^M$ of jumps $[u_n]$. The vector F involves known jumps $[u]$ and $[f]$ and boundary conditions on the computational domain. To see how we formulate this system, we provide a pseudo-code for a subroutine we could use to compute $CV + F$. For the following subroutine, we assume that we have a fast Poisson solver which for notational convenience, we call `FPS`. This solver takes as input a right hand side F which is defined on an $(N + 1) \times (N + 1)$ grid. Boundary data is supplied on the boundary of the grid. To obtain the solution, we call this solver using the syntax $U = FPS(F)$. The algorithm is shown in Figure 3.8.

This system could be solved either directly or using an iterative method, on the system $CV = g - F$. In either case, we should modify the above subroutine so that only homogenous terms are involved. A second subroutine could then be written which determines the right hand side $g - F$. To solve the system directly, we form the matrix C explicitly apply the modified subroutine C to columns of the identity matrix. The resulting matrix is dense and non-symmetric, but is small and so can be solved quite reasonably using a direct method. Alternatively, we could solve the system using a linear iterative method, such as GMRES.

Once we have V, we can then use the jumps to form the correction term C_{ij} at all irregular grid points and solve the resulting system one more time to get the solution at all grid points.

Before continuing onto a section where we describe the details of the above, we first describe how we can solve parabolic equations.

Function C(V)

1. For each grid point near the interface, form the vector $\mathbf{v}_{ij}$ using V and the temporary right hand side $R^{tmp} \in \mathbb{R}^{(N+1)\times(N+1)}$, where the entry corresponding to the irregular grid point x_{ij} is given by

$$R_{ij}^{tmp} \equiv \mathcal{D}'_{ij}\mathbf{w}_{ij} + \mathcal{D}_{ij}\mathbf{v}_{ij} + \mathcal{D}_{ij}^{H}\mathbf{r}_{ij}$$

   For regular grid points, $R_{ij}^{tmp} = 0$.

2. Use the fast poisson solver to get $U = \texttt{FPS}(R^{tmp} + B)$. Use B, defined on the grid to apply appropriate boundary conditions on the boundary of the computational domain.

3. The k^{th} entry of the vector $C(V)$ is given by

$$(C(V))_k \equiv \mathcal{I}_k U_{ij}^k + \mathcal{I}_k^w \mathbf{w}_k + \mathcal{I}_k^v \mathbf{v}_k^* + \mathcal{I}_k^f \mathbf{r}_k$$

   for each $k = 1, \dots M$.

End Function C

Figure 3.8: Subroutine for computing $CV + F$ needed to determine unknown jumps V.

3.3.5 Solving parabolic equations

In this section, we show how the ideas presented can be used to solve parabolic boundary value problems. We begin by considering the model problem

$$u_t = \nu \nabla^2 u, \quad (x, y) \in \Omega \tag{3.88}$$

subject to Neumann (flux) boundary conditions

$$\frac{\partial u}{\partial n} = g \tag{3.89}$$

on $\partial\Omega$ and given initial conditions. The parameter ν is the diffusion coefficient and is assumed to be constant and continuous across the boundary. As we pointed out in the first section of this chapter, the coefficient vectors needed to form the correction term C_{ij} depend upon the PDE we are solving. As we have seen in the preceding sections, this dependence comes about in part through the matrix Σ, first introduced in (3.27). Here, we show how this matrix is modified for the parabolic problem.

The major differences between the elliptic problem and the parabolic problem are that (1) we don't automatically have the equation needed to form the last row in the matrix Σ, defined in (3.27), and (2) in the parabolic problem the jumps are now time dependent and must be determined at each time step.

To get the last row of Σ, we must first discretize the equation to get something of the form

$$\nabla^2 u + \lambda u = f. \tag{3.90}$$

or alternatively, something of the form

$$u + \lambda \nabla^2 u = f. \tag{3.91}$$

To get these coefficients, we discretize the parabolic equation in time using either backward Euler or Crank-Nicolson and write down a semi-discrete equation as

$$\left(1 - \Delta t\, \nu\, \beta_1\, \nabla^2\right) u^{m+1} = \left(1 + \Delta t\, \nu\, \beta_2\, \nabla^2\right) u^m \tag{3.92}$$

where we have used the superscript m to indicate the time level t_m. The constants β_1 and β_2 identify the time discretization scheme used. The values $(\beta_1, \beta_2) = (1, 0)$ correspond to a Backward Euler scheme, and $(\beta_1, \beta_2) = (0.5, 0.5)$ correspond to a Crank-Nicolson scheme.

With the PDE written in this way, we see immediately that for the parabolic case, λ in (3.91) should be set to $\lambda = -\Delta t\, \nu\, \beta_1$. Using these coefficients, we form a matrix we call $\overline{\Sigma}$, to distinguish it from the Σ used for elliptic problems. The right hand side f is then given by

$$f \equiv \left(1 + \Delta t\, \nu\, \beta_2\, \nabla^2\right) u^m \tag{3.93}$$

which we will need to evaluate at expansion points at the boundary of our domain using additional interpolation formulas.

To get the jump information, we again use the boundary conditions, as we did in the elliptic case. If we assume that outside of the flow domain (the Ω^- region), the solution is identically 0, then we have that the following jump information.

$$\begin{aligned}
[u_n^{m+1}] &= -g^{m+1} \\
[(u_n^{m+1})_s] &= -g_s^{m+1} \\
[f] &= (1 + \Delta t\, \nu\, \beta_2\, \nabla^2)\; u^m \\
&= [u^m] + \Delta t\, \nu\, \beta_2\, ([u_{xx}^m] + [u_{yy}^m])
\end{aligned} \tag{3.94}$$

To form the jump $[f]$, we need to form a second set of interpolation coefficients, which we label here as $\mathcal{I}_n^{f^m}$. Using the fact that $[U_c] = \overline{\Sigma}^{-1}[U_s]$, the coefficients of this vector are

$$\mathcal{I}_n^{f^m} \equiv (e_1 + \Delta t\, \nu\, \beta_2(e_4 + e_5))\overline{\Sigma}^{-1} \tag{3.95}$$

and we have

$$[f] = \mathcal{I}_n^{f^m}[U_s]^m \tag{3.96}$$

These coefficients should be stored along with the others, and when the jump vector r^{m+1} is needed, we can form it using

$$\mathbf{r}^{m+1} \equiv [0,\ 0,\ 0,\ 0,\ 0,\ \mathcal{I}_n^{f^m}[U_s]^m]. \tag{3.97}$$

Note that the sixth entry in the vector $[U_s]^m$ is r^m, the jump $[f]$ from time level m.

Using the jump information given by the problem data, we can form the jump vectors $\mathbf{v}$ and $\mathbf{r}$. The set of scalar equations we need to solve for the unknown vector $\mathbf{w}$ are

$$\begin{aligned}
\mathcal{H}U_{ij}^{m+1} &= \mathcal{H}'U_{ij}^m + \overline{\mathcal{D}}'_{ij}\overline{\mathbf{w}}_{ij}^m + \overline{\mathcal{D}}_{ij}\overline{\mathbf{v}}_{ij}^m + \overline{\mathcal{D}}_{ij}^H\overline{\mathbf{r}}_{ij}^m \\
&\quad -(\overline{\mathcal{D}}'_{ij}\overline{\mathbf{w}}_{ij}^{m+1} + \overline{\mathcal{D}}_{ij}\overline{\mathbf{v}}_{ij}^{m+1} + \overline{\mathcal{D}}_{ij}^H\overline{\mathbf{r}}_{ij}^{m+1}), \quad x_{ij} \in R \\
\overline{\mathcal{I}}_{n,k}U_{ij}^{m+1} + \overline{\mathcal{I}}_{n,k}^w\overline{\mathbf{w}}_k^{m+1} &+ \overline{\mathcal{I}}_{n,k}^v\overline{\mathbf{v}}_k^{m+1} + \overline{\mathcal{I}}_{n,k}^f\overline{\mathbf{r}}_k^{m+1} = g_k, \quad k = 1, \ldots M
\end{aligned} \tag{3.98}$$

where the jumps from time level m and $m+1$ are needed. Also, we use the overline here to distinguish the jumps and jump coefficients here from those used for elliptic problems. The coefficients $\mathcal{H}$ and $\mathcal{H}'$ are the usual coefficients used in the parabolic case and, using our notation, are given by

$$\begin{aligned}
\mathcal{H} &\equiv (e_1 - \Delta t\, \nu\, \beta_1(e_4 + e_5))^T(\Lambda^h)^{-1} \\
\mathcal{H}' &\equiv (e_1 + \Delta t\, \nu\, \beta_2(e_4 + e_5))^T(\Lambda^h)^{-1}
\end{aligned} \tag{3.99}$$

The coefficients $\overline{\mathcal{D}}'_{ij}$, $\overline{\mathcal{D}}_{ij}$ and $\overline{\mathcal{D}}^H_{ij}$ are given in terms the vector $\overline{\mathcal{B}}$, given by

$$\overline{\mathcal{B}} \equiv (e_1 - \Delta t\, \nu(e_4 + e_5))^T (\Lambda^h)^{-1} \Phi_{ij} \overline{\Sigma}^{-1}. \tag{3.100}$$

where matrices Λ^h and Φ_{ij} are defined exactly as they are for the elliptic problem. Using the entries b_i of $\overline{\mathcal{B}}$, we have that

$$\begin{aligned} \overline{\mathcal{D}}'_{ij} &\equiv [b_1,\ 0,\ b_3,\ b_4,\ 0,\ 0] \\ \overline{\mathcal{D}}_{ij} &\equiv [0,\ b_2,\ 0,\ 0,\ b_5,\ 0] \\ \overline{\mathcal{D}}^H_{ij} &\equiv [0,\ 0,\ 0,\ 0,\ 0,\ b_6] \end{aligned} \tag{3.101}$$

The vector $\mathcal{I}_{n,k}$ is defined exactly as it is for the elliptic case, with Σ replaced by $\overline{\Sigma}$:

$$\mathcal{I}_{n,k} \equiv e_2^T \left[\overline{\Sigma} \left(\Lambda^h \Lambda^\alpha \right)^{-1} \right] \tag{3.102}$$

The coefficient vectors $\overline{\mathcal{I}}^w_k$, $\overline{\mathcal{I}}^v_k$, and $\overline{\mathcal{I}}^f_k$ are defined in terms of the vector $\mathcal{B}''$, given by

$$\mathcal{B}'' \ \equiv \ e_2^T \overline{\Sigma} (\Lambda^h \Lambda^\alpha)^{-1} \Phi_{ij} \overline{\Sigma}^{-1} \tag{3.103}$$

Using $\mathcal{B}''$ instead of $\mathcal{B}'$, these coefficients are defined exactly in (3.101).

Unlike the elliptic case, it may not always be possible to create the jump coefficients once as a preprocessing step, since these coefficients depend on the time step Δt. In an involved computation, where the time step is determined by stability considerations of some other part of the problem, one will need to recompute these coefficients whenever the time step changes. Fortunately, computing these coefficients does not require much overhead.

The resulting matrix system for the unknowns u_{ij} can be solved for using a fast Poisson solver. The C matrix for the unknowns is well conditioned, as it was in the elliptic case.

Before continuing to numerical examples for the elliptic and parabolic problems, we discuss the truncation error of the parabolic scheme. Unlike the elliptic case, the transformation $U_s = \overline{\Sigma} U_c$ has a truncation error associated with the last row of $\overline{\Sigma}$. Assuming that we have $[\nabla^2 u] = [u_{xx}] + [u_{yy}]$ exactly, we have that

$$U_s = \overline{\Sigma} U_c + \tau_{\text{BE}}(\Delta t) \tag{3.104}$$

for the backward Euler scheme, and

$$U_s = \overline{\Sigma} U_c + \tau_{\text{CN}}(\Delta t) \tag{3.105}$$

for the Crank-Nicolson scheme. The vectors $\tau_{\text{BE}}(\Delta t)$ and $\tau_{\text{CN}}(\Delta t)$ are given by

$$\begin{aligned} \tau_{\text{BE}}(\Delta t) &\sim [0,\ 0,\ 0,\ 0,\ 0,\ O(\Delta t)]^T \\ \tau_{\text{CN}}(\Delta t) &\sim [0,\ 0,\ 0,\ 0,\ 0,\ O(\Delta t^2)]^T \end{aligned} \tag{3.106}$$

When using backward Euler, equations (3.66) become

$$\begin{aligned}\widetilde{U}_{ij} &= U_{ij} + \Phi_{ij}[U_c] + \tau^1(h) \\ &= U_{ij} + \Phi_{ij}\Sigma^{-1}\left\{[U_s] + \tau_{\text{BE}}(\Delta t)\right\} + \tau^1(h)\end{aligned} \tag{3.107}$$

The equation for Crank-Nicolson is the same, with $\tau_{\text{CN}}(\Delta t)$ replacing $\tau_{\text{BE}}(\Delta t)$. Using the symbolic package Maple, we evaluate the approximation error $\Sigma^{-1}\tau_{\text{BE}}(\Delta t)$ and $\Sigma^{-1}\tau_{\text{CN}}(\Delta t)$. These vectors have form

$$\begin{aligned}\Sigma^{-1}\tau_{\text{BE}}(\Delta t) &\sim [0,\ 0,\ 0,\ O(1),\ O(1),\ O(1)]^T \\ \Sigma^{-1}\tau_{\text{CN}}(\Delta t) &\sim [0,\ 0,\ 0,\ O(\Delta t),\ O(\Delta t),\ O(\Delta t)]^T\end{aligned} \tag{3.108}$$

Recalling that the last three columns of Φ_{ij}, defined in (3.64), have entries that are all $O(h^2)$, we can derive the truncation error expressions for $\widetilde{U}_{ij}$ in (3.107) and get for backward Euler

$$\widetilde{U}_{ij} = U_{ij} + \Phi_{ij}\Sigma^{-1}[U_s] + O(h^2) \tag{3.109}$$

and for Crank-Nicolson,

$$\widetilde{U}_{ij} = U_{ij} + \Phi_{ij}\Sigma^{-1}[U_s] + O(h^2\Delta t). \tag{3.110}$$

The error in these last two equations applies to each element in the vectors in $\widetilde{U}_{ij}$. When we approximated the Laplacian in (3.71), we relied on the fact that we could approximate $\widetilde{U}_{ij}$ to $O(h^3)$ in order to get an expression for the Laplacian that was $O(h)$. Using the same line of reasoning, we see that when using backward Euler, we only can expect an $O(1)$ approximation to the Laplacian. When using Crank-Nicolson, we still expect an $O(h)$ approximation, assuming that $\Delta t \sim h$. However, we want to approximate $(1-\nu\Delta t\beta_1\nabla^2)\widetilde{U}_{ij}^{m+1}$ and $(1+\nu\Delta t\beta_2\nabla^2)\widetilde{U}_{ij}^{m}$, which we can do to at least $O(\Delta t)$, regardless of the scheme that we use. This is more than sufficient to ensure that the backward Euler scheme will be first order accurate. For Crank-Nicolson, it is also sufficient for second order accuracy, since we only use these approximations near the boundary.

In the next section, we solve a particular parabolic problem and show some typical results. We verify that we get at least first order accuracy for backward Euler, and second order for Crank-Nicolson. In the case of the Neumann problem, we solve for the unknown value of u at the boundary and show that this also converges at a second order rate.

3.4 *Implementation details of the Immersed Interface Method*

We are ready to use the correction formulas derived above to solve a variety of linear boundary value problems. In particular, we will solve an elliptic and parabolic equations that arise in potential flow and Stokes flow.

Before starting with the actual solution process, however, we first discuss the pre-processing needed to obtain the coefficient vectors $\mathcal{D}'_{ij}$, $\mathcal{D}_{ij}$ etc. We begin by discussing the representation of a general interface using Fourier splines. In the following discussion, we use the term interface to mean the boundary of a single embedded object.

Representing the interface and normal and tangential derivatives

In each of the following problems, we assume that we have chosen M points $\bar{\alpha}^k, k = 1, \dots M$ equally spaced in arclength along the boundary of each embedded object. These points, called *interface points* or *interface nodes* are chosen so that they represent the interface well, but the number of points M does not need to depend on the width of the underlying mesh. These M points will be the points at which we discretize the boundary conditions.

There are at least two advantages to relying on as few points as possible to represent the interface. First, as we will see, for the following BVPs, we will need to invert an $M \times M$ system to obtain unknown jump conditions at the interface. The fewer points that are needed to represent the interface and the solution along the interface, the smaller this linear system will be. This becomes especially important when we consider domains with several embedded objects.

Second, and probably the more fundamental reason is that as we will see, it will be important that the ratio of the average arclength between interface points to the mesh width be at least a certain minimum critical value. Roughly, this ratio scales with M, and so as we increase M, we must increase the number of grid points. On the other hand, if we refine the mesh without increasing the number of interface points, we show that the conditioning of the resulting linear system remains essentially constant, assuming the minimum arclength to mesh width ratio has been reached.

In her PhD thesis [58], Yang obtained acceptable results by taking $M \sim N$, the number of grid points in, say, the x direction. She did, however, find that the conditioning of her resulting linear system was quite sensitive to the location of these interface points (also used as expansion points), and could in fact be so poor as to render the solution completely unreliable. We have essentially eliminated this problem by choosing very few points on which to base our linear system. To obtain expansion points, however, we must interpolate between these points, as we describe next.

One potential drawback to choosing as few interface points as possible is that in general, interface points will not coincide with the expansion points α needed to form the correction term C_{ij}. The expansion point for a particular grid point x_{ij} should be no more than a mesh width away from the grid point, and so we expect to need many more expansion points than we have interface points. For this reason, we will need to be able to interpolate between interface points to obtain expansion points sufficiently close to the grid point of interest. Moreover, we will need normal and tangential derivative information at expansion points and interface points. To obtain all this information, we have found it convenient to use Fourier interpolation for all closed interfaces. In choosing this representation, however, we are only able to handle sufficiently smooth interfaces. By representing the interface $\sigma(s) = (x(s), y(s))$ as

$$\begin{aligned} x(s) &= \sum_{m=-(M/2+1)}^{M/2} \hat{x}(m)e^{ims} \\ y(s) &= \sum_{m=-(M/2+1)}^{M/2} \hat{y}(m)e^{ims} \end{aligned} \tag{3.111}$$

using Fourier coefficients $\hat{x}(m)$ and $\hat{y}(m)$ obtained from the Fast Fourier Transform, we get spectrally accurate approximations to the locations of interface and expansion points, as well as to derivative information x_s, y_s, x_{ss} and y_{ss} needed to form the matrix Σ at arbitrary points on the boundary. To obtain the Fourier coefficients $\hat{x}(m)$ and $\hat{y}(m)$, we used the standard FFT package `fftpack`, available in both scalar and vector form from Netlib [49]. When using this package, it is convenient to have $s \in [0, M]$. Interface points then correspond to integral values of s and we have $\bar{\alpha}^k \equiv (\bar{\alpha}_x^k, \bar{\alpha}_y^k) = (x(k-1), y(k-1)), k = 1, \ldots M$. Arbitrary expansion points are identified by non-integral values of s in the interval $[0, M]$. As before, we denote these points $\alpha^\mu \equiv (x(s^\mu), y(s^\mu)), \mu = 1, \ldots P$, where $P \gg M$. To choose these expansion points, we first create a set of indices $\{s^\nu\}_{\nu=1}^{P_{max}}$ where $P_{max} \gg M$. Using these indices, we form points $\alpha^\nu = (x(s^\nu), y(s^\nu))$ using the Fourier coefficients obtained from the FFT and the summation formula for $(x(s), y(s))$ given above. Expansion points α^μ are then chosen so that μ minimizes $\| \alpha^\nu - \alpha^\mu \|$ over all ν. We then store these expansion points and associated indices s^μ. To get derivatives, at the expansion points, we use the above summation formulas with coefficients $im\hat{x}(m)$, $im\hat{y}(m)$, $-m^2\hat{x}(m)$ and $-m^2\hat{y}(m)$.

Because the expansion points have associated indices s^μ which are in general non-integral, we must use the above summation formulas explicitly. However, the interface points correspond to only the integral values of $s \in [0, M-1]$ and so we can use the inverse FFT on the coefficients $\hat{x}(s)$, $\hat{y}(s)$ and coefficients for derivatives to get x_s, y_s, x_{ss} and y_{ss} at these points. This may be more a convenience than anything, however, since the computational overhead associated with even doing the Fourier interpolation explicitly is minimal, since this information need only be determined once at the beginning of the calculations.

Forming the coefficient vectors

Once we have identified expansion points $\alpha^{\mu(i,j)}$ for each irregular grid point x_{ij}, and the derivatives along the interface at these points, we can begin to form the coefficient vectors at these points. One of the many advantages of the Immersed Interface Method is that at this point, determining these vectors is a very mechanical process. We first identify the set of irregular grid points $x_\mu, \mu = 1, \dots P$. Recall that irregular grid points are the anchor points for which a five points stencil straddles the interface. For each irregular grid point, we choose a point α^μ on the interface that is no more than one mesh width away from x^μ. Then using the Fourier interpolation described above, we obtain the interface information needed to form matrix Σ at the point α^μ.

To get matrices Φ and Λ^h, we assume we have an underlying ordered stencil S_{ij}. In most cases, this can be taken to the usual five point stencil plus an extra sixth point chosen arbitrarily. We assume the same stencil for all Φ and Λ^h, unless otherwise specified. Using this stencil, we can form Λ^h and Φ, using definition (3.12),

$$e_p^T(\Lambda^h) = \begin{pmatrix} 1 & k_p h & \ell_p h & \frac{1}{2}(k_p h)^2 & \frac{1}{2}(\ell_p h)^2 & (k_p h)(\ell_p h) \end{pmatrix}.$$

and definition (3.64)

$$e_p^T(\Phi_{ij}) = \sigma_{ij}^{k_p,\ell_p} \widetilde{P}(x_i, y_j; k_p h, \ell_p h)^T.$$

The matrix Λ^α is defined using (3.19).

With the matrices Σ, Λ^h, Φ and Λ^α, we can construct coefficient vectors $\mathcal{D}'_{ij}$, $\mathcal{D}_{ij}$, $\mathcal{D}^H_{ij}$, $\mathcal{I}_k$, $\mathcal{I}^w_k$, $\mathcal{I}^v_k$ and $\mathcal{I}^f_k$ using formulas derived earlier. In practice, we make use of the fact that

$$\mathcal{D}'_{ij}\mathbf{w} + \mathcal{D}_{ij}\mathbf{v} + \mathcal{D}^H_{ij}\mathbf{r} = (\mathcal{D}'_{ij} + \mathcal{D}_{ij} + \mathcal{D}^H_{ij}) \cdot (\mathbf{w} + \mathbf{v} + \mathbf{r}),$$

and only store $\mathcal{B} \equiv \mathcal{D}'_{ij} + \mathcal{D}_{ij} + \mathcal{D}^H_{ij}$. We store an equivalent vectors $\overline{\mathcal{B}}$ for the parabolic coefficients $\overline{\mathcal{D}}'_{ij}$, $\overline{\mathcal{D}}_{ij}$ and $\overline{\mathcal{D}}^H_{ij}$. To remind the reader, this vector $\mathcal{B}$ for the elliptic case is given by

$$\mathcal{B} \equiv (e_4 + e_5)^T (\Lambda^h)^{-1} \Phi_{ij} \Sigma^{-1}$$

and for the parabolic problem is given by

$$\overline{\mathcal{B}} \equiv (e_1 - \Delta t\, \nu(e_4 + e_5))^T (\Lambda^h)^{-1} \Phi_{ij} \Sigma^{-1}.$$

Vectors $\mathcal{I}_k$, $\mathcal{I}_{n,k}$, $\overline{\mathcal{I}}_k$, $\overline{\mathcal{I}}_{n,k}$ and associated jump coefficients are formulated in the same way. We also have

$$\mathcal{I}^w_k \mathbf{w} + \mathcal{I}^v_k \mathbf{v} + \mathcal{I}^f_k \mathbf{r} = (\mathcal{I}^w_k + \mathcal{I}^v_k + \mathcal{I}^f_k) \cdot (\mathbf{w} + \mathbf{v} + \mathbf{r})$$

and so we only need to store $\mathcal{B}' \equiv \mathcal{I}^w_k + \mathcal{I}^v_k + \mathcal{I}^f_k$. Similarly, we store $\mathcal{B}'' \equiv \mathcal{I}^w_{n,k} + \mathcal{I}^v_{n,k} + \mathcal{I}^f_{n,k}$. Coefficients for the parabolic problem are stored in $\overline{\mathcal{B}}'$ and $\overline{\mathcal{B}}''$. The coefficients for the elliptic problem are given by

$$\mathcal{B}' \equiv e_1^T \Sigma (\Lambda^h \Lambda^\alpha)^{-1} \Phi_{ij} \Sigma^{-1}$$
$$\mathcal{B}'' \equiv e_2^T \Sigma (\Lambda^h \Lambda^\alpha)^{-1} \Phi_{ij} \Sigma^{-1}$$

The coefficients for the parabolic problem are identical, with Σ replaced by $\overline{\Sigma}$.

With these coefficients, we can continue on to the details of forming the linear system $C(V)$.

Forming the modified linear system $C(V)$

In Figure 3.8, we introduced a linear system $CV + F = g$. We suggested that we could solve this system iteratively for a vector of unknowns $V \in \mathbb{R}^M$. In this section, we show how we can rearrange the terms of this system to get an explicit linear system of the form $C(V) = R_{rhs}$ which we can solve using either a direct solver or a linear iterative method. We refer to the Poisson equation (3.86) subject to Dirichlet conditions as the model problem.

The discretization of the model and associated boundary conditions given in (3.87) is

$$\begin{aligned} \mathcal{A}U_{ij} + \mathcal{D}'_{ij}\mathbf{w}_{ij} + \mathcal{D}_{ij}\mathbf{v}_{ij} + \mathcal{D}^H_{ij}\mathbf{r}_{ij} &= f_{ij}, \quad x_{ij} \in \Omega \\ \mathcal{I}_k U^k_{ij} + \mathcal{I}^w_k \mathbf{w}_k + \mathcal{I}^v_k \mathbf{v}_k + \mathcal{I}^f_k \mathbf{r}_k &= g_k, \quad k = 1, \dots M \end{aligned} \tag{3.112}$$

We have vectors $\mathbf{w}$ and $\mathbf{r}$ from the boundary information and the PDE, but need to compute $\mathbf{v}$. We now go through this steps in this algorithm in more detail and then we show how we can rearrange the terms to get a linear system which we can solve either directly, or using linear iterative methods.

First, we describe briefly how we form the vectors $\mathbf{w}_{ij}$, $\mathbf{v}_{ij}$ and $\mathbf{r}_{ij}$ given the M values (either known or unknown) $w = [u]$, $v = [u_n]$ and $r = [f]$ along the boundary. The algorithm requires that we be able to form these vectors at each of our expansion points $\alpha^\mu, \mu = 1, \dots P$. To do so, we use the data w_k, v_k and r_k, $k = 1, \dots M$ to get Fourier coefficients $\hat{w}(m)$, $\hat{v}(m)$ and $\hat{r}(m)$ for the summations

$$\begin{aligned} w(s) &= \sum_{m=-(M/2+1)}^{M/2} \hat{w}(m)e^{ims} \\ v(s) &= \sum_{m=-(M/2+1)}^{M/2} \hat{v}(m)e^{ims} \\ r(s) &= \sum_{m=-(M/2+1)}^{M/2} \hat{r}(m)e^{ims} \end{aligned} \tag{3.113}$$

We can then evaluate these summations at points s^μ corresponding to a point α^μ. To get the derivatives w_s, w_{ss}, and v_s, we use the coefficients $im\hat{w}(m)$, $-m^2\hat{w}(m)$ and $im\hat{v}(m)$ instead of the original Fourier coefficients. We note that the Fourier transform is linear, and so the map which takes, for example, the vector $W \in \mathbb{R}^M$ to the vector $\mathbf{w}_{ij}$ is also linear.

The subroutine described in Figure 3.8 is not in a very useful form. Because it is in the form $CV + F = g$, it cannot be immediately used in a linear iterative method. We now want to show how we can rearrange the terms in the linear iteration described above to form a system $\mathbf{C}(V) = F_{rhs}(W, R)$. For this subroutine, we assume that we have a Poisson solver $U = \texttt{FPS}(F)$.

Function C(V) :

1. Form the temporary right hand side $R^{tmp}(V) \in \mathbb{R}^{(N+1)\times(N+1)}$, where the entry corresponding to irregular grid point x_{ij} is given by

$$R^{tmp}_{ij}(V) \equiv -\mathcal{D}_{ij}\mathbf{v}_{ij}$$

For regular grid points, $R^{tmp} \equiv 0$.

2. Use a fast Poisson solver to get $U = \texttt{FPS}(R^{tmp})$. Use homogenous boundary data on the boundary of the computational domain.

3. The k^{th} entry of the vector $\mathbf{C}(V) \in \mathbb{R}^M$ is then given by

$$\mathbf{C}_k(V) \equiv \mathcal{I}_k U^k_{ij} + \mathcal{I}^v_k \mathbf{v}_k$$

End Function C(V)

Figure 3.9: Subroutine describing linear system CV for unknown vector V.

To get the right hand side vector $\mathbf{F}_{rhs}$, we use a similar algorithm, given in Figure 3.10.
We now have two general approaches we can take to solving the system $\mathbf{C}(V) = \mathbf{F}_{rhs}$ for V. One, we can form the matrix $\mathbf{C}$ explicitly. The columns of $\mathbf{C}$ are given by $\mathbf{C}(\xi_j), j = 1, \ldots M$, where ξ_k is the k^{th} column of the $M \times M$ identity matrix, and so forming $\mathbf{C}$ will require M fast Poisson solves. For small M, this is probably the most reasonable approach.

For larger M. we want to consider using linear iterative methods designed to solve linear systems $\mathbf{C}(V) = \mathbf{F}_{rhs}$. In the context of linear iterative methods, the function evaluation $\mathbf{C}(V)$ is typically referred to as a *matrix-vector multiply.* One advantage of using an iterative method is that it can in principle require fewer than M matrix multiplies, especially if one does not need a high degree of accuracy in the resulting solution. Since our matrix vector multiply is expensive (requiring a fast Poisson solve), we will want to use an iterative method that is likely to converge in

Function $\mathbf{F}_{rhs}(G, R, W)$:

1. Form the temporary right hand side $R^{tmp}(W, R) \in \mathbb{R}^{(N+1)\times(N+1)}$, where the entry corresponding to irregular grid point x_{ij} is given by

$$R^{tmp}_{ij}(W, R) = f_{ij} - \mathcal{D}'_{ij}\mathbf{w}_{ij} - \mathcal{D}^H_{ij}\mathbf{r}_{ij}$$

2. Use a fast Poisson solver to get $U = \mathtt{FPS}(R^{tmp} + B)$. Use B to impose inhomogenous boundary conditions on the boundary of the computational domain.

3. The k^{th} entry of the vector $F_{rhs} \in \mathbb{R}^M$ is then

$$(F_{rhs})_k(G, R, W) \equiv g(\alpha^k) - \mathcal{I}_k U^k_{ij} - \mathcal{I}^w_k \mathbf{w}_k - \mathcal{I}^f_k \mathbf{r}_k$$

End Function $\mathbf{R}_{rhs}(G, R, W)$

Figure 3.10: Right hand side of linear system $g - F$.

as few iteration as possible. Ideally, it should require far fewer than M iterations, or the iterative method is not competitive with forming the matrix directly. We also need a method which can handle the fact that our matrix is non-symmetric. Probably the method most ideally suited to our problem is the Generalized Minimum Residual Algorithm, or GMRES, developed by Saad [47].

Once we have solved for V, we can then solve the full equations to get a complete solution to the PDE. We solve for the PDE with the subroutine in Figure 3.11.
The solution U obtained by this procedure is the solution to the PDE in the irregular domain.

In the following, we apply the above general framework to specific linear PDEs that we will use in the following chapters. As we will see, very little needs to be done to modify the above to these particular systems.

3.5 Numerical results

Because our final goal is to solve the full streamfunction-vorticity equations, we now consider numerical examples related to these equations, given in Chapter 1. In particular, we want to solve Laplace's equation for potential flow, a diffusion equation for the diffusion of vorticity and finally Stokes flow equations.

Function solvePDE(F,W,V,R) :

1. Form the temporary right hand side $R^{tmp} \in \mathbb{R}^{(N+1)\times(N+1)}$ where the entry corresponding to irregular grid point x_{ij} is given by

$$R^{tmp}_{ij} = F_{ij} - \mathcal{D}'_{ij}\,\mathbf{w}_{ij} - \mathcal{D}_{ij}\mathbf{v} - \mathcal{D}^{H}_{ij}\,\mathbf{r}_{ij}$$

2. Use a fast Poisson solver to get $U = \texttt{FPS}(R^{tmp} + B)$. Use B to impose inhomogenous boundary conditions on the boundary of the computational domain.

End solvePDE(F,W,V,R)

Figure 3.11: Subroutine used to solve PDE once all jumps are known.

3.5.1 Solving potential flow problems

A convenient representation of two dimensional, incompressible flow assumes that we have a stream-function ψ for which

$$u = \psi_y, \quad v = -\psi_x \tag{3.114}$$

where u and v are fluid velocities in the x and y directions respectively. By representing $(u, v) \equiv \vec{u}$ in this manner, we automatically satisfy the incompressibility condition $\nabla \cdot \vec{u}$, since

$$\nabla \cdot \vec{u} = u_x + v_y = \psi_{yx} - \psi_{xy} = 0 \tag{3.115}$$

Moreover, ψ has the property that $(\nabla\psi) \cdot \vec{u} = (\vec{u} \cdot \nabla)\psi = 0$. This means that ψ is constant along stream-lines of the flow and level sets of ψ are particle paths. In general, $\nabla^2\psi = \psi_{xx} + \psi_{yy} = -v_x + u_y \equiv -\omega$, the vorticity in the flow. In the present case, however, we consider only irrotational flow, and so assume that $\nabla^2\psi = 0$.

Potential flow around a cylinder

One classic problem in potential theory is that of potential flow around a circular cylinder. For this problem, we assume that we are given an infinite domain and a mean flow in the positive x direction. A circular cylinder is embedded in the flow field. The problem is to determine the flow field around the cylinder so that the surface of the cylinder is a streamline, but that at infinity, the streamlines are parallel to the direction of the mean flow.

From potential theory, we have a closed form expression for the streamfunction satisfying this set of conditions. In Cartesian coordinates, this streamfunction is given by

$$\widetilde{\psi}(x,y) = U_0 y \left(1 - \frac{r_a^2}{x^2+y^2}\right). \tag{3.116}$$

As $y \to \infty$, we have that $\widetilde{\psi} \to U_0 y$, or $u \to U_0$, $v \to 0$. On the cylinder itself, we have that $\widetilde{\psi} = 0$, and the surface of the cylinder is a streamline, as required.

To test how well the Immersed Interface Method solves this exterior Dirichlet problem, we solve the above problem in a finite computational domain $R = [-2,2] \times [-2,2]$. At the origin, we embed a cylinder of radius $r_a = 0.5$. The problem will be somewhat contrived in that we are going to use boundary data from the exact solution $\widetilde{\psi}$, but this will enable us to compare our computed results to the exact solution. The problem that we wish to solve is given by

$$\begin{aligned} \nabla^2\psi &= 0, \quad r \geq r_a \\ \psi &= 0, \quad r = r_a \\ \psi &= \widetilde{\psi}, \qquad \text{on } \partial R \end{aligned} \tag{3.117}$$

where $r = \sqrt{x^2+y^2}$. Although not strictly part of the problem definition, we make the additional assumption that $\psi = 0$ for $r < a$. This additional assumption, along with our original boundary conditions, gives us the jump $w \equiv [\psi] = 0$. Also, since our right hand side is identically zero, it too is continuous at the interface, and so $r \equiv [f] = 0$. From this we then have that $\mathbf{w} = \mathbf{r} = 0$.

To obtain the unknown jump $v = [\psi_n]$ at M points along the interface, we solve the system given in (3.87). In the present situation, these equations are given by

$$\begin{aligned} \mathcal{A}\psi_{ij} + \mathcal{D}_{ij}\mathbf{v}_{ij} &= 0, \quad x_{ij} \in R \\ \mathcal{I}_k\psi_{ij}^k + \mathcal{I}_k^v\mathbf{v}_k &= 0, \quad k = 1, \dots M \end{aligned} \tag{3.118}$$

For a single interface, we solved the system $\mathbf{C}(V) = \mathbf{R}_{rhs}$ directly for V, the $M \times 1$ vector of unknown jumps $[u_n]$. With a known V, we then solved for the streamfunction, using **solvePDE**, given in the previous section.

We solved for the stream-function using the method outlined above, and plotted the results on a 100×100 grid in Figure 3.12. The interface was represented using 16 interface points. This figure shows quite well that even close to the surface of the cylinder, the contours remain quite smooth.

Using (3.116) as a reference solution, we can compute the error e_{ij} of our computed solution at each grid point x_{ij} and confirm that the errors are $O(h^2)$. We compute the norms of the errors using the definition of the norm given by

$$\| e \|_p = \left(\frac{1}{\text{area}(\Omega)} \sum_{i,j=1}^{N+1} |e_{ij}|^p \, \Delta x \, \Delta y\right)^{1/p}, \qquad 1 \leq p < \infty \tag{3.119}$$

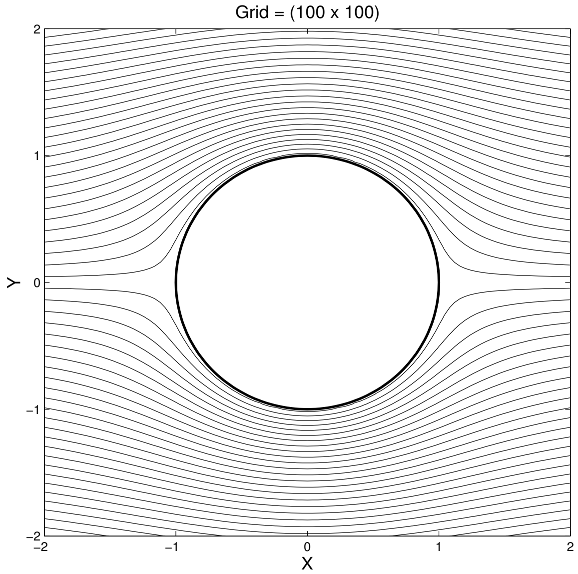


Figure 3.12: Computed solution to potential flow problem using Immersed Interface Method. Computational grid is 100×100

The max norm definition is the standard definition given by

$$\| e \|_\infty = \max_{i,j} |e_{ij}| \tag{3.120}$$

The norm of the error $e(s)$ on the curve Γ is defined as

$$\begin{aligned} \| e \|_p^\Gamma &= \left(\frac{1}{\text{length}(\Gamma)} \sum_{j=1}^{M} |e_j|^p \; ds \right)^{1/p} \\ &= \left(\frac{1}{M} \sum_{j=1}^{M} |e_j|^p \right)^{1/p}, \qquad 1 \le p < \infty \end{aligned} \tag{3.121}$$

where we have used the definition $ds \equiv \text{length}(\Gamma)/\text{M}$. Also, we have assume that Γ is a single curve. The max-norm for functions defined on the curve Γ is given by

$$\| e \|_p^\Gamma = \max_{j} |e_j| \tag{3.122}$$

where $|e_j|$ is the error of the j^{th} interface point.

In Figure 3.13, we plot the 1 norm and max-norm errors of our computed solution versus mesh width. In this plot we also show the convergence rate for the derivative computed from the solution to the linear system. We again use the reference solution

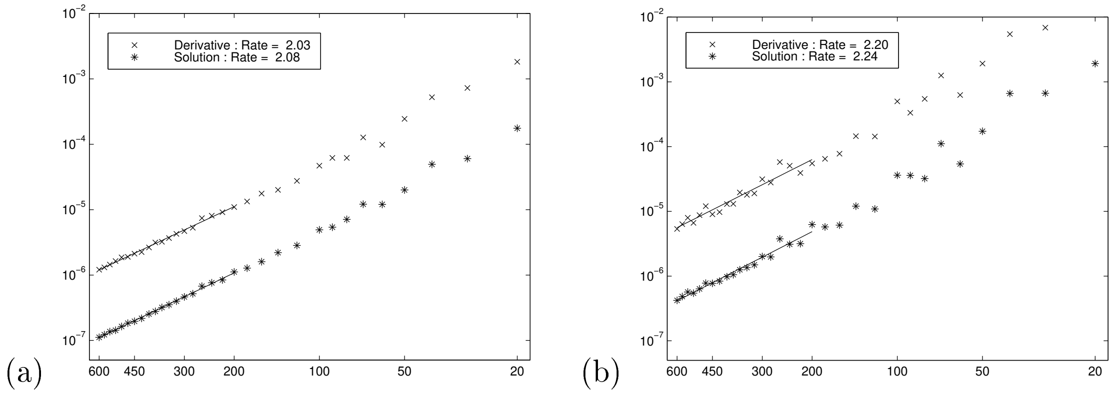


Figure 3.13: (a) 1-norm and (b) max-norm convergence rates for errors of computed stream function for potential problem. The solid line is the best-fit line used to compute the convergence rates shown in the legend.

to derive the exact solution $\widetilde{\psi}_n(\theta) = 2U_0 \sin(\theta)$. The convergence for both the solution and the derivative is clearly second order in both norms, although the solution is more accurate than the derivative calculation.

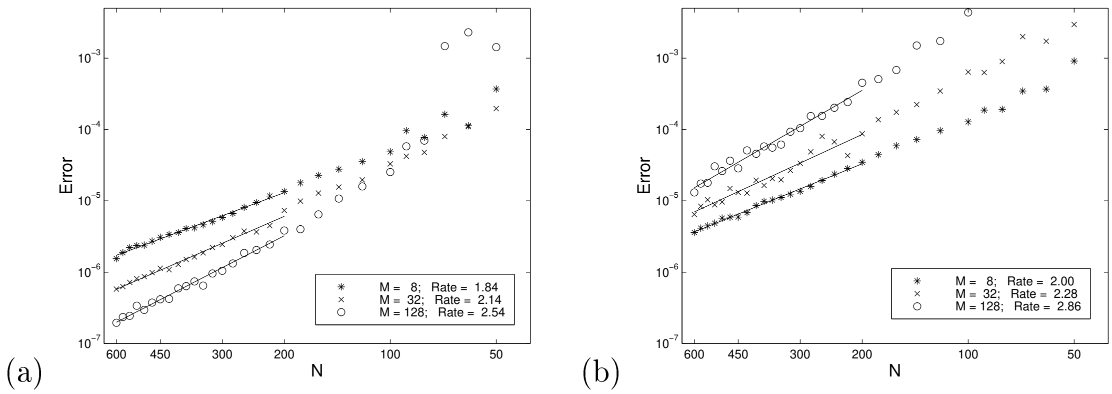


Figure 3.14: (a) 1-norm and (b) max-norm in errors in derivative for different values of M. Solid lines are the best-fit lines used to compute the convergence rates.

To see the effect that the number of interface points M has on the computed derivative, we computed the solution and derivative on several different grids for several different values of M. In Figure 3.14, we show the error in the derivative for three different values of M, $M = 8$, $M = 32$, and $M = 64$. In the 1-norm, we see that as we increase the number of interface points, we get more accurate approximations to the derivative, whereas in the max-norm this situation is reversed. Since the exact derivative, $\widetilde{\psi}_n(\theta) = 2U_0 \sin(\theta)$ is represented exactly by as few as four interface points, we wouldn't expect to increase the accuracy by adding more points. The fact that in the one-norm, the accuracy does increase is probably due to the fact that we are imposing the boundary conditions at more points, thereby placing more constraints on the computed solution. In the max-norm, we observe that the accuracy *decreases* as we increase the number of points. We attribute this to the fact that increasing the number of points increases the size and condition number (a fact we verify shortly) of the linear system we must solve to obtain the derivative, thereby increasing the role that numerical roundoff error plays in solving that system. However, the convergence rates in the max-norm favor the solutions with more points, and so we expect that for large enough N, we will obtain more accurate solutions with more points.

The fact that we can use the same value of M on all grids and still get convergence is only due to the fact the exact derivative, $\widetilde{\psi}_n(\theta) = 2U_0 \sin(\theta)$ can be represented using as few as four points. In general, we would expect to need to increase the number of interface points as we refine the grid in order to see convergence. After we have discussed the issue of conditioning the linear system we must solve to get unknown jump information, and described how to handle the multiply connected domain, we return to a more general boundary problem.

Conditioning of the linear system

To see how the condition number of the matrix $\mathbf{C}$ is affected by grid size and M, we plotted the condition number versus grid size for five different value of M in Figure 3.15. We also show the best-fit curve through the points (N_{min}, κ_M) where N_{min} is approximately the smallest value of N for which the condition number κ takes on its constant value κ_M.

From this plot, we draw two main conclusions from this investigation of condition number and its relation to mesh size and number of interface points. The first conclusion is that one must take some care to choose a mesh width that is fine enough for the particular choice of M. If the mesh is too coarse, then the matrix C could be quite poorly conditioned, and the resulting solution and derivative along the interface unreliable. The second and more interesting conclusion is that once we have reached this minimum ratio, the condition number of C remains essentially constant as we refine the grid further. From this investigation, this constant seems to scale as $\kappa \sim O(M)$. We hope to confirm this result in other examples.

Finally, using the observation that for this problem, $\kappa \sim M$ and $\kappa_M \sim \sqrt{N_{min}}$, we might conclude that to obtain a reliable solution V, we should require that the number of grid points N for the underlying mesh be approximately M^2. This value is probably too large in practice, and for this particular problem, increasing M to 256 should only require an N of about 1700, a number based on the trend of the dashed curve shown in Figure 3.15. Nonetheless, increasing the number of interface points beyond a certain number may require a grid that is so fine that the problem

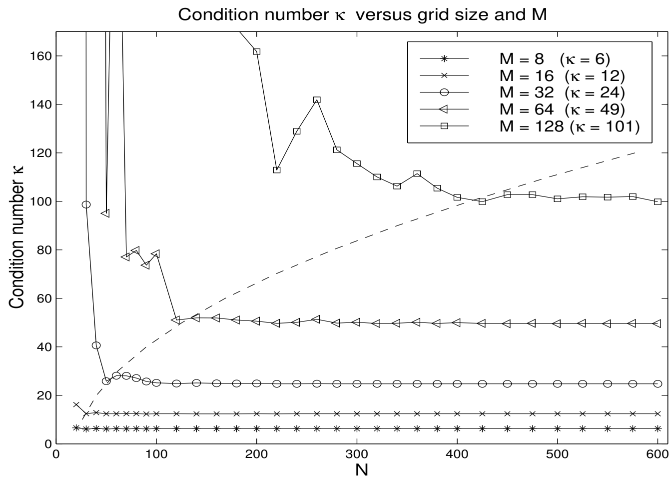


Figure 3.15: Condition number as a function of grid size N for various M. The dashed line is condition number as function of smallest N allowable for particular M.

can become computationally quite expensive, and judging from the plot of the errors in the derivatives given in Figure 3.14, there may be little gain in terms of accuracy.

Handling multiply-connected domains

Before continuing on to more realistic examples, we first want to solve this problem again, this time also solving for the value of the stream-function on the boundary of the cylinder. The reason for this is that in general, particularly in the multibody case, we don't know this value, and must impose instead the condition that no circulation be generated around the object. This latter condition ensures that the velocity field remain single valued, and is imposed by requiring that

$$\int_\Gamma \vec{u} \cdot \tau \, ds = \int_\Gamma \psi_n \, ds = 0, \tag{3.123}$$

where τ is the unit tangent vector at the surface of the cylinder, and ψ_n is the normal derivative of ψ at the cylinder surface.

The modified system of equations that we now solve are given by

$$\begin{aligned} \nabla^2 \psi &= 0, \quad r \geq r_a \\ \psi &= \overline{\psi}, \quad r = r_a \\ \psi &= \widetilde{\psi}, \qquad \text{on } \partial R \\ \int_\Gamma \psi_n(s) \, ds &= 0 \end{aligned} \tag{3.124}$$

where $\overline{\psi}$ is the unknown value of the streamfunction on $r = r_a$. To satisfy the last constraint, we must add an extra row and column to the matrix C. We call this new matrix C', where now $C' \in \mathbb{R}^{(M+1)\times(M+1)}$. If we require that for $\psi = \overline{\psi}$ for $r < r_a$, as we did in the previous problem, the jump conditions for this problem are the same as those used in the first example. Our system of equations for this modified problem is given by

$$\begin{aligned} \mathcal{A}\psi_{ij} + \mathcal{D}_{ij}\mathbf{v}_\mu &= 0, \quad x_{ij} \in R \\ \mathcal{I}_k\psi_{ij}^k + \mathcal{I}_k^v\mathbf{v}_k &= \overline{\psi}, \quad k = 1, \ldots M \end{aligned} \tag{3.125}$$

Except for the right hand side in the second equation, these equations are exactly those given in (3.118).

We first observe that the condition number of the matrix $\mathbf{C}'$ is $O(M^2)$, but after a minimum level of refinement is reached, this condition remains constant for successive levels of refinement. We point this out only because it was a surprise to see that adding just a single row and column could have such a dramatic affect on the condition number. But when we consider the form of the modified matrix, which looks like

$$C' = \left(\begin{array}{c|ccc} 0 & 1 & \ldots & 1 \\ \hline -1 & & & \\ \vdots & & C & \\ -1 & & & \end{array}\right) \tag{3.126}$$

we can reasonably argue that the $O(M^2)$ behavior of the condition number in this situation is due mainly to the fact that the scaling of the extra row and column added to C is orders of magnitude greater than that for C.

We solved the modified potential flow problem, this time looking at the errors in the value $\overline{\psi}$, as well as the errors in the solution and the normal derivative on the object. These errors and convergence rates are plotted in Figure 3.16 for $M = 16$. We observe that the errors in the solution and the derivative are essentially the same as the errors we got when we imposed a known value of the stream-function on the surface of the cylinder. As in the first case, these errors are converging at a second order rate. The fact that the convergence of $\overline{\psi}$ appears to be $O(h^3)$ is somewhat misleading. While the interpolation formula for the Dirichlet boundary condition in (3.125) is $O(h^3)$, our approximation to ψ, which is used in this interpolation formula, is only $O(h^2)$, and so we do not expect to be able to approximate the boundary conditions to third order. Moreover, from Figure 3.16, we can see that the convergence to $\overline{\psi}$ is quite erratic, and so it is difficult to say anything more than the convergence is at least second order, which is all we expect.

Convergence on a more general boundary

As we mentioned earlier, we don't necessarily expect convergence of our solution or derivative if we hold M fixed as we refine the grid. The only reason we were only able

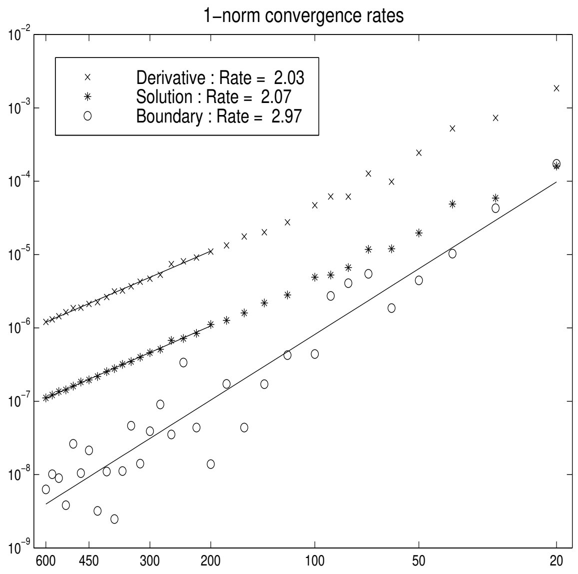


Figure 3.16: 1-norm convergence rates for potential flow problem in which value of stream-function on the boundary is unknown. The solid lines are the best-fit lines used to determine the convergence rates shown in the legend.

to get convergence in the examples we considered earlier was that the exact derivative could be represented exactly for all $M \geq 4$. Now, we consider a situation in which this is not the case.

The problem we consider now is to determine the potential flow field around an ellipse in a spatially periodic domain. We choose an ellipse, rather than some more general shape so that that for each value of M, we represent the same boundary.[2] For this problem, we set the average velocity at the left edge equal to 1, rather than requiring that the the velocity all along this edge be 1, as we did earlier. In Figure 3.17, we show the potential flow field computed for this problem.

To see how the value of the derivative changes as we refine the grid and refine the number of interface points, we computed to solution for pairs (N, M) set to $(100, 8)$, $(200, 16)$, $(400, 32)$ and $(800, 64)$. In Figure 3.18, we plot the the normal derivative ψ_n along the boundary of the ellipse obtained for each pair. The first plot shows that the derivative is clearly converging as the grid is refined. In the second plot, we show the difference between the derivative computed on the three coarsest grids ($N = 100$, $N = 200$ and $N = 400$) and that computed on the finest grid ($N = 800$). Again we see quite clearly that the solution is converging as the grid is refined. We

[2]One important change we need to make to the code is to be able to represent the interface using a different number of points that what we use to represent the unknown jumps.

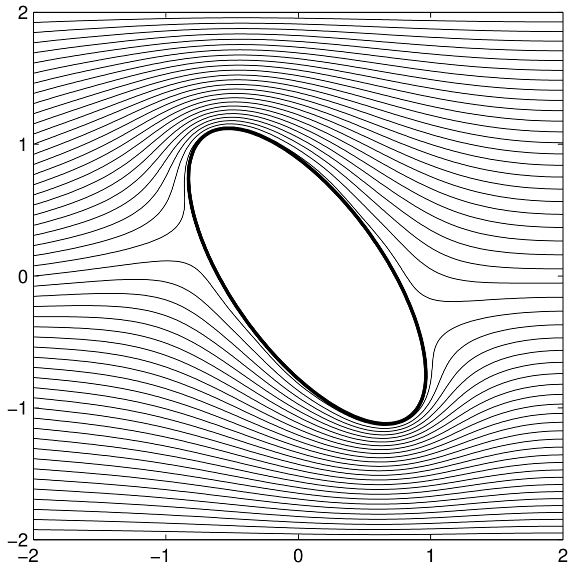


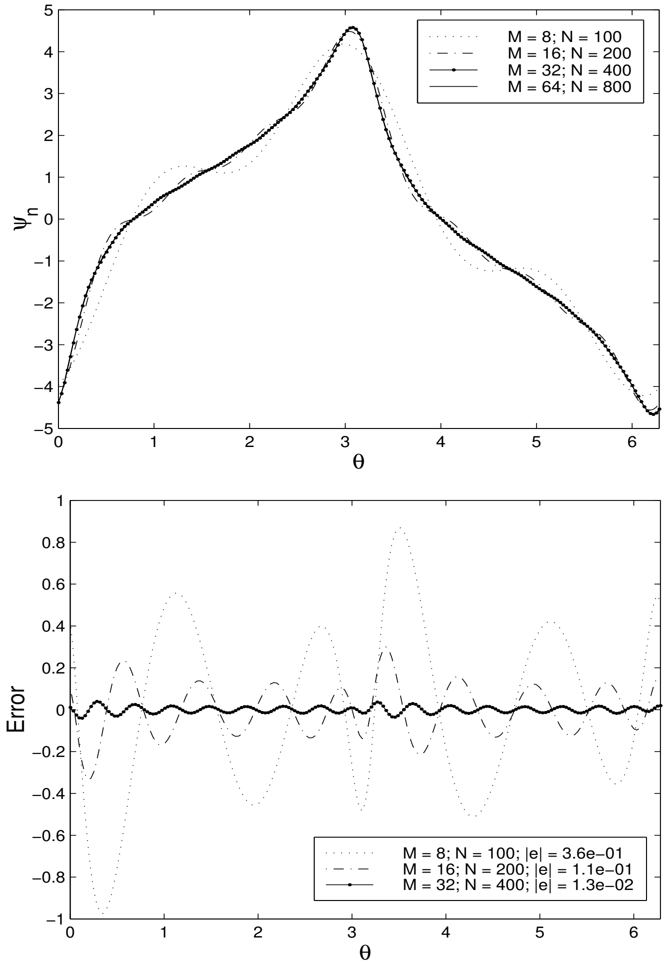
Figure 3.17: Solution to potential flow around single cylinder in spatially periodic domain. Computational grid is 200×200 and $M = 16$.

note that for this particular problem, the derivative on the coarsest grid, with $M = 8$ interface points is not very accurate, although the matrix for the unknown jumps was well conditioned. We conclude that for problems with irregular boundaries, it may be especially important to do grid refinement studies in order to be sure that the solution has an acceptable accuracy.

Solving for stream-lines in a general multiply-connected domain

In this next example, we solve for stream-lines around several embedded irregular objects. The objects were designed using a `Matlab` drawing package we have written for such purposes and are placed arbitrarily in the square computational domain. Each object is represented by 16 interface points. As with all objects presented so far, we compute normal and tangential interface derivative information using Fourier interpolation. We impose the no-circulation condition on the flow around each object and require that the surface of each object be a stream-line. On the computational domain, we impose periodic boundary conditions in both x and y. We compute the streamfunction on a 240×240 grid, about the minimum size allowable for the given shapes.

The computed streamfunction for these objects is shown in Figure 3.19. We see that the Immersed Interface Method apparently has no difficulty handling the more complicated geometry. In Figure 3.20, we show a close up of closely packed contours



(a)

(b)

Figure 3.18: (a) Normal derivatives along boundary of single ellipse shown in Figure 3.17. Derivative computed on four different grids are shown. (b) Difference between solution on three coarsest grids ($N = 100$, $N = 200$ and $N = 400$ and finest grid $N = 800$. Horizontal axis value θ is the angle from the lower tip of the ellipse, measured in a counter clockwise direction.

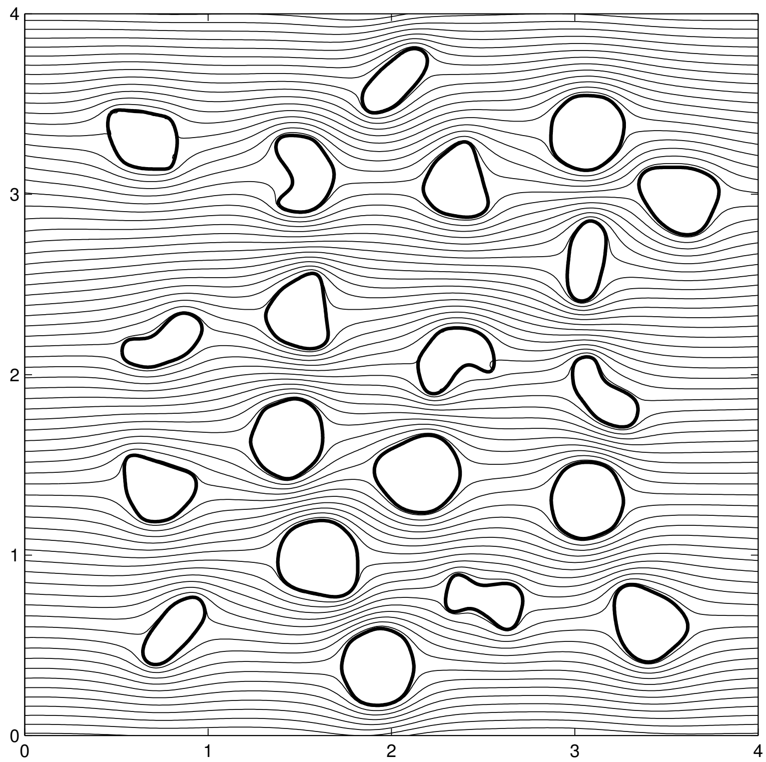


Figure 3.19: Solution to multi-body flow problem with several embedded objects. Computational grid is 240×240.

around the object located centered at approximately $(2.5, 0.8)$. We also show the same object embedded in the 240×240 computational grid with interface points marked.

3.5.2 Solving parabolic equations

In this example, we used the approach described in 3.3.5 to solve the parabolic equation given by

$$u_t = \nu \nabla^2 u, \qquad (x, y) \in \Omega \tag{3.127}$$

subject to the flux boundary conditions $\partial\omega/\partial n = g$ a the boundary of Ω. The general parabolic equation is used to model heat and mass transfer by diffusive processes and arises in a number of physical contexts. We are particularly interested in solving the parabolic equation so that we can solve Stokes flow problem, which we model as a coupled parabolic-elliptic system to obtain an appropriate flux of vorticity needed to satisfy the no-slip boundary condition on rigid boundaries.

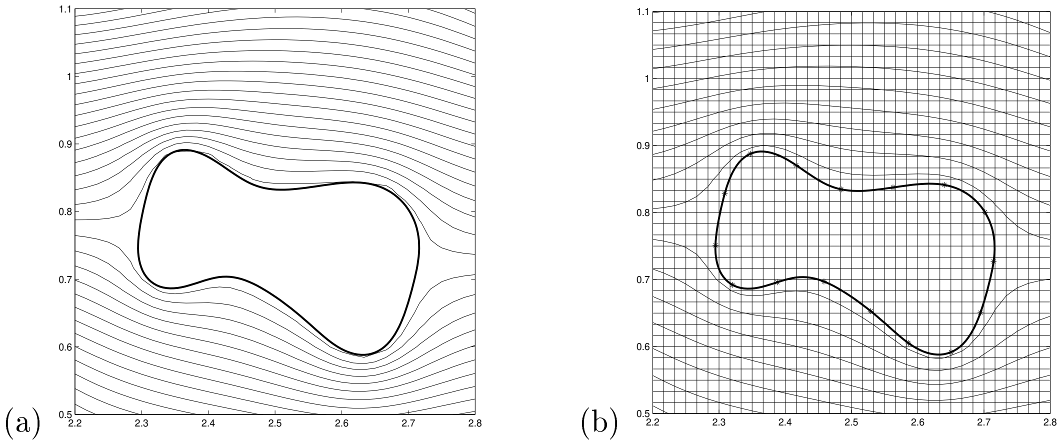


Figure 3.20: (a) Close-up of streamlines around object in multiply-connected domain. (b) Close-up with 240×240 computational grid. Interface points are marked by $*$'s.

We tested our parabolic solver for both Euler and Crank-Nicolson discretization schemes and determined the order of convergence by comparing our computed results to a reference solution. Our computational domain is the square domain $[0,1] \times [0,1]$ minus a circle of radius 0.25 centered at the point $(0.5, 0.5$. For our reference solution $u_{ref}(x,y,t)$, we use

$$u_{ref}(x,y,t) = e^{-t\nu\pi^2(k_x^2+k_y^2)} \cos(\pi k_x x)\ \cos(\pi k_y y) \tag{3.128}$$

where (k_x, k_y) are integer wave numbers, both set to 2 in the present example. Using u_{ref}, we initialize the solution at time $t = 0$ using

$$u_{ij}^0 \equiv \cos(\pi k_x x_i)\ \cos(\pi k_y y_j) \tag{3.129}$$

We also use the reference solution to determine flux boundary conditions on the boundary of the computational domain and on the circle. On the computational domain, we impose a no-flux condition, and on the boundary of the circle, we impose the condition

$$\begin{aligned} \frac{\partial u}{\partial n} &\equiv \nabla u \cdot \vec{n} \\ &= -4\pi e^{-t\nu\pi^2(k_x^2+k_y^2)} \left(k_x \sin(k_x \pi x)(x-0.5) + k_y \sin(k_y \pi y)(y-0.5)\right) \end{aligned} \tag{3.130}$$

At the 8 interface points, we initialize the jumps in the solution and its derivatives at time level $t = 0$, again using the reference solution. The jumps are

$$\begin{aligned} w_k &\equiv -\cos(\pi k_x \alpha_x^k)\ \cos(\pi k_y \alpha_y^k) \\ v_k &\equiv k_x \pi\ \sin(\pi k_x \alpha_x^k)\ \cos(\pi k_y \alpha_x^k)\, n_x^k + k_y \pi\ \cos(\pi k_x \alpha_x^k)\ \sin(\pi k_y \alpha_x^k)\, n_y^k \\ r_k &\equiv \left(1 + \Delta t\, \nu\, \beta_1\, \pi^2\, (k_x^2 + k_y^2)\right) w_k \end{aligned} \tag{3.131}$$

where (n_x^k, n_y^k) is the normal to the circle at the interface point $\alpha^k, k = 1, \dots 8$.

Using these initial conditions, we solved the parabolic problem for time steps $t = 0, 0.5, \dots 5$. In Figure 3.21, we show a mesh plot of our computed solution at time level $t = 5$. In interior of the circle (outside of the domain Ω) the solution should be identically zero, since this is the condition implicitly imposed by our choice of jump conditions. Figure 3.21 shows that we resolve the discontinuity in u at the boundary $r = \sqrt{(x-0.5)^2 + (y-0.5)^2} = 0.25$ quite well. The agreement with the reference

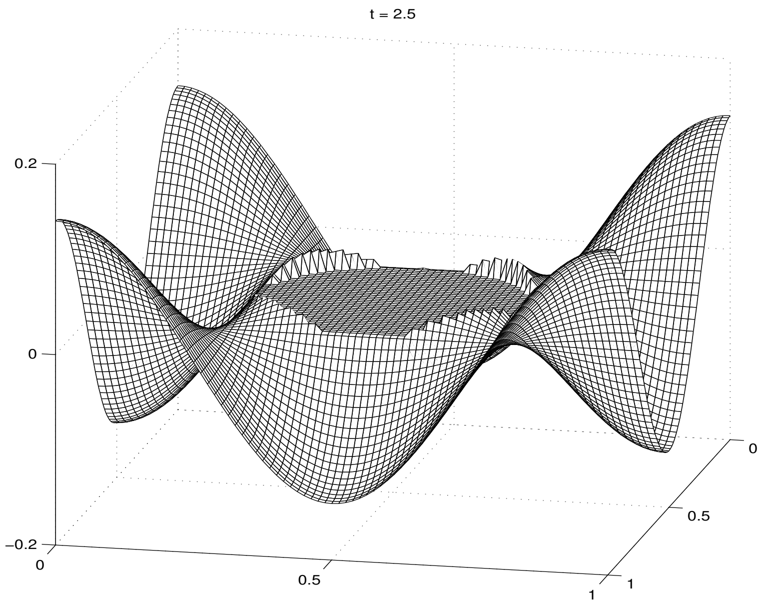


Figure 3.21: Solution at $t = 2.5$. to parabolic problem using the Immersed Interface Method. Solution is zero in inner circle. Computational grid is 80×80.

solution is seen even better in Figure 3.22, where we show a slice through the solution at times $t = 0, 1, \dots 5$. Here, the reference solution, also taken to be zero outside of the problem domain, is plotted on top of the computed solution.

To test the convergence of the scheme, we computed the solution on several different grids and plot the errors in Figure 3.23 for Backward Euler and Crank-Nicolson. Backward Euler for this problem actually converges at a super-linear rate, while Crank-Nicolson is very nearly second order. Also, the results using Crank-Nicolson are more accurate than those obtained using Backward Euler. One unsettling discovery we made was that Crank-Nicolson did not converge on all grids, and in the plots, we only show results for those grids that converged. We are not entirely sure whether the problem lies with some particular alignment of the irregular boundary with the grid, or whether there is something more subtle occurring with the Crank-Nicolson

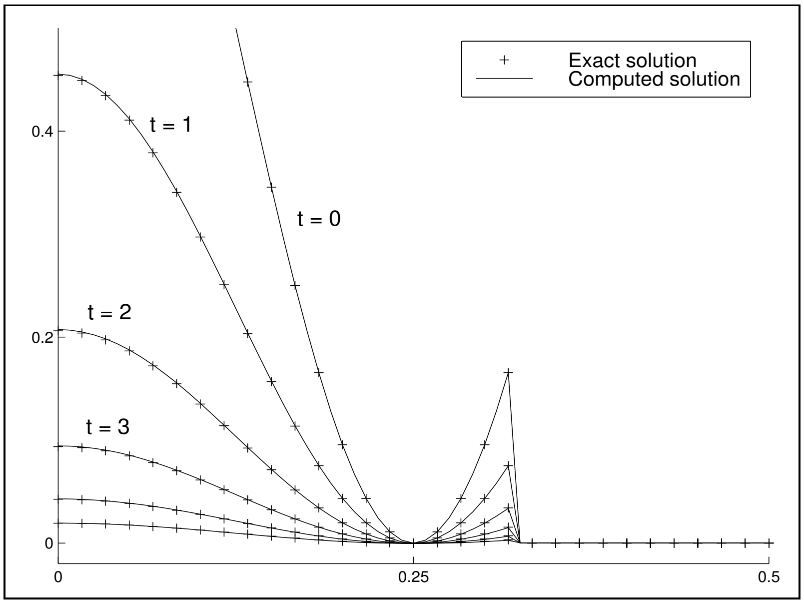


Figure 3.22: Slices of reference and computed solution along the diagonal line segment $y = x$ at times $t = 0, 1, \ldots 5$. Computational grid is 120×120. The reference solution is shown only at 30 points.

scheme when used with the Immersed Interface Method in the manner we have described. Crank-Nicolson is not an L-stable scheme, whereas Backward Euler is, and so Crank-Nicolson may be quite sensitive to small oscillations in the data. Because of the unpredictable behavior of Crank-Nicolson for this problem, we use Backward Euler for all of our calculations involving the streamfunction vorticity equations.

Finally, we give an example in which we use the parabolic solver in a multiply-connected domain. The domain we use consists of several ellipses embedded in an insulated box. At the boundary of each ellipse, tracer material is allowed to diffuse into the domain at a rate of one unit per second. No-flux boundary conditions are imposed on the boundary of the computational domain. In Figure 3.24, we show the contours of the tracer material at time level 5.

3.5.3 Solving the Stoke's flow equations

In our first example, we solved the potential flow equations which are the solution to steady, incompressible, inviscid flow. As a step towards solving the full Navier-Stokes equations in stream-function vorticity formulation, we now solve the steady and unsteady Stokes flow equations. Like the potential flow equations, Stokes flow governs incompressible flow. But unlike the potential flow equations, where we only

(a) (b)

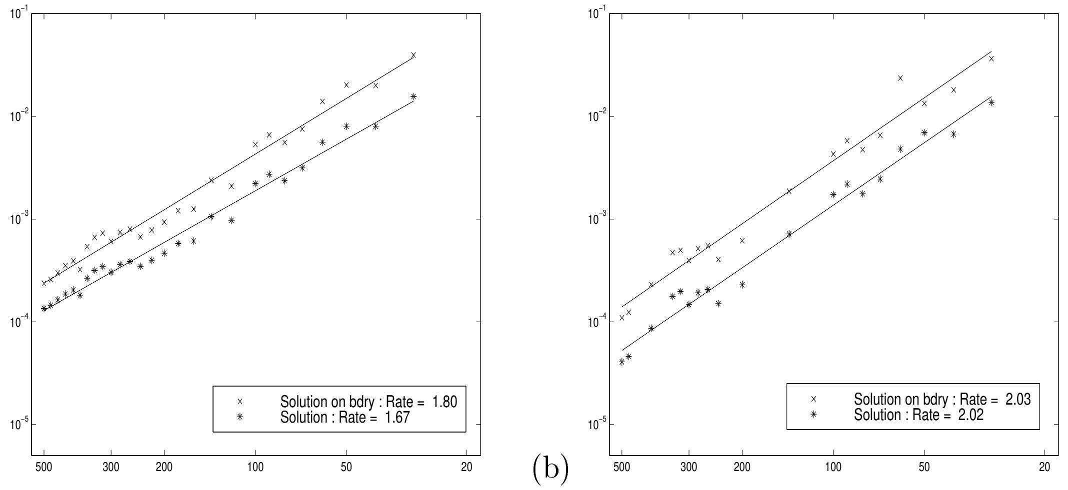


Figure 3.23: One-norm convergence rates for (a) Backward Euler and (b) Crank-Nicolson time discretization schemes. Errors for both global solution and solution on the boundary are shown. Solid line is best-fit line used to compute convergence rates.

required that a no normal-flow condition on each object be satisfied, we we now require that a no-slip condition be satisfied as well.

In the Stokes flow approximation, we assume that advection term $(\vec{u} \cdot \nabla)\omega \ll 1$ in (1.1) is negligible, and so we drop this term to get the equations

$$
\begin{array}{rlrl}
\widehat{\omega}_t = \nu \nabla^2 \widehat{\omega}, & (x,y) \in \Omega & \dfrac{\partial \widehat{\omega}}{\partial n} = g, & (x,y) \in \partial\Omega \\
\nabla^2 \psi = -\widehat{\omega}, & (x,y) \in \Omega & \psi = \psi_j, & (x,y) \in \partial\Omega \\
\displaystyle\int_{\Omega_j} \frac{\partial \widehat{\omega}}{\partial n}\, ds = 0, & j = 1, M_{bodies} & \dfrac{\partial \psi}{\partial n} = 0, & (x,y) \in \partial\Omega
\end{array}
\tag{3.132}
$$

for $\widehat{\omega}(x,y,t)$ and $\psi(x,y,t)$. Here, we use $\widehat{\omega}$ instead of ω in (1.1) to distinguish between the solution $\widehat{\omega}$ we obtain using the Immersed Interface Method, and the solution ω we obtain using the finite volume method described in 2. Also, in the following discussion, we assume that we only have a single embedded object defined by the region $\Omega^+ = R\backslash\Omega$. This assumption is made for notational convenience only; the method naturally generalized to multiple objects.

The Stoke's flow equations describe what is sometimes called "creeping flow" and models flow in regimes for which the Reynolds number, the non-dimensional parameter characterizing the ratio of the inertial terms to diffusive terms, is small. We are primarily interested in solving these equations to obtain the unknown flux condition g, which we will use as the diffusive flux in our finite volume scheme, although the Stokes flow equations are of interest in their own right.

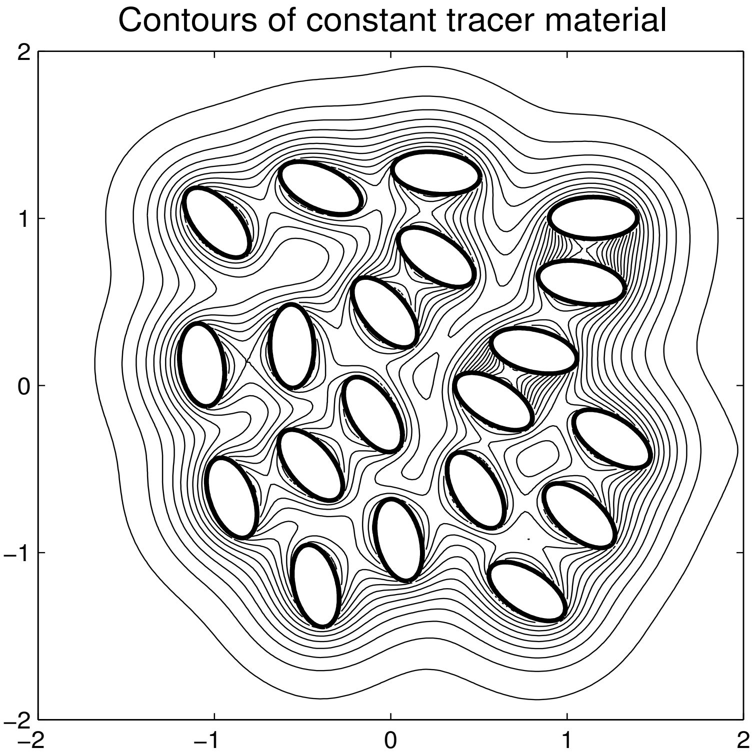


Figure 3.24: Contours at time $t = 5$ of tracer material diffused into domain through boundary of each embedded ellipse. Flux of material is set to 1 at each boundary. Flux on the boundary of the computational domain is set to 0. Computational grid was 200×200.

LeVeque and Li, in [36], describe how one can solve a similar set of equations in primitive variable formulation for solving Stoke's flow induced by movement of a deformed elastic interface as it returns to its equilibrium shape. They solved an elliptic equation for pressure and used a quasi-Newton iteration to determine the location of the elastic interface. There were no solid boundaries involved in the flow, however, and so they were not concerned with imposing no-slip boundary conditions. Also, they had dropped the inertial terms in the equation and so the only time dependent term in their equations was the forcing term due to the elastic membrane. The problem we solve is simpler than their problem in the sense that our boundaries are fixed; however, we solve the unsteady Stoke's flow in the presence of solid boundaries and so must impose the no-slip condition.

One difficulty in solving these equations is that the physical problem explicitly gives us two boundary conditions for ψ but no conditions on $\widehat{\omega}$. The two conditions on ψ correspond to the physical requirement that on all solid boundaries, we must have no normal-flow (ψ = constant) and no-slip ($\psi_n = 0$). There is no such clearly defined boundary conditions on $\widehat{\omega}$, and in fact the existence of such conditions would lead to an overdetermined system. There is, however, the physical notion that vorticity is generated at solid boundaries in response to forces imposed by the solid boundary on the fluid which act to keep the solid body at rest. We model this notion of vorticity generation by imposing flux boundary conditions on solid boundaries. This condition is given by $\partial\omega/\partial n = g$, given in (3.132), for some unknown flux g at solid boundaries. This unknown value of g must be obtained by solving the coupled system given in (3.132). This type of boundary condition is different than that used by traditional approaches, most of which impose a Dirichlet condition on the vorticity.

Since both the jumps $[\widehat{\omega}]$ and $[\widehat{\omega}_n]$ are unknown, we do not discretize any boundary conditions for $\widehat{\omega}$; instead we set up a linear system for which both jumps for $\widehat{\omega}$ are unknowns. The only boundary conditions we explicitly impose are those on ψ. Discretizing the parabolic and elliptic equations for $\widehat{\omega}$ and ψ, we get

$$\begin{aligned} \mathcal{H}\widehat{\omega}_{ij}^{m+1} + \overline{\mathcal{D}}'_{ij}\overline{\mathbf{w}}_{ij}^{m+1} + \overline{\mathcal{D}}_{ij}\overline{\mathbf{v}}_{ij}^{m+1} + \overline{\mathcal{D}}_{ij}^{H}\overline{\mathbf{r}}_{ij}^{m+1} = \\ \mathcal{H}'\widehat{\omega}_{ij}^{m} + \overline{\mathcal{D}}'_{ij}\overline{\mathbf{w}}_{ij}^{m} + \overline{\mathcal{D}}_{ij}\overline{\mathbf{v}}_{ij}^{m} + \overline{\mathcal{D}}_{ij}^{H}\overline{\mathbf{r}}_{ij}^{m}, \quad x_{ij} \in R \\ \mathcal{A}\psi_{ij} + \mathcal{D}_{ij}^{H}\mathbf{r}_{ij} = -\,\widehat{\omega}_{ij}^{m+1}, \quad x_{ij} \in R \end{aligned} \tag{3.133}$$

The unknowns in this equation are $\widehat{\omega}_{ij}$, ψ_{ij}, and the jump vectors $\overline{\mathbf{w}}$, $\overline{\mathbf{v}}$. Jump vectors $\mathbf{w}$ and $\mathbf{v}$ are known and equal to zero, since we have that $\psi = \psi_n = 0$ on the irregular boundary. To close the system with respect to the unknown jumps in $\widehat{\omega}$, we discretize the two boundary conditions for ψ. The conditions, $\psi = \overline{\psi}_j$ and $\psi_n = 0$ are discretized using the Immersed Interface Method as

$$\begin{aligned} \mathcal{I}_k\psi_{ij}^k + \mathcal{I}_k^f\mathbf{r}_k &= \overline{\psi}, \qquad k = 1,\dots M \\ \mathcal{I}_{n,k}\psi_{ij} + \mathcal{I}_{n,k}^f\mathbf{r}_k &= 0, \qquad k = 1,\dots M \end{aligned} \tag{3.134}$$

These expressions are the $2M$ equations we need to close the system with respect to the unknown jumps $\overline{\mathbf{w}}$ and $\overline{\mathbf{v}}$.

Finally, in the general multiply-connected domain case, the value of the stream-function on the boundaries is unknown. Only now, unlike in the potential flow case, we have already prescribed the circulation around each object by specifying the value of ψ_n as part of the problem data. We no longer have this degree of freedom, as we did in the potential flow case. However, in the viscous flow case, the pressure enters into the problem, and is given as a function of position on the boundary of a single inclusion $\partial\Omega_j$ as

$$P(s) + P_0 = \rho\nu \int_0^s \frac{\partial\omega}{\partial n}\, ds \tag{3.135}$$

where P_0 is a known pressure at some point on the boundary and ρ is the density of the fluid. This will be a single valued function only if

$$\int_{\partial\Omega} \frac{\partial \omega}{\partial n}\, ds = 0 \tag{3.136}$$

This equation replaces the no-circulation condition given earlier and closes the system with respect to the unknown values of the streamfunction on the embedded objects.

To solve the equations, we set up an iteration on the unknown vectors $\overline{W}$ and $\overline{V}$, the entries of which are the jumps $\overline{w}_k$ and $\overline{v}_k$ the $2M$ interface points. With guesses to these vectors, we could solve the parabolic equation for $\widehat{\omega}$ (skipping the step involving solving for unknown jumps, all jumps are given by $\overline{W}$ and $\overline{V}$). This gives us the right hand side for the elliptic equation for ψ as well as the jump $\mathbf{r} = -\overline{\mathbf{w}}$. Other jumps $\mathbf{w}$ and $\mathbf{v}$ needed for the elliptic solve are known and identically zero, as we explained above. Formulating the above procedure as a matrix-vector multiply for the unknown vectors $\overline{W}$ and $\overline{V}$, we can solve for these unknowns.

Since each matrix-vector multiply in this iteration requires two fast Poisson solves (one for the parabolic equation, and one for the elliptic equation), we want to form the matrix as infrequently as possible. At each time step, only the right hand side changes, and so we have found it to be worthwhile to form the resulting matrix explicitly, factor it once and then backsolve the system at each time step. Of course, as soon as the time step Δt changes, the matrix $\mathbf{C}$ must be reformed, since its coefficients depend on this time step. However, even when we include the advection, we have found that one can use the factored matrix for quite a few time steps, and usually only needs to form and re-factor the matrix very infrequently.

We implemented and tested the above scheme and used it to solve the Stoke's flow equations given in 3.132. The domain we use is the square domain $[0, 4] \times [0, 4]$ embedded with two circular cylinders of radius 0.5, one centered in the upper right quadrant of the domain and one centered in the lower left quadrant of the square domain. On the boundaries of the computational domain, we impose biperiodic boundary conditions given by

$$\begin{aligned} \psi(4, y) =& \psi(0, y) \\ \psi(x, 4) =& \psi(x, 0) + \triangle\psi. \end{aligned} \tag{3.137}$$

where $\triangle\psi$ is the desired shift in ψ from the bottom of the domain to the top. Imposing this condition on the given domain is equivalent to solving the problem for $\triangle\psi = 0$ and then adding $\triangle\psi y/4$ the resulting solution.

The equations for Stoke's flow were solved to steady state (reached at about t = 20). The vorticity and streamline contours are plotted in Figure 3.25 and are in excellent agreement with those plots given in [50] and reproduced in Appendix A.

As we will see in Chapter 4, the above scheme for solving the Stoke's equations is used to determine the flux of vorticity g needed in the full stream-function vorticity equations.

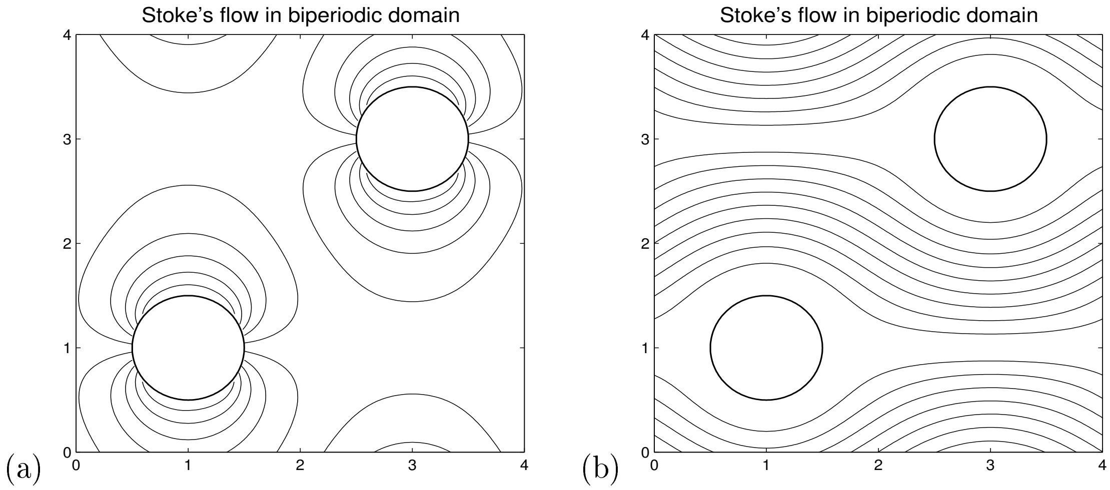


Figure 3.25: Contours of (a) vorticity and (b) the streamfunction for Stokes flow problem in biperiodic domain. The computational grid is 160×160.

Chapter 4

SOLVING THE STREAMFUNCTION-VORTICITY EQUATIONS IN IRREGULAR REGIONS

In Chapter 2 where we described our advection-diffusion algorithm for complex geometries, we assumed that our underlying velocity field was a steady state velocity field. Furthermore, we only imposed no-normal flow boundary conditions on the boundaries of the irregular obstacles. In this Chapter, we describe how we can solve the full Navier-Stokes equations in streamfunction-vorticity formulation. In particular, we discuss how we satisfy the no-slip boundary condition on the boundary of irregular objects.

The streamfunction-vorticity equations, introduced in Chapter 1, are given by

$$\begin{aligned}
\omega_t + (\vec{u}\cdot\nabla)\omega =& \nu\nabla^2\omega, \quad (x,y)\in\Omega \\
\nabla^2\psi =& -\omega, \quad (x,y)\in\partial\Omega \\
u = \psi_x, & \qquad v = -\psi_y \\
\psi =& \psi_j, \quad (x,y)\in\partial\Omega \\
\psi_n =& 0, \quad (x,y)\in\partial\Omega
\end{aligned} \tag{4.1}$$

The boundary values for the streamfunction are chosen to enforce the no-normal flow and no-slip boundary conditions. A no-normal flow condition is satisfied when $\vec{u}\cdot\mathbf{n} = 0$, or equivalently when ψ is a constant along contiguous segments of the boundary. In a channel flow, for example, one usually sets the value of ψ to be one constant at the lower wall and a second constant on the upper wall. The mean velocity, or average velocity across the channel is then the difference of these two boundary values. In the channel flow setup, we can also impose a no-slip condition by requiring that the tangential velocity, or equivalently, the normal derivative of ψ be equal to the wall speed. In many physical situations, the wall speed is 0 and so imposing $\psi_n = 0$ at both the upper and lower boundaries is the appropriate boundary condition. However, a few classic test problems involve a moving wall. In the driven cavity problem, for example, ψ_n is set to the velocity of the moving lid. As was discussed in Section 3.5.3, we do not have explicit conditions on ω, and so in order to solve the transport equation numerically, we must first solve for an unknown flux of vorticity.

This formulation of the Navier-Stokes equations is used extensively for two-dimensional flows because of the fact that the divergence-free condition $\nabla\cdot u = 0$ is automatically satisfied. For our purposes, the equations are especially convenient because they naturally decouple into an advection-diffusion equation

$$\omega_t + (\vec{u} \cdot \nabla)\omega = \nu \nabla^2 \omega \tag{4.2}$$

governing the transport of vorticity and an elliptic equation

$$\nabla^2 \psi = -\omega \tag{4.3}$$

for the streamfunction. As we showed in Chapter 2, the velocities $(u(x, y, t), v(x, y, t))$ are computed by differencing the streamfunction. Since vorticity is a conserved quantity in two dimensions, we can use the finite volume methods developed in the Chapter 2 to solve the vorticity transport equation. To solve the elliptic equation for the streamfunction, we use the Immersed Interface Methods developed in Chapter 3. The main challenge in solving these equations will be imposing the no-slip boundary condition.

In what follows, we describe in detail how we combine the finite volume methods described in Chapter 2 with the finite difference methods described in Chapter 3 to solve the full streamfunction-vorticity equations. First, we use our hybrid finite volume/finite difference scheme to solve a simplified set of equations which do not satisfy the no-slip boundary condition. The method described for these inviscid equations turns out to be a straightforward application of the advection-diffusion algorithm described in the previous chapter. With the basic algorithm laid out, we can then discuss how we use the solution to a Stoke's flow problem to impose the no-slip boundary conditions. To handle multiply-connected domains, we impose condition (3.136) requiring no net flux of vorticity across embedded objects.

Throughout the early part of this chapter, we always have in the back of our mind that we wish to solve these equations in irregular, multiply-connected domains. Initially, however, we omit some of the details associated with the proper treatment of the such domains in order to provide an overview of our approach. However, all of the details necessary to evaluate our algorithm and implement it will be provided, either in a later section of this chapter, or in appendices, where they are described in full.

4.1 Survey of methods

Methods for solving the streamfunction vorticity equations has a long history, dating at least back to 1933 when Thom proposed his now well-known "Thom's Formula" [51]. At the center of most of these discussions is the treatment of the two boundary conditions on ψ and the absence of conditions on ω.

Most of the early methods for solving the viscous streamfunction vorticity equations employed what have been termed "boundary vorticity formulas" [44]. These formulas are usually obtained by manipulating finite difference discretizations of the elliptic equation coupling ψ and ω, and the extra boundary condition on ψ to obtain a formula for ω on the boundary of the domain. But because these formulas usually involve both ω and ψ at the same time level, one must iterate in order to obtain the desired value of the vorticity at the boundary.

To illustrate these ideas more concretely, we derive Thom's formula for a value of vorticity $\omega_{1,j}$ at the left edge of a rectangular domain. We assume that we have node based values $\omega_{i,j}$ and ψ, $i, j = 1, \ldots N$. At the boundary of that domain, we have the elliptic equation $\nabla^2\psi = -\omega$ linking ψ and ω, and the extra boundary condition $\psi_n = 0$ on ψ. These two equations can be discretized using finite differences as

$$\begin{aligned} \frac{\psi_{1,j+1} + \psi_{0,j} + \psi_{2,j} + \psi_{1,j-1} - 4\psi_{1,j}}{h^2} &= -\omega_{1,j} \\ \frac{\psi_{2,j} - \psi_{0,j}}{2\Delta x} &= 0 \end{aligned} \tag{4.4}$$

Substituting the value for $\psi_{0,j}$ obtained from the second equation into the first equation and using the Dirichlet conditions $\psi = 0$ on ψ, we get that

$$\text{(Thom's Formula)} \qquad \omega_{1,j} = \frac{2\psi_{2,j}}{\Delta x^2} \tag{4.5}$$

which is Thom's formula for vorticity. These formula is only first order accurate, however, and so researchers have derived more accurate formulas. One classic formula, derived by Briley in 1971 is given by

$$\text{(Briley's Formula)} \qquad \omega_{1,j} = \frac{1}{18\Delta x^2}\left(108\psi_{2,j} - 27\psi_{3,j} + 4\psi_{4,j}\right). \tag{4.6}$$

Briley's formulas is fourth order accurate. E in [54] gives a table of these formulas shown above, as well as several others. Such formulas have been commonly called "local formulas" in the literature, because they depend only on values of ψ near the boundary.

These formulas by themselves, however, do not provide the necessary boundary conditions on ω. They must be incorporated into the coupled $\omega - \psi$ equations, which must be solved to obtain the boundary values of vorticity. Using a explicit discretization of the advection term and an implicit discretization of the diffusion term in (4.1), we arrive at the discretized set of equations given as

$$\begin{aligned} \frac{\omega^{n+1} - \omega^n}{\Delta t} &= -\left(u^n D_x + v^n D_y\right)\omega^n + \nu\nabla_h^2\left(\frac{\omega^{n+1} + \omega^n}{2}\right) \\ \nabla_h^2\psi^{n+1} &= -\,\omega^{n+1} \\ u^{n+1} = D_y\psi^{n+1}, &\qquad v^{n+1} = -D_x\psi^{n+1} \\ \psi &= \overline{\psi}_j, \quad (x,y) \in \partial R \\ \omega_b^{n+1} &= \frac{2\psi_{in}^{n+1}}{2h^2} \end{aligned} \tag{4.7}$$

Here, we have used Thom's formula as the representative boundary vorticity formula. The value ψ_{in} is a value of ψ on any of the four boundaries in the rectangular flow domain, and D_x and D_y represent the appropriate approximations to first derivatives in the x and y directions, respectively. The mesh width is given by $\Delta x = \Delta y = h$. The boundary value ω_b^{n+1} is required by the diffusion term, and so the equations are coupled through their dependence on ω_b^{n+1}. For this reason, we are led to a linear system of equations which we must solve for ω_b^{n+1}.

Although the system for ω_b^{n+1} is linear, attempts to solve this using simple iteration schemes have evidently only met with limited success. In general the iteration typically used has been found to diverge unless some sort of relaxation parameter is used [44], [54]. In fact, several early papers in this field were devoted to determining optimal values for this relaxation parameter [31]. Surprisingly, few researchers, to our knowledge, have considered formulating this set of equations as a linear system of the form $A\omega_b^{n+1} = R$, and solving the system directly. Gresho, in [29], hints at a formulation of this type, but then later discounts the approach as impractical.

To perhaps avoid the problems associated with solving the above system iteratively, E and colleagues [54] have argued that it is not necessary to discretize the diffusion equation implicitly. Instead, they discretize the diffusion term explicitly, and eliminate the need for boundary conditions on ω_b at time level $n+1$. Local formulas are still used to obtain ω_b^{n+1} in terms of boundary values of ψ^{n+1}, but only ω_b^n is needed in the first equation. The chief advantage of this approach is that the equations are decoupled and so it is no longer necessary to solve a linear system to obtain ω_b^{n+1} at each time step.

An issue not addressed by many of the early approaches to solving the boundary vorticity formulas is that the vorticity is singular at the corners, and so boundary formulas at the corner can run into numerical problems. Huang, in [30] uses a variant of the local formula approach to devise a scheme which doesn't require boundary values of vorticity at the corner (or for that matter, anywhere along the boundary). Instead, he manipulates the formulas to obtain conditions in the interior of the domain, thus avoiding the corner singularity problem. Huang and his co-author do not solve the above linear system of equations for ω^{n+1} and ψ^{n+1} at each time step, but rather solve a non-linear system to obtain a steady state solution only. Additionally, they provide a convergence proof of their non-linear iteration.

In light of the fact that one must solve a linear system (either iteratively, or directly) to obtain the boundary values of vorticity, it seems somewhat misleading to call these formulas *local formulas*. Through the iteration procedure, the boundary value of vorticity is coupled to all values of ψ, not just those on the boundary. This notion that the boundary values of vorticity are really coupled globally to the streamfunction ψ is made explicit in another set of approaches, commonly called "global formulas". The main proponent of these formulas is Quartapelle, who derived an integral condition for vorticity that decouples the vorticity equation from the streamfunction equation. This vorticity integral condition, proposed by Quartapelle in [45], is based on the following theorem, proved by Quartapelle.

Theorem 1 (Quartapelle and Valz-Griz, 1981). *A function ω defined in Ω is such that $\nabla^2\psi = -\omega$, with $\psi = a$ and $\psi_n = b$ on the boundary of Ω if and only if*

$$\int_\Omega \omega\eta \, dxdy = \int_{\partial\Omega} \left(a\frac{\partial \eta}{\partial n} - b\eta \right) \, ds. \tag{4.8}$$

for any function η harmonic in Ω, i.e. any function η such that $\nabla^2\eta = 0$.

This leads to the following "split" formulation of the vorticity equations

$$\begin{aligned} \omega_t + (\vec{u}\nabla\omega) &= \nu\nabla^2\omega, & \int_\Omega \omega\eta \, dxdy &= \int_{\partial\Omega} \left(a\frac{\partial \eta}{\partial n} - b\eta \right) \, ds \\ \nabla^2\psi &= -\omega, & \psi &= a \text{ on } \partial\Omega \end{aligned} \tag{4.9}$$

The advantage of this formulation is that is doesn't involve ψ, and so again the equations in (4.1) can be solved independently. One can solve the advection-diffusion equation for ω using the vorticity integral condition, and then solve the equation for ψ using the Dirichlet (or "natural") condition. Another advantage to this approach, and one that is relevant to this thesis, is that the vorticity integral condition applies to irregular regions as well as regular regions. Unlike the local formulas derived above, it is valid in general regions, not just rectangular regions. The drawback most often cited in the literature to the implementation of this condition is, however, that harmonic functions η must be computed and stored before the computations can begin. These makes this approach of limited usefulness in, for example, a 3d setting, although a variant of this method, the Glowinski-Pironneau method eliminates the need for these harmonic functions. It is curious that this approach is also criticized for being complicated to implement [54], in part because the implementation requires the solution to a system $\mathbf{A}\lambda = \beta$ that must be solved at each time step.

Another representative "global method" for determining the correct boundary conditions on vorticity is given by Anderson [2]. Here, Anderson derives a constraint equation that the flux of vorticity $(\partial\omega/\partial n)$ must satisfy. This flux constraint is given by

$$\frac{\partial \omega}{\partial n} + \frac{\partial}{\partial n}\int_{\partial\Omega} \frac{\partial G}{\partial n}(\alpha, s)\omega(s,t) \, ds = \frac{1}{\nu}\frac{\partial}{\partial n}\int_\Omega G(\alpha, s) \left[\vec{u}\nabla\omega(\mathbf{x},t)\right]; dV \tag{4.10}$$

where $\alpha \in \mathcal{R}^2$ is the point on the boundary, and $G(x,s)$ is the free space Green's function satisfying $\nabla^2 G(x,s) = \delta(x-s)$. Because he is computing a flux condition on vorticity, Anderson emphasizes that his method exhibits vorticity creation at the boundary, and so is similar to vortex sheet methods proposed by Chorin. In fact, Anderson demonstrates that at least in one case, his formula for the flux of vorticity is exactly that required by Chorin's method, described in Section 1.2.5.

Other methods involve solving the fully coupled equations, or eliminating the vorticity term altogether and solving a biharmonic equation for ψ. An excellent survey of these different methods can be found in [44]

4.2 A Hybrid Immersed Interface/Finite-volume Method

To introduce our approach to solving the above system of equations in a finite volume setting, we first discuss a simpler set of equations, the inviscid streamfunction vorticity equations.

4.2.1 Solving the inviscid equations ($\nu = 0$)

Using methods from the previous chapters, it is fairly straightforward to solve the *inviscid* streamfunction vorticity equations, given by

$$\begin{aligned} \omega_t + \vec{u} \cdot \nabla \omega &= 0, \quad (x, y) \in \Omega \\ \nabla^2 \psi &= -\omega, \quad (x, y) \in \Omega \\ u = \psi_y, &\qquad v = -\psi_x \\ \psi &= \overline{\psi}, \quad (x, y) \in \partial\Omega \end{aligned} \tag{4.11}$$

in an irregular region Ω. The difference between these equations and the full equations given above is that here we only impose a no-normal flow condition on the velocity field at solid boundaries. The tangential component of the velocity ψ_n is not specified in the problem, and in general, the velocity at solid boundaries will not match the wall velocity.

In the finite volume setting, we treat the vorticity as a volume averaged quantity located at cell centers, and the streamfunction as a node-based quantity located at the corners of cells. Velocities are the edge averaged velocities described in Chapter 2. See Figure 4.1, for a diagram of this discretization.

Specifically, we use the discrete capacity function κ_{ij} to define ω_{ij} as

$$\omega_{ij}^n = \frac{1}{\kappa_{ij} \Delta x \ \Delta y} \int_{C_{ij}} \omega(x, y, t_n) \ dxdy. \tag{4.12}$$

The streamfunction values ψ_{ij} are defined as

$$\psi_{ij} \approx \psi(x_{i-1/2}, y_{j-1/2}), \tag{4.13}$$

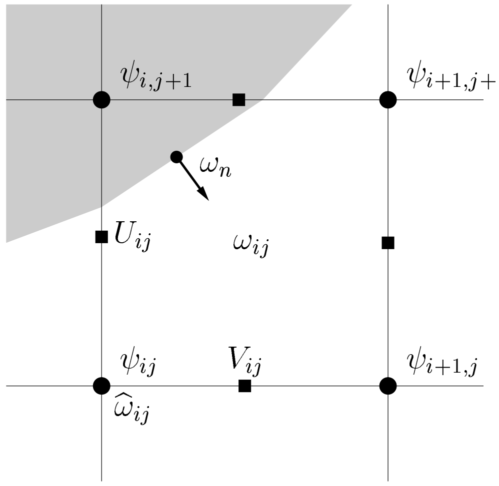


Figure 4.1: Discretization used in present algorithm for solving the streamfunction-vorticity equations.

and from 2.27 and 2.28, we have that

$$U_{i-1/2,j} = \frac{1}{\Delta y}\int_{y_{j-1/2}}^{y_{j+1/2}} \psi_y(x,y)\,dy \tag{4.14}$$

$$= \frac{1}{\Delta y}[\psi(x_{i-1/2}, y_{j+1/2}) - \psi(x_{i-1/2}, y_{j-1/2})] \tag{4.15}$$

and

$$V_{i,j-1/2} = -\frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} \psi_x(x,y)\,dy \tag{4.16}$$

$$= -\frac{1}{\Delta x}[\psi(x_{i+1/2}, y_{j-1/2}) - \psi(x_{i-1/2}, y_{j-1/2})]. \tag{4.17}$$

Using discretizations of the advection-diffusion equation given in Chapter 2, and the elliptic solver described in Chapter 3, we can outline the algorithm for solving the above set of inviscid equations in an irregular, multiply-connected region Ω. This algorithm is given in Figure 4.2. The multiply-connected domain is handled using the techniques described in Section 3.5.1.

The no-flow boundary conditions used in the advection step are imposed through our choice of $U^n_{i-1/2,j}$ and $V^n_{i,j-1/2}$, obtained by differencing the streamfunction. By requiring that ψ be equal to a constant $\overline{\psi}_j$ along the boundary of each embedded object

Solving the inviscid streamfunction-vorticity equations

Given cell-averaged values ω^0, obtain $\widehat{\omega}^0$ by averaging ω^0 onto grid nodes. Solve $\nabla^2\psi^0 = -\widehat{\omega}^0$ subject to Dirichlet boundary conditions $\psi = \overline{\psi}$ on each irregular boundary. The exact value of the Dirichlet condition is to be determined as part of the solution process. Impose a no-circulation condition around each object, as described in Chapter 3, and free-stream conditions on ψ^{n+1} on the boundary of the computational domain. Define velocities $U_{i-1/2,j}$ and $V_{i,j-1/2}$ using ψ^0.

For $n = 0, 1, \ldots$

1. Advect the vorticity using the finite volume advection scheme and capacity form differencing in CLAWPACK to get ω^{n+1}. Impose no-flow conditions on all solid boundaries.
2. Average ω^{n+1} onto grid nodes to get $\widehat{\omega}^{n+1}$.
3. Use the Immersed Interface Method to solve $\nabla^2\psi^{n+1} = -\widehat{\omega}^{n+1}$, subject to Dirichlet conditions $\psi = \overline{\psi}_j^{n+1}$ on the boundary of object Ω_j. A no-circulation condition is imposed to obtain unknown values $\overline{\psi}_j$.
4. Compute $U^{n+1}_{i-1/2,j}$ and $V^{n+1}_{i,j-1/2}$ using ψ^{n+1}.

Figure 4.2: Algorithm for solving the inviscid streamfunction vorticity equations

Ω_j, we obtain velocities satisfying the no-flow condition. At the boundaries of the computational domain, which for our purposes here, we assume do not coincide with physical boundaries, we impose a no-inflow boundary condition and an appropriate inflow and out-flow conditions. If our flow is exactly parallel to some boundary of our computational domain, then it doesn't matter what values we choose to put in ghost cells — no vorticity will flow across that computational boundary.

Only two items need to be described in more detail. These are the averaging process we allude to above, and the manner in which we obtain the jumps $[\omega]^{n+1}$ needed by the Immersed Interface Method at the irregular boundaries.

The averaging process we use is designed so that the value $\widehat{\omega}_{ij}$ is roughly the average value of ω_{ij} in neighboring cells. In particular, we didn't want to introduce any unnecessary oscillations in the data.

Away from the interface, we determine $\widehat{\omega}_{ij}$ using

$$\widehat{\omega}_{ij} = \frac{\omega_{i-1,j} + \omega_{i-1,j-1} + \omega_{i,j-1} + \omega_{i,j}}{4}. \tag{4.18}$$

Near the interface, the averaging must take into account partial cells. For each cell, we determine the number of corners n_{ij} of the cell that are in the flow-domain. For cells completely in the no flow-domain, $n_{ij} = 0$ and for cells completely in the flow domain, $n_{ij} = 4$. For partial cells, $0 < n_{ij} < 4$. Using this number, the vorticity in each cell is distributed to nodes using the formula

$$\widehat{\omega}_{ij} = \frac{1}{\sum_{k,\ell \in \{-1,0\}} \kappa_{i+k,j+\ell}} \sum_{k,\ell \in \{-1,0\}} \frac{\kappa_{i+k,j+\ell}}{n_{i+k,j+\ell}} \omega_{i+k,j+\ell}. \tag{4.19}$$

These formulas preserve constant functions, a desirable feature which prevents the introduction of unnecessary oscillations.

Second, in order to solve the elliptic equation for ψ, we need jumps in ω at the boundary in order to form the vector $\mathbf{r} = [0, 0, 0, 0, 0, -[\omega]]^T$ required by the Immersed Interface Method. We obtain the necessary jumps by using a second order accurate one-sided approximation to ω at a point on the boundary. Assuming that in the no-flow domain, $\omega_{ij} \equiv 0$, the jump in the ω at the boundary is equal to this approximation, with a possible sign change. Obtaining the stencil for the one-sided formula requires some care, because not just any choice of six points can be used in the stencil. In Section 3.3, we describe which stencils can be used.

To illustrate this algorithm, we investigate vortex dynamics in an enclosed square region embedded with two circular cylinders. In this simulation, we place a circular cylinder in the upper left and lower right quadrants of a square domain. In the other two quadrants, we place two circular patches of vorticity of equal strength. We impose no-flow conditions on all solid boundaries. The initial vorticity patches induce a velocity field in the domain which then advects the vorticity further. Eventually, the two smaller vortex patches coalesce into a single bigger patch of vorticity. The results of this simulation at four different times are shown in Figure 4.3.

The algorithm described in Figure 4.2 provides the framework for the algorithm we will use to solve the fully viscous equations. Aside from the addition of a diffusion term in the viscous case, the main difference between the inviscid algorithm presented in Figure 4.2 and its viscous counterpart is that in the latter case, we must impose an inhomogenous boundary condition on the vorticity in order to satisfy the no-slip condition at solid boundaries.

4.2.2 Solving the viscous equations ($\nu > 0$)

Our goal now is to modify Algorithm 4.2 so that we can handle no-slip boundary conditions. In particular, what we seek is a flux g of vorticity which we can introduce in the diffusion step of our algorithm and which will cancel out the slip velocity on the boundary of each object at each time step. Unlike the vortex method described in Section 1.2.5, our method solves an implicit system to obtain this desired flux.

In Chapter 2, we described how we used a Godunov splitting to split the advection term from the diffusion term in the vorticity transport equation. In Algorithm 4.2,

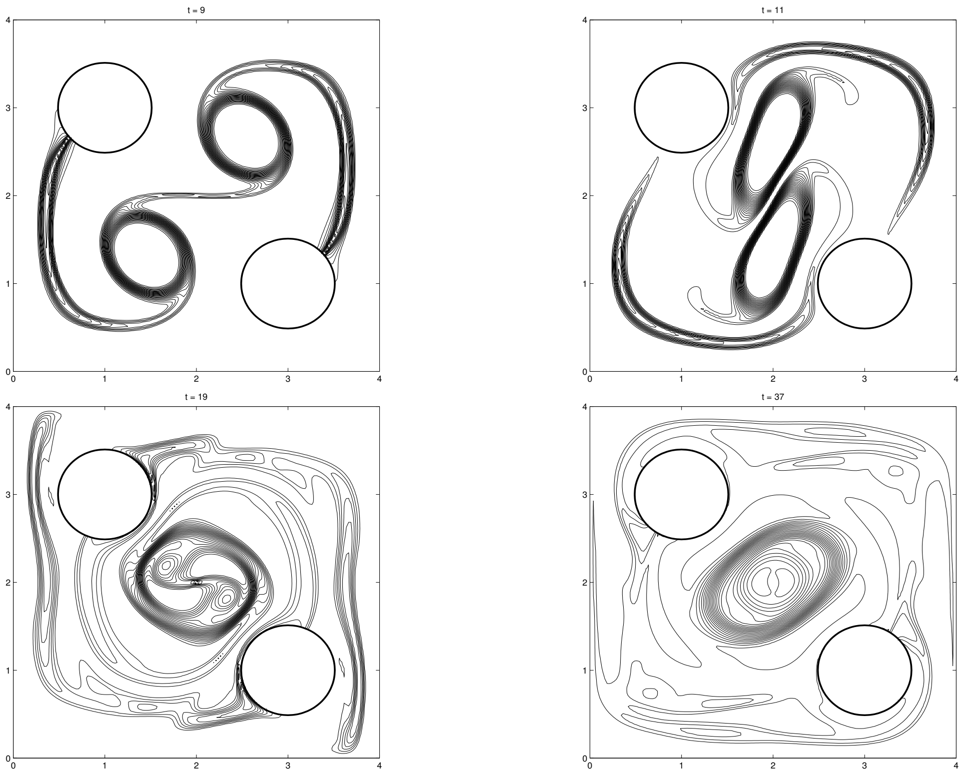


Figure 4.3: Inviscid calculation of two vortex patches coalescing into a single vortex in the presence of two obstacles. Initially, two circular patches of constant vorticity are positioned in the lower left and upper right corners of the domain. No-flow boundary conditions are imposed on all domain boundaries. The computational grid is 160×160.

we made use of the splitting to solve the transport equation. Now, we view this fractional step method as not only splitting the diffusion term from the advection term, but as splitting the transport equation into a convection-dominated, inviscid step and a diffusion-dominated, Stoke's flow step. In the Stoke's flow step, we solve the equations given in Section 3.5.3 for the flux of vorticity g needed to ensure that the no-slip condition $\psi_n = 0$ is satisfied.

In Figure 4.4 we give viscous counterpart to the inviscid algorithm presented in Figure 4.2. Note that instead of imposing the no-circulation condition, as we did in the inviscid case, we now impose no net-flux condition

$$\int_{\Omega_j} \frac{\partial \omega}{\partial n} ds = 0 \tag{4.20}$$

described in Section 3.5.3 on the flux of vorticity across solid boundaries.

In Algorithm 4.4, we determine the unknown vorticity flux g^{n+1} after we have

Solving the viscous streamfunction-vorticity equations

Given cell-averaged values ω^0, obtain $\widehat{\omega}^0$ by averaging ω^0 onto grid nodes. Solve $\nabla^2\psi^0 = -\widehat{\omega}^0$ subject to Dirichlet boundary conditions $\psi = \overline{\psi}$ on each irregular boundary. Dirichlet conditions to be determined as part of the solution process. Impose a no net-flux condition (4.20) on the vorticity. Impose appropriate conditions on ψ^{n+1} on the boundary of the computational domain.
Define $U^0_{i-1/2,j}$ and $V^0_{i,j-1/2}$ using ψ^0.
For $n = 0, 1, \ldots$

1. Advect the vorticity using the finite volume advection scheme and capacity form differencing in CLAWPACK to get ω^*. Impose no-flow conditions on all solid boundaries.
2. Solve the Stoke's flow problem in Section 3.5.3 to obtain the unknown flux of vorticity g^{n+1} needed to ensure $\psi^{n+1}_n = 0$. Use node based solution $\widehat{\omega}^{n+1}$ to solve $\nabla^2\psi^{n+1} = -\widehat{\omega}^{n+1}$ subject to Dirichlet boundary conditions $\psi = \overline{\psi}_j$.
3. Diffuse ω^* using the finite volume scheme presented in Chapter 2. Impose flux g^{n+1} at physical boundaries.
4. Compute $U^{n+1}_{i-1/2,j}$ and $V^{n+1}_{i,j-1/2}$ using ψ^{n+1}.

Figure 4.4: Algorithm for solving the viscous streamfunction vorticity equations

completed the advection step of our splitting scheme, but before we have carried out the diffusion step. When we solve the Stoke's flow equations, therefore, we are solving the system

$$\begin{aligned} \frac{\widehat{\omega}^{n+1} - \widehat{\omega}^*}{\Delta t} &= \nu\nabla^2\left(\frac{\widehat{\omega}^{n+1} + \widehat{\omega}^*}{2}\right) \\ \nabla^2\psi &= -\widehat{\omega}^{n+1} \\ \psi_n &= 0 \end{aligned} \tag{4.21}$$

for g^{n+1}. subject to boundary conditions

$$\begin{aligned} \frac{\partial\widehat{\omega}^{n+1}}{\partial n} &= g^{n+1} \\ \psi &= \overline{\psi} \end{aligned} \tag{4.22}$$

The value $\widehat{\omega}$ is the value of ω averaged onto grid nodes. This is the same coupled system we described for Stoke's flow in (3.132).

There is one additional detail we must include in order to ensure the stability of the Algorithm 4.4. As we recall, the Stoke's flow equations described in Chapter 3 required jumps at time level n. These were required to form the right hand side for the parabolic solve. In the present situation, we have introduced an advection step between diffusion steps, and so jumps computed from the previous solution ω^n of the parabolic equation are no longer consistent with jumps in the quantity ω^*. We must therefore recompute appropriate jumps w^*, v^*. This is done using one-sided approximations described in the inviscid case. Using jumps w^n, v^n and r^n leads to an algorithm which is stable only for very small Reynolds numbers ($Re \ll 1$).

4.3 Numerical results

4.3.1 Flow past a cylinder

One classic, non-trivial 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 of our algorithm applied to this problem with those results published in the literature.

In this problem, we have a circular cylinder embedded in a rectangular domain. As suggested by Behr et. al. in [4], we use a computational mesh that extends 8 cylinder units (1 unit = diameter of the cylinder) to the left and in the cross-flow directions from the center of the cylinder. In the downstream direction, the domain extends 18.5 cylinder units. The grid used is 530×320. See Figure 4.5 for a sketch of the computational domain.

To represent the circular cylinder, we use either $M = 8$ or $M = 16$ interface points, depending on the Reynolds number. For this value of M, we can explicitly form and solve the linear systems for the flux of vorticity directly. The size of the time step was restricted by the stability requirements of the advection scheme. See Figure 4.5 for a plot of how the cylinder and interface points are embedded in the mesh used for these calculations.

On the boundaries of the computational domain, we impose no-flux conditions ($\partial\omega/\partial n = 0$) on all four boundaries. For the streamfunction, we impose free-stream conditions given by

$$
\begin{aligned}
\psi =& U_\infty(y-8) \quad \text{at } x = 0, \\
\frac{\partial \psi}{\partial n} =& 0 \quad \text{at } x = 26.5, \\
\psi =& -2U_\infty, \quad \text{at } y = 0, \text{ and} \\
\psi =& 2U_\infty, \quad \text{at } y = 16.
\end{aligned}
\tag{4.23}
$$

These conditions are actually considered quite crude and if highly accurate values of vorticity and pressure on the wall are desired, then one should impose more sophisticated far field conditions [24], [46]. However, according to Behr et. al. [4], even using

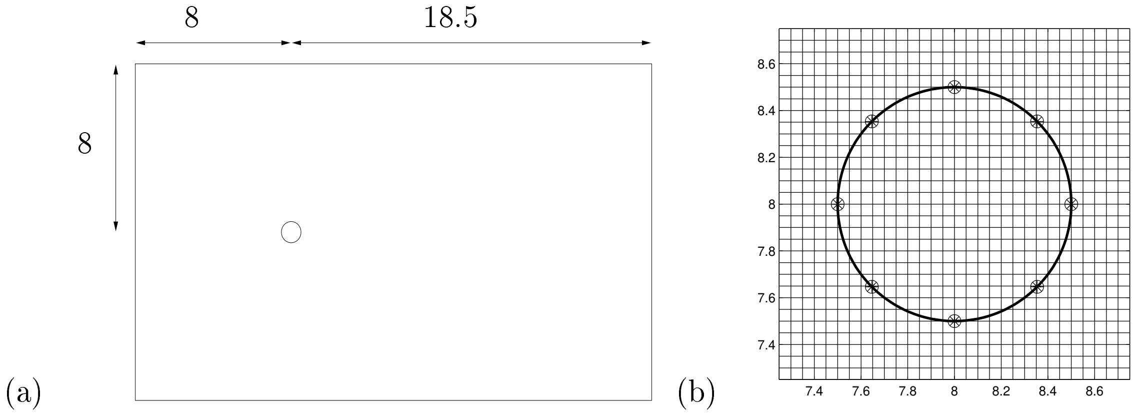


Figure 4.5: (a) Computational domain for flow past cylinder test case. (b) Cylinder embedded in 530×320 grid used for this problem. The 8 interface points used for $Re = 20$ this problem are marked with the "*". For $Re = 40$ and $Re = 100$, we used 16 interface points, also equally spaced on the cylinder.

these crude conditions on the smaller domain we are considering, we should be able to get values of the drag coefficient that are within 5% of the value we would get on a larger domain.

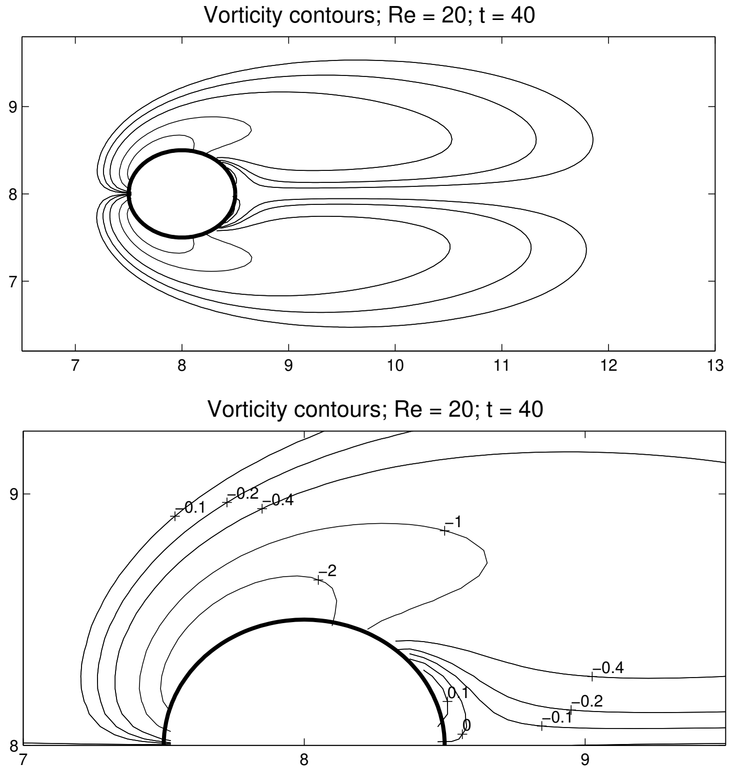
There is a wealth of papers on the subject of flow around a cylinder. We found a handful of these papers particular useful. The first, by Fornberg [24] is a numerical study of steady flow past a cylinder. In his work, he determines the cylinder wake for steady flow at Reynolds numbers from 2 to 300. These results are fairly detailed and so provide a good primary source for numerical results. The second paper by Braza et. al. [9] also did a numerical study and in the early section of their paper, make many useful comparisons between their work and others reported in the literature.

In addition to numerical work, there have also been some detailed experimental work done for the flow around a cylinder. One very early and often cited work is that of Tritton [52]. Tritton carried out the most extensive set of calculations for the drag coefficient and his results are still cited even in the most recent papers. One particularly comprehensive set of experiments was carried out by Coutanceau and Bouard ([16], [17]). In their series of papers, they report detailed information regarding several geometrical parameters such as wake length and separation angle. Since these are values which we can easily determine from our results, we refer often to these values to see that our results are agreeing with experimental work.

The difficulty with comparing our results to those cited above is that very rarely were numeric values for the various quantities of interest given. We usually had to resort to reading values from graphs presented in these papers and so as a result could only get very crude estimates for some of these values. In some cases, we were able to cite secondary sources for numeric values that apparently the authors of those sources were able to obtain from the primarily authors. While we tried to go to the primary sources whenever possible, we often could not get the exact numbers quoted. In some cases, we only compare our plots visually with those published plots, and so to aid the reader in making these comparisons, we reproduce a selection of these plots in Appendix A.

Our numerical results are divided into the three sections, one for each Reynolds number. While this is not the standard way to group the results, we wanted to look at three different regimes: the clearly steady regime (Re = 20), a regime in transition (Re = 40) and the clearly unsteady regime (Re = 100). Each of these regimes show different features of our algorithm.

Steady flow : Re = 20

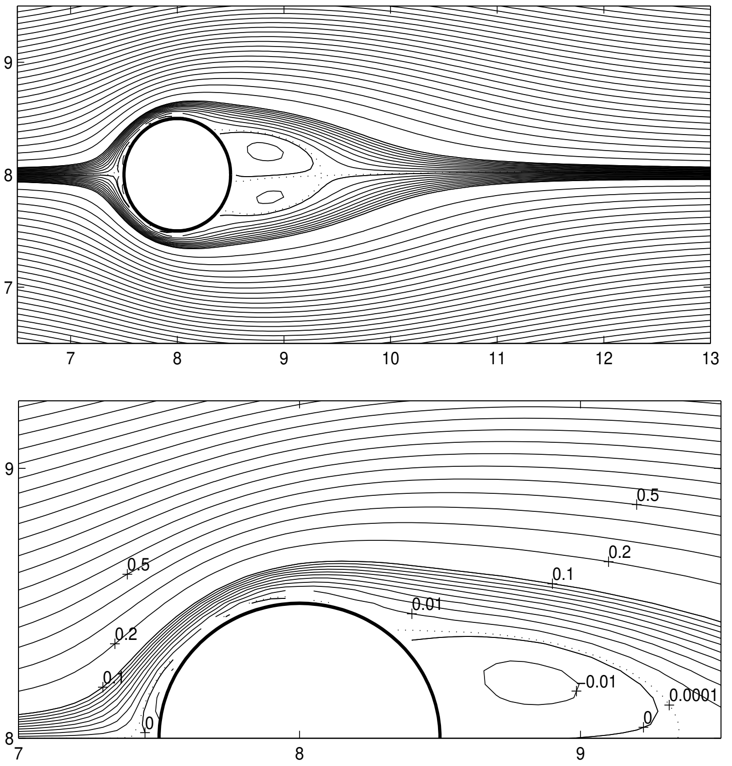


(a)

(b)

Figure 4.6: Contours of nondimensional vorticity $\omega r_a/U_\infty$ for Re = 20.

To see that we are getting the right general behavior, we plot in Figure 4.7 the streamlines and vorticity contours computed for Re = 20. In order to compare our results with those found in Fornberg's study of steady wakes behind a cylinder [24],



(a)

(b)

Figure 4.7: Contours of non-dimensional streamfunction ψ/r_aU_∞ for Re = 20. The dotted line delineates the boundary of the closed wake.

we plot the non-dimensional quantities $\psi/(r_aU_\infty$ and $\omega r_a/U_\infty$ and show the same contour levels as are cited in that paper. In these plots and all subsequent plots, we have masked out (using the Matlab `NaN` value) the vorticity values corresponding to $\kappa = 0$ and the streamfunction values corresponding to nodes strictly inside of the no-flow domain. The contour levels that cross the cylinder boundary are a result of the fact that Matlab assumes that all values are cell centered, which is not true for those values in partial cells.

In both sets of plots, we see very good agreement with Fornberg's plots, which we have reproduced in Appendix A. Only the zero level contour for both plots seems to deviate noticeably from that shown by Fornberg. In both plots, this contour is not symmetric with respect to the x axis. Also, in the zero level streamline is separated from the surface of the front (left) edge of the cylinder and doesn't delineate perfectly a boundary of the closed wake. We attribute this slight lack of symmetry to the fact that the six point stencils that we use to approximate the boundary conditions are not symmetric with respect to the horizontal centerline dividing the domain. Also, the one-sided stencils we use to approximate the jumps in vorticity after the advection step are also not symmetric with respect to this line. We expect, however, that on a finer grid, our results would exhibit more symmetry.

The contour that seems to follow the wake boundary most closely is shown in Figure 4.7 by the dotted line. The value of this contour is quite small (10^{-4}) and close to the truncation error of our numerical scheme. We use this contour to establish the length of the wake boundary, as measured from the downstream edge of the cylinder. Using Matlab, we estimate this length to be 0.85, a value that falls well within the range of experimental values (0.6 to 1) reported by Coutanceau and Bouard [16].

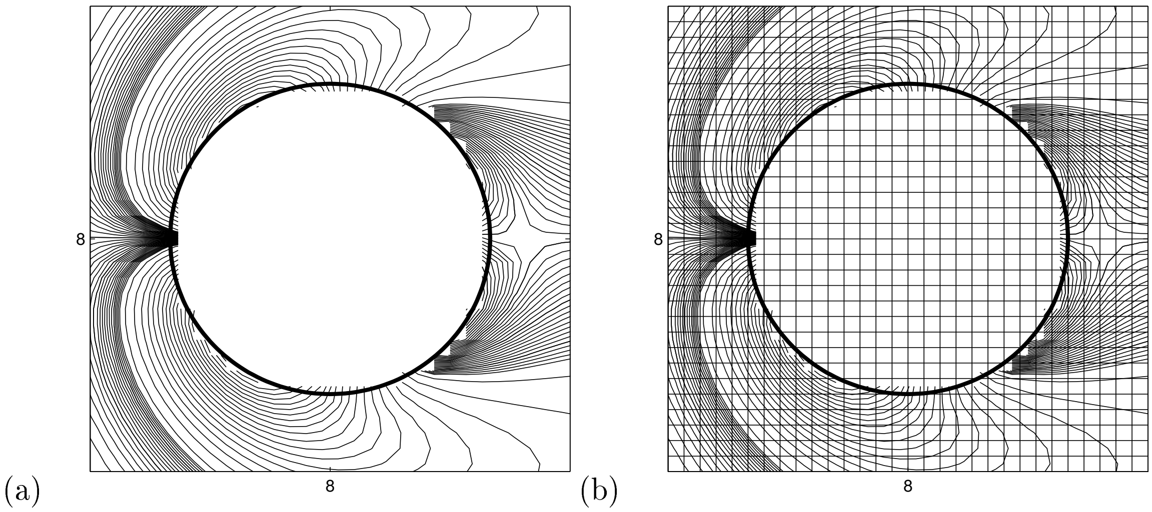


Figure 4.8: Close-up of vorticity contours for non-dimensional vorticity at $Re = 20$. The two sets of contour levels used for each plot are $-4.6 : 0.2 : 4.6$ and $-0.5 : 0.025 : 0.5$.

To see that the solution is very sensitive to the value in a few grid cells near the back (right) edge of the cylinder, we show in Figure 4.8 a close up of the vorticity contours, this time for closely packed contour levels. In both plots, we show the same contour levels, but in the second plot we overlay the plot with Cartesian mesh on which the solution was computed to show that the deviation from symmetry occurs only in about four cells near the right edge of cylinder.

We now look at quantities on the surface of the cylinder and see how those compare to the results reported in the literature. In Figure 4.9, we plot the vorticity along the surface of the cylinder. The maximum value of vorticity that we report is higher than those reported by either Fornberg or Braza et. al. [9], but by no more than those two reported values deviate from each other. We attribute our larger value to the fact that we are applying the freestream boundary conditions on a relatively small computational domain. Fornberg shows that this maximum value in particular is very sensitive to the boundary conditions chosen and the domain size, and generally increases as the domain size is decreased when using these free stream conditions. Extrapolating from his data, we are well within the error bounds that he reports.

From this plot of the wall vorticity, we can get a good estimate of the separation angle for the cylinder wake. The location of the separation point is determined by the point on either the upper or lower half of the cylinder where the vorticity changes sign. We have estimated this separation angle for our case to be about 45.2 degrees, where we measure the angle in a counter-clockwise direction from the back (right) edge of the cylinder. This agrees quite well with those results reported by Fornberg (at least as far as we could read on the graph). When compared with the summary experimental values reported by Coutanceau and Bouard [16], our value is somewhat high. The more recently reported values, cited in this paper are in the range 43 to 43.8 degrees.

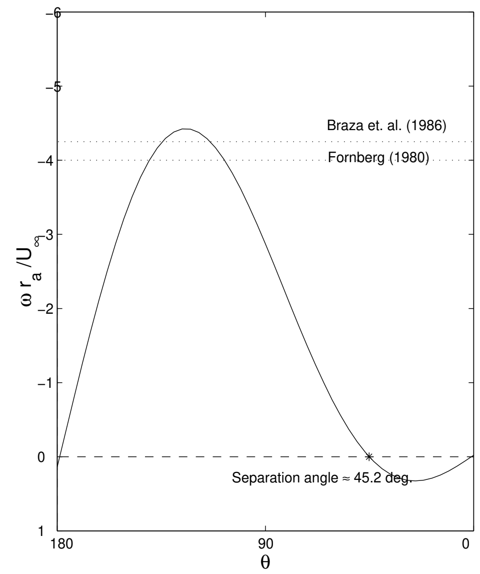


Figure 4.9: Vorticity on the surface of the cylinder for $Re = 20$. The angle θ is measured from the back (right) of the cylinder counter-clockwise. The dotted horizontal lines indicate maximum values reported by the authors cited.

Finally, we compute the pressure on the surface of the cylinder, using the formula

$$P(\theta) = P_0 - \rho\nu a \int_0^{\theta} \frac{\partial \omega}{\partial n}\, d\theta \tag{4.24}$$

where now, we have taken θ to be the angle from the stagnation point at the front edge of the cylinder, as measured in a clockwise direction. In all of our computations,

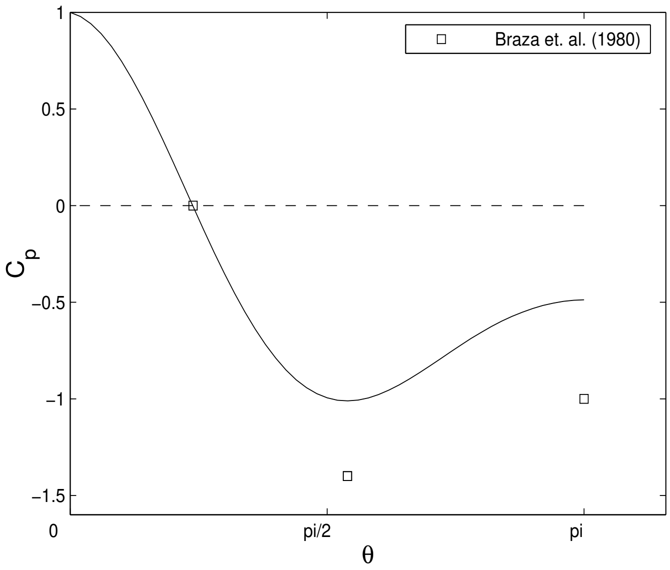


Figure 4.10: Pressure coefficient C_p calculated on the surface of the cylinder for $Re = 20$. Angle θ is measured from the stagnation point on the front of the cylinder in a clockwise direction. Values reported by Braza et. al. are shown.

ρ, the density of the liquid, is taken to be 1. In Figure 4.10, we plot the non-dimensional pressure coefficient, defined as $C_p = (P - P_0 + \frac{1}{2}\rho U_\infty^2)/(\frac{1}{2}U_\infty^2\rho)$, where P_0 is the stagnation pressure. We do not have exact measurements against which to evaluate our data, but along with our results, we plot our best estimates of those values reported by Braza et. al. [9] for Reynolds number 20. Qualitatively, our calculations of pressure are in good agreement with those in Braza et. al. However, our actual numeric values for the pressure are quite a bit lower in absolute value than those reported in that paper. We would like to be able to argue convincingly that again, the discrepancies are due to the fact that our domain is much smaller than it should be to get good agreement. However, we have found nothing in the literature that supports this hypothesis. We can only conclude that future efforts should be directed towards uncovering the source of this discrepancy.

In Figure 4.15, we plot the time history of this coefficient as computed using our algorithm. We see that the value is low. While we don't have a good explanation for this discrepancy from that value reported by Fornberg, we believe that because both the pressure coefficient C_p and now the drag coefficient depend on the integral of $\partial\omega/\partial n$, the fact that both of these values are low suggests that our computation of the flux of vorticity is low. We again will consider it a priority to determine the source of this discrepancy in future work.

Transition Regime : Re = 40

Experimentally, the transition to an unstable wake occurs at around $Re = 40$ [16]. To see how our code behaves around this Reynolds number, we computed the vorticity and streamlines for Reynolds numbers 40 and 50 to see if we can detect the beginnings of an instability. Our numerical results for $Re = 40$ exhibit slight asymmetries in the eddy structure in the wake, but the wake remains closed and the perturbations in the flow are damped. For $Re = 50$, these asymmetries in the flow are more pronounced. In Figure 4.11, we plot the streamlines for the two flows and in Figure 4.12,we plot the vorticity for the two cases. The contour levels shown in all plots are the same as those shown for the corresponding $Re = 20$ cases.

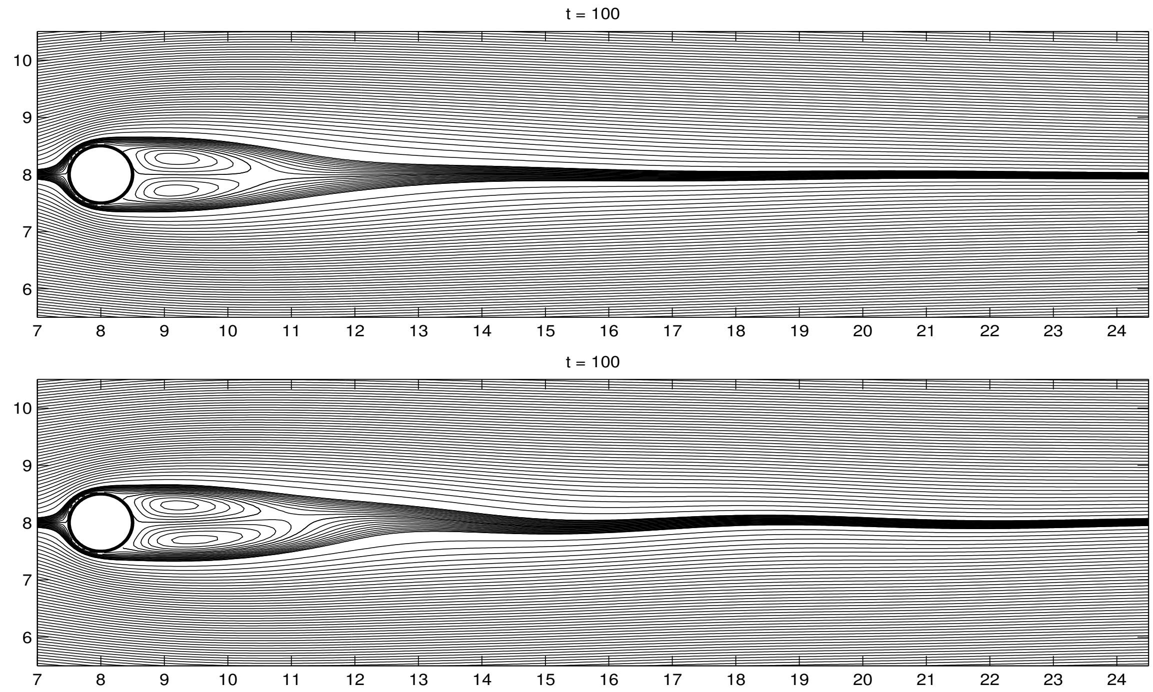


Figure 4.11: Streamlines for (top) $Re = 40$ and (bottom) $Re = 50$ at $t = 100$. Two sets of contours levels shown are in both plots. These are $-10 : 0.1 : 10$ and $-0.1 : 0.01 : 0.1$, the same levels used for the $Re = 20$ case.

To establish the wake boundary for $Re = 40$, the highest Reynolds for which Coutanceau and Bouard [16] report data, we locate the point along the centerline behind the cylinder at which the velocity changes sign. In Figure 4.13, we plot these velocities at three different time levels. First, we confirm that the velocities along this centerline have essentially reached a steady state by time level 100. We then compare these results with the those in [16] and see that we have excellent agreement. We

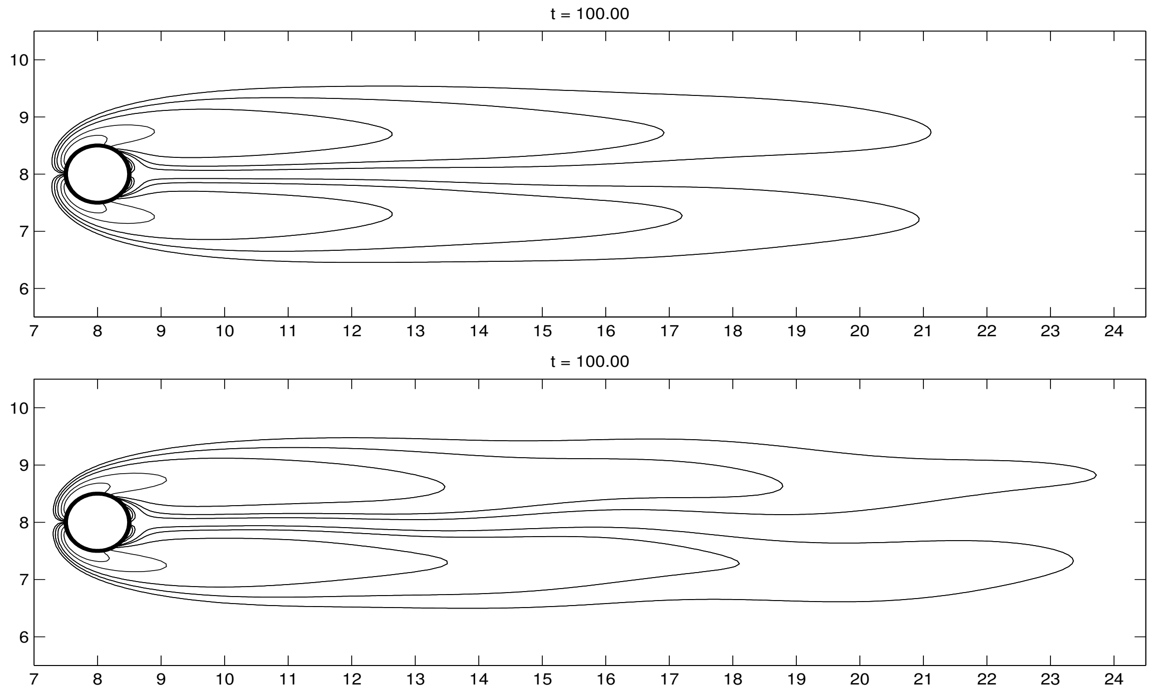


Figure 4.12: Vorticity contours for (top) $Re = 40$ and (bottom) $Re = 50$ at $t = 100$. The contour levels shown are the same levels used for the $Re = 20$ case.

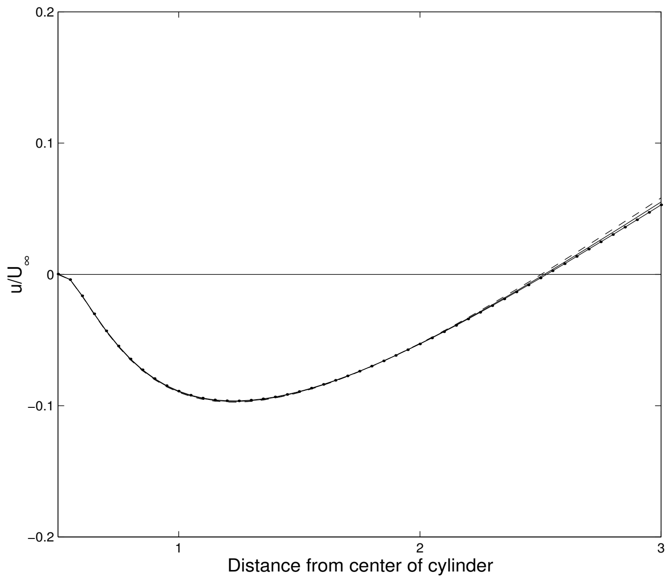


Figure 4.13: Horizontal velocities for $Re = 40$ along centerline as function of distance from center of cylinder. Markers on curve for $t = 100$ are values at individual cell edges. Markers on other times are not shown. The three time levels plotted are : "—", $t = 80$; "—", $t = 90$; ".—", $t = 100$.

have reproduced this plot in Appendix A. Even the location of the velocity minimum is well within their experimental results. Finally, we determine the length of the wake at time level $t = 100$ by locating the point at which the velocity at that time level changes sign. From our results, we determine that the wake length is about 2.05, which well within the results reported in [16].

One signature of the birth of an instability in flow past a cylinder is the onset of a difference in lengths separating the two vortex cores from the vertical plane through the stagnation point [16]. In Figure 4.11, we observe that the vortex cores are about the same distance from the stagnation point, but that by $Re = 50$, the top vortex is closer than the bottom vortex to this point. Furthermore, we see that the streamlines have become warped with respect to the y axis. We would need to run the code for a much longer time to determine if we actually get vortex shedding at $Re = 50$, but more likely this is a regime in which the wake merely oscillates, but doesn't actually separate from the cylinder.

From this numerical study of this transition regime, we conclude that somewhere between Reynolds number 40 and Reynolds number 50, perturbations in the flow behind the cylinder are no longer damped by the flow and as a result affect the downstream wake. This behavior is in good agreement with other results reported.

Results on the drag coefficients for these two cases are reported in 4.3.1.

Unsteady Vortex Shedding : Re = 100

In this regime, we clearly have vortex shedding past the cylinder. These unsteady flows are more complicated to analyze, since most quantities are now time dependent, and all that one can really hope to assess is the basic behavior of the time dependent quantity; establishing time accuracy of the results is quite difficult, given that the onset of the instability depends on the manner in which the solution was perturbed from symmetry. In our case, there is no need to perturb the solution at all; the slight asymmetry in our numerical scheme, due to the asymmetric choice of six point stencils, was enough to initiate the unsteady behavior.

Despite the fact that the higher Reynolds number flows are known to be more difficult to compute, we were able to get good agreement with most quantities that we computed.

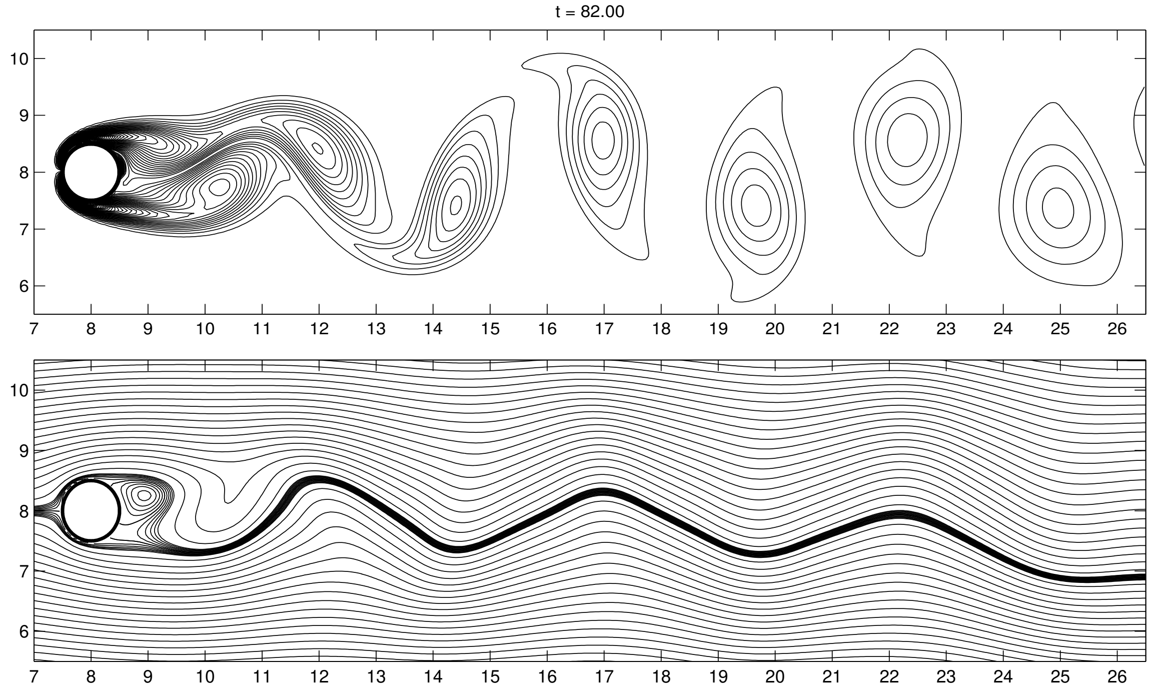


Figure 4.14: Contours of vorticity and streamlines for Re =100. Time corresponds to crest of lift coefficient.

To see that we are simulating basic behavior of the vortex wake behind the cylinder, we show in Figure 4.14 plots of the shedding vortices and streamlines at a time corresponding to a crest in the lift coefficient. The general behavior agrees with results found, for example, in [4] for the size grid we are working on. We also note that we see very little diffusion in the vortices as they travel downstream. This may be due in part to the fact that the high resolution algorithms in CLAWPACK minimizes the diffusion normally seen in first order upwind advection algorithms. Also, we note that our results exhibit virtually no dispersive oscillations associated with second order finite difference schemes. Again, the flux limiters of CLAWPACK minimize these oscillations. The outflow boundary conditions that we use in CLAWPACK also seem to be doing a very good job of allowing the detached vortices to cleanly exit the domain; no noise propagates back into the domain at the outflow boundary.

To compute the Strouhal number, the non-dimensional frequency given by $\mathrm{Str} = f 2r_a/U_\infty$, we tracked the time history of vorticity at the point $(x_{288}, y_{148}) = (14.375, 7.375)$ corresponding to the center of the lower left most detached vortex in Figure 4.14. Using this time history, we could estimate an average frequency f based on several periods of the shedding. The Strouhal number corresponding to this shedding was about $\mathrm{Str} = 0.17$, which is in good agreement with other results in for example [52].

Other values such as the pressure coefficient, vorticity on the wall and drag and lift coefficients could in principle be computed as well, but because these values are all time dependent, one should compute these quantities for several hundreds of times steps to be sure that the desired periodic pattern has been established. Moreover, one should compute these values more frequently than we do to get results that compare favorably with those published. We did compute the lift coefficient as a function of time and do see the beginnings of a periodic pattern, but this really only appeared to establish itself near $t = 100$, our final time step for this time step. The periodic behavior in the drag coefficient is much less pronounced,and so our results didn't show any behavior that we could clearly identify as periodic. Again, we assume that we hadn't run the simulation long enough to establish the pattern.

The drag coefficient that we did compute was within the range of results published by Tritton [52], Braza et. al. [9], and Fornberg [24]. Our value drifted a bit from a low value of 1.03, computed at $t = 28$ to a higher value of 1.17 at $t = 100$. Tritton reports a value of about 1.25, whereas Fornberg reports a much lower value of about 1.058. Our results, along with drag coefficient results for other Reynolds numbers that we have computed are reported in Section 4.3.1.

Drag coefficients

In this section, we briefly discuss the drag coefficient calculations for each of the Reynolds numbers discussed above. We compare our results to those found in the experimental work of Tritton [52] and the computational results of Fornberg [24].

The coefficient of drag C_d is computed as the sum of a viscous and pressure drag terms and is given by

$$C_D = \frac{\nu}{U_\infty^2} \int_0^{2\pi} \left(r_a \frac{\partial \omega}{\partial n} + \omega \right) \sin(\theta)\, d\theta \tag{4.25}$$

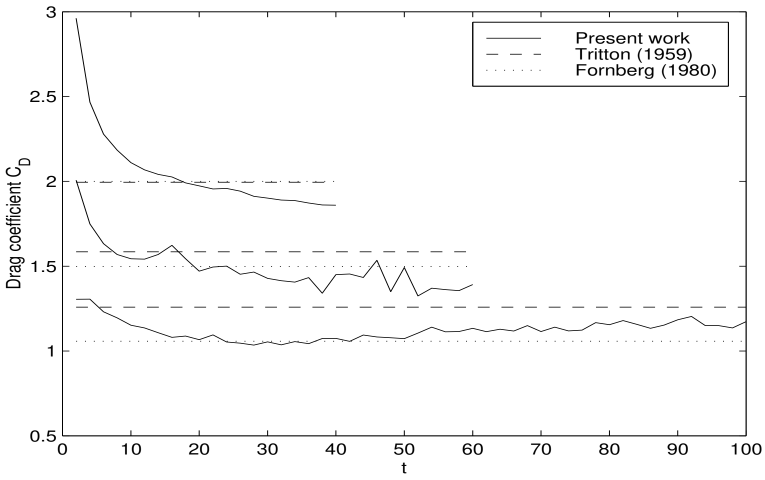


Figure 4.15: Time history of the drag coefficients for Re = 20, 40 and 100. The largest drag coefficient is associated with $Re = 20$ and the smallest with $Re = 100$.

where θ is the angle measured from the front stagnation point in a clockwise direction.

Generally, we found that we got reasonable agreement with the results reported by these researchers. We plot the time histories of these coefficients in Figure 4.15, along with the values reported by Tritton [52] and Fornberg [24]. The results quoted for Tritton were taken from graphs, whereas the Fornberg published the numeric values. Our general observations are that our results match the published values more closely for the higher Reynolds number than for the lower one. This is most likely due to the fact that for these lower Reynolds numbers, the pressure pressure drag is a larger percentage of the total drag than for the higher values. Since the pressure seems to be the value that we can approximate with the least degree of accuracy, we are not surprised at our results for the drag coefficient.

Second, we noticed that we didn't generally get smooth behavior unless we arranged the computations of the flux of vorticity in certain ways. The computations we are referring to are those involved in the solving the coupled parabolic-elliptic system in Section 3.5.3. For example, if we enforce the condition $\psi_n = 0$, by requiring that the normal derivative of the streamfunction, as approximated using the ψ that is returned from the elliptic solver be zero, then we get much smoother results than if we simply require that the jumps $\mathbf{v}_k$ returned from the elliptic solver be set to zero. The difference seems to be most dramatic for $Re = 20$. On the other hand, the smoothed solutions are not necessarily more accurate than those shown in Figure 4.15. Understanding the effect that the arrangement of the computation of the flux has on the drag coefficient is clearly worth future investigation.

4.3.2 Flow past several cylinders

While the single cylinder problem has gained an enormous amount of attention, flow past several cylinders has received considerably less attention. The reason for this is in part because it is not longer possible to use the cylindrical transformation used by most researchers for the single cylinder case. We present here results from computing flow around three cylinders, embedded in a 20×10 grid. The cylinders are each of radius 0.5 and The computational grid was 400×200.

We again see a periodic vortex shedding, but now because the cylinders are not placed symmetrically with respect to the y axis, the shedding is not vertically symmetric. In Figure 4.16, we show the results at $Re = 100$ of our calculations and show four different times during the evolution of a single period. We took the starting point for a single period to be the time at which the largest vortex sheds from the upper cylinder. In the final frame, this shedding is just about to occur. We measured the Strouhal number for this shedding and found it to be about 0.14.

4.3.3 Flows in biperiodic domains

The great majority of calculations published in the literature for flow around a cylinder are done using a cylindrical coordinate transformation. While this is the obvious transformation for this problem, it doesn't extend to multiple cylinders, nor does it lend itself very well to computing flows in periodic domains. Boundary conditions which are naturally imposed at a distance r_∞ of a circular domain are difficult to impose on a rectangular domain. One advantage of the rectangular domain is that we can in fact impose periodic boundary conditions quite naturally. Furthermore, by doing so, we can avoid many of the problems associated with far-field boundary conditions.

For this test problem, we revisit the biperiodic flow problem used as our example in the Stoke's flow case. Only this time, we include the affects of convection. The setup for this problem is identical to that for the Stoke's flow problem, so we refer the reader back to 3.5.3 for the domain configuration. In the present problem, we run the code at $Re = 50$, where the Reynolds number is based on the average upstream velocity. The vorticity and streamfunction contours are shown for several different time levels in Figures 4.17 and 4.18

In the literature, we found at least one set of results with which we could compare our computations. This set of results, by Tezduyar and Liou [50] and were reported for Reynolds number 100, we can see that we get similar flow structures to what they report, although we see the structures much earlier than they do. This is probably due to the fact that they start with a Stokes flow solution, whereas we start with a fluid at rest.

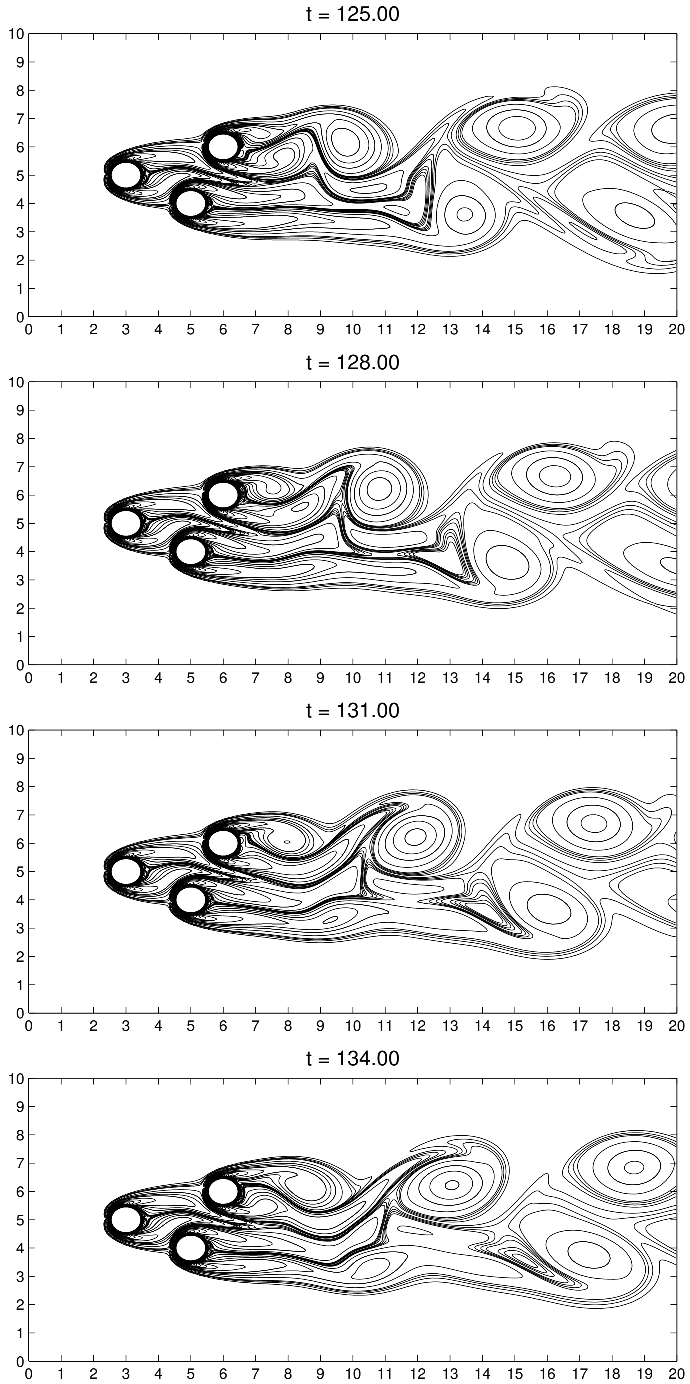


Figure 4.16: Flow past three cylinders. The last vortex from the upper cylinder was shed at time 121. In the bottom frame, another vortex is about to shed off of the upper cylinder.

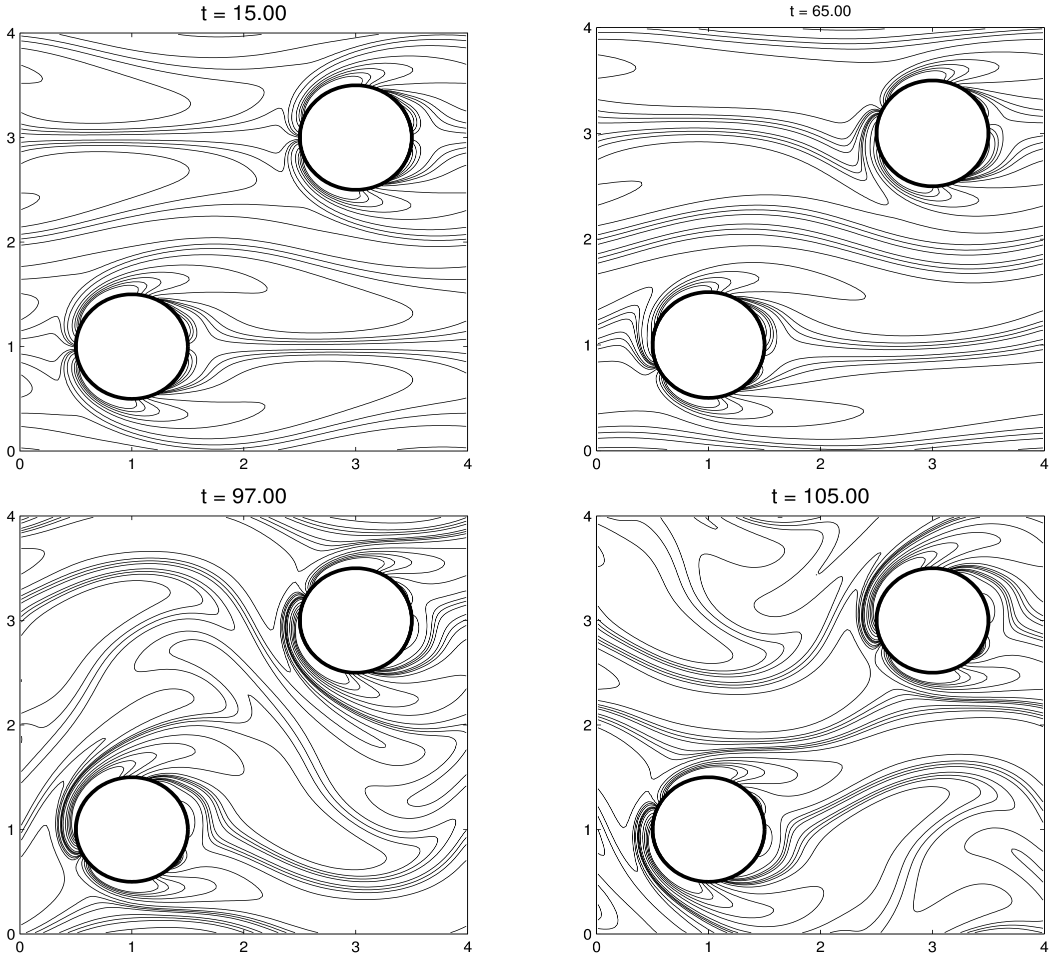


Figure 4.17: Contours of vorticity for flow at $Re = 100$ in a biperiodic domain.

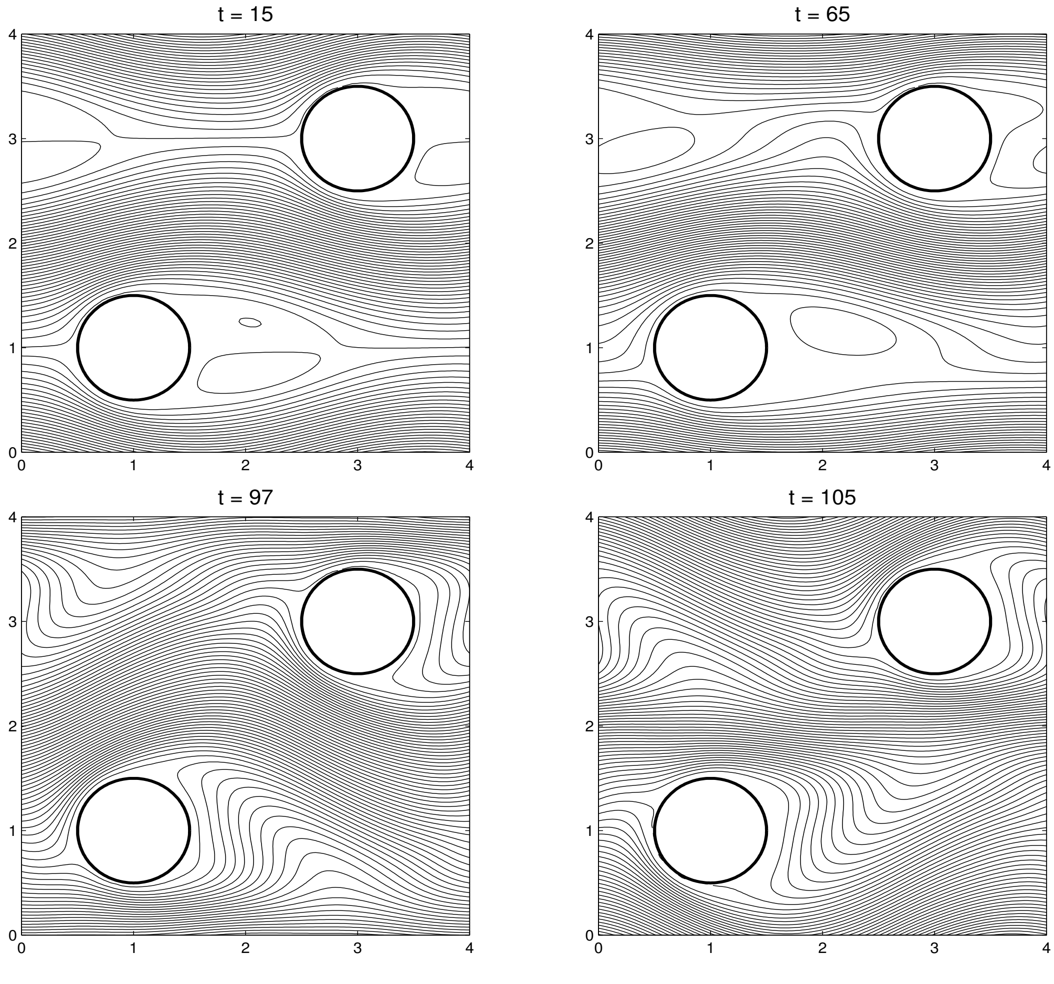


Figure 4.18: Contours of the streamfunction for flow at $Re = 100$ in a biperiodic domain.

4.3.4 Flow around irregular objects in a biperiodic domain

In our previous examples, we have only shown flow around circular cylinders. But as we saw in Chapters 2 and 3, neither our advection-diffusion algorithm, nor the Immersed Interface Method is restricted to simple shapes. Moreover, the code is written to handle any general smooth shapes which can be represented by a Fourier spline. Our choice of cylinders in the previous examples was motivated mainly by the fact that these are the shapes that show up most frequently in the literature.

In this set of examples, we show that our code works quite well on arbitrarily shaped objects. The domain we consider is the same square biperiodic domain as in the previous set of examples. This time, we embed three irregularly shapes objects in the domain and consider flow at Reynolds number approximately equal to 20, where we chose a length scale based on the diameter of the largest circle circumscribing each of the objects. The average flow at the left edge was equal to 0.1.

In Figure 4.19, we show the results of this investigation as the run approaches a steady state.

To see that the solution is converging, we ran the code at three different grid resolutions: $N = 80$, $N = 160$ and $N = 320$. Plots of the results of the grid refinement investigation are shown at time level 30 in Figure 4.20.

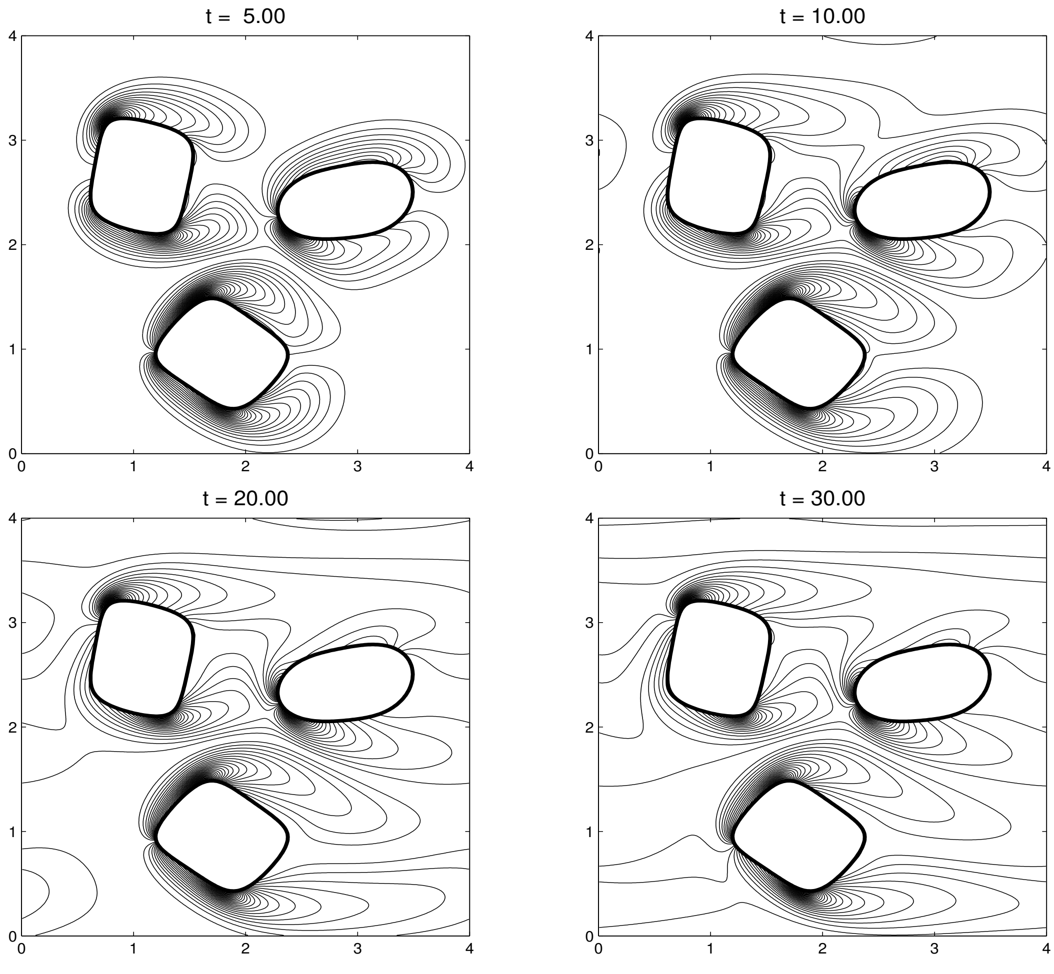
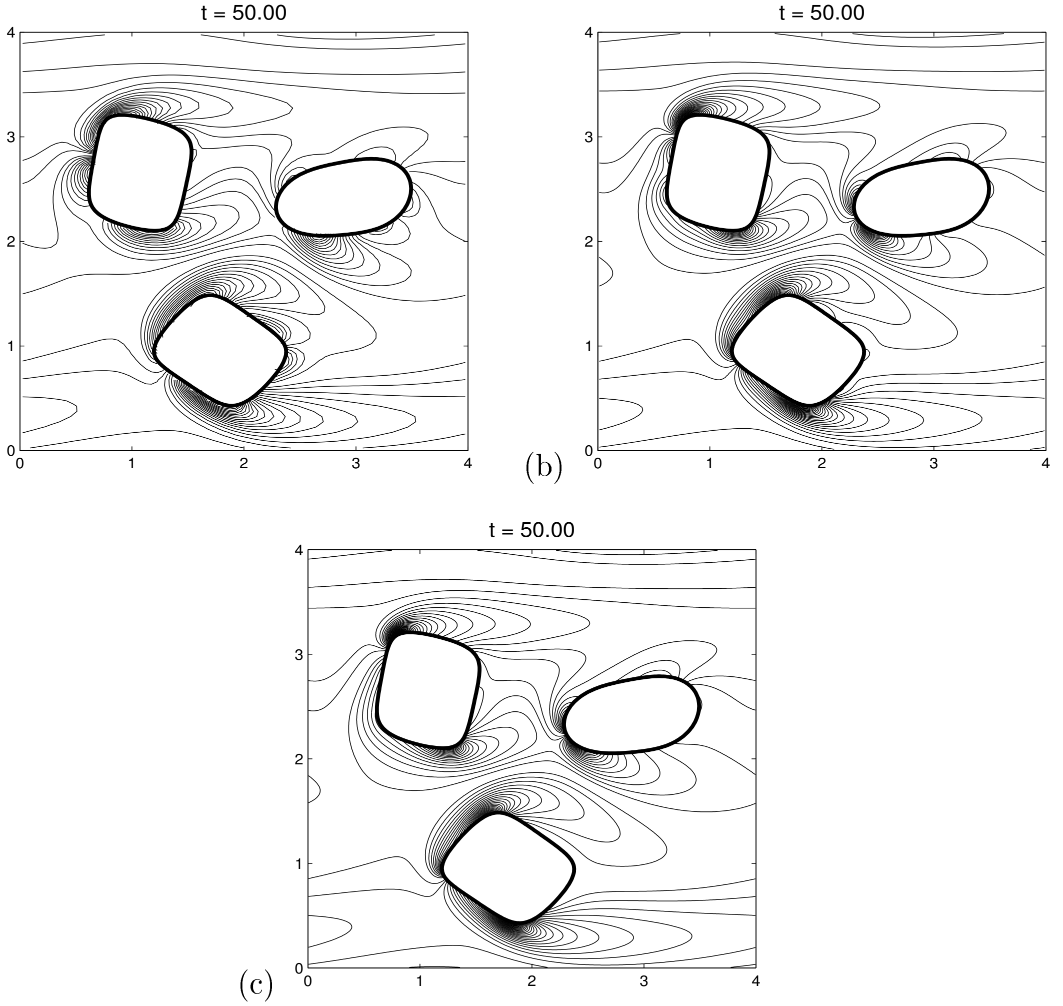


Figure 4.19: Vorticity contours for flow in a biperiodic domain. Computational domain is 320×320.



(a) (b) (c)

Figure 4.20: Vorticity contours for flow in a biperiodic domain at time level 50, as computed on three different grids. Plot (a) was computed on an 80×80 grid, plot (b) on a 160×160 grid and plot (c) on a 320×320 grid.

Chapter 5

CONCLUSIONS AND FUTURE WORK

In this thesis, I have described in detail a Cartesian grid method for solving the streamfunction vorticity equations in irregular geometries. Using the Immersed Interface Method (IIM), I solved the elliptic equation for the streamlines of the flow, and using a finite volume scheme, I solve the advection-diffusion equation modeling the transport of vorticity. Flux at the boundary of solid objects is generated by solving a coupled elliptic-parabolic system of equations in which I enforced the no-slip boundary condition on the surface of solid objects. The code can be used to solve these equations in complicated domains with multiple embedded inclusions.

The highlighs of this thesis include the following:

- A conservative finite volume algorithm is developed for solving the advection-diffusion equation in irregular regions. This is later used to solve for the transport of vorticity in the streamfunction vorticity equations. The algorithm has an order of accuracy between 1.2 and 2, depending on the geometry and the Peclet number. The stability problem associated with small cells is eliminated by modifying the capacity of each cell. This allows the code to run stably at Courant numbers near 1 relative to the uniform grid spacing.

- The Immersed Interface Method is rederived and extended to solve parabolic boundary value problems and coupled elliptic-parabolic problems in complex, multiply-connected regions. It was shown that I can use the IIM to solve the Stokes flow problem in a complicated domain. Also, the new derivation makes the computer implementation of the IIM quite straightforward.

- The Immersed Interface Method is used to determine the flux of vorticity at solid boundaries. This vorticity is diffused into the domain during the diffusion step of the algorithm. Determining how much vorticity to generate requires very little computational overhead, as the linear system I solve for the flux of vorticity needs to be formed and factored very infrequently. Moreover, the size of this system depends only the number of points used to represent the interface, and not on the number of grid points.

- The code was designed to take advantage of the high resolution algorithms in CLAWPACK. Using this high resolution algorithm, the results show very little evidence of numerical oscillations or diffusion.

The algorithm reproduces several published computational and experimental results for low Reynolds number flow around a circular cylinder. In Chapter 4 I investigated the behavior of the code for Reynolds numbers 20, 40, 50 and 100 and found that at $Re = 20$ and 40, the flow remained essentially steady and symmetric and any small perturbations in the wake were damped. At Reynolds number 50, the streamline wake begins to oscillate, and by Reynolds number 100, vortices are shed behind the cylinder forming the well known von Karman vortex street. The values of vorticity and streamfunction for the lower Reynolds numbers were shown to be in good agreement with published results. At $Re = 100$, I show that I get the proper shedding frequency. The drag coefficients and pressure calculations were not as accurate as the other values I report, but still within an acceptable tolerance for many applications. I also show good agreement with at least one set of published results on flow in a biperiodic domain.

5.1 Future Work

There are still numerical verifications that I would like to consider in the future. First it would be interesting to test the problem of flow around the cylinder on a finer grid to verify that the results remain essentially unchanged from those computed on the coarser grid. Second, it would be very interesting to run the code for longer times and for more Reynolds numbers between 40 and 100 to establish how well the code can determine critical Reynolds number for the onset of vortex shedding. Another interesting flow regime is flow past a rotating cylinder. There is small set of published results on this problem, and it would serve as a good test case for our algorithm. Finally, I would like to find more results on flow past non-circular blunt bodies so that I can have something to compare our results in these less well studied problems.

With our code, one can solve may interesting and challenging problems involving viscous incompressible flow in multiply-connected domains. I hope that such a code, which I intend to make available, will be useful to researchers in a variety of fields who need to solve low Reynolds number flow problems in complicated geometries. I include in this group biologists working in bio-fluid dynamics and material scientists interested in the role that convection could play in the evolution of a phase change interface. Also, I expect that mechanical engineers would be interested in our work, particularly those designing heat exchangers, which in the simplest cases can be modeled as an array of cylinders.

Eventually, I would like to extend this code to problems in multiphase flow with moving boundaries. My original motivation for developing the present code was to model Raleigh-Benard convection in the presence of a melting or solidifying interface. I plan to return to this problem in the future.

BIBLIOGRAPHY

[1] L. Adams. A multigrid algorithm for immersed interface problems. In *Copper Mountain Multigrid Conference*. NASA, available anonymous ftp from mgnet, 1995.

[2] C. Anderson. Vorticity Boundary Conditions and Boundary Vorticity Generation for Two-dimensional Viscous Incompressible Flows. *J. Comput. Phys.*, 80:72–97, 1989.

[3] M. Baker and D. Calhoun. Vapor Growth of Atmospheric Crystals. *Journal of Atmospheric Science*, to appear.

[4] M. Behr, J. Liou, R. Shih, and T. E. Tezduyar. Vorticity-streamfunction formulation of unsteady incompressible flow past a cylinder: Sensitivity of the computed flow field to the location of the outflow boundary. *Int. J. Numer. methods. fluids*, 12:323–342, 1991.

[5] J. Bernsdorf, F. Durst, and M. Schafer. Comparison of cellular automata and finite volume techniques for simulation of incompressible flows in complex geometries. *Int. J. Numer. methods. fluids*, 29:251–264, 1999.

[6] A. Bertozzi, 1999. private communication.

[7] R. P. Beyer. A computational model of the cochlea using the immersed boundary method. *J. Comput. Phys.*, 98:145–162, 1992.

[8] R. P. Beyer and R. J. LeVeque. Analysis of a one-dimensional model for the immersed boundary method. *SIAM J. Num. Anal.*, 29:332–364, 1992.

[9] 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.*, 165:79–130, 1986.

[10] L.N. Brush and R.F. Sekerka. A Numerical Study of Two-Dimensional Crystal Growth Forms in the Presence of Anisotropic Growth Kinetics. *J. of Crystal Growth*, 96:419–441, 1989.

[11] D. Calhoun and R. J. LeVeque. A Cartesian Grid Finite-Volume Method for the Advection-Diffusion Equation in Irregular Geometries. *J. Comput. Phys.*, 1999. submitted.

[12] S. Chen and G. Doolen. Lattice Boltzmann Method for Fluid Flow. *Annu. Rev. Fluid Mech.*, 30:329–64, 1998.

[13] I. Chern and P. Colella. A conservative front tracking method for hyperbolic conservation laws. Report UCRL-97200, LLNL, 1987.

[14] A. J. Chorin. Numerical study of slightly viscous flow. *J. Fluid Mech.*, 75(4):785–96, 1973.

[15] A. J. Chorin and J. E. Marsden. *A Mathematical Introduction to Fluid Mechanics.* Springer-Verlag, 1979.

[16] M. Coutanceau and R. Bouard. Experimental determination of the main features of the vicous flow in the wake of a circular cylinder in uniform translation. part 1. steady flow. *J. Fluid Mech.*, 79:231–256, 1977. Part 1.

[17] M. Coutanceau and R. Bouard. Experimental determination of the main features of the vicous flow in the wake of a circular cylinder in uniform translation. part 2. unsteady flow. *J. Fluid Mech.*, 79:257–272, 1977. Part 2.

[18] M. S. Day, P. Colella, M. J. Lijewski, C. A. Rendleman, and D. L. Marcus. Embedded boundary alogirthms for solving the Poisson equation on complex domains. Preprint LBNL-41811, Lawrence Berkeley Lab, 1998.

[19] H. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992.

[20] R. Dillon, L. Fauci, A. Fogelson, and D. Gaver. Modeling Biofilm Processes Using the Immersed Boundary Method. *J. Comput. Phys.*, 129:57–73, 1996.

[21] R. Dillon, L. Fauci, and D. Gaver. A Microscale Model of Bacterial Swimming, Chemotaxis and Substrate Transport. *J. Theor. Biol.*, 177:325–340, 1995.

[22] A. Fogelson and J. Keener. A software system for immersed boundary and software simulations. Available on the web at `http://www.math.utah.edu/IBIS/`, 1998.

[23] A. L. Fogelson. A mathematical model and numerical method for studying platelet adhesion and aggregation during blood clotting. *J. Comput. Phys.*, 56:111–134, 1984.

[24] B. Fornberg. A numerical study of steady viscous flow past a circular cylinder. *J. Fluid Mech.*, 98:819–855, 1980.

[25] R. Glowinski, T.-S. Pan, and J. Periaux. A fictitious domain method for Dirichlet problem and applications. *Comp. Meth. Appl. Mech. Eng.*, 111:283–303, 1994.

[26] R. Glowinski, T.-S. Pan, and J. Periaux. A fictitious domain method for external incompressible viscous flow modeled by Navier-Stokes equations. *Comp. Meth. Appl. Mech. Eng.*, 112:133–148, 1994.

[27] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, 1997.

[28] L. Greengard, M. C. Kropinski, and A. Mayo. Integral equation methods for stokes flow and isotropic elasticity in the plane. *J. Comput. Phys.*, 125:403–414, 1996.

[29] P. M. Gresho. Incompressible Fluid Dynamics: Some Fundamental Formulation Issues. *Annu. Rev. Fluid Mech.*, 23:413–53, 1991.

[30] H. Huang and B. R. Seymour. The no-slip boundary condition in finite difference approximations. *Int. J. for Num. Meth. in Fluids*, 22:713–729, 1996.

[31] M. Israeli. On the Evaluation of Iteration Parameters for the Boundary Vorticity. *Studies in Applied Mathematics*, LI(1):67–71, March 1972.

[32] H. Johansen and P. Colella. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *J. Comput. Phys.*, 147:60–85, 1998.

[33] R. J.. LeVeque. CLAWPACK software. available from `netlib.att.com` in `netlib/pdes/claw` or on the Web at the URL `http://www.amath.washington.edu/~rjl/clawpack.html`.

[34] R. J. LeVeque. High-resolution conservative algorithms for advection in incompressible flow. *SIAM J. Numer. Anal.*, 33:627–665, 1996.

[35] R. J. LeVeque. Wave propagation algorithms for multi-dimensional hyperbolic systems. *J. Comput. Phys.*, 131:327–353, 1997.

[36] R. J. LeVeque and Z. Li. Immersed interface methods for Stokes flow with elastic boundaries or surface tension. to appear in SIAM J. Sci. Comput. (`ftp://amath.washington.edu/pub/rjl/papers/rjl-li:stokes.ps.Z`).

[37] R. J. LeVeque and Z. Li. The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *SIAM J. Numer. Anal.*, 31:1019–1044, 1994.

[38] Z. Li. *The Immersed Interface Method — A Numerical Approach for Partial Differential Equations with Interfaces.* PhD thesis, University of Washington, 1994.
(`ftp://math.ucla.edu/pub/zhilin/Papers/thesis.ps.Z`).

[39] Z. Li and A. Mayo. ADI methods for heat equations with discontinuties along an arbitrary interface. *Proc. Symp. Appl. Math.*, 48:311–316, 1994.

[40] A. Mayo. The fast solution of Poisson's and the biharmonic equations on irregular regions. *SIAM J. Num. Anal.*, 21:285–299, 1984.

[41] A. Mayo and A. Greenbaum. Fast parallel iterative solution of Poisson's and the biharmonic equations on irregular regions. *SIAM J. Sci. Stat. Comput.*, 13:101–118, 1992.

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

[43] C. S. Peskin and B. F. Printz. Improved volume conservation in the computation of flows with immersed elastic boundaries. *J. Comput. Phys.*, 105:33–46, 1993.

[44] L. Quartapelle. *Numerical Solution of the Incompressible Navier-Stokes Equations*, volume 113 of *International series of numerical mathematics.* Birkhauser, 1993.

[45] L. Quartapelle and F. Valz-griz. Projection conditions on the vorticity in viscous incompressible flows. *Int. J. Numer. methods. fluids*, 1:129–144, 1981.

[46] J. Y. Sa and K. S. Chang. On Far Field Stream Function Condition for Two-Dimensional Incompressible Flow. *J. Comput. Phys.*, 91:398–412, 1990.

[47] Y. Saad. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[48] S. S. Samant, J. E. Bussoletti, F. T. Johnson, R. H. Burkhart, B. L. Everson, and R. G. Melvin. TRANAIR: a computer code for transonic analyses of arbitrary configurations. AIAA Paper 87-0034, Reno, Nevada, 1987.

[49] P. Swarztrauber. FFTPACK : A package of fortran subprograms for the fast fourier transform of periodic and other symmetric sequences, 1985. available on the web at `http://www.netlib.org`.

[50] T. E. Tezduyar and J. Liou. Computation of spatially periodic flows based on the vorticity-stream function formulation. *Computer Methods in Applied Mechanics and Engineering*, 1990.

[51] A. Thom. The flow past circular cylinders at low speeds. *Proc. Roy. Soc. A*, 141:651, 1933.

[52] D. J. Tritton. Experiments on the flow past a circular cylinder at low Reynolds numbers. *J. Fluid Mech.*, 6:547–567, 1959.

[53] P. W. Vorhees, G. B. McFadden, and R. F. Boisvert. Numerical Simulation of Morphological Development during Ostwald Ripening. *Acta Metall.*, 36(1):207–222, 1988.

[54] E. Weinan and J. Liu. Vorticity Boundary Condition and Related Issues for Finite Difference Schemes. *J. Comput. Phys.*, 124:368–382, 1996.

[55] A. A. Wheeler, B. T. Murray, and R. J. Schaefer. Computation of Dendrites using a Phase Field Model. Technical report, National Institute of Standards and Technology, 1992.

[56] A. Wiegmann. *The Explicit Jump Immersed Interface Method.* PhD thesis, University of Washington, 1997.

[57] A. Wiegmann and K. P. Bube. The Explicit Jump Immersed Interface Method. to appear in SIAM J. Numer. Anal.

[58] Z. Yang. *A Cartesian Grid Method for Elliptic Boundary Value Problems In Irregular Regions.* PhD thesis, University of Washington, 1996.

[59] C. Zhang. *Immersed Interface Methods for Wave Equations.* PhD thesis, University of Washington, in preparation, 1996.

Appendix A

FIGURES REPRODUCED FROM OTHER SOURCES

Below are the figures cited in previous chapters.

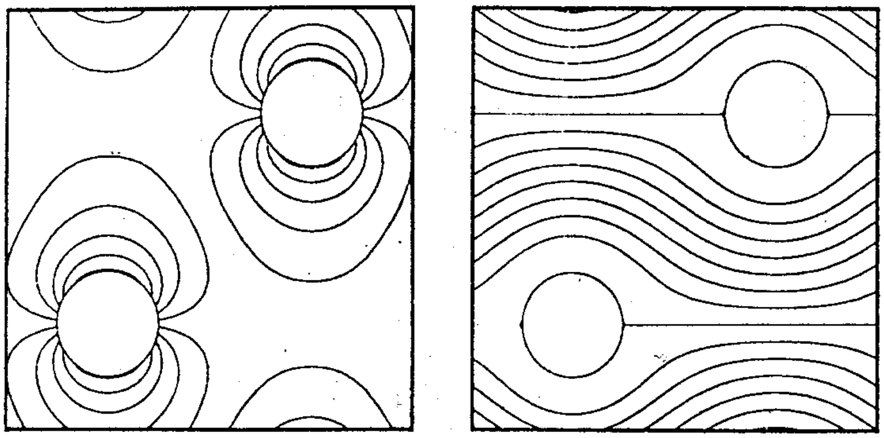
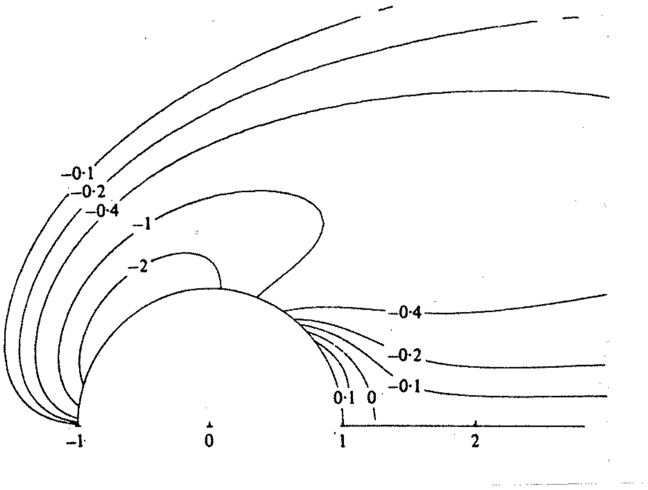
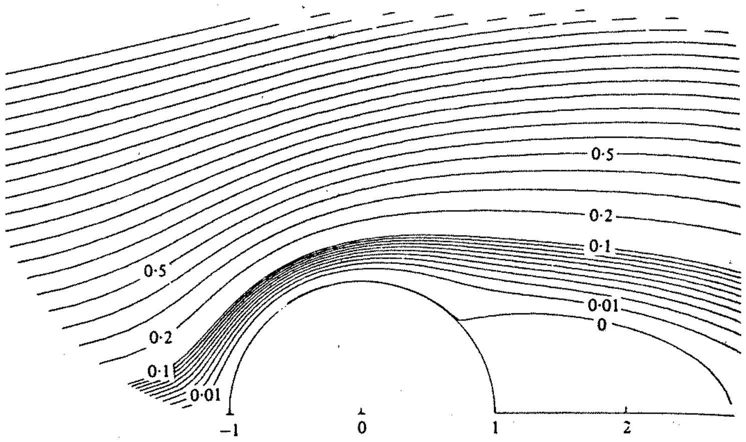


Figure A.1: Vorticity contours (left) and streamlines (right) of the Stoke's flow problem with biperiodic boundary conditions. Reproduced from Tezduyar and Liou [50].



(a)



(b)

Figure A.2: Vorticity contours (a) and streamlines (b) for flow around cylinder at $Re = 20$. Reproduced from Fornberg [24].

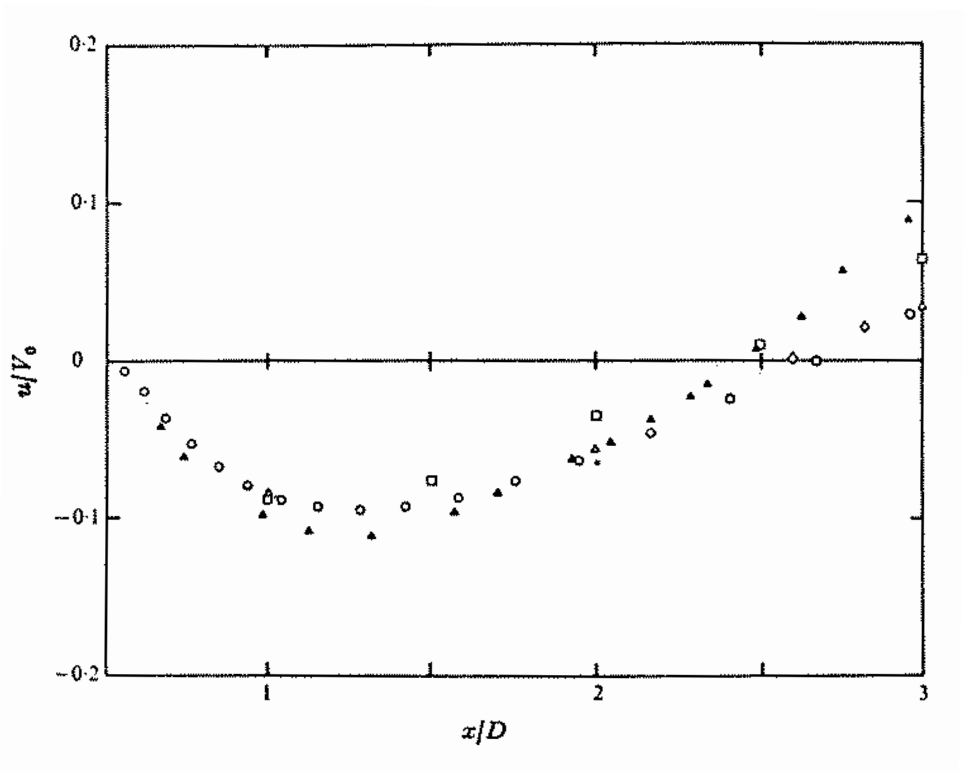


Figure A.3: Experimental results of horizontal velocity along centerline behind cylinder moving with speed V_0. Variable x is the distance from the center of the cylinder, and D is the diameter of the cylinder. The triangle symbol are the experimental results reported by authors of paper from which this plot was taken. Other symbols are from other researchers. Reproduced from Coutanceau and Bouard [16].

VITA

Donna Calhoun
Department of Applied Mathematics
University of Washington
P.O. Box 352420
Seattle, Washington 98195-2420

Education

- *University of Washington*, Seattle, Washington
  Ph.D. in Applied Mathematics, 1999

- *University of Washington*, Seattle, Washington
  M.S. in Mathematics, 1988

- *Pomona College*, Claremont, CA
  B.A. in Mathematics, 1985

Employment History

- Aptech Systems (1989 - 1991)

- National Oceanic and Atmospheric Administration (1991 - 1994)