Development of a Two-Dimensional Model of Blood Microcirculation Flows

by

Kevin Sabo

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Signature redacted

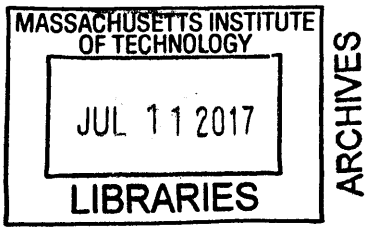
Author ... .
Department of Aeronautics and Astronautics
May 25, 2017

Signature redacted

Certified by..... .....
Wesley L. Harris
C.S. Draper Professor of Aeronautics and Astronautics
Thesis Supervisor

Signature redacted

Accepted by . .....
Youssef Marzouk
Chairman, Department Committee on Graduate Theses



Development of a Two-Dimensional Model of Blood Microcirculation Flows

by

Kevin Sabo

Submitted to the Department of Aeronautics and Astronautics
on May 25, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis presents the development of a dimensionless blood microcirculation model for the study of blood microcirculation flows. It is a two dimensional, axially symmetric, incompressible, Newtonian-flow, Krogh cylinder model subjected to axially periodic boundary conditions. This model formulation allows for the use of the streamfunction-vorticity formulation of the Navier-Stokes equation, which offers simplification to boundary conditions and also allows for the use of a non-uniform, collocated mesh. A streamfunction vorticity formulation of the Immersed Boundary Method is also developed, specifically for the boundary conditions along the immersed boundary (red blood cell membrane). Periodic boundary conditions are used, with the assumption of fully-developed flow, in order to focus on the effects of the transient diffusion of oxygen into the surrounding tissue, orthogonal to the capillary flow direction.

Thesis Supervisor: Wesley L. Harris
Title: C.S. Draper Professor of Aeronautics and Astronautics

Acknowledgments

Grandma Sabo, this is for you.
I know you said that I could be a doctor,
but this is as close as I will ever get to blood.

I have many people whom I want to thank. If I forget anyone, I offer my sincerest apologies because so many have helped me get to where I am today.

First and foremost, thank you to Professor Wesley L. Harris. Professor Harris has given me a wonderful opportunity to work on a problem, one which I have never seen before, that offers a complete overview of the engineering process. Also, Professor Harris has also been an excellent source of wisdom in regards to general daily life as well. Having known him since I was a wee freshman here at MIT and having him advise me throughout my undergraduate and graduate careers, I can confidently say that I would not be here if it were not for his guidance and wisdom.

Next, I want to thank my parents, Steve and Sonya Sabo. They have watched me grow throughout the years and have always been an unwavering source of love and support. From sharing in my joy of getting into graduate school to taking a phone call at four in the morning because I was too stressed out to sleep, I cannot thank them enough.

I also want to thank all of my friends here at MIT who have helped me out in many ways. A special thanks goes to Eric Tu, the guy who has been there through pretty much all of MIT, and is even sitting here next to me as I write this statement (he has no idea - Hi Eric!). To my friends who have been at my side throughout this endeavor: Dominique Hoskin, Andrew Koff, Billy Moses, Alex Luh, Zachery Miranda, Andrew Zmolek, Louis Chen, Chris Gilmore, Jacquie Thomas, Tim Galligan, Haofeng Xu, Ben Couchman, Tony Tao, Theo Mouratidis, Jinwook Lee, Albert Gnat, Vince Wang, Derek Paxson, Georgi Subashki, Juju Wang, Maddie Jansson, Dolly Yuan,

Dakota Pierce (平俊哲), Grace Yin, and the entire Maseeh 2 gang (past and present), thanks for being supportive and keeping life around MIT fun.

Finally, I want to thank the entirety of the faculty and staff here at MIT's Department of Aeronautics and Astronautics. Thank you Marie Stuppard and Beth Marois, both of whom keep all the deadlines in order and make it easy for us students to focus on courses and research. Thank you to Todd Billings and David Robertson, both of whom have given me countless hours of technical and life advice, as well as increased my repertoire of jokes and humor topics. Thank you Mark Drela for the laughs and for answering my endless stream of varied questions, despite being on sabbatical.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Modeling Blood Microcirculation Flows

The modeling of blood microcirculation flows is important for the study of numerous physiological processes, including the study of oxygen transport and other molecular transport to muscular tissues as well as various diseases, such as sickle cell disease. Results from such modeling and simulations can be used to help guide scientists, engineers, and medical professionals in creating new and effective treatment options for patients afflicted with various ailments and diseases.

There currently exists a large body of work on blood microcirculatory flows. As summarized by Gompper and Fedosov[1], two prominent observations have come from the study of these flows: the Fahraeus effect and the Fahraeus-Lindqvist effect. These effects lead to a seemingly reduced hematocrit in the capillary and an apparent viscosity reduction in the blood flow, respectively. Understanding these effects provides and understanding for the counter-intuitive results that they are responsible for.

The Fahraeus effect describes a seeming inconsistency regarding the hematocrit, the volume fraction of red blood cells (RBCs) in a blood vessel, in capillary flows. The overall hematocrit of the capillary flow is lower than that of the hematocrit observed at the capillary discharge. The underlying physics behind this effect resides with the RBCs occupying the center of the capillary flow, thus moving faster on average than the blood plasma which moves through the capillary.

The Fahraeus-Lindqvist effect describes the apparent viscosity reduction in the blood microcirculation flow as a function of the capillary diameter. This effect arises from the occupation of RBCs in the center of the capillaries, but in context with the RBC-free region near the capillary edges. This RBC-free region effects the apparent viscosity depending on the diameter of the capillary. The maximal reduction in viscosity is achieved in capillaries that are 7-8 µm in diameter. Below this diameter ranges results in RBCs producing increased blood plasma shear near the wall. Above this diameter range, the effect becomes negligible and the viscosity increases. This effect is quantified for the equivalent Hagen-Poiseuille flow viscosity, which can be expressed as follows[2]:

$$\mu_{app} = -\Delta p \frac{\pi}{8} \frac{R_w^4}{(Q + \pi R^2 U)} \tag{1.1}$$

Here, Δp is the pressure drop, Q is the volume flow rate between the RBC and capillary wall, U is the average velocity, and R_w is the capillary wall radius.

These effects, along with numerous others, have been studied extensively in microcirculation flows. While these effects provide key insight into the underlying physics, they do not provide a full description of all the effects occurring in these flows.

Instead of tackling specific effects, the aim of this research is to develop a framework which tracks high-level, non-dimensional effects of various physical blood microcirculation properties. While some assumptions are made regarding the properties of the microcirculation flow and surrounding tissue, the methodology behind the dimensionless formulation is general and can be applied to most systems. Instead of varying specific coefficients and running a plethora of experiments or simulations, varying dimensionless groupings of these coefficients can lead to more insight while decreasing experimentation and simulation time.

1.2 Sickle Cell Anemia: Example of The Importance of Understanding Blood Microcirculation Flows

According to the National Institute of Health's Heart, Lung, and Blood Institute, sickle cell disease is a genetic blood disorder in which the afflicted patient has abnormal hemoglobin, *hemoglobin S* (HbS) or sickle hemoglobin, which causes the RBCs to sickle upon their deoxygenation (source: *https://www.nhlbi.nih.gov/health/health-topics/topics/sca*). It is an autosomal recessive disease, requiring a gene mutation from both parents in order to be phenotypically active.

Sickle cell disease is found all over the world, but is largely concentrated in Africa and some parts of Asia. The leading hypothesis for this occurrence is that the disease offers resistance to malaria, a parasitic disease with mosquitoes acting as a primary carrier. This hypothesis has been confirmed by Piel et al[4]. A geographical map showing their Bayesian geostatistical analysis is shown in Figure 1-1. As can be seen, the malaria *holoendemic* (defined as when essentially every individual in a population is infected with a disease) and the frequency of the HbS gene go hand-in-hand. Understanding how to combat the negative effects of sickle cell anemia would directly benefit a large population.

The phenotypical signature that sickle cell disease is commonly known for is the formation of RBCs with a sickled shape, as can be seen in Figure 1-2. When an RBC releases oxygen (oxygen unbinds from the hemoglobin protein), the RBC retains its normal shape in a healthy patient. However, in afflicted patients, the RBC sickles, which can lead to capillary vessel occlusion, a condition when the capillary becomes clogged. When this event happens, the tissue downstream of the blockage receives little to no oxygen, leading to painful events called *crises*.

An understanding of the transient conditions which lead to RBCs sickling can potentially be found from a blood microcirculation model. In particular, a dimensionless model can isolate which physical effects dominate the transient state from normal RBC behavior to sickled RBC behavior. While the sickled RBC model requires some extra steps[2], a dimensionless microcirculation model framework will help

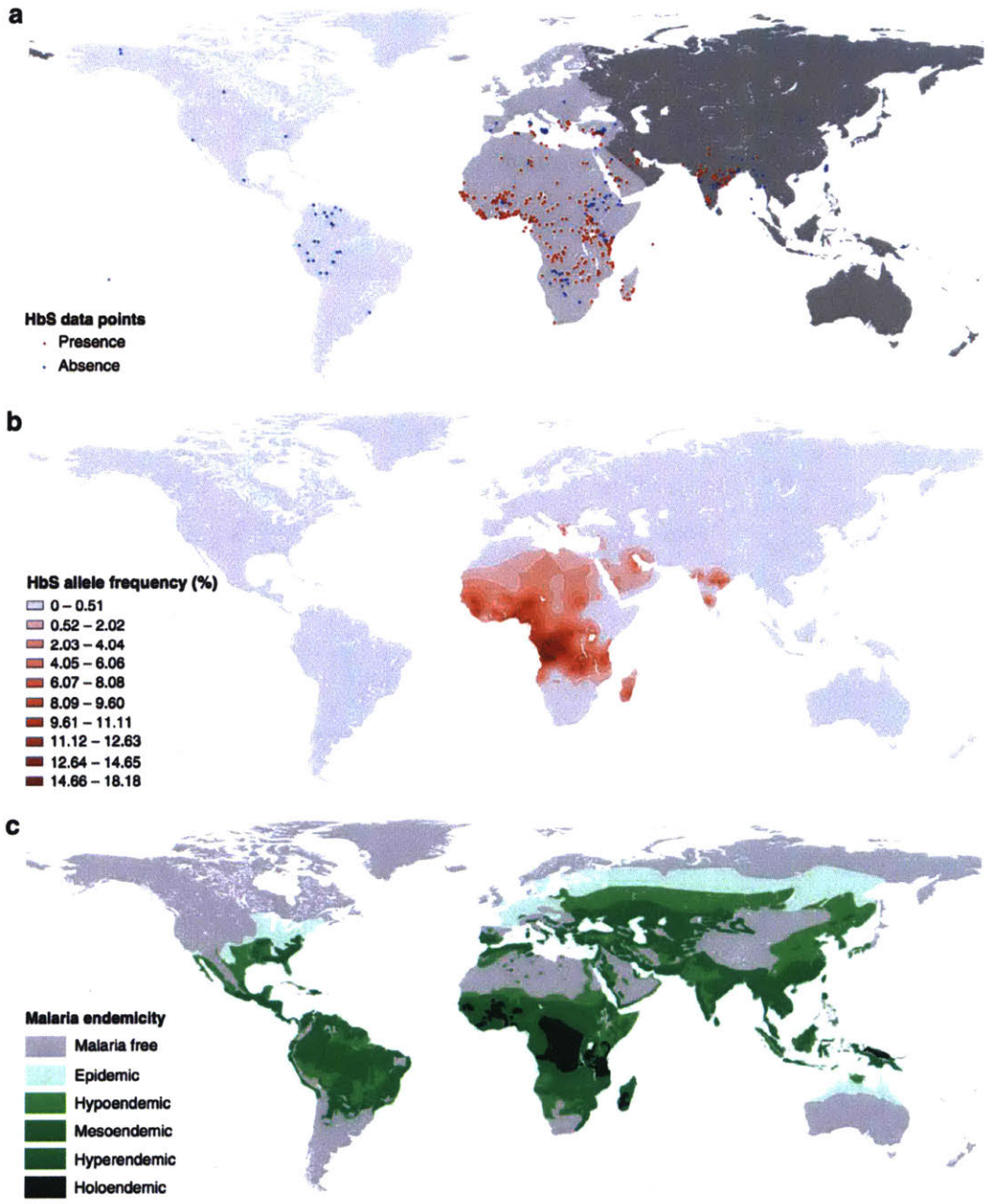


Figure 1-1: (**a**) Data points collected, red dots representing presence of HbS gene and blue dots representing absence of HbS gene. (**b**) Raster map of HbS gene frequency. (**c**) Historical map of malaria endemicity. Figure and caption from Piel et al.[4]

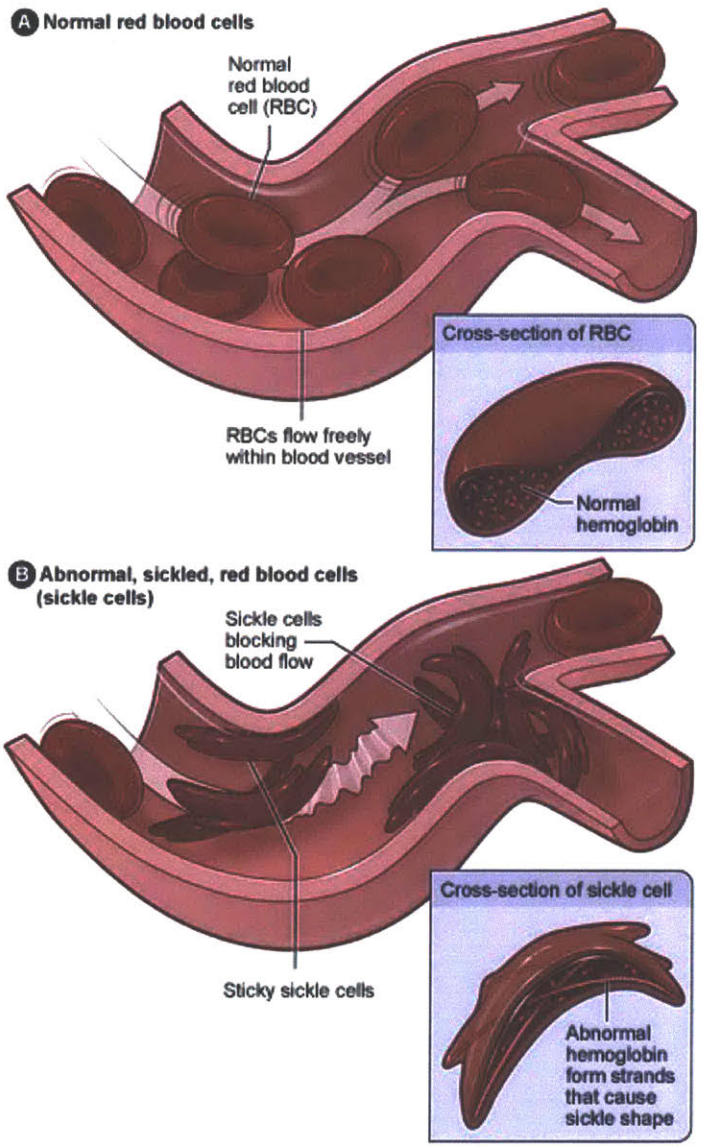


Figure 1-2: Graphic depicting normal RBC versus sickled RBC behavior with key difference being blockage of inlet to microcirculation passage. Image courtesy of *NIH* (*https://www.nhlbi.nih.gov/health/health-topics/topics/sca*).

to uncover trends which govern RBC sickling.

1.3 Outline of Thesis

This thesis is composed of five chapters.

Chapter 1:

The first chapter covers the general importance of modeling blood flows, specifically that of blood microcirculation flows. An example for context is given in the study of sickle-cell anemia.

Chapter 2:

The second chapter provides an overview of previous research and motivation for the continual study of research problems in this field. It will first review the background work done on blood microcirculation models, covering the fluid mechanics, oxygen transport, and membrane mechanics. Next, the importance of non-dimensionalization is discussed in the context of canonical fluid mechanics problems and how those techniques may be applied to hemodynamics problems.

Chapter 3:

The third chapter develops the physical model used in this work for the blood microcirculation problem. The domain and sub-domains are established and the assumptions pertaining to the problem domain and sub-domains are laid out. The fluid mechanics of the blood plasma and cellular cytoplasm are explored in depth with the appropriate modeling assumptions stated. The oxygen transport equations are also developed for the various sub-domains. The cellular membrane mechanics are developed from first principles and the resulting constituent relations are found with the appropriate modeling assumptions. Lastly, the resulting system of equations are non-dimensionalized in order to obtain the non-dimensional parameters that are of interest to hemodynamicists.

Chapter 4:

The fourth chapter covers the simulation algorithm and code implementation. The Newton-Raphson Method (or Newton Method) for root-finding is discussed. How this method is applied to finite difference code is also covered. Non-uniform mesh techniques are developed for the problem in order to mitigate numerical boundary layer issues. The resulting domain discretization (meshing) is then discussed, paying special attention to unique requirements for the Immersed Boundary Technique. Finally, the discretization of the governing equations is reviewed and the appropriate boundary conditions discussed for the domain, sub-domains, and cellular membrane.

Chapter 5:

The fifth chapter of this thesis is dedicated to the discussing of the results of the research. Comparisons to previous work are made and recommendations for future work are presented. The key focuses for future work involve the extended use of the non-dimensional framework developed as well as potentially better techniques for simulating the problem, such as a finite element method with adaptive grid generation.

Chapter 2

Previous Research and Motivation

2.1 Current Blood Microcirculation Models

This chapter presents the high-level thoughts and modeling assumptions of some previous blood microcirculation models. There is a large body of literature regarding blood microcirculation modeling techniques, and a summary of this body may be found in Gompper and Fedosov[1]. This chapter provides the essential motivation for the modeling done in this thesis work.

The model problem is the Krogh cylinder model, developed by August Krogh in 1919. It first started out as a 3-layer model, consisting of a red blood cell, a capillary containing blood plasma, and muscular tissue. Its intended use was to model the effects of oxygen diffusion through a cylindrical capillary tube, and has done so successfully for nearly 100 years. Vadapalli, Goldman, and Popel[7] and Le Floch-Yin[2] have used a more advanced 5-layer model, adding a vascular wall layer and an interstitial space layer in between the capillary and muscular tissue layers. Also, the red blood cell contains the oxygen-fixing protein, hemoglobin, and the muscular tissue includes the oxygen-fixing protein, myoglobin, both layers having their own set of governing equations.

2.1.1 Fluid Mechanics: Viscous Fluid Modeling of Microculation

Blood microcirculation flows occur in narrow capillaries ranging from 2 µm to 10 µm in diameter. The flow is not homogeneous due to the fact that red blood cells exist inside the blood plasma, leading to a coupled interaction between the red blood cells and the blood plasma.

Blood plasma is approximately 92% water.[9] Previous models [2][8][13] have assumed that the blood plasma in the microcirculation is an incompressible, Newtonian, viscous fluid that can be modeled by the relevant Navier-Stokes equations. Most recent works also make this assumption, as stated by Sousa et al.[14] However, it has been experimentally demonstrated with sufficient confidence that blood is a complex, shear-thinning fluid (refer to Figure 2-1), leading the authors to strongly recommend the modeling community to incorporate the shear-thinning effects into new models.[9]

The red blood cells are assumed to be fluid-filled "sacks" that contain cellular cytoplasm and hemoglobin proteins.[2] The fluid properties of the cell are assumed to be similar to that of the blood plasma (e.g. incompressible, Newtonian, viscous fluid) surrounding the cell, with a few minor differences (e.g. density and dynamic viscosity). From a dimensionless perspective, the Reynolds number of the blood plasma and cellular cytoplasm, relative to the capillary diameter, are different.

Since these fluids have different Reynolds number, it is reasonable to expect that the red blood cell membrane mechanics will be affected by this change. In previous work,[2] the density was chosen to remain constant between the blood plasma and cellular cytoplasm while only varying the dynamic viscosity. The blood plasma and cellular cytoplasm density was taken as $1025\,\mathrm{kg/m^3}$, despite the latter being reported to be slightly larger at $1125\,\mathrm{kg/m^3}$. In the Newtonian fluid assumption, the affect on the Reynolds number due to this difference would be negligible,[2] but this affect is not negligible in the copmlex, shear-thinning fluid assumption.

In the context of the current research that this thesis presents, the fluid will be assumed incompressible, Newtonian, and viscous for simplicity in deriving and imple-

menting a new non-dimensional model using a streamfunction-vorticity formulation of the fluid transport equations. It is strongly recommended that viscoelasticity effects be considered in future research, as Reynolds number of blood microcirculation flows are O(1), with Figure 2-1[9] showing the sensitivity to Reynolds number at these lower values.

Limitations of Newtonian Fluid Assumption

The blood microcirculation flow in this work is assumed to be Newtonian in nature, or for the isothermal, incompressible assumption, that the viscosity coefficient μ is constant. However, it is observed in reality that blood is a complex fluid due to the suspension of blood cells, proteins, mineral ions, hormones, and glucose.[9]

The work done by Brust et al.[9] show that blood plasma is a shear-thinning complex fluid (i.e. the viscosity decreases under increased strain rate). Brust et al. experimentally showed, via the use of a microfluidic contraction-expansion device, the dimensionless pressure drop as a function of the Reynolds number for human blood plasma versus that of water (see Figure 2-1).

Figure 2-1 clearly indicates a non-linear, shear-thinning behavior of human blood plasma as the Reynolds number increases. However, water exhibits no change in dimensionless pressure drop as the Reynolds number increases over the indicated range.

In the upper-right corner of Figure 2-1, the actual pressure drop is measured against the volume flow rate. Water (black) exhibits a linear relationship while the human blood plasma (red) deviates slightly in the intermediate flow rate range of 100-600 micro-liters.

Brust et al. conclude that the viscoelastic behavior (shear-thinning) of human blood plasma is significant and recommend that it should not be ignored in future blood flow modeling. They also indicate that the viscoelasticity may lead to viscoelastic flow instabilities, especially when RBCs are present at values around 50% hematocrit.

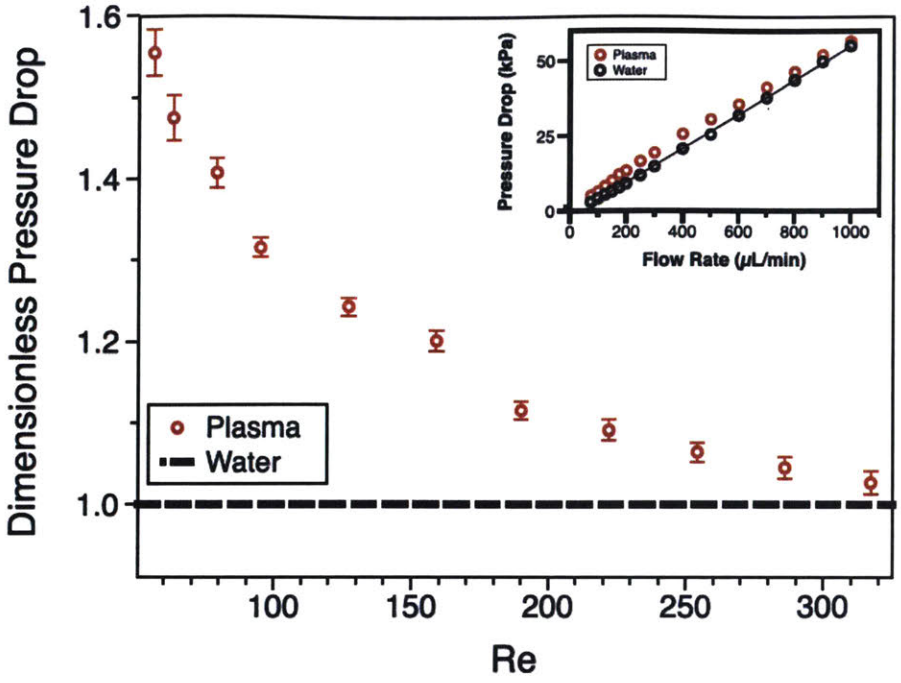


Figure 2-1: Experimentally measured dimensionless pressure drop of human blood plasma (red circles) versus that of water (black dashed line), indicating non-linear, shear-thinning behavior (figure provided by Brust et al.[9]).

2.1.2 Species Transport: Chemical Modeling of Oxygen Transport in Capillaries and Surrounding Tissue

As stated previously, the Krogh cylinder model was developed for the study of oxygen flow in blood microcirculation flows. The original 3-layer model has developed into an advanced 5-layer model[2][13] which has proven versatile in its development over the past nearly 100 years.

The model used by Secomb et al and Le Floch-Yin consists of a simple cylindrical geometry that is layered similar to that of a cake (see Figure 3-1). The innermost layer is the capillary domain which includes the blood plasma and the red blood cell layer. The capillary is surrounded by the vascular wall, the vascular wall is then surrounded by the interstitial space, and the interstitial space is surrounded by the muscular tissue, thus yielding 5 layers. The red blood cell layer contains hemoglobin proteins which can bind with oxygen and the muscular tissue layer consists of myoglobin proteins which can also bind with oxygen.

In order to model the oxygen concentration dynamics, advecting-diffusion transport equations were developed in the form

$$\frac{\partial c}{\partial t} + (\vec{v} \cdot \nabla) c = D\nabla^2 c + R \tag{2.1}$$

where c represents a scalar quantity (in this case, the concentration of oxygen). Upon invoking Henry's law

$$\alpha_{O_2} = \frac{c_{O_2}}{p_{O_2}} \tag{2.2}$$

where α_{O_2} is the oxygen solubility coefficient and p_{O_2} is the oxygen partial pressure. This statement leads to the governing equations for oxygen taking the form

$$\frac{\partial p_{O_2}}{\partial t} + (\vec{v} \cdot \nabla) p_{O_2} = \frac{D}{\alpha}\nabla^2 p_{O_2} + \frac{R}{\alpha} \tag{2.3}$$

The first term represents unsteady effects, the second advective effects, the third diffusive effects, and the fourth reaction rate chemistry with the oxygen-fixing protein effects.

A similar transport equation is developed for the saturation of the oxygen-fixing proteins, which is coupled to the oxygen partial pressure equations via the reaction rate term.

$$\frac{\partial S}{\partial t} + (\vec{v} \cdot \nabla) S = D\nabla^2 S - R' \tag{2.4}$$

Through the reaction rate terms R and R' (which are not equivalent in this representation), the equations are coupled and the effects of the binding and unbinding of oxygen to the proteins is adequately captured.

The detailed derivation and explanation of these equations can be seen in Chapter 3 and Appendix B.

2.1.3 Membrane Mechanics: Cell Suspension Modeling of Red Blood Cells

In the previous work of Le Floch-Yin, the red blood cell membrane was assumed to be an axisymmetric, deformable shell. The shell undergoes deformations in time due to the fluid interactions of the blood plasma and cellular cytoplasm with the membrane. These interactions are captured via a body force model, represented as $\vec{f}$, which is accounted for in the governing equations for the fluid momentum (for a full derivation, see Appendix A):

$$\frac{D\vec{v}}{Dt} = -\nabla p + \nu\nabla^2\vec{v} + \frac{1}{\rho}\vec{f} \tag{2.5}$$

However, how the body forces are calculated depends on the mechanics of the red blood cell membrane. For these mechanics, constituent relations are derived.

The first constituent relations for the red blood cell membrane deformations are developed in Evans and Skalak's *Mechanics and Thermodynamics of Biomembranes*:[5]

$$\bar{t} = \sigma_0 + K\frac{dA}{dA_0} \tag{2.6}$$

$$t_d = \frac{\kappa}{2\left(1+\frac{dA}{dA_0}\right)^2}\left(\lambda_1^2 - \lambda_2^2\right) \tag{2.7}$$

$$m = B\left(1+\frac{dA}{dA_0}\right)(k - k_0) \tag{2.8}$$

where $\bar{t}$ is the isotropic mean tension, t_d is the shear deviatoric term, and m is the bending moment about a membrane point in any principal direction. Here, σ_0 is the membrane's reference state isotropic tension, K is the isothermal area compressibility modulus, κ is the 2-D shear modulus, B is the isotropic bending modulus, $\frac{dA}{dA_0}$ is the area change with respect to the reference state, λ_1 and λ_2 are extension ratios in the reference directions such that $\frac{dA}{dA_0} = \lambda_1\lambda_2 - 1$, k is the total local curvature, and k_0 is the reference local curvature.

While this model proves to be successful, some modifications to better capture the interaction between bending and tension forces were carried out by Secomb in 1988.[12] Secomb added the assumption of axisymmetry to the model and removed terms related to area changes that were not of leading order. The new constituent relations used are:

$$\bar{t} = \frac{t_s + t_\theta}{2} = \sigma_0 + K\frac{dA}{dA_0} \tag{2.9}$$

$$t_d = \frac{t_s - t_\theta}{2} = \frac{1}{2}\kappa\left(\lambda_s^2 - \lambda_s^{-2}\right) - \frac{1}{2}B\left(k_s - k_\theta\right)\left(k_s + k_\theta - k_0\right) \tag{2.10}$$

$$m = B\left(k_s + k_\theta + k_0\right) \tag{2.11}$$

where s and θ are curvilinear coordinates following the membrane surface.

Equilibrium equations were derived in order to incorporate the constituent relations into the body force model:

$$\Delta p = -t_s k_s - t_\theta k_\theta - \frac{1}{R}\frac{d\left(Rq_s\right)}{ds} \tag{2.12}$$

$$-\tau = \frac{1}{R}\frac{d\left(Rt_s\right)}{ds} - t_\theta\frac{1}{R}\frac{dR}{ds} - q_s k_s \tag{2.13}$$

$$0 = \frac{dm}{ds} + q_s \tag{2.14}$$

where Δp is the local pressure difference between the external and internal fluids (local normal force per unit area), τ is the local shear stress in the s direction (local tangential force per unit area), and the equation (2.14) represents the local moment per unit area, which equals zero in this instance as the red blood cell is suspended in fluid (thus unanchored, so no external moments apply). The new terms introduced are R, which pertains to the local membrane radius and is a function of s, and q_s is the shear force per unit length in the membrane and is also a function of s. For a

complete derivation, see Appendix C.

For future research, it is recommended to use a viscoelastic model which captures the behavior of the red blood cell membrane with non-negligible viscoelastic effects. Work done by Tżeren et al[15][16] develop constitutive relations that model the effects from the viscoelastic behavior. The viscoelastic modification to equations (2.9) - (2.11) is only seen in equation (2.10), and is shown below:

$$t_d = \frac{1}{2}\kappa\left(\lambda_s^2 - \lambda_s^{-2}\right) - \frac{1}{2}B\left(k_s - k_\theta\right)\left(k_s + k_\theta - k_0\right) + 2\mu_{RBC}\left(\frac{1}{\lambda_s}\frac{\partial\lambda_s}{\partial t}\right) \tag{2.15}$$

where the viscoelasticity is a time-dependent behavior.

2.2 Motivation for Dimensionless Equations

Studying problems from a dimensionless viewpoint offer advantages for both the formulation of the problem and the insight gleaned from the results of the problem. These benefits can be explained in three parts. For a highly detailed explanation, please refer to B. Zohuri's textbook on dimensional analysis.[10]

1. Reduction of Variable Count

First, finding dimensionless parameters may reduce the number of variables in the problem. Taking the Reynolds number as a common example of fluid mechanics, one can see that it is made of up four variables. If we were to look for the solution of some function, the functional form would be as follows:

$$F = f(U, L, \rho, \mu) \tag{2.16}$$

However, upon non-dimensionalizing the function in equation (2.16), the Reynolds number is formed:

$$F = f(Re = \frac{\rho U L}{\mu}) \tag{2.17}$$

The effects of this dimensionless form are profound. By reducing the number of

variables from four to one, a dimensionless curve can be formed. If a high fidelity mapping of the function f is desired, such that a minimum of 100 points must be taken, in the dimensional form, 100^4 experiments (calculations, simulations, or other) are required. By reducing the variables non-dimensionally to one, now only 100 experiments are required.

2. Insight into Governing Equations

The second benefit is that pertaining to the models developed for a particular governing equation or set of governing equations. As seen in Appendices A, B, and C, the governing equations naturally form dimensionless parameters. This formation of dimensionless parameters reveals key insights into the governing physics of the problem.

For instance, the Reynolds number describes (non-dimensionally) the ratio of advective forces to frictional forces. By adjusting this parameter, one may characterize how much advection and diffusion occur in the fluid.

3. Similitude

The third benefit pertains to that of similitude. Similitude has three requirements: geometric similarity, kinematic similarity, and dynamic similarity.

Take an airfoil as an example. In order to test its flight characteristics, it may be cumbersome or infeasible to test a full-scale model. Engineers instead will test a scale model, preserving the shape of the airfoil, but shrinking it to an appropriate and manageable size for testing.

However, preserving the shape (enforcing geometric similarity) is only one requirement. The second requirement is kinematic similarity. Formally, the fluid over the airfoil in both the full-scale and scaled test cases must undergo similar time rates of change or change of motions. In other words, quantities related to motions or how things move must be similar.

The final similarity requirement is that of dynamic similarity. This requirement pertains to the ratio of forces acting on the system. In the case of the airfoil, the Reynolds number is a key measure of the ratio of inertial (advective) forces to viscous

(frictional) forces. If the full-scale and scaled cases are ran at the same Reynolds number, they are said to have dynamic similarity.

If these three criteria are met, a problem is said to have similitude. The non-dimensional solution to the model that governs a particular problem can then be rescaled to the required dimensions. Doing this for blood microcirculation flows allows for the scaling of physics pertaining to blood plasma viscosity, oxygen diffusion throughout the capillary to the muscular tissue, study the effects of variable hemoglobin and myoglobin concentration, as well as other mechanical and chemical effects.

2.3 Motivation: Thesis Hypothesis and Objectives

The first goal of this thesis work is to provide a dimensionless perspective to blood microcirculation flows in order to reduce the number of necessary simulations required to gain valuable insight into the problem and to take advantage of similitude. With these three principles evoked in the presented model, the results from a database of parametric studies (future work) should be able to guide medical professionals in developing new drugs and new therapeutic treatments that directly affect sickle cell crises.

The second goal of this thesis work is to generate a user-friendly simulation using the Immersed Boundary Method (IBM) technique. While the IBM is conceptually intuitive to understand, it presents unique challenges in the approximation of a moving boundary and its boundary conditions on the flow fields inside and outside of the moving boundary. In the context of the red blood cell membrane, a zero-velocity boundary condition and a normal oxygen mass flux boundary condition must be satisfied on and across the membrane, respectively. Enforcing these boundary conditions is a challenging task and leads to some undesirable arbitrariness in their implementation. The new goal is to understand why the arbitrariness arises and to provide a clear path forward in the discretization of this problem.

Chapter 3

Development of Blood Microcirculation Model

This chapter presents the blood microcirculation model. First, an overview of the model is presented, including key assumptions and model geometry. Next, the details of the model are presented, including the fluid dynamics of the blood microcirculation environment, oxygen species transport, and red blood cell membrane mechanics. Finally, the governing equations are summarized and non-dimensionalized.

3.1 Physical Model of Blood Microcirculation and Red Blood Cell System

The model used in this thesis is a modified approach to the work done by Le Floch-Yin.[2] It consists of a two-dimensional, axially-periodic domain made up of five sub-domains: the red blood cell, the capillary, the vascular tissue, the interstitial space, and the surrounding muscle tissue. A visualization of this can be seen in Figure 3-1.

It is assumed to be symmetric about the centerline of the capillary, neglecting any variations in capillary flow area (and effects associated with vary vessel size) along the main axis of the capillary. This assumption ensures symmetry across the capillary while simultaneously reducing code complexity and runtime.

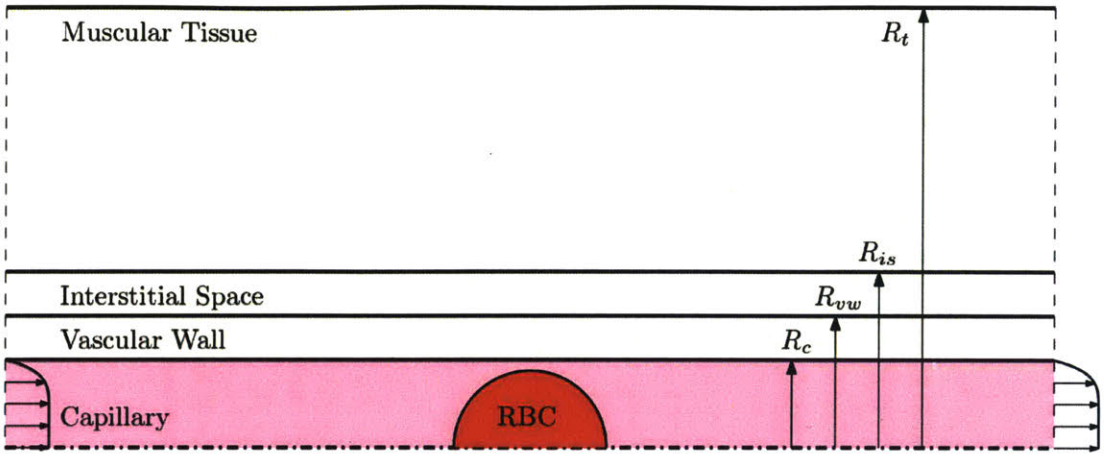


Figure 3-1: Visualization of the problem domain including the capillary, vascular wall, interstitial space, and muscular tissue sub-domains.

Each sub-domain has its own set of physical coefficients and parameters. While some of the coefficients are equivalent, this fact is not generally the case. These values are listed in Table 3.1 (the data are taken from work done by Vadapalli, Goldman, and Popel[7]).

	$\vec{v}$ (m/s)	ν (m^2/s)	D_{O_2} (m^2/s)	α (mol/m^3 /mmHg)	$[Hb]$ (mol/m^3)	$[Mb]$ (mol/m^3)	M ($mol/m^3/s$)
RBC Cytoplasm	(v_x, v_y)	$5.2356 \cdot 10^{-6}$	$9.47 \cdot 10^{-10}$	$1.3118 \cdot 10^{-3}$	21.099	0	0
Blood Plasma	(v_x, v_y)	$1.3659 \cdot 10^{-6}$	$2.40 \cdot 10^{-9}$	$1.0906 \cdot 10^{-3}$	0	0	0
Vascular Wall	0	-	$8.73 \cdot 10^{-10}$	$1.5097 \cdot 10^{-3}$	0	0	$3.8811 \cdot 10^{-3}$
Interstitial Space	0	-	$2.18 \cdot 10^{-9}$	$1.0906 \cdot 10^{-3}$	0	0	0
Tissue	0	-	$2.41 \cdot 10^{-9}$	$1.5059 \cdot 10^{-3}$	0	0.4	$6.1321 \cdot 10^{-3}$

Table 3.1: Velocity and physical coefficients and parameters associated with separate sub-domains (from Vadapalli, Goldman, and Popel[7]).

The velocity vector $\vec{v}$ only exists in the RBC cytoplasm and the blood plasma as those two sub-domains are the only sub-domains with bulk fluid motion. Thus, the kinematic viscosity ν ($\equiv \mu/\rho$) only takes on a value in those two sub-domains. D represents the diffusion coefficient for oxygen, α the oxygen solubility constant, $[Hb]$ and $[Mb]$ the molar concentration of hemoglobin and myoglobin , and M a hypot-

hesized oxygen consumption rate constant. Not that in some of the sub-domains, the molar concentration of hemoglobin and myoglobin go to zero, thus eliminating the associated reaction rate effects in those regions. Also note that the vascular wall and muscular tissue are the only sub-domains hypothesized in previous work[2] to consume free oxygen.

These coefficients and parameters arise in the governing equations presented in the model. However, for the formation of the oxygen transport equations, these parameters do not make up the complete set of required model inputs. The complete derivation of these equations can be seen in Appendix B. The additional inputs required are listed in Table 3.2. Table 3.1 and Table 3.1 complete the list of required input parameters for the model continuum model.

Input	Symbol	Value
Kinetic dissociate rate constant, oxyhemoglobin backwards reaction	k_{-1}^{Hb}	44 s^{-1}
Kinetic dissociate rate constant, oxymyoglobin backwards reaction	k_{-1}^{Mb}	15 s^{-1}
Oxygen partial pressure at equilibrium, 50% hemoglobin saturation	$p_{O_2 50\%}^{Hb}$	5.3 mmHg (706.6 Pa)
Oxygen partial pressure at equilibrium, 50% myoglobin saturation	$p_{O_2 50\%}^{Mb}$	29.3 mmHg (3906.3 Pa)
Diffusivity of oxyhemoglobin	D_{Hb}	$6.10 \cdot 10^{-11}$ m^2/s
Diffusivity of oxymyoglobin	D_{Mb}	$1.3783 \cdot 10^{-11}$ m^2/s

Table 3.2: Additional model input values required for generation of governing equations for oxygen species transport.

The last set of parameters needed are those for the structural mechanics and associated constituent relations of the RBC membrane. These parameters are listed in Table 3.3.

The initial isotropic tension in the RBC membrane's resting, unstressed state is represented as σ_0. The isothermal area compressibility modulus K, often referred to as the 2-D bulk modulus, measures the membrane's resistance to surface area changes.

Input	Symbol	Value
Initial isotropic tension in membrane	σ_0	$7 \cdot 10^{-2}$ kg/s^2
Isothermal area compressibility modulus	K	0.5 kg/s^2
Membrane bending modulus	B	$1.8 \cdot 10^{-19}$ kg m^2/s^2
Membrane shear modulus	κ	$4.2 \cdot 10^{-6}$ kg/s^2

Table 3.3: Parameters required for the constituent relations of the red blood cell membrane structural mechanics.

The bending modulus B quantifies the membrane's resistance to bending. Finally, the membrane's shear modulus κ represents the membrane's resistance to shear stresses acting on the membrane. These values used are in accordance with the work of *Evans and Skalak*[5] and *Halpern and Secomb*[8].

3.2 Fluid Mechanics of the Microcirculation

The blood plasma and red blood cell cytoplasm are assumed to be incompressible, Newtonian viscous fluids. This assumption allows for the use of the incompressible forms of mass continuity and the Navier-Stokes equations to be used as the fluid equations of motion, with associated body force $\vec{f}$:

$$\nabla \cdot \vec{v} = 0 \tag{3.1}$$

$$\frac{D\vec{v}}{Dt} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v} + \frac{1}{\rho}\vec{f} \tag{3.2}$$

Given that the model is assumed to be fully two dimensional, the problem can be simplified to a streamfunction vorticity formulation:

$$\omega + \nabla^2 \Psi = 0 \tag{3.3}$$

$$\frac{\partial \omega}{\partial t} + (\vec{v} \cdot \nabla)\,\omega = \nu \nabla^2 \omega + \frac{1}{\rho}\left(\nabla \times \vec{f}\right) \tag{3.4}$$

This formulation has the advantage of reducing a three variable vector transport system of equations to a two variable scalar transport system. The approach allows for a reduced computation time due to the reduced number of independent variables as well as a simpler use of a collocated mesh, as opposed to a staggered mesh, to compute on. While a collocated mesh can be used for the traditional representation of the Navier-Stokes equations, it is often difficult to run (refer to section 3.2.2). Bueno and Harris [3] also used this formulation for the blood plasma fluid dynamics, but did so on a staggered mesh and also left the formulation in its dimensional form.

In order to better characterize the solutions to these governing equations and isolate the dynamics of the problem, the nondimensional form of equations (3.3) and (3.4) are used:

$$\omega^* + \nabla^{*2}\Psi^* = 0 \tag{3.5}$$

$$\frac{\partial \omega^*}{\partial t^*} + \left(\vec{v^*} \cdot \nabla^*\right)\omega^* = \frac{1}{Re}\nabla^2 \omega^* + \nabla^* \times \vec{f^*} \tag{3.6}$$

From the nondimensionalization, the Reynolds number Re is found as a key nondimensional parameter of interest. The Reynolds number measures the ratio of inertial (advective) forces to viscous (diffusive) forces and can be used to isolate the mechanics of the problem. For instance, in high Reynolds number flows, viscous forces are small such that momentum diffusion is largely contained primarily in small viscous layers along solid surfaces (known as boundary layers).

For a more detailed derivation of this model, refer to Appendix A. For a more in-depth discussion of the Reynolds number, refer to Section ??.

3.2.1 Fluid Mechanics: Boundary Conditions

The boundary conditions for the fluid mechanics of the blood plasma are similar to that of pipe flows.

First, along the outer edge of the capillary sub-domain, a solid wall zero-velocity (no-slip and no-flux) boundary condition is applied. This boundary condition corresponds to a constraint on the first-derivative of the streamfunction at the capillary wall, namely the first-derivatives in both the x-direction and y-direction are equal to 0:

$$u\big|_{wall} = \frac{\partial \Psi}{\partial y} = 0 \tag{3.7}$$

$$v\big|_{wall} = -\frac{\partial \Psi}{\partial x} = 0 \tag{3.8}$$

In the streamfunction vorticity formulation, these are enforced via setting the wall to be a streamline (i.e. Ψ is equal to a constant) and substituting equations (3.7) and (3.8) into equation (3.5). Doing so results in the following set of residual equations:

$$R_1\big|_{wall} \equiv \Psi^* - c = 0 \tag{3.9}$$

$$R_2\big|_{wall} \equiv \omega^* + \frac{\partial^2 \Psi^*}{\partial y^{*2}} = 0 \tag{3.10}$$

where c is a constant, which equals the non-dimensional capillary radius length in order to satisfy the non-dimensional volume flow rate of the blood plasma through the capillary.

The boundary condition for the centerline is a symmetric boundary condition. Physically, this corresponds to a streamline with zero vorticity. Thus, the boundary conditions are as follows:

$$R_1\big|_{CL} \equiv \Psi^* = 0 \tag{3.11}$$

$$R_2\big|_{CL} \equiv \omega^* = 0 \tag{3.12}$$

Note that the first residual equation has no constant here as the radius is 0 by de-

finition. This value of Ψ corresponds to it being the first streamline in the domain, while also being the lower bound of the integration over Ψ in order to find the non-dimensional volume flow rate.

The inlet and outlet of the capillary are periodic boundary conditions. In order to enforce this boundary condition, it is assumed that the outlet nodes along edge of the domain are equal to the inlet nodes. The inlet of the domain takes a standard finite difference stencil, while the residuals for the outlet of the domain take the following form:

$$R_1\big|_{outlet} \equiv \Psi^*\big|_{outlet} - \Psi^*\big|_{inlet} = 0 \tag{3.13}$$

$$R_2\big|_{outlet} \equiv \omega^*\big|_{outlet} - \omega^*\big|_{inlet} = 0 \tag{3.14}$$

Lastly, the boundary conditions for the RBC membrane is also a zero-velocity boundary condition (no-slip and no-flux). Although the RBC membrane is modeled as an infinitely thin shell, it is a extensible (moveable and deformable) solid boundary and still acts as a barrier to fluid transit across it. Thus, the first derivative of the streamfunction in both the x and y-direction is required to be zero along the length of the membrane. This boundary conditions is expressed as:

$$R_1\big|_{BC,u} \equiv X^{n+1} - \frac{\Psi(\vec{x})}{dy}\bigg|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{3.15}$$

$$R_2\big|_{BC,v} \equiv Y^{n+1} + \frac{\Psi(\vec{x})}{dx}\bigg|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{3.16}$$

Note that this expression is implicitly solving for the RBC membrane's updated location at a time-step $n+1$. Formulating the boundary condition in this way ensures a strong (implicit, minimized numerical round-off error) versus weak (explicit, larger numerical round-off errors) enforcement of the zero-velocity boundary condition. but it requires solving the equations simultaneously, which results in a generally longer code run-time.

These are the completed boundary conditions for the capillary domain blood plasma fluid dynamics. For the derivation of the overall system of equations, see Appendix A. For a detailed derivation of the boundary conditions relating the plasma-RBC body force interaction, refer to the detailed explanation in Appendix E.

3.2.2 Benefits of Streamfunction-Vorticity Formulations

As alluded to in section 3.2, the streamfunction-vorticity formulation in two dimensions has a few benefits worth noting.

A first benefit is the reduction of variables. In transforming the system from a dynamic system (u-v-p) to a kinematic system (Ψ-ω), the total number of variables required to generate a physical solution is reduced from 3 to 2. Reducing the number of variables without losing model fidelity is one excellent reason to use the streamfunction-vorticity formulation, as the fewer variables results in a (typically) faster simulation runtime.

A second benefit pertains to the boundary conditions of the fluid problem. Instead of specifying dynamic boundary conditions in velocities or pressure levels, a kinematic boundary condition is applied. The boundary conditions on the stream-function Ψ is set by the volume flow rate of the fluid. Dimensionlessly, this is established when using the capillary radius as the problem length-scale. The boundary conditions on the vorticity ω are also easily set, either by the definition from equation (3.5) or by it being 0 across a line-of-symmetry. For a more detailed treatment of the boundary conditions on Ψ and ω, refer to section 4.6.

A third benefit pertains to code simplicity. Having fewer dependent variables results in requiring fewer governing equations to be implemented. This simplifies the construction of the Newton Method Jacobian matrix (see section 4.7) while simultaneously requiring less time to invert and solve the residual system. Incompressible Navier-Stokes codes commonly use projection methods, an efficient technique developed by Alexandre Chorin in 1967, which decouple the pressure and velocity terms. An intermediate velocity term is calculated, but is then corrected by the pressure field in subsequent step in order to satisfy the velocity divergence constraint ($\nabla \cdot \vec{v} = 0$) found

from the incompressible continuity equation. With a streamfunction-vorticity code, the equations do not require a projection method and the $n+1$ time-step solution can be solved for in one step, either explicitly or implicity.

The last benefit worth mentioning pertains to domain meshing strategy. Navier-Stokes codes tend to work better with staggered meshes in order to overcome the problem of spurious, high-frequency pressure oscillations, known as *odd-even decoupling*[6]. These oscillations are non-physical in nature as they do not dampen out and leave a sawtooth, or checkerboard, pattern in flow solutions. In order to overcome this problem, staggered meshes are used. However, streamfunction-vorticity methods do not have this problem, thus a collocated mesh can be used, simplifying the implementation of the code.

3.3 Oxygen Species Transport within the Microcirculation and Surrounding Tissue

The concentration of oxygen is assumed to exist as a continuum so that advecting-diffusion equations may be used to simulate its transport in the domain. In regions where no hemoglobin or myoglobin proteins are present, the concentration of free oxygen, O_2, molecules is modeled via the oxygen partial pressure p_{O_2}. In regions where there is hemoglobin or myoglobin, a saturation transport equation is also present for measuring the amount of bound and unbound oxygen molecules. (For a detailed derivation of the oxygen transport equations, please refer to Appendix B).

The first sub-domain examined is the RBC cytoplasm. In this work, hemoglobin is only assumed to be present inside the RBC. While free hemoglobin proteins may exist in the blood plasma, those proteins and their associated effects are neglected. From this assumption, the first set of governing equations for the inside of the RBC are as follows:

$$\frac{\partial p_{O_2}}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial p_{O_2}}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial p_{O_2}}{\partial y} = D_{O_2}\nabla^2 p_{O_2} + k_{-1}^{Hb}\frac{[Hb]}{\alpha}\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Hb}}\right)^n\right) \tag{3.17}$$

$$\frac{\partial S^{Hb}}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial S^{Hb}}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial S^{Hb}}{\partial y} = D^{Hb}\nabla^2 S^{Hb} - k_{-1}^{Hb}\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Hb}}\right)^n\right) \tag{3.18}$$

Equation (3.17) monitors the transport of oxygen in the form of p_{O_2}. Due to the motion of the RBC, the oxygen transport is unsteady; this effect is quantified via the first term on the LHS. Also, since the RBC is in motion, the oxygen must be advected along, thus the LHS velocity terms (second and third) are present (as streamfunction derivatives). Given that free oxygen is able to diffuse throughout the cytoplasm, the Laplacian diffusion operator is present on the RHS (fourth term). Since hemoglobin is found in the RBC, oxygen is free to bind and unbind with it, thus the presence of the reaction rate term on the RHS (fifth term).

Equation (3.18) is analogous to equation (3.17), except that it measures the advection and diffusion of the hemoglobin saturation. A key difference is that the reaction rate term is now negative, as it should be opposite to that of the partial pressure equation (otherwise, an unphysical and numerical accumulation of oxygen would ensue).

The second sub-domain treated is the blood plasma. As mentioned previously, it is assumed to not have any free hemoglobin present, thus no saturation transport equation is required as the saturation of hemoglobin (as well as myoglobin) is zero by definition. With this assumption in mind, the reaction rate term is now zero as $[Hb] = 0$, and the resulting governing equation is:

$$\frac{\partial p_{O_2}}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial p_{O_2}}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial p_{O_2}}{\partial y} = D_{O_2}\nabla^2 p_{O_2} \tag{3.19}$$

Equation (3.19) models the unsteadiness of the oxygen transport as well as its ad-

vection and diffusion throughout the capillary.

The third sub-domain treated is the vascular wall. In this region (as well as the remaining regions), no bulk fluid motion occurs, thus the LHS advection terms are zero by definition. There are also no hemoglobin or myoglobin proteins present, thus their respective saturations are zero by definition and reaction rate terms are also zero by definition. However, a hypothesized oxygen consumption coefficient M is present, as done previously by LeFloch-Yin.[2] This additional term shows up as a RHS source term:

$$\frac{\partial p_{O_2}}{\partial t} = D_{O_2}\nabla^2 p_{O_2} - M \tag{3.20}$$

The fourth sub-domain treated is the interstitial space. This region also contains no advection or oxygen binding proteins, nor does it contain a oxygen mass consumption coefficient. Thus, the related terms are zero and no saturation governing equation is required. As such, the single governing equation is as follows:

$$\frac{\partial p_{O_2}}{\partial t} = D_{O_2}\nabla^2 p_{O_2} \tag{3.21}$$

The final sub-domain treated is the muscular tissue. This region has no advection, but does have a reaction rate term due to the presence of myoglobin as well as a hypothesized oxygen mass consumption rate term M.[2]

$$\frac{\partial p_{O_2}}{\partial t} = D_{O_2}\nabla^2 p_{O_2} + k_{-1}^{Mb}\frac{[Mb]}{\alpha}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Mb}}\right)\right) - M \tag{3.22}$$

$$\frac{\partial S^{Mb}}{\partial t} = D^{Mb}\nabla^2 S^{Mb} - k_{-1}^{Mb}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Mb}}\right)\right) \tag{3.23}$$

Note that although M is present, it acts as a source term for the oxygen partial pressure. As oxygen is removed from the system by tissue consumption, the saturation will adapt as required.

In order to avoid unnecessary redundancy, the dimensionless forms of the previous

equations are succinctly presented in subsection 3.5.1 (alongside the dimensionless equations for the fluid mechanics). For a detailed derivation and explanation of the non-dimensional parameters, refer to Appendix B and Chapter 5, section ??, respectively.

3.3.1 Oxygen Species Transport: Boundary Conditions

The boundary conditions for the oxygen transport equations result from periodicity of the domain and an oxygen flux constraint (e.g. analogous to that of a heat flux boundary condition).

The periodic boundary condition at inlet and outlet are enforced in a similar manner to those of the fluid mechanics. For the capillary domain, the advection and diffusion terms are approximated at the inlet using information from the outlet and preceding points (as required). The outlet is set equal to the inlet by definition. (This enforcement of the boundary conditions is one way to construct it. Many other methods exist, and this method was chosen for its simplicity).

Across non-periodic boundaries, a normal flux constraint on the oxygen partial pressure is applied. It is mathematically expressed as:

$$\left(\alpha D_{O_2} \nabla p_{O_2} \cdot \hat{n}\right)\Big|_1 = -\left(\alpha D_{O_2} \nabla p_{O_2} \cdot \hat{n}\right)\Big|_2 \tag{3.24}$$

where $\Big|_1$ and $\Big|_2$ indicate regions 1 and 2 (e.g. cytoplasm region and blood plasma region).

First, in the capillary, vascullar wall, interstitial space, and tissue sub-domains, the gradient exists only in the y-direction at the intersections of both domains (see Figure 3-1). This boundary condition is expressed as:

$$\left(\alpha D_{O_2} \frac{dp_{O_2}}{dy}\right)\Bigg|_1 = -\left(\alpha D_{O_2} \frac{dp_{O_2}}{dy}\right)\Bigg|_2 \tag{3.25}$$

However, across the RBC membrane, the normal derivative is in both the x and y-directions. The boundary condition for this region is

$$\left[(\alpha D_{O_2} \nabla p_{O_2} \cdot \hat{n})\, \delta\left(\vec{x} - \vec{X}(s)\right)\right]\Big|_p = -\left[(\alpha D_{O_2} \nabla p_{O_2} \cdot \hat{n})\, \delta\left(\vec{x} - \vec{X}(s)\right)\right]\Big|_c \quad (3.26)$$

where $[\cdot]_p$ indications the blood plasma region and $[\cdot]_c$ indicates the cytoplasm region. For the detailed, dimensionless treatment of equation (3.26), refer to Appendix E.

3.4 Red Blood Cell Membrane Mechanics and Interactions with the Microcirculation

The RBC membrane is modeled as a segmented beam. It is suspended in the blood plasma and surrounds the cellular cytoplasm. Motion of these surrounding fluids impart momentum to the membrane and the membrane provides a resistance to motion in return. The interaction is a two-way reaction, i.e. the fluid motion affects the RBC membrane motion and the RBC membrane motion affects the motion of the fluid.

In order to capture this interaction, a body force model is used. The RBC membrane is assumed infinitely thin and is massless, thus it acts as a solid boundary which separates two distinct fluid regions. Given that it is extensible (can move and deform), not only must it be integrated into the fluid equations of motion as a body force, but it must also have unique boundary conditions that update its location while enforcing the zero-velocity boundary condition in the reference frame of the local RBC membrane location.

The body force term $\vec{f} = (f_x, f_y)$ is expressed as follows:

$$f_x = \frac{dF_t}{dA}\sin\phi - \frac{dF_n}{dA}\cos\phi \quad (3.27)$$

$$f_y = \frac{dF_t}{dA}\cos\phi + \frac{dF_n}{dA}\sin\phi \quad (3.28)$$

where

$$\frac{dF_t}{dA} = \frac{dt_s}{ds} - q_s \frac{d\phi}{ds} \tag{3.29}$$

$$\frac{dF_n}{dA} = -t_s \frac{d\phi}{ds} - \frac{dq_s}{ds} \tag{3.30}$$

and the constituent relations used to determine t_s and q_s are

$$t_s = \sigma_0 + K\left(\frac{ds}{ds_0}\frac{R}{R_0} - 1\right) - B\frac{d\phi}{ds}\left(\frac{d\phi}{ds} - k_0\right) + \frac{\kappa}{2}\left(\left(\frac{ds}{ds_0}\right)^2 - \left(\frac{R}{R_0}\right)^2\right) \tag{3.31}$$

$$q_s = \frac{dm_s}{ds} = -B\frac{d^2\phi}{ds^2} \tag{3.32}$$

Note that there are no external moments applied to the RBC membrane as it is suspended in the fluid (thus it is unanchored). As such, the net bending moment, modeled as

$$\frac{dM_z}{dA} = \frac{dm_s}{ds} - q_s = 0 \tag{3.33}$$

which yields equation (3.32). Differentiating each quantity with respect to s yields the appropriate necessary terms for the body force quantities mentioned previously.

In order to satisfy the fluid boundary condition along the RBC membrane, the velocity of the local fluid must be equal to the velocity of the local point on the membrane. As mentioned in subsection 3.2.1, this boundary condition is implicitly satisfied for a strong enforcement of the boundary conditions. This statement is mathematically expressed as:

$$X^{n+1} = \left.\frac{\Psi(\vec{x})}{dy}\right|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{3.34}$$

$$Y^{n+1} = -\left.\frac{\Psi(\vec{x})}{dx}\right|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{3.35}$$

For a detailed derivation of the RBC membrane mechanics, refer to Appendix C

and for a detailed derivation of the required RBC membrane mechanical boundary conditions, refer to Appendix E.

3.5 Governing Equations for Blood Microcirculation Model

The governing equations from the previous sections are summarized below for each sub-domain. Note that due to the sub-domain and the varying parameters (as presented in section 3.1), some terms vanish.

The RBC cytoplasm sub-domain, the interior of the RBC:

$$\omega + \nabla^2 \Psi = 0 \tag{3.36}$$

$$\frac{\partial \omega}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial \omega}{\partial y} = \nu \nabla^2 \omega + \nabla \times \vec{f} \tag{3.37}$$

$$\frac{\partial p_{O_2}}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial p_{O_2}}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial p_{O_2}}{\partial y} = D_{O_2}\nabla^2 p_{O_2} + k_{-1}^{Hb}\frac{[Hb]}{\alpha}\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Hb}}\right)^n\right) \tag{3.38}$$

$$\frac{\partial S^{Hb}}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial S^{Hb}}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial S^{Hb}}{\partial y} = D^{Hb}\nabla^2 S^{Hb} - k_{-1}^{Hb}\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Hb}}\right)^n\right) \tag{3.39}$$

The capillary sub-domain, $0 < r < R_c$:

$$\omega + \nabla^2 \Psi = 0 \tag{3.40}$$

$$\frac{\partial \omega}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial \omega}{\partial y} = \nu \nabla^2 \omega + \nabla \times \vec{f} \tag{3.41}$$

$$\frac{\partial p_{O_2}}{\partial t} + \frac{\partial \Psi}{\partial y}\frac{\partial p_{O_2}}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial p_{O_2}}{\partial y} = D_{O_2}\nabla^2 p_{O_2} \tag{3.42}$$

The vascular wall sub-domain, $R_c < r < R_{vw}$:

$$\frac{\partial p_{O_2}}{\partial t} = D_{O_2}\nabla^2 p_{O_2} + M \tag{3.43}$$

The interstitial space sub-domain, $R_{vw} < r < R_{is}$:

$$\frac{\partial p_{O_2}}{\partial t} = D_{O_2}\nabla^2 p_{O_2} \tag{3.44}$$

The muscular tissue sub-domain, $R_{is} < r < R_t$:

$$\frac{\partial p_{O_2}}{\partial t} = D_{O_2}\nabla^2 p_{O_2} + k_{-1}^{Mb}\frac{[Mb]}{\alpha}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Mb}}\right)\right) + M \tag{3.45}$$

$$\frac{\partial S^{Mb}}{\partial t} = D^{Mb}\nabla^2 S^{Mb} - k_{-1}^{Mb}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Mb}}\right)\right) \tag{3.46}$$

It should emphasized that the vascular wall, interstitial space, and muscular tissue sub-domains do not have any governing equations relating to the streamfunction and vorticity as there is no bulk fluid motion in those regions.

Corresponding to Table 3.1, the capillary, vascular wall, and interstitial space sub-domains do not have a saturation equation for either of the oxyhemoglobin or oxymyoglobin protein complexes as they are not usually found in these regions. While free hemoglobin molecules can exist in blood plasma, this model assumes their affects to be negligible in comparison to the transport of free oxygen within the blood plasma. Also, the capillary domain does not have myoglobin proteins, nor does the muscular tissue have hemoglobin proteins, thus no corresponding saturation transport equation exists in these sub-domains by definition.

3.5.1 Non-dimensionalization of Governing Equations

This subsection summarizes the dimensional governing equations presented previously. They are now cast into their final non-dimensional form for code implementation. As can be seen, the non-dimensional parameters at play reveal themselves, allowing for a non-dimensional characterization of the physics to be performed.

The RBC cytoplasm sub-domain, the interior of the RBC:

$$\omega^* + \nabla^{*2}\Psi^* = 0 \tag{3.47}$$

$$\frac{\partial \omega^*}{\partial t^*} + \frac{\partial \Psi^*}{\partial y^*}\frac{\partial \omega^*}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial \omega^*}{\partial y^*} = \frac{1}{Re}\nabla^{*2}\omega + \nabla^* \times \vec{f^*} \tag{3.48}$$

$$\frac{\partial p^*_{O_2}}{\partial t^*} + \frac{\partial \Psi^*}{\partial y^*}\frac{\partial p^*_{O_2}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial p^*_{O_2}}{\partial y^*} = \frac{1}{Pe_{O_2}}\nabla p^*_{O_2} + Da_{Hb}\left(\frac{[Hb]}{\alpha p^{Hb}_{O_2 50\%}}\right)\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(p^*_{O_2}\right)^n\right) \tag{3.49}$$

$$\frac{\partial S^{Hb}}{\partial t^*} + \frac{\partial \Psi^*}{\partial y^*}\frac{\partial S^{Hb}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial S^{Hb}}{\partial y^*} = \frac{1}{Pe_{Hb}}\nabla^{*2}S^{Hb} - Da_{Hb}\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(p^*_{O_2}\right)^n\right) \tag{3.50}$$

The capillary sub-domain, $0 < r < R^*_c$:

$$\omega^* + \nabla^{*2}\Psi^* = 0 \tag{3.51}$$

$$\frac{\partial \omega^*}{\partial t^*} + \frac{\partial \Psi^*}{\partial y^*}\frac{\partial \omega^*}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial \omega^*}{\partial y^*} = \frac{1}{Re}\nabla^{*2}\omega^* + \nabla^* \times \vec{f^*} \tag{3.52}$$

$$\frac{\partial p^*_{O_2}}{\partial t^*} + \frac{\partial \Psi^*}{\partial y^*}\frac{\partial p^*_{O_2}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial p^*_{O_2}}{\partial y^*} = \frac{1}{Pe_{O_2}}\nabla^{*2}p^*_{O_2} \tag{3.53}$$

The vascular wall sub-domain, $R^*_c < r < R^*_{vw}$:

$$\frac{\partial p^*_{O_2}}{\partial t^*} = \frac{1}{Pe_{O_2}} \nabla^{*2} p^*_{O_2} - M^* \tag{3.54}$$

The interstitial space sub-domain, $R^*_{vw} < r < R^*_{is}$:

$$\frac{\partial p^*_{O_2}}{\partial t^*} = \frac{1}{Pe_{O_2}} \nabla^{*2} p^*_{O_2} \tag{3.55}$$

The muscular tissue sub-domain, $R^*_{is} < r < R^*_t$:

$$\frac{\partial p^*_{O_2}}{\partial t^*} = \frac{1}{Pe_{O_2}} \nabla^{*2} p^*_{O_2} + Da_{Mb} \left(\frac{[Mb]}{\alpha p^{Mb}_{O_2 50\%}} \right) r \left(S^{Mb} - \left(1 - S^{Mb}\right) \left(r p^*_{O_2}\right) \right) - r M^* \tag{3.56}$$

$$\frac{\partial S^{Mb}}{\partial t} = \frac{1}{Pe_{Mb}} \nabla^{*2} S^{Mb} - Da_{Mb} \left(S^{Mb} - \left(1 - S^{Mb}\right) \left(r p^*_{O_2}\right) \right) \tag{3.57}$$

3.5.2 Non-dimensional Parameters

As seen in this chapter, the governing equations have a set of dimensionless parameters pertaining to the desired physics in the problem. For convenience, these parameters are listed in Table 3.4 with a brief description of what they measure.

Parameter	Definition	Description
Reynolds Number (Re)	$\frac{UL}{\nu}$	$\frac{\text{Inertial Forces}}{\text{Viscous Forces}}$
Pêclet Number (Pe)	$\frac{UL}{D}$	$\frac{\text{Advection}}{\text{Diffusion}}$
Damköhler Number (Da)	$\frac{k_{-1}U}{L}$	$\frac{\text{Reaction Rate}}{\text{Passage Rate}}$
Concentration Number	$\frac{[\cdot]}{\alpha p^{\cdot}_{O_2 50\%}}$	$\frac{\text{Saturated Concentration}}{\text{Free Oxygen Concentration}}$
Mass Consumption Number	$\frac{M(L/U)}{\alpha p^{Hb}_{O_2 50\%}}$	$\frac{\text{Oxygen Consumption}}{\text{Oxygen in Equilibrium}}$
Partial Pressure Ratio (r)	$\frac{p^{Mb}_{O_2 50\%}}{p^{Hb}_{O_2 50\%}}$	$\frac{\text{Free Oxygen in Equilibrium with Oxymyoglobin}}{\text{Free Oxygen in Equilibrium with Oxyhemoglobin}}$

Table 3.4: Dimensionless parameters to be studied in the simulations.

Chapter 4

Implementation of Numerical Model

4.1 Immersed Boundary Method

The immersed boundary method (IBM) is a numerical technique used to approximate the effects of an immersed boundary in a fluid[11]. Its development largely stemmed from the need to simulate biomembranes suspended inside a blood microcirculation flow. While it has practical applications elsewhere, its original intent was for the use in extensible membrane problems.

For background, the IBM is a simplified approach (in terms of coding) to deforming boundary problems. It uses two types of meshes: a static (Eulerian) mesh and a dynamic (Lagrangian) mesh. The former is the mesh in which all the calculations take place, and the latter is "sampled" onto the former mesh. The Lagrangian mesh acts as an immersed boundary in which it provides a feedback mechanism to the flow field lying underneath it on the Eulerian mesh, thus acting as an effective boundary to the underlying fluid.

In the context of red blood cells, the membrane is modeled as an immersed boundary (the Lagrangian mesh) and is extensible. It also affects the motion of the underlying fluid. The sampling technique used in capturing of this coupled interaction is the hallmark feature of the method.

In order to couple the interactions, a discretized delta function, $\delta(r)$, must be developed in order to properly sample the membrane body forces, $\vec{F}$, acting upon the

fluid, $\vec{f}$. This coupling is done in the following way:

$$\vec{f}(\vec{x}(t)) = \int \vec{F}(\vec{X}(t))\delta\left(\vec{x}(t) - \vec{X}(t)\right) d\vec{X} \tag{4.1}$$

where $\vec{x}(t)$ represents the Eulerian mesh points at time-varying locations and $\vec{X}(t)$ represents the Lagrangian mesh points at time-varying locations.

The delta function, $\delta(r)$, originally developed by Peskin, but presented here from Lai and Peskin[17], is expressed in the following way:

$$\delta(r) = \begin{cases} \frac{1}{8h}\left(3 - 2\frac{|r|}{h} + \sqrt{1 + 4\frac{|r|}{h} - 4\left(\frac{|r|}{h}\right)^2}\right) & |r| \leq h \\ \frac{1}{8h}\left(5 - 2\frac{|r|}{h} - \sqrt{-7 + 12\frac{|r|}{h} - 4\left(\frac{|r|}{h}\right)^2}\right) & h < |r| \leq 2h \\ 0 & |r| \geq 2 \end{cases}$$

It is simply the discretized variant of the Dirac delta function. Consequently, because this function also allows for the derivative of the solution to be continuous, it is only first-order accurate (for more on the accuracy, see Lai and Peskin[17]).

Expressed in a more tractable form for computations, equation (4.1) can be discretized into

$$\vec{f}(\vec{x}(t)) = \sum_s \vec{F}(\vec{X}(s,t))\delta\left(\vec{x}(t) - \vec{X}(s,t)\right)\Delta s \tag{4.2}$$

or for a particular time-step n

$$\vec{f}^n(\vec{x}) = \sum_s \vec{F}^n(\vec{X}^n(s))\delta\left(\vec{x}(t) - \vec{X}^n(s)\right)\Delta s \tag{4.3}$$

Equation (4.3) is a matrix equation that samples the Lagrangian body force vector $\vec{F}$ and smears the effects onto the Eulerian mesh via the Eulerian body force vector, $\vec{f}$. In the context of the code, this is an explicitly calculated (time-step n) sub-routine, which provides for a simpler implementation and reduced runtime.

4.2 Newton-Raphson Iteration Solver

The solver of choice used in this work is the iterative Newton's Method. It is a zero-finding algorithm (root solver) that iterates over a system of residual equations, driving them to zero by systematically adjusting the values of the dependent variables at each iteration step. Newton's Method also has the property of quadratic convergence and is a very robust algorithm, thus being a suitable solver.

Mathematically, the residual function is defined as

$$R(x) = 0 \tag{4.4}$$

where x is a single variable.

Equation (4.4) is known from the governing equation(s) of the system a priori. Essentially, all terms in the governing equation(s) are moved to one side of the equation, equaling 0, and then the residual $R(x)$ is defined to be that resulting equation.

In order to iterate, the derivative of equation (4.4) is first taken and is multiplied by a Δx:

$$\frac{\partial R(x)}{\partial x}\Delta x = R'(x)\Delta x = 0 \tag{4.5}$$

Numerically, an initial guess for the roots of the function $R(x)$ is made, x_0. From here, the necessary Δx can be solved for by adding equations (4.4) and (4.5) and rearranging:

$$\Delta x = -\frac{R(x_0)}{R'(x_0)} \tag{4.6}$$

From here, the next x can be solved for: $x_1 = x_0 + \Delta x$. Note that this x_1 may not satisfy equation (4.4) and will require multiple iterations for convergence. These multiple iterations occur for non-linear problems, such as those encountered in fluid mechanics, as the residual is linearized about a point x_k. Newton's Method iterates in linear steps from x_k to x_{k+1}, thus requiring potentially new linearizations about the successively iterated solutions. In general, for any Newton step k, the $k+1$ step

can be solved for by

$$x_{k+1} = x_k - \frac{R(x_k)}{R'(x_k)} \tag{4.7}$$

Generally, a non-linear problem requires multiple iterations to converge to an accurate solution that satisfies equation (4.4) within a user-chosen tolerance. It is possible for a non-linear residual function to be satisfied in one iteration, but only if the initial guess is already close to the actual solution.

4.3 Domain Discretization

The problem domain (see Figure 3-1) must be discretized for the numerical simulation to take place. The Eulerian mesh chosen is a non-uniform, rectilinear mesh.

The reason for choosing a rectilinear mesh is two-fold. First, it is relatively simple to implement and keep track of point data (e.g. grid location and variable values). Second, no extensive modifications need to be made to the sampling delta function $\delta(r)$, thus the immersed boundary method can still be used.

Near non-periodic domain boundaries and non-symmetric domain boundaries, the grid must contract (i.e. become progressively more dense) in order to better resolve numerical boundary layers. For example, in fluid simulations, boundary layers form near solid boundaries and form high regions of fluid shear. Thus, the value of the local derivative of the velocity field near these solid boundaries can increase to large values, requiring higher resolution to more accurately resolve the velocity values and not over or underestimate the values of their derivatives. Figure 4-1 displays the contraction of the mesh near the capillary edge at the top of the capillary sub-domain. Note that no contraction occurs on the left or right edges due to the domain periodicity and no contraction occurs on the bottom edge due to the domain being axially symmetric about the length-axis.

For the remaining sub-domains (vascular wall, interstitial space, and muscular tissue), the boundary interfaces between them are similarly contracted, while the left and right sub-domain edges are left with uniform grid spacing since axial periodicity still applies.

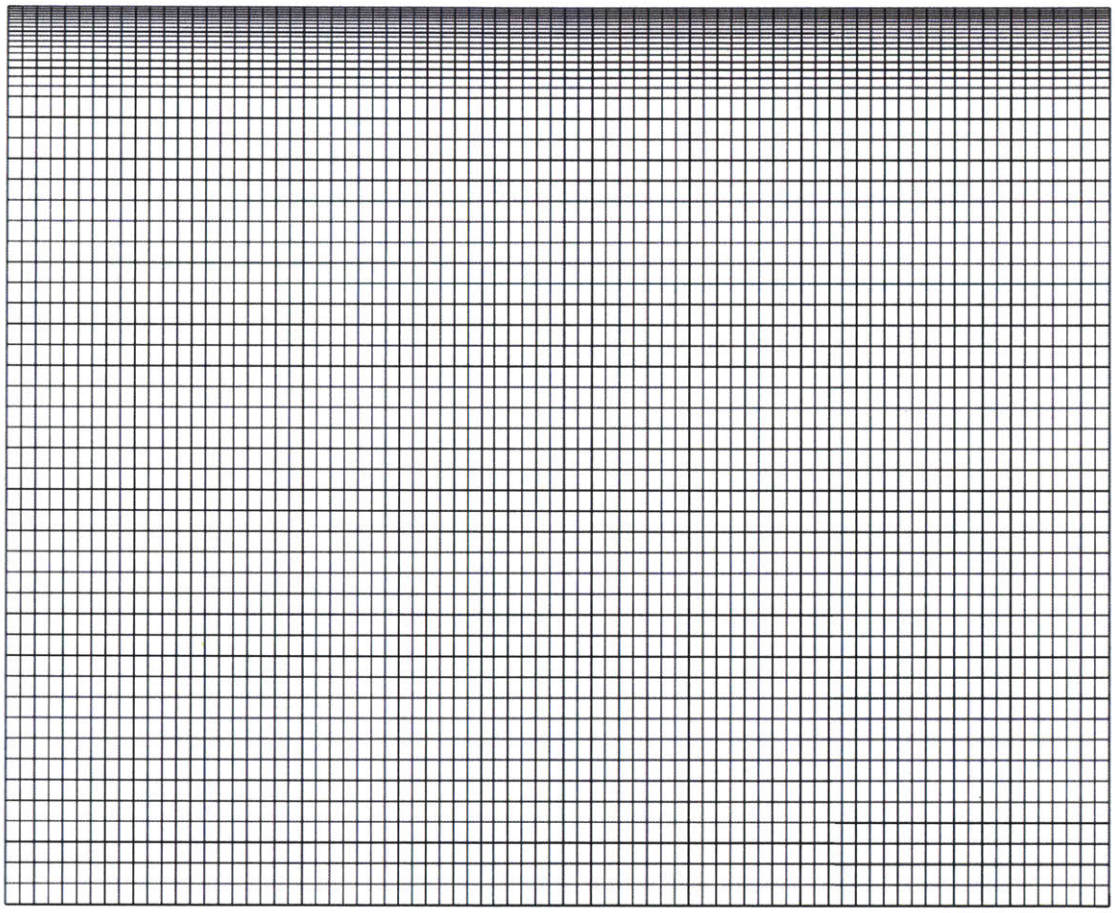


Figure 4-1: Non-uniform rectilinear grid for capillary sub-domain (unscaled). Notice that the top capillary edge has a gradually denser mesh to account for the boundary layer formation.

The Lagrangian RBC membrane mesh is made up of only a curve of points which obey the constituent relations derived in Appendix C. They are overlaid upon the Eulerian mesh and sampled according to the delta function shown in section 4.1. The Lagrangian mesh is shown in Figure 4-2 with the Lagrangian mesh points shown in red and the Eulerian mesh shown in black. (This mesh is the starting configuration for a RBC, which is a semi-circle.)

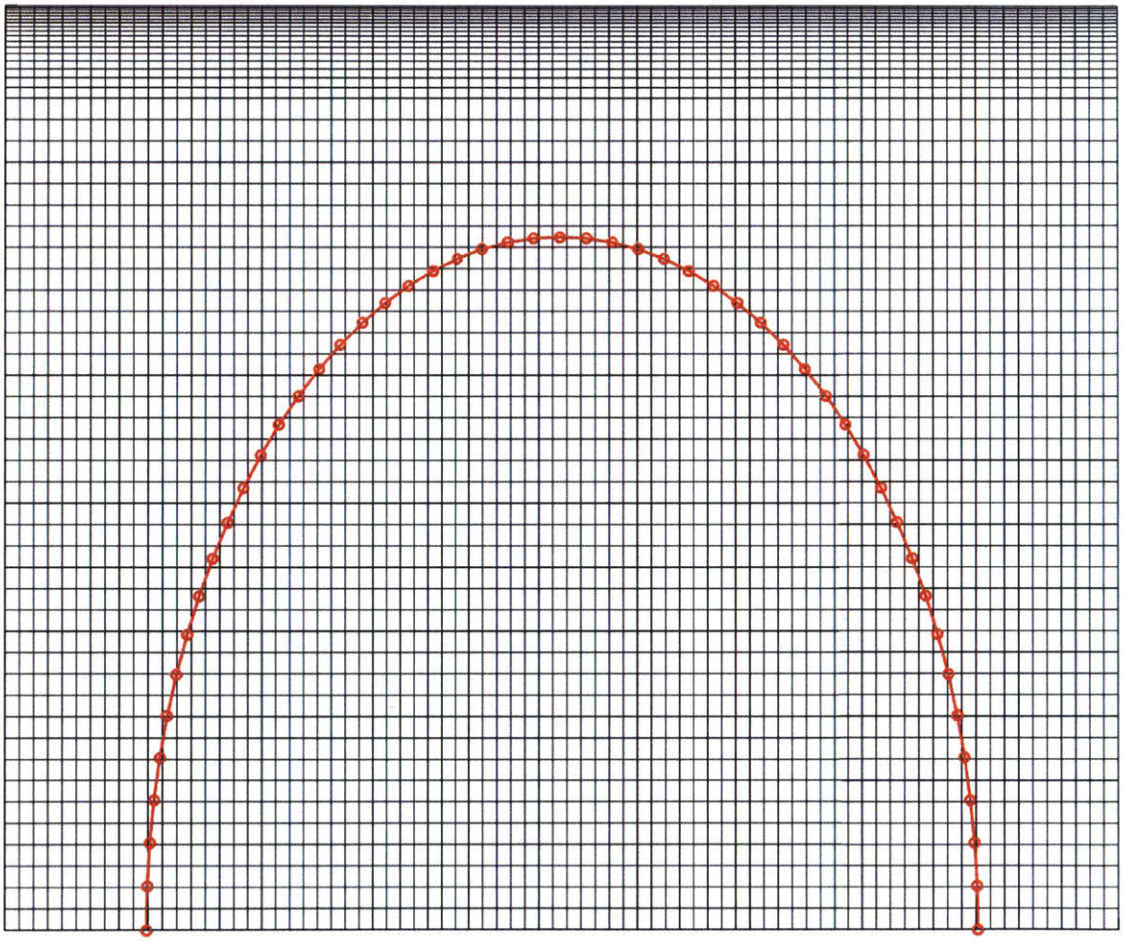


Figure 4-2: Lagrangian mesh (shown in red with circles highlighting individual mesh points) overlaying the Eulerian mesh (shown in black).

4.4 Finite Difference Modeling of Non-Uniform Grids

Finite difference modeling typically involves the Taylor expansion of a function about a set of points. The expansions are then summed, canceling out terms, leaving

a stencil used for numerical approximations. The most common stencils seen is that for the approximation of the slope of a line about a point i:

$$\frac{\partial f}{\partial x} = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} \tag{4.8}$$

Similarly, the slope of a line may also be calculated by taking a point behind x_i as follows:

$$\frac{\partial f}{\partial x} = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} = \frac{f(x_i) - f(x_{i-1})}{\Delta x} \tag{4.9}$$

The slope of this same line can also be approximated to higher order by using a point $i+1$ and $i-1$:

$$\frac{\partial f}{\partial x} = \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} \tag{4.10}$$

where $\Delta x = (x_{i+1} - x_i) = (x_i - x_{i-1})$. These equations, which are intuitive to understand, can also be approximated by the Taylor series expansion about the points x_{i-1}, x_i, and x_{i+1} and appropriate summations.

However, these formulas do not readily work to higher orders of accuracy for non-uniform grids (meshes) as the Δx in the denominators are no longer equivalent. Also, in the case of equation (4.10), $f(x_i)$ will no longer have a weighting coefficient equal to 0 (instead of 1 or -1), thus the value of $f(x_i)$ will be required in order to approximate the derivative. This work circumvents this issue by using Lagrange basis polynomials for the derivation of the desired stencils on a non-uniform mesh.

4.4.1 Lagrange Basis Polynomials for Finite Differentiation

Functions can be locally approximated by a polynomial. How this approximation is carried out can be done multiple ways, one of which is via the use of Lagrange basis polynomials. To start, the polynomial is defined as

$$f(x_i) \approx L(x_i) = \sum_{i=1}^{N} f(x_i) l_i(x) \tag{4.11}$$

where $l_i(x)$ is a Lagrange basis polynomial:

$$l_i(x) = \prod_{\substack{m=1 \\ m \neq i}}^{N} \frac{x - x_m}{x_i - x_m} \tag{4.12}$$

The Lagrange polynomial can be extended and differentiated out for N points x_i in order to give the desired stencils for numerical approximation. The advantage of this procedure is that the weights for each point x_i are retained until the grid spacing between points (i.e. the Δx for each grid cell) is defined. Once defined, the weights are solved for based upon the user-desired mesh resolution and spacing. The full derivation and explanation of this process and the resulting approximations used (of which they are boxed) can be seen in Appendix D.

4.5 Discretization of Governing Equations

The governing equations are discretized according to a traditional forward-time centered-space (FTCS) finite differencing scheme with a second-order upwinding of the first-derivative terms. This discretization scheme is a commonly employed scheme, has been extensively studied, and is also used as a common teaching example in graduate level numerical methods courses, leading to it becoming the discretization scheme of choice.

The spatial stencils used for the all of the required derivative approximations are boxed in Appendix D. Note that the first-derivative approximation for upwinding involves the velocity terms such that

$$c\frac{\partial u}{\partial x} = \begin{cases} c\frac{\partial u}{\partial x}\Big|_f, & c < 0 \\ c\frac{\partial u}{\partial x}\Big|_b, & c \geq 0 \end{cases}$$

where c is the local grid speed, u is the generic solution variable, $\Big|_f$ indicates a forward differencing, and $\Big|_b$ indicates and backwards differencing (whose stencils are, once again, boxed in Appendix D).

Where required, depending on the local RBC mesh location, forwards and backwards differencing of the first- and second-derivatives may be strongly enforced,

regardless of upwinding, to ensure that the diffusion being approximated is physically realistic. These instances will be explained in more detail in section 4.6.3.

The governing equations are discretized temporally by a semi-implicit forward Euler scheme, where the linear terms (e.g. diffusion) are implicit in time and the non-linear terms are explicit (e.g. advection, body force, reaction rate) in time. All constants are also treated explicitly. The implicit terms are denoted by $[\cdot]\Big|^{n+1}$ and the explicit terms are denoted by $[\cdot]\Big|^{n}$ (note that the Hill coefficient n is unchanged inside the reaction rate terms, and it does not denote a time-step).

For the RBC cytoplasm sub-domain, in the interior of the RBC:

$$\left[\omega^* + \nabla^{*2}\Psi^*\right]\Big|^{n+1} = 0 \tag{4.13}$$

$$\frac{\partial \omega^*}{\partial t^*} + \left[\frac{\partial \Psi^*}{\partial y^*}\frac{\partial \omega^*}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial \omega^*}{\partial y^*}\right]\Big|^{n} = \left[\frac{1}{Re}\nabla^{*2}\omega\right]\Big|^{n+1} + \left[\nabla^* \times \vec{f^*}\right]\Big|^{n} \tag{4.14}$$

$$\begin{aligned}\frac{\partial p^*_{O_2}}{\partial t^*} + \left[\frac{\partial \Psi^*}{\partial y^*}\frac{\partial p^*_{O_2}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial p^*_{O_2}}{\partial y^*}\right]\Big|^{n} &= \left[\frac{1}{Pe_{O_2}}\nabla p^*_{O_2}\right]\Big|^{n+1} \\ &+ \left[Da_{Hb}\left(\frac{[Hb]}{\alpha p^{Hb}_{O_2 50\%}}\right)\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(p^*_{O_2}\right)^n\right)\right]\Big|^{n}\end{aligned} \tag{4.15}$$

$$\begin{aligned}\frac{\partial S^{Hb}}{\partial t^*} + \left[\frac{\partial \Psi^*}{\partial y^*}\frac{\partial S^{Hb}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial S^{Hb}}{\partial y^*}\right]\Big|^{n} &= \left[\frac{1}{Pe_{Hb}}\nabla^{*2}S^{Hb}\right]\Big|^{n+1} \\ &- \left[Da_{Hb}\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(p^*_{O_2}\right)^n\right)\right]\Big|^{n}\end{aligned} \tag{4.16}$$

For the capillary sub-domain, $0 < r < R_c$:

$$\left[\omega^* + \nabla^{*2}\Psi^*\right]\Big|^{n+1} = 0 \tag{4.17}$$

$$\frac{\partial \omega^*}{\partial t^*} + \left[\frac{\partial \Psi^*}{\partial y^*}\frac{\partial \omega^*}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial \omega^*}{\partial y^*}\right]\Bigg|^n = \left[\frac{1}{Re}\nabla^{*2}\omega^*\right]\Bigg|^{n+1} + \left[\nabla^* \times \vec{f}^*\right]\Bigg|^n \tag{4.18}$$

$$\frac{\partial p^*_{O_2}}{\partial t^*} + \left[\frac{\partial \Psi^*}{\partial y^*}\frac{\partial p^*_{O_2}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial p^*_{O_2}}{\partial y^*}\right]\Bigg|^n = \left[\frac{1}{Pe_{O_2}}\nabla^{*2}p^*_{O_2}\right]\Bigg|^{n+1} \tag{4.19}$$

For the vascular wall sub-domain, $R_c < r < R_{vw}$:

$$\frac{\partial p^*_{O_2}}{\partial t^*} = \left[\frac{1}{Pe_{O_2}}\nabla^{*2}p^*_{O_2}\right]\Bigg|^{n+1} - [M^*]\Big|^n \tag{4.20}$$

For the interstitial space sub-domain, $R_{vw} < r < R_{is}$:

$$\frac{\partial p^*_{O_2}}{\partial t^*} = \left[\frac{1}{Pe_{O_2}}\nabla^{*2}p^*_{O_2}\right]\Bigg|^{n+1} \tag{4.21}$$

For the muscular tissue sub-domain, $R_{is} < r < R_t$:

$$\begin{aligned}\frac{\partial p^*_{O_2}}{\partial t^*} = &\left[\frac{1}{Pe_{O_2}}\nabla^{*2}p^*_{O_2}\right]\Bigg|^{n+1} \\ &+ \left[Da_{Mb}\left(\frac{[Mb]}{\alpha p^{Mb}_{O_2 50\%}}\right) r\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(rp^*_{O_2}\right)\right)\right]\Bigg|^n - [rM^*]\Big|^n\end{aligned} \tag{4.22}$$

$$\frac{\partial S^{Mb}}{\partial t} = \left[\frac{1}{Pe_{Mb}}\nabla^{*2}S^{Mb}\right]\Bigg|^{n+1} - \left[Da_{Mb}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(rp^*_{O_2}\right)\right)\right]\Big|^n \tag{4.23}$$

4.5.1 Red Blood Cell Body Forces:

In order to calculate the constituent relations for the body force term $\vec{f}^*$, the geometry from the Lagrangian mesh is needed. A schematic of the RBC geometry is shown in Figure 4-3.

Recapitulating the body force vector and constituent relations for the purpose of

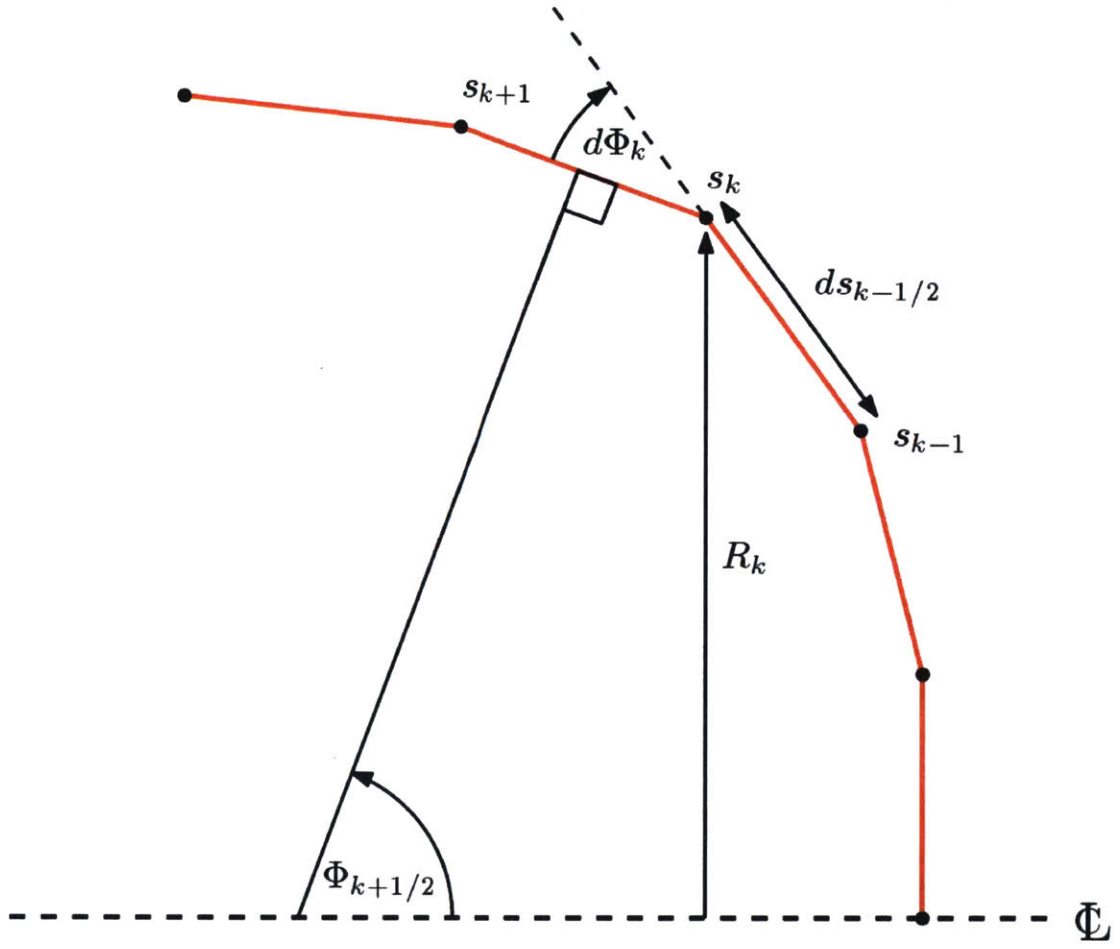


Figure 4-3: Red blood cell mesh geometry for calculation of constituent relations, visually defining all necessary variables.

clarity (for their derivations, refer to Appendix C):

$$
\begin{aligned}
f_x^* &\equiv \frac{f_x}{\rho U^2} = \left(\frac{dt_s^*}{ds^*} - q_s^*\frac{d\phi}{ds^*}\right)\sin(\phi) - \left(-t_s^*\frac{d\phi}{ds^*} - \frac{dq_s^*}{ds^*}\right)\cos(\phi) \\
f_y^* &\equiv \frac{f_y}{\rho U^2} = \left(\frac{dt_s^*}{ds^*} - q_s^*\frac{d\phi}{ds^*}\right)\cos(\phi) + \left(-t_s^*\frac{d\phi}{ds^*} - \frac{dq_s^*}{ds^*}\right)\sin(\phi) \\
t_s^* &= \sigma_0^* + K^*\left(\frac{ds^*}{ds_0^*}\frac{R^*}{R_0^*} - 1\right) - B^*\frac{d\phi}{ds^*}\left(\frac{d\phi}{ds^*} - k_0^*\right) + \frac{\kappa^*}{2}\left(\left(\frac{ds^*}{ds_0^*}\right)^2 - \left(\frac{R^*}{R_0^*}\right)^2\right) \\
\frac{dt_s^*}{ds^*} &= K^*\left(\frac{d}{ds^*}\left(\frac{ds^*}{ds_0^*}\right)\frac{R^*}{R_0^*} + \frac{d}{ds^*}\left(\frac{R^*}{R_0^*}\right)\frac{ds^*}{ds_0^*}\right) \\
&\qquad - B^*\frac{d^2\phi}{ds^{*2}}\left(2\frac{d\phi}{ds^*} - k_0^*\right) + \frac{\kappa^*}{2}\left(\frac{d}{ds^*}\left(\frac{ds^*}{ds_0^*}\right)^2 - \frac{d}{ds^*}\left(\frac{R^*}{R_0^*}\right)^2\right) \\
q_s^* &= -B^*\frac{d^2\phi}{ds^{*2}} \\
\frac{dq_s^*}{ds^*} &= -B^*\frac{d^3\phi}{ds^{*3}}
\end{aligned}
$$

As can be seen, the constituent relations depend only upon the RBC geometry and predetermined constant coefficients.

First, the definition of ϕ and the stencils for the derivatives of ϕ must be shown. First, ϕ is defined from the geometry as

$$\phi_{k+1/2} = \arccos\left(\frac{R_{k+1}^* - R_k^*}{ds_{k+1/2}^*}\right) \tag{4.24}$$

where upon a line-element weighted average

$$\phi_k = \frac{\phi_{k+1/2}ds_{k-1/2}^* + \phi_{k-1/2}ds_{k+1/2}^*}{ds_{k+1/2}^* + ds_{k-1/2}^*} \tag{4.25}$$

Using a combination of the midpoint formula and the line-element weighted average (as shown in equation (4.25)), the other derivatives for ϕ can also be shown as:

$$\left.\frac{d\phi}{ds^*}\right|_k = \frac{\phi_{k+1/2} - \phi_{k-1/2}}{\left(ds_{k+1/2}^* + ds_{k-1/2}^*\right)/2} \tag{4.26}$$

$$\left.\frac{d^2\phi}{ds^{*2}}\right|_k = \frac{\left.\frac{d^2\phi}{ds^{*2}}\right|_{k+1/2} ds^*_{k-1/2} + \left.\frac{d^2\phi}{ds^{*2}}\right|_{k-1/2} ds^*_{k+1/2}}{ds^*_{k+1/2} + ds^*_{k-1/2}} \tag{4.27}$$

$$\left.\frac{d^3\phi}{ds^{*3}}\right|_k = \frac{\left.\frac{d^2\phi}{ds^{*2}}\right|_{k+1/2} - \left.\frac{d^2\phi}{ds^{*2}}\right|_{k-1/2}}{\left(ds^*_{k+1/2} + ds^*_{k-1/2}\right)/2} \tag{4.28}$$

The remaining terms that are needed for the constituent relations are listed below in equations (4.29) through (4.34). Once again, a combination of line-element weighted averaging and the midpoint formula is used.

$$\left.\frac{ds^*}{ds^*_0}\right|_k = \frac{\frac{ds^*_{k+1/2}}{ds^*_{0,k+1/2}} ds^*_{k-1/2} + \frac{ds^*_{k-1/2}}{ds^*_{0,k-1/2}} ds^*_{k+1/2}}{ds^*_{k+1/2} + ds^*_{k-1/2}} \tag{4.29}$$

$$\left.\frac{R^*}{R^*_0}\right|_k = \frac{R^*_k}{R^*_{0,k}} \tag{4.30}$$

$$\left.\frac{d}{ds^*}\left(\frac{ds^*}{ds^*_0}\right)\right|_k = \frac{\frac{ds^*_{k+1/2}}{ds^*_{0,k+1/2}} - \frac{ds^*_{k-1/2}}{ds^*_{0,k-1/2}}}{\left(ds^*_{k+1/2} + ds^*_{k-1/2}\right)/2} \tag{4.31}$$

$$\left.\frac{d}{ds^*}\left(\frac{R^*}{R^*_0}\right)\right|_k = \frac{\frac{R^*_{k+1} + R^*_k}{R^*_{0,k+1} + R^*_{0,k}} - \frac{R^*_k + R^*_{k-1}}{R^*_{0,k} + R^*_{0,k-1}}}{\left(ds^*_{k+1/2} + ds^*_{k-1/2}\right)/2} \tag{4.32}$$

$$\left.\frac{d}{ds^*}\left(\frac{ds^*}{ds^*_0}\right)^2\right|_k = \frac{\left(\frac{ds^*_{k+1/2}}{ds^*_{0,k+1/2}}\right)^2 - \left(\frac{ds^*_{k-1/2}}{ds^*_{0,k-1/2}}\right)^2}{\left(ds^*_{k+1/2} + ds^*_{k-1/2}\right)/2} \tag{4.33}$$

$$\left.\frac{d}{ds^*}\left(\frac{R^*}{R^*_0}\right)^2\right|_k = \frac{\left(\frac{R^*_{k+1} + R^*_k}{R^*_{0,k+1} + R^*_{0,k}}\right)^2 - \left(\frac{R^*_k + R^*_{k-1}}{R^*_{0,k} + R^*_{0,k-1}}\right)^2}{\left(ds^*_{k+1/2} + ds^*_{k-1/2}\right)/2} \tag{4.34}$$

Now, substituting these terms into the constituent relations, and the constituent

relations into the body force components f_x^* and f_y^*, completes the calculation for the RBC body force interaction at time-step n. This body force calculation is done explicitly in the semi-implicit scheme, which is consistent with the calculation of the other non-linear terms in the governing equations.

4.6 Boundary Conditions and Interface Points

This section treats the boundary conditions for the entirety of the problem domain. It will recapitulate the physics of the governing equations as well as discretize the governing equations for use in the IBM.

4.6.1 Periodic Boundary Conditions Along Domain Inlet and Outlet

The inlet and outlet of the domain (left and right sides, respectively) are periodic boundary conditions.

The inlet boundary condition takes data from the outlet boundary and preceding points in the x-direction, as necessary, for advective-upwinding effects and diffusion effects (the latter being dependent on the RBC membrane position, discussed in section 4.7.2).

The outlet boundary values are set equal to the inlet values via residual solving, ensuring that the inlet and outlet are equal.

4.6.2 Flux Boundary Conditions Along Sub-Domain Interfaces

The interfacial boundaries between the capillary centerline, capillary, vascular wall, interstitial space, muscular tissue, and muscular tissue edge form two sets of boundary conditions. The two sets are related to the kinematics of the fluid (e.g. centerline streamline with zero vorticity, no-slip streamline at capillary edge) and the oxygen transport (e.g. oxygen flux across interfacial boundaries). The capillary sub-domain employs both of these sets of boundary conditions while the remaining sub-domains only employ the oxygen transport boundary conditions. (Note that the

RBC membrane in the capillary has its own specific boundary conditions - see section 4.6.3 for details.)

Capillary Sub-Domain, $0 < r < R_c$:

As briefly mentioned previously, the capillary domain (excluding the RBC membrane) has two sets of boundary conditions. For the streamfunction Ψ^*

$$\Psi^*\Big|_{y*=0} = 0 \tag{4.35}$$

$$\Psi^*\Big|_{y*=1} = 1 \tag{4.36}$$

and for the vorticity ω^*

$$\omega^*\Big|_{y*=0} = 0 \tag{4.37}$$

$$\omega^*\Big|_{y*=1} = -\frac{\partial^2 \Psi^*}{\partial y^{*2}} \text{ , where } \frac{\partial \Psi^*}{\partial y^*} = 0 \tag{4.38}$$

The latter boundary condition where $\omega^*\Big|_{y^*=1}$ has $\frac{\partial \Psi^*}{\partial y^*} = 0$ is enforced via the ghost point method. This implementation will be explained in section 4.7.2.

The oxygen transport boundary conditions for the oxygen partial pressure $p^*_{O_2}$ are, where $\hat{n} = \hat{y}$:

$$\frac{\partial p^*_{O_2}}{\partial y^*}\Big|_{y^*=0} = 0 \tag{4.39}$$

$$\left(\frac{\alpha_p}{\alpha_{vw}}\right) \frac{1}{Pe_p} \frac{\partial p^*_{O_2}}{\partial y^*}\Big|_{y^*=1} = -\frac{1}{Pe_{vw}} \frac{\partial p^*_{O_2}}{\partial y^*}\Big|_{y^*=1} \tag{4.40}$$

Recall that the saturation is defined to be zero in the capillary sub-domain as no hemoglobin or myoglobin are assumed to exist here.

Vascular Wall Sub-Domain, $R_c < r < R_{vw}$:

This sub-domain only contains free oxygen transport, thus the saturation is once again defined to be zero as no hemoglobin or myoglobin are present. With that fact stated, the flux interface boundary conditions are:

$$\left(\frac{\alpha_p}{\alpha_{vw}}\right)\frac{1}{Pe_p}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=1} = -\frac{1}{Pe_{vw}}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=1} \tag{4.40}$$

$$\left(\frac{\alpha_{vw}}{\alpha_{is}}\right)\frac{1}{Pe_{vw}}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{vw}/R_{cap}} = -\frac{1}{Pe_{is}}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{vw}/R_{cap}} \tag{4.41}$$

Interstitial Space Sub-Domain, $R_{vw} < r < R_{is}$:

Similar to the vascular wall sub-domain, this sub-domain also contains only free oxygen transport. Thus, the saturation is once again zero. The interface boundary conditions similarly are:

$$\left(\frac{\alpha_{vw}}{\alpha_{is}}\right)\frac{1}{Pe_{vw}}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{vw}/R_{cap}} = -\frac{1}{Pe_{is}}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{vw}/R_{cap}} \tag{4.41}$$

$$\left(\frac{\alpha_{is}}{\alpha_m}\right)\frac{1}{Pe_{is}}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{is}/R_{cap}} = -\frac{1}{Pe_m}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{is}/R_{cap}} \tag{4.42}$$

Muscular Tissue Sub-Domain, $R_{is} < r < R_t$:

The final sub-domain contains both free oxygen transport as well as the oxygen-binding protein myoglobin. Therefore, boundary conditions for both the oxygen partial pressure and the myoglobin saturation are needed. The oxygen partial pressure interface boundary conditions are:

$$\left(\frac{\alpha_{is}}{\alpha_t}\right)\frac{1}{Pe_{is}}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{is}/R_{cap}} = -\frac{1}{Pe_t}\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_{is}/R_{cap}} \tag{4.42}$$

$$\frac{\partial p^*_{O_2}}{\partial y^*}\bigg|_{y^*=R_t/R_{cap}} = 0 \tag{4.43}$$

with a no flux boundary condition at the edge of the muscular tissue sub-domain. The myglobin saturation boundary conditions are simple no-flux boundary conditions at both interface edge locations:

$$\left.\frac{\partial S^{Mb}}{\partial y^*}\right|_{y^*=R_{is}/R_{cap}} = 0 \tag{4.44}$$

$$\left.\frac{\partial S^{Mb}}{\partial y^*}\right|_{y^*=R_t/R_{cap}} = 0 \tag{4.45}$$

4.6.3 Lagrangian Mesh: Membrane Boundary Conditions

This subsection treats the details of the RBC membrane boundary conditions as mentioned in subsection 4.6.2. The physics will be recapitulated and mathematically discretized for use in the IBM.

No-Slip on RBC Boundary

When a fluid comes into contact with a solid boundary (such as a wall or membrane), the fluid comes to rest with respect to the solid boundary. For instance, in the case of the stationary capillary wall, the velocity of the blood plasma comes to zero. In the case of the membrane, the blood plasma will move at the speed at which the local membrane point is moving through the plasma. In essence, the fluid moves at the speed at which the solid boundary is moving.

Mathematically, this is expressed for the membrane as

$$\frac{d\vec{X}(t)}{dt} = \vec{v}(\vec{X}(t)) \tag{4.46}$$

where $\vec{X}(t)$ represents the local membrane location and $\vec{v}(\vec{X}(t))$ represents the underlying blood plasma velocity at that $\vec{X}(t)$ location.

In the component form of the streamfunction-vorticity format, the vector equation (4.46) takes the following form:

$$\frac{dX(t)}{dt} = \frac{d\Psi(\vec{X}(t))}{dy} \tag{4.47}$$

$$\frac{dY(t)}{dt} = -\frac{d\Psi(\vec{X}(t))}{dx} \tag{4.48}$$

In the discretized format, and solving for the boundary condition implicitly (thus strongly enforcing it), the equations can be approximated as

$$X(t+\Delta t) = \frac{\Psi(\vec{X}(t+\Delta t))}{dy}\Delta t \tag{4.49}$$

$$Y(t+\Delta t) = -\frac{\Psi(\vec{X}(t+\Delta t))}{dx}\Delta t \tag{4.50}$$

These equations must be solved simultaneously (thus the "implicit" label). A more descriptive and implementable form of the above equations are listed below:

$$X^{n+1} = \left.\frac{\Psi(\vec{x})}{dy}\right|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{4.51}$$

$$Y^{n+1} = -\left.\frac{\Psi(\vec{x})}{dx}\right|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{4.52}$$

Equations (4.51) and (4.52) represent the matrix equations that are implemented in the code. They translate the membrane (Lagrangian) mesh location to the underlying blood plasma and cellular cytoplasm (Eulerian) mesh, correctly identifying where the cell boundary is. The plasma velocity $\vec{v}(t)$ along the membrane is thus equal to the local membrane velocity, $d\vec{X}(t)/dt$.

Do note that equations (4.51) and (4.52) form their own residuals due to the introduction of the new variables X and Y, corresponding to the Lagrangian mesh point locations. A more in depth explanation is given in section [#REF].

Oxygen Mass Flux Conservation Across RBC Membrane

The RBC membrane not only separates the cellular cytoplasm from the blood plasma, but also acts as a semi-permeable membrane for oxygen transport. The appropriate boundary condition is not a direct oxygen-in/oxygen-out, but rather ensuring that the normal flux of oxygen times the appropriate diffusion coefficient is satisfied. This statement is generally written mathematically as

$$\alpha_p D_p \left(\nabla p_{O_2} \cdot \hat{n}\right) = -\alpha_c D_c \left(\nabla p_{O_2} \cdot \hat{n}\right) \tag{4.53}$$

The dimensionless form of equation (4.53) can be expressed as:

$$\left(\frac{\alpha_p}{\alpha_c}\right) \frac{1}{Pe_p} \left(\nabla^* p^*_{O_2} \cdot \hat{n}\right) = -\frac{1}{Pe_c} \left(\nabla^* p^*_{O_2} \cdot \hat{n}\right) \tag{4.54}$$

where $Pe.$ is the Péclet number, ∇^* is the nondimensional gradient operator, $p^*_{O_2}$ is the nondimensional partial pressure of oxygen, and $\hat{n}$ is the unit-normal vector pointing outwards from the RBC membrane.

Equation (4.54) must be enforced along the RBC membrane, thus along the Lagrangian mesh. Given this requirement, equation (4.54) is simply multiplied by the delta function, resulting in:

$$\left(\frac{\alpha_p}{\alpha_c}\right) \frac{1}{Pe_p} \left(\nabla^* p^*_{O_2} \cdot \hat{n}\right) \delta\left(\vec{x} - \vec{X}(s)\right) = -\frac{1}{Pe_c} \left(\nabla^* p^*_{O_2} \cdot \hat{n}\right) \delta\left(\vec{x} - \vec{X}(s)\right) \tag{4.55}$$

Expanding terms and simplifying helps to reach a more tractable form of the above result:

$$\left(\frac{\alpha_p}{\alpha_c}\right) \frac{1}{Pe_p} \left(\frac{\partial p^*_{O_2}}{\partial x} \cos(\Phi) + \frac{\partial p^*_{O_2}}{\partial y} \sin(\Phi)\right) \Bigg|_s \delta\left(\vec{x} - \vec{X}(s)\right) = - \frac{1}{Pe_c} \left(\frac{\partial p^*_{O_2}}{\partial x} \cos(\Phi) + \frac{\partial p^*_{O_2}}{\partial y} \sin(\Phi)\right) \Bigg|_s \delta\left(\vec{x} - \vec{X}(s)\right) \tag{4.56}$$

$$\left(\frac{\alpha_p}{\alpha_c}\right)\frac{1}{Pe_p}\left(\cos(\Phi)\bar{\bar{D}}_{x,p}\vec{p}^*_{O_2}+\sin(\Phi)\bar{\bar{D}}_{y,p}\vec{p}^*_{O_2}\right)\Big|_s = -\frac{1}{Pe_c}\left(\cos(\Phi)\bar{\bar{D}}_{x,c}\vec{p}^*_{O_2}+\sin(\Phi)\bar{\bar{D}}_{y,c}\vec{p}^*_{O_2}\right)\Big|_s \quad (4.57)$$

where $\bar{\bar{D}}_{x,p}$ and $\bar{\bar{D}}_{y,p}$ correspond to a matrix resulting form the multiplication of the first derivative operator and the delta function on the plasma side of the membrane, and $\bar{\bar{D}}_{x,c}$ and $\bar{\bar{D}}_{y,c}$ correspond to a matrix resulting form the multiplication of the first derivative operator and the delta function on the cytoplasm side of the membrane. Note that the resulting difference operators are one-sided differences in that they take the first-derivative approximation only within the corresponding domain that the derivative exists in.

Equation (4.57) must be substituted into the governing equations in the residual formulation. No new variables are introduced, but rather constraints (i.e. oxygen flux across the membrane is enforced) are put on the current ones in the problem domain. The nature of this substitution is actually arbitrary, but will be explained in detail in section 4.7.2.

4.7 Residual Formulations

4.7.1 Residual Formulation for Interior Points

In order to solve the equations of motion for the blood plasma and the oxygen transport equations, a system of residuals must be formulated for each point in the domain. This section will examine the capillary domain, as it requires the most residual equations and the physical interactions between the RBC membrane and the surrounding blood plasma is nontrivial. The three exterior domains (vascular wall, interstitial space, and muscular tissue) only use the oxygen transport equations, therefore requiring fewer residuals and reverting to solving advective-diffusion equations on a static mesh.

The residual equations for each point (x, y) in the capillary domain are:

$$R_1 \equiv \left(\omega^* + \nabla^{*2}\Psi^*\right)\Big|^{n+1} \tag{4.58}$$
$$= 0$$
$$R_2 \equiv \frac{\partial \omega^*}{\partial t^*} + \left(\frac{\partial \Psi^*}{\partial y^*}\frac{\partial \omega^*}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial \omega^*}{\partial y^*}\right)\Big|^n - \frac{1}{Re}\left(\frac{\partial^2 \omega^*}{\partial x^{*2}} + \frac{\partial^2 \omega}{\partial y^{*2}}\right)\Big|^{n+1} - \left(\frac{\partial f_x^*}{\partial y^*} - \frac{\partial f_y^*}{\partial x^*}\right)\Big|^n \tag{4.59}$$
$$= 0$$
$$R_3 \equiv \frac{\partial p_{O_2}^*}{\partial t^*} + \left(\frac{\partial \Psi^*}{\partial y^*}\frac{\partial p_{O_2}^*}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial p_{O_2}^*}{\partial y^*}\right)\Big|^n - \frac{1}{Pe.}\left(\frac{\partial^2 p_{O_2}^*}{\partial x^{*2}} + \frac{\partial^2 p_{O_2}^*}{\partial y^{*2}}\right)\Big|^{n+1} - Da_{Hb}\left(\frac{[Hb]}{\alpha p_{O_2}^*}\right)\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(p_{O_2}^*\right)^n\right)\Big|^n + M^*\Big|^n \tag{4.60}$$
$$= 0$$
$$R_4 \equiv \frac{\partial S^{Hb}}{\partial t^*} + \left(\frac{\partial \Psi^*}{\partial y^*}\frac{\partial S^{Hb}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial S^{Hb}}{\partial y^*}\right)\Big|^n - \frac{1}{Pe.}\left(\frac{\partial^2 S^{Hb}}{\partial x^{*2}} + \frac{\partial^2 S^{Hb}}{\partial y^{*2}}\right)\Big|^{n+1} + Da_{Hb}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(p_{O_2}^*\right)^n\right) \tag{4.61}$$
$$= 0$$

Note that all the residuals are defined to be an equation that equals 0. When posed this way, the Newton Method can iterate over the multiple variables and guide the resulting non-linear system to a vector solution of zeros, thereby converging to a numerical solution. (The resulting numerical solution must be inspected further to ensure physical plausibility. In most simulation software, this process is referred to as *code verification.*)

Equations (4.58) through (4.61) take on various discretizations depending on whether or not the Eulerian mesh point lies inside or outside the RBC membrane. The blood plasma and cellular cytoplasm have different valued non-dimensional parameters (e.g. Reynolds, Péclet, and Damkhöler numbers) based on the matter that makes up the plasma and cytoplasm. As a result of this domain dependence, the values of those parameters must be changed accordingly and the approximation of the deriva-

tives must be physically consistent (i.e. the approximations must be made inside the appropriate domain, either inside or outside the cellular membrane, depending upon the location of the point being approximated).

The residual Jacobian matrix must be constructed and understood. When the Newton Method is extended to multiple variables and the residual discretizations are semi-implicit or fully-implicit, a matrix formulation must be used in order to solve all of the necessary equations simultaneously. The matrix equation takes the general form as follows:

$$\bar{\bar{K}}\delta\vec{U} = -\vec{R} \tag{4.62}$$

where $\bar{\bar{K}}$ is the Jacobian matrix comprised of local derivative approximations linearized about the $n+1$ time-step solution, $\delta\vec{U}$ is the variable step-change vector (to be solved for), and $\vec{R}$ is the residual vector composed of both time-step n and $n+1$ solution data. This system of equations is solved via the MATLAB \ (backslash) command:

$$\delta\vec{U} = -\bar{\bar{K}}\backslash\vec{R} \tag{4.63}$$

In order to visualize what comprises the $\bar{\bar{K}}$ matrix, equation (4.58) will be discretized via the standard central difference formula as an example:

$$R_1 = \omega^*_{i,j} + \left(\frac{\Psi_{i+1,j} - 2\Psi^*_{i,j} + \Psi^*_{i-1,j}}{\Delta x^{*2}} + \frac{\Psi^*_{i,j+1} - 2\Psi^*_{i,j} + \Psi^*_{i,j-1}}{\Delta y^{*2}}\right) \tag{4.64}$$

where the subscripts i, j indicate the local mesh point along with those adjacent to it.

In order to fill the Jacobian matrix $\bar{\bar{K}}$ for the first residual R_1, the derivative of R_1 must be taken with respect to each variable at each point. By inspection, all but six entries in this residuals respective rows will be zeros. The resulting six derivatives are:

$$\frac{\partial R_1}{\partial \omega^*_{i,j}} = 1 \tag{4.65}$$

$$\frac{\partial R_1}{\partial \Psi^*_{i,j}} = -\left(\frac{2}{\Delta x^{*2}} + \frac{2}{\Delta y^{*2}}\right) \tag{4.66}$$

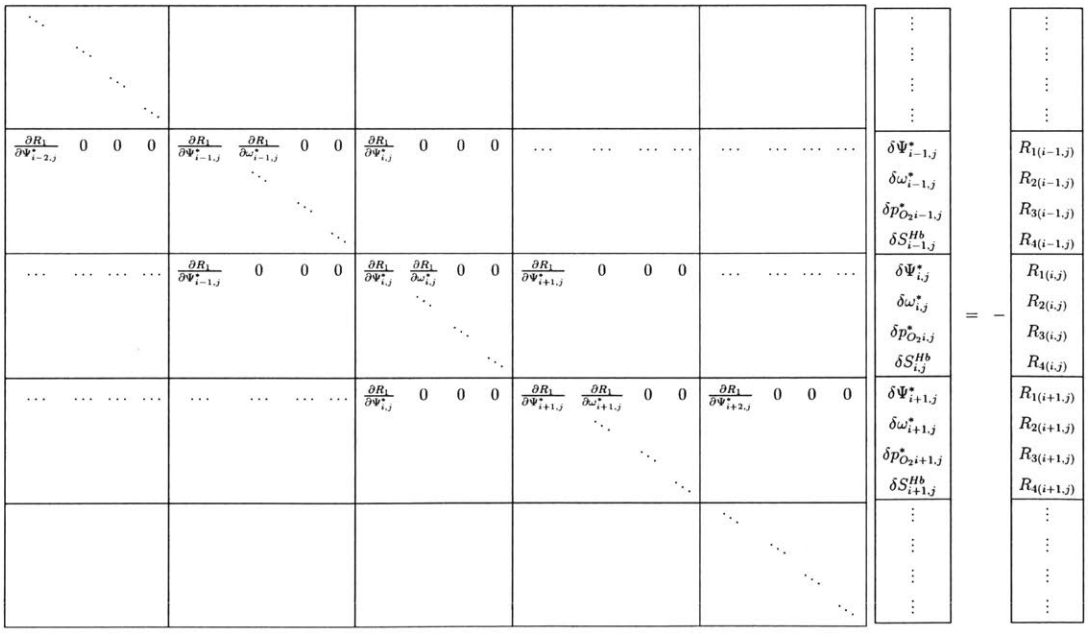
$$\frac{\partial R_1}{\partial \Psi^*_{i+1,j}} = \frac{1}{\Delta x^{*2}} \tag{4.67}$$

$$\frac{\partial R_1}{\partial \Psi^*_{i-1,j}} = \frac{1}{\Delta x^{*2}} \tag{4.68}$$

$$\frac{\partial R_1}{\partial \Psi^*_{i,j+1}} = \frac{1}{\Delta y^{*2}} \tag{4.69}$$

$$\frac{\partial R_1}{\partial \Psi^*_{i,j-1}} = \frac{1}{\Delta y^{*2}} \tag{4.70}$$

With these derivatives, $\bar{\bar{K}}$ can be built up by inserting those values at the indicated variable location (e.g. $\Psi^*_{i,j}$) in the appropriate columns, in which said columns' numbers correspond to the rows' numbers in the solution vector $\delta\vec{U}$. The conceptual visualization of this matrix is (partially) seen below:



Note that not all of the residual derivatives are explicitly shown (e.g. $\frac{\partial R_1}{\partial \Psi^*_{i,j+1}}$ and

$\frac{\partial R_1}{\partial \Psi^*_{i,j-1}}$ for $R_{1(i,j)}$). These derivatives are generated for all four residual equations at each Eulerian grid point (x_i, y_j) for $i \in [0, M]$ and $j \in [0, N_{cap}]$, where M is the total number of x-mesh points and N_{cap} is the total number of capillary sub-domain y-mesh points (for a total of MN_{cap} Eulerian mesh points).

The other residual equations R_2, R_3, and R_4 and their associated derivatives are constructed in a similar fashion to R_1 and are appropriately inserted into their respective rows and columns, based on the chosen discretization scheme. Once the matrix is constructed, it is then solved via equation (4.63) and the solution values are updated via:

$$U_{k+1} = U_k + \delta U \tag{4.71}$$

If the solution U_{k+1} does not satisfy the residual system within the user designated tolerance, U_{k+1} is now used as the new Newton Method initial guess and is iterated upon again until the residual tolerance is satisfied.

Red Blood Cell Membrane Considerations

In order to prevent nonphysical diffusion modeling (Laplacian terms in fluid momentum and oxygen transport equations), the location of the Lagrangian points must be checked so that the underlying Eulerien capillary point approximations can be adjusted accordingly.

For instance, if an Eulerian mesh point lies directly to the left of the RBC membrane, it should not use a centered difference approximation for the second derivative, as the $i + 1$ point will cross into the interior of the RBC boundary. Thus, a backwards difference scheme should be used so that the local information regarding diffusion comes solely from the domain outside the RBC membrane. The appropriate approximations for this are boxed in Appendix D.

4.7.2 Residual Formulation for Boundary Conditions

Given that the boundary conditions are formed and the residual Jacobian matrix is generally constructed, modifications to it must occur to properly account for

boundary conditions. Most boundary condition residuals directly replace the main governing equations in the residual matrix system. There are a few exceptions to this replacement rule, which will be exemplified and discussed in depth in the following sections.

Red Blood Cell sub-domain:

In order to implement no-slip and normal flux boundary conditions on a moving, deformable mesh, special care must be taken. The boundary condition that must be implemented is that presented in equations (4.51) and (4.52). Fortunately, two new variables are introduced in X and Y, the Lagrangian mesh point locations. What this means is that these equations can form their own residuals and be independently implemented into the matrix equation (4.62) without the need to modify previously constructed residuals.

The matrix system that equations (4.51) and (4.52) form for a Lagrangian point l can be added to the top of the matrix in the matrix equation (4.62). The residuals are defined as follows:

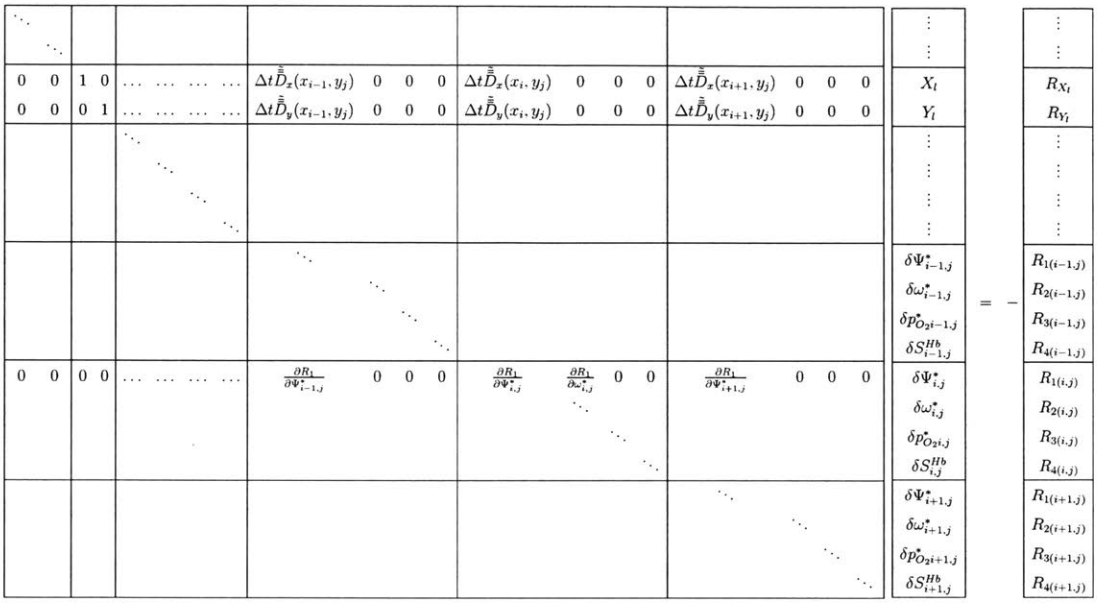
$$R_{X_l} \equiv X^{n+1} - \left.\frac{\Psi(\vec{x})}{dy}\right|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{4.72}$$

$$\equiv X^{n+1} - \Delta t \bar{\bar{D}}_y \vec{\Psi}^* = 0$$

$$R_{Y_l} \equiv Y^{n+1} + \left.\frac{\Psi(\vec{x})}{dx}\right|^{n+1} \delta\left(\vec{x}^{n+1} - \vec{X}^{n+1}(s)\right)\Delta t \tag{4.73}$$

$$\equiv Y^{n+1} + \Delta t \bar{\bar{D}}_x \vec{\Psi}^* = 0$$

where $\bar{\bar{D}}_x$ and $\bar{\bar{D}}_y$ are the differentiation sampling matrices similar to those in equation (4.57), but are discretized on both sides of the cellular membrane. Upon differentiating, l^{th} row and the associated $\vec{\Psi}^*(x_i, y_j)$ column entries of $\bar{\bar{D}}_x$ and $\bar{\bar{D}}_y$ are inserted into the Jacobian matrix $\bar{\bar{K}}$ as follows:



Now, the Lagrangian mesh points (X_l, Y_l) can be implicitly calculated such that the flow along the cellular membrane satisfies the no-slip condition. In other words, the time-step $n+1$ location of the Lagrangian mesh is solved for such that the velocity along the wall is 0, thus strongly enforcing the no-slip boundary condition while moving and deforming the mesh appropriately.

The enforcement of the oxygen flux boundary condition across the cellular membrane has additional complications. There are no new variables present in the boundary condition, equation (4.57). Because of this, the existing governing equations contained in the matrix system must be modified. The modifications involve substitutions for terms in the governing residual equations for oxygen partial pressure (namely R_3). However, which (x_i, y_j) point gets modified is arbitrary.

If there are 100 Lagrangian mesh points, 100 substitutions for the oxygen normal flux boundary condition must be applied to 100 of the R_3 residuals (which exist on the Eulerian mesh). Which 100 R_3 residuals are to be chosen are arbitrary. As long as the residuals and residual derivatives do not form a matrix singularity, a solution will be converged upon. However, the solutions from different choices of R_3 substitutions may not be consistent with each other. At the time of writing, the author is not

aware of this issue being seen in practice, but acknowledges that it may diminish the quality of the results of the simulation.

Capillary sub-domain:

Recalling the capillary boundary conditions in equations (4.35)-(4.40), their residuals are much more tractable and do not require complex substitutions.

For the streamfunction Ψ^*, the residuals are:

$$R_{BC1,(i,1_{cap})} = \Psi^*_{i,1_{cap}} \tag{4.74}$$

$$R_{BC1,(i,N_{cap})} = \Psi^*_{i,N_{cap}} - 1; \tag{4.75}$$

For the vorticity ω^*, the boundary condition for $y^* = 0$ is straightforward:

$$R_{BC2,(i,1_{cap})} = \omega^*_{i,1_{cap}} \tag{4.76}$$

However, the boundary condition for $y^* = 1$ requires one extra step. This extra step is referred to as a ghost point substitution, which is a technique used to enforce the first-derivative boundary condition.

$$R_{BC2,(i,N)} = \omega^*_{i,N_{cap}} + \frac{\Psi^*_{i,N_{cap}+1} - 2\Psi^*_{i,N_{cap}} + \Psi^*_{i,N_{cap}-1}}{\Delta y^{*2}_{N_{cap}}} \tag{4.77}$$

$$= \omega^*_{i,N_{cap}} + \frac{-2\Psi^*_{i,N_{cap}} + 2\Psi^*_{i,N_{cap}-1}}{\Delta y^{*2}_{N_{cap}}} \tag{4.78}$$

where a first-derivative mid-point approximation, set equal to 0 for the boundary condition on Ψ^*, is used as a substitution for $\Psi^*_{i,N+1}$ (see equation (4.10)).

In a similar fashion, the oxygen partial pressure equation (4.19) also requires this substitution. For the $y^* = 0$, where the normal oxygen mass flux across the capillary centerline is 0, the boundary residual is:

$$\frac{\partial p^*_{O_2}}{\partial t^*} + \left[\frac{\partial \Psi^*}{\partial y^*}\frac{\partial p^*_{O_2}}{\partial x^*} - \frac{\partial \Psi^*}{\partial x^*}\frac{\partial p^*_{O_2}}{\partial y^*}\right]\Bigg|^n =$$
$$\left[\frac{1}{Pe_{O_2}}\left(\frac{p^*_{O_2,(i+1,1_{cap})} - 2p^*_{O_2,(i,1_{cap})} + p^*_{O_2,(i-1,1_{cap})}}{\Delta x^{*2}} + \frac{2p^*_{O_2,(i,1_{cap})} - 2p^*_{O_2,(i,1_{cap})}}{\Delta y^{*2}_{1_{cap}}}\right)\right]\Bigg|^{n+1} \tag{4.79}$$

For the capillary sub-domain edge where $y^* = 1$, one can substitute the normal flux boundary condition, equation (4.40), into equation (4.19). A more ad-hoc, but still valid, residual is to directly discretize equation (4.40) and make it the residual, being careful to use a one-sided difference for each derivative corresponding to its relevant sub-domain. With the non-uniform mesh, the weights of the discretization depend on the mesh chosen by the user. Using the approximations in Appendix D, a residual can be readily constructed.

Vascular Wall and Interstitial Space sub-domain:

The residuals for the interface boundary conditions for oxygen partial pressure for the vascular wall and interstitial space sub-domains can be constructed in a similar manner to that of the capillary sub-domain. Refer to the previous section for details.

Muscular Tissue sub-domain:

The muscuar tissue sub-domain has to account for both the oxygen partial pressure and the saturation of myoglobin. The oxygen partial pressure boundary conditions are implemented in the same manner as above, for both the interstitial space muscular tissue interface and for the muscular tissue edge, using a ghost point method for the latter condition.

For the myoglobin saturation transport equation, a ghost point method is also used, but for both domain edges as they both use the no-flux boundary condition. For completeness, they are listed below:

$$\frac{\partial S^{Mb}}{\partial t} = \left[\frac{1}{Pe_{Mb}}\left(\frac{S^{Mb}_{i+1,1_t} - 2S^{Mb}_{i,1_t} + S^{Mb}_{i-1,1_t}}{\Delta x^{*2}} + \frac{2S^{Mb}_{i,2_t} - 2S^{Mb}_{i,1_t}}{\Delta y^{*2}_{1_t}}\right)\right]\Bigg|^{n+1} - \left[Da_{Mb}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(rp^{*}_{O_2}\right)\right)\right]\Bigg|^{n} \quad (4.80)$$

$$\frac{\partial S^{Mb}}{\partial t} = \left[\frac{1}{Pe_{Mb}}\left(\frac{S^{Mb}_{i+1,N_t} - 2S^{Mb}_{i,N_t} + S^{Mb}_{i-1,N_t}}{\Delta x^{*2}} + \frac{2S^{Mb}_{i,N_t-1} - 2S^{Mb}_{i,N_t}}{\Delta y^{*2}_{N_t}}\right)\right]\Bigg|^{n+1} - \left[Da_{Mb}\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(rp^{*}_{O_2}\right)\right)\right]\Bigg|^{n} \quad (4.81)$$

Chapter 5

Concluding Remarks

This chapter presents and discusses the results from the current presented research.

5.1 Discussion of Results

This thesis provides two contributions to be used in the modeling of blood microcirculation flows. The first contribution is the development of a dimensionless streamfunction vorticity model, which offers a host of benefits for the computational setup and speed. The second contribution is the development of an implicit formulation for the boundary conditions along the immersed boundary in the streamfunction vorticity form. It is shown how to be implemented in a Newton-Raphson residual solver Jacobian matrix.

5.1.1 Parametric Study of Blood Microcirculation Flows: A Non-dimensional Viewpoint

The dimensionless framework and model developed in this work provides versatility for a variety of parametric studies. While the specifics of certain blood disease models need to be implemented, the general process of the data collection and analysis is similar.

Dimensionless models offer the ability to adjust physical parameter inputs in a dimensionless fashion. For example, if the effects from viscosity are wished to be studied, the Reynolds number is adjusted for a range of values and the effects are

then studied in comparison to other dimensionless values (for example, see Figure 2-1). Assessing physical mechanisms in this dimensionless way offers insights about the blood microcirculation flow and can help doctors assess the importance of effects such as blood plasma viscosity in comparison to other effects such as the pressure drop through the capillary, and can quantify when these effects can and cannot be ignored for the development of new treatments.

5.1.2 Immersed Boundary Method Using Streamfunction Vorticity Formulation

The IBM has been used extensively in simulations with Navier-Stokes (NS) (u-v-p) formulations as opposed to streamfunction vorticity (SFV) (Ψ-ω) formulations. While the reasons for this preference are unknown to the author, using the latter formulation is not restricted by the IBM.

The SFV formulation has numerous advantages to its use over NS formulations. The first advantage is the reduction of variables, leading to decreased runtimes. The second advantage is that the boundary conditions are kinematic in nature as opposed to dynamic in nature, leading to a more direct implementation. The third advantage is the relative simplification of code, in that a simultaneous reduction of governing equations as well as the form of the governing equations (purely advecting-diffusion) leads to a direct semi-implicit solve as opposed to, for example, a predictor-corrector method. The final benefit noted is the use of a collocated mesh as opposed to a staggered mesh, which simplifies the implementation of the governing equations and does not suffer from the *odd-even decoupling* produced by pressure terms in collocated Navier-Stokes simulations.

The difficulty in using the SFV formulation in IBM arises from the enforcement of a no-slip boundary condition on a moving surface. In NS formulations, the velocity is set equal to the speed of the moving boundary surface. However, in the SFV formulation, the streamfunction derivatives are set equal to the speed of the moving boundary surface. The formation of differentiation sampling matrices, $\tilde{\tilde{D}}_x$ and $\tilde{\tilde{D}}_y$ (as seen in section 4.7.2), with the appropriate substitution of their individual entries

into the residual Jacobian matrix, elegantly resolves the issue of boundary condition enforcement for SFV IBM techniques. It also offers an elegant way to implicitly solve for the location of the immersed boundary, strongly enforcing the no-slip boundary condition along the boundary.

5.2 IBM Coding Challenges

The IBM technique has conjured up a few challenges that were difficult in implementing. The first challenge pertains to the grid construction and one-sided differencing in finite difference codes. The second challenge pertains to the arbitrariness that is seen in the implementation of the normal oxygen flux boundary condition across the RBC membrane.

The first challenge of grid construction issue arises in the one-sided differencing of first and second derivatives when inside the red blood cell. In regions of small radius of curvature, there exists the potential for one-sided difference stencils to overlap both the RBC interior and blood plasma domains, which leads to an incorrect diffusion approximation, both physically (RBC acts as a material barrier, thus no overlap is allowed) and dimensionlessly (differing Reynolds or Péclet numbers in Laplacian terms for transport equations). Circumventing this issue requires either a smaller stencil, resulting in reduced accuracy, or requires some form of interpolation, resulting in increased code complexity and reduced accuracy.

The second challenge arising with the arbitrariness of the normal oxygen flux boundary condition comes form the substitution of the boundary condition terms into the governing residual equations. For example, if there are 100 Lagrangian mesh points (RBC mesh points), there are 100 oxygen normal flux boundary condition equations, one for each point. However, given the method in which the IBM samples the Eulerian mesh beneath the Lagrangian mesh, this sampling results in a dependency on more than 100 Eulerian mesh points. The question arises: which Eulerian mesh point (i.e. row in the residual Jacobian matrix) does the boundary condition substitute into for its enforcement? In conversations with colleagues[18], it has been determined that the substitution is arbitrary so that the 100 Lagrangian boundary

conditions can be substituted into any 100 of the 100+ Eulerian mesh point candidates. These generally arbitrary substitutions are concerning because one user-chosen algorithm for the substitutions, as opposed to different substitution algorithm, may produce differing simulation results.

5.3 Future Work

This research presents a solid framework for the future studies of a variety of problems.

Le-Floch Yin[2] has developed a model for sickle cell anemia microcirculation flows that can be implemented into the dimensionless framework of this research model. The physical effects that he observed can be dimensionlessly studied in order to better describe the dominant effects that lead to the sickling of red blood cells. With a dimensionless parametric study of this type, doctors can clearly distill which physical effects need to be altered in sick patients (e.g. blood plasma viscosity, oxygen diffusion constants and rates) and how altering those physical properties of the blood flow affects other blood properties. Ultimately, the dimensionless studies can be used to guide doctors to develop improved treatment options to afflicted patients.

An important effect that should be accounted for is that of the viscoelastic (shear-thinning) behavior of blood microcirculation flow[9]. The shear-thinning behavior, as seen in Figure 2-1, is non-linear. Blood microcirculation flows have Reynolds number of O(1) to O(10), dependent upon the capillary diameter. As can be seen in Figure 2-1, the variability in the dimensionless pressure drop is large for Reynolds number of this small order.

Another research opportunity for blood microcirculation flows, pertaining particularly to the field of aerospace, is that of the blood flows of astronauts during highly accelerated rocket launches and in microgravity environments, such as that on the Internation Space Station (ISS) or on a trip from Earth to Mars. Some questions that arise are whether or not blood flows in these environments contribute to degradation of surrounding tissue, and if so, what can be done to mitigate this impact for healthy living in long-duration space missions.

Appendix A

Derivation of Equations of Motion - Fluid Mechanics

Given the length scales of this research problem, an appropriate model for the blood plasma and red blood cell cytoplasm is that of continuous medium, or a continuum, which happens to be that of a fluid. Therefore, the equations of motion for the blood plasma and cellular cytoplasm will be governed by fluid mechanics.

The differential expression for the law of mass conservation is as follows:

$$\frac{D\rho}{Dt} + \rho\nabla \cdot \vec{v} = 0 \tag{A.1}$$

Here, $\rho(\vec{x}, t)$ is the fluid density, $\vec{v}(\vec{x}, t)$ is the fluid velocity, and ∇ is the spatial differential del vector operator. Note that $\vec{x}$ refers to the Eulerian coordinates and that t refers to time.

The differential expression for the law of momentum conservation in a continuum, known as the Cauchy Momentum Equation, is as follows:

$$\rho\frac{D\vec{v}}{Dt} = \nabla \cdot \bar{\bar{\sigma}} + \vec{f} \tag{A.2}$$

Two new variables are introduced in this equation, namely the Cauchy stress tensor $\sigma(\vec{x}, t)$ and the body force vector $\vec{f}(\vec{x}, t)$.

The stress tensor $\bar{\bar{\sigma}}$ can be modeled in many different ways, resulting in the analysis

of a many different materials (e.g. steel, shaving cream, air, etc.). For the purposes of fluid mechanics, it is convenient to split the tensor into its isotropic stress and deviatoric (traceless) stress tensors. Doing so results in

$$\bar{\bar{\sigma}} = -p\bar{\bar{I}} + \bar{\bar{\tau}}, \tag{A.3}$$

where p is the mechanical pressure, or the normal stress applied to a fluid element, $\bar{\bar{I}}$ is the identity matrix, and $\bar{\bar{\tau}}$ is the deviatoric stress tensor, or the stress resulting from non-hydrostatic states.

How $\bar{\bar{\tau}}$ is modeled determines the type of flow that is examined. The present work assumes the Newtonian observation of linear strain rates, the fluid is isotropic, and $\nabla \cdot \bar{\bar{\tau}} = 0$ when the fluid comes to rest. A more complex model can be used to account for other non-Newtonian effects, which may be of interest for future work. However, applying these assumptions leads to the following result:

$$\bar{\bar{\tau}} = \mu \left(\nabla \vec{v} + \nabla^T \vec{v} \right) + \left(\lambda - \frac{2}{3}\mu \right) (\nabla \cdot \vec{v}) \tag{A.4}$$

The first term contains the information regarding the shear stress acting on the fluid and the second term contains the information for the normal stresses in regards to volume changes of the fluid element. The term μ is the dynamic viscosity (or shear viscosity), which is a proportionality coefficient indicating the capability of a fluid to dissipate momentum. The term λ is the bulk viscosity (or volume viscosity) that is a proportionality coefficient for the ability of the volume change of a fluid element to dissipate momentum. These terms are assumed to be constant in this work, but they can have dependencies on certain fluid properties or variables, such as the local temperature of the fluid.

Substituting equations (A.3) and (A.4) into equation (A.2), the equations of motion for the fluid are as follows:

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \vec{v} = 0 \tag{A.1}$$

$$\rho\frac{D\vec{v}}{Dt} = -\nabla p + \nabla \cdot \left[\mu\left(\nabla\vec{v} + \nabla^T\vec{v}\right) + \left(\lambda - \frac{2}{3}\mu\right)(\nabla \cdot \vec{v})\right] + \vec{f} \tag{A.5}$$

These equations represent the complete equations of motion for a fluid (excluding conservation of energy and an equation of state, which are not relevant for this research - see below).

Given that we are in a low Mach number flow, the fluid may be accurately approximated as an incompressible fluid and may be assumed isothermal (energy conservation and equation of state are no longer required to solve for the fluid state variables). These approximations allow for us to neglect temperature changes (eliminating the need for conservation of energy and an equation of state). This approximation also states that the substantial derivative of the density ρ is zero, resulting in the following velocity divergence constraint from equation (A.1):

$$\nabla \cdot \vec{v} = 0 \tag{A.6}$$

This new constraint allows us to simplify equation (A.5) to the following, incompressible form of Navier-Stokes:

$$\rho\frac{D\vec{v}}{Dt} = -\nabla p + \mu\nabla^2\vec{v} + \vec{f} \tag{A.7}$$

The incompressible fluid equations of motion are now presented as equations (A.6) and (A.7). These equations examine the dynamical effects of the fluid; however, a kinematic approach can be taken when using a streamfunction vorticity formulation.

In order to arrive at the kinematic representation of the equations of motion in a two dimensional, Cartesian flow, a streamfunction must first be defined as follows:

$$u \equiv -\frac{\partial\Psi}{\partial y} \tag{A.8}$$

$$v \equiv \frac{\partial\Psi}{\partial x} \tag{A.9}$$

Next, by taking the curl of equation (A.7), a purely kinematic vector equation

results:

$$\rho\frac{D\vec{\omega}}{Dt} = \mu\nabla^2\vec{\omega} + \nabla \times \vec{f} \tag{A.10}$$

where

$$\vec{\omega} \equiv \nabla \times \vec{v} \tag{A.11}$$

Equation (A.10) is known as the incompressible vorticity transport equation. However, when simplified to the two dimensional case, it becomes a scalar equation (more precisely, a vector equation entirely orthogonal to the fluid velocity field).

Applying equations (A.8) and (A.9) to equation (A.11) and using the two dimensional case of equation (A.10) gives the desired governing equations for the fluid equations of motion:

$$\omega + \nabla^2\Psi = 0 \tag{A.12}$$

$$\rho\frac{D\omega}{Dt} = \mu\nabla^2\omega + \nabla \times \vec{f} \tag{A.13}$$

In order to complete the derivation of the kinematic form for the equations of motion, the divergence of equation (A.7) may be taken to yield the pressure Poisson equation:

$$\nabla^2 p = 2\rho\left[\frac{\partial^2\Psi}{\partial x^2}\frac{\partial^2\Psi}{\partial y^2} - \left(\frac{\partial^2\Psi}{\partial x \partial y}\right)^2\right] \tag{A.14}$$

This equation is used in order to compute the pressure levels ex-post-facto in most solvers.

In order to isolate the dynamic effects which govern the problem, nondimensionalizing the governing equations is required. Using dimensional analysis techniques, the following definitions are arrived at:

$$x^* \equiv \frac{x}{L} \qquad y^* \equiv \frac{y}{L} \qquad t^* \equiv \frac{tU}{L}$$

$$\omega^* \equiv \frac{\omega L}{U} \qquad \Psi^* \equiv \frac{\Psi}{UL} \qquad \vec{f^*} \equiv \frac{\vec{f} L}{\rho U^2} \tag{A.15-A.20}$$

Here, the values for L, U, and ρ are independently chosen so that the geometric and dynamic scaling of the problem is preserved. The time t is simply the solver time at some time-step.

Upon substitution in equations (A.12), (A.13), and (A.14), the nondimensional forms of the governing equations of motion are as follows:

$$\omega^* + \nabla^{*2}\Psi^* = 0 \tag{A.15}$$

$$\frac{\partial \omega^*}{\partial t^*} + (\nabla^*_\perp \Psi^*) \cdot \nabla^* \omega * = \frac{1}{Re} \nabla^{*2} \omega^* + \left(\nabla^* \times \vec{f^*} \right) \tag{A.16}$$

$$\nabla^{*2} p^* = 2 \left[\frac{\partial^2 \Psi^*}{\partial x^{*2}} \frac{\partial^2 \Psi^*}{\partial y^{*2}} - \left(\frac{\partial^2 \Psi^*}{\partial x^* \partial y^*} \right)^2 \right] \tag{A.17}$$

where $\nabla^* = (\partial/\partial x^*, \partial/\partial y^*, \partial/\partial z^*)$, $\nabla^*_\perp = (\partial/\partial y^*, -\partial/\partial x^*, 0)$, and the Reynolds number is

$$Re \equiv \frac{\rho U L}{\mu} = \frac{UL}{\nu} \tag{A.18}$$

which measures the ratio of inertial forces to viscous forces. Here it is seen that the Reynolds number is the only unique nondimensional quantity of interest that governs the mechanics of the system.

Equations (A.15), (A.16), and (A.17) are the final equations of motion for the blood plasma and cellular cytoplasm. In the same manner as before, the nondimen-

sional pressure can be computed ex-post-facto in most solvers.

Appendix B

Derivation of Equations of Motion - Oxygen Transport

Physical phenomena dominated by advection and diffusion can be modeled by the advective-diffusion equation (often referred to as the convective-diffusion equation). In the case of oxygen transport, this equation is used to model the concentration of oxygen in the blood microcirculation and tissue, as well as how it varies in time and space:

$$\frac{\partial c}{\partial t} + \nabla \cdot (\vec{v}c) = \nabla \cdot (D\nabla c) + R \tag{B.1}$$

Each term in the equation (B.1) corresponds to a specific physical process. The first term pertains to the unsteadiness of the process. The second term corresponds to the advection of the variable c by the fluid traveling at velocity $v(\vec{x}, t)$. The third term corresponds to the diffusion of c, which is modeled linearly by the divergence of its gradient multiplied by a diffusion coefficient D. The final term pertains to the reaction rate R at which a chemical species undergoes a reaction with its surrounding environment.

It is useful to note that the modeling of oxygen transport in this problem is carried out with an incompressible fluid, which provides the velocity divergence constraint developed in Appendix A, equation (A.6):

$$\nabla \cdot \vec{v} = 0 \tag{A.6}$$

With this constraint, and given that D is a diffusivity coefficient, equation (B.1) simplifies to:

$$\frac{\partial c}{\partial t} + (\vec{v} \cdot \nabla)\, c = D\nabla^2 c + R \tag{B.2}$$

giving the working form of the advective-diffusion equation for this problem.

Oxygen transport in the capillary is carried out in three ways. First, oxygen is hypothesized to be consumed at a constant rate M in the capillary wall and tissue. Second, oxygen binds and is released by hemoglobin in the red blood cell. Third, oxygen binds and is released by myoglobin in the tissue. The latter two rates are governed by the law of mass action as derived from oxyhemoglobin and oxymyoglobin dissociation reactions.

The oxyhemoglobin dissociation reaction is assumed to be occurring in equilibrium:

$$\mathrm{Hb_4} + 4\,\mathrm{O_2} \rightleftharpoons \mathrm{Hb_4O_8} \tag{B.3}$$

$$R^{Hb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Hb} = 4k_{-1}^{hb}[Hb_4O_8] - 4k_1^{hb}[Hb_4]c^4 \tag{B.4}$$

where k_1^{HB} and k_{-1}^{HB} are the kinetic dissociation rate constants of the forward and backward reactions, respectively, and $[Hb_4]$ and $[Hb_4O_8]$ are the concentrations of the unbound hemoglobin and bound oxyhemoglobin, respectively. For simplicity, it is assumed that the oxygen has infinite cooperativity with the hemoglobin proteins, such that all the oxygen binds or unbinds at once.

In order to make R^{Hb} more tractable to solve, a substition for $[Hb_4]$ by the total hemoglobin conentration $[Hb_4]_{tot}$ (comprising both bound and unbound states) and defining the oxyhemoglobin saturation S^{Hb} is necessary:

$$[Hb_4] = [Hb_4]_{tot} - [Hb_4O_8] \tag{B.5}$$

$$K_{Hb} = \frac{[Hb_4O_8]}{[Hb_4][O_2]^4} \tag{B.6}$$

$$S^{Hb} = \frac{[Hb_4O_8]}{[Hb_4]_{tot}} = \frac{K_{Hb}[O_2]^4}{1 + K_{Hb}[O_2]^4} \tag{B.7}$$

$$R^{Hb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Hb} = [Hb_4]_{tot}\left(4k_{-1}^{Hb}S^{Hb} - 4k_1^{Hb}\left(1 - S^{Hb}\right)c^4\right) \tag{B.8}$$

where K_{Hb} is the reaction equilibrium coefficient for hemoglobin, with dependencies on temperature, blood plasma pH, and other molecules such as CO_2 and 2,3-DPG. Noting that the ratio of the reaction constants yields the equilibrium oxygen concentration at 50% saturation, $c_{50\%}^{Hb}$, and substituting yields:

$$c_{50\%}^{Hb} = \left(\frac{k_{-1}^{Hb}}{k_1^{Hb}}\right) \tag{B.9}$$

$$R^{Hb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Hb} = 4k_{-1}^{Hb}[Hb_4]_{tot}\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{c}{c_{50\%}^{Hb}}\right)^4\right) \tag{B.10}$$

Because the concentration of oxygen is coupled to the saturation of oxyhemoglobin, an advective-diffusion equation for it is necessary to determine the saturation $S^{Hb}(\vec{x}, t)$:

$$\frac{\partial S^{Hb}}{\partial t} + (\vec{v} \cdot \nabla) S^{Hb} = D^{Hb}\nabla^2 S^{Hb} - \frac{R^{Hb}}{4[Hb_4]_{tot}} \tag{B.11}$$

where D^{Hb} is the diffusivity of oxyhemoglobin, and the production rate R is determined by using the above reaction. Now introducing the total hemoglobin concentration in terms of moles of protein subunits per unit volume, $[Hb]$, equations (B.10) and (B.11) become:

$$R^{Hb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Hb} = k_{-1}^{Hb}[Hb]\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{c}{c_{50\%}^{Hb}}\right)^4\right) \tag{B.12}$$

$$\frac{\partial S^{Hb}}{\partial t} + \left(\vec{v}\cdot\nabla\right)S^{Hb} = D^{Hb}\nabla^2 S^{Hb} - \frac{R^{Hb}}{[Hb]} \tag{B.13}$$

In reality, infinite cooperativity does not occur in the binding of oxygen to hemoglobin nor the unbinding of oxyhemoglobin. In order to account for this effect without adding complexity in the form of more reactions and transport equations, experimental data is used to determine the Hill coefficient n, which replaces the exponent 4 at equilibrium in equation (B.12). This allows us to lie on a realistic hemoglobin dissociation curve (see Figure B-1). Thus, the resulting in the semi-empirical form of the reaction rate R^{Hb}:

$$R^{Hb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Hb} = k_{-1}^{Hb}[Hb]\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{c}{c_{50\%}^{Hb}}\right)^n\right) \tag{B.14}$$

where n is approximately equal to 2.2 for normal blood microcirculation flows.

The oxymyoglobin dissociation reaction is also assumed to be occurring in equilibrium in the muscle tissue. The reaction is similar to that of hemoglobin, except oxygen has one binding site to the myoglobin protein:

$$\mathrm{Mb} + \mathrm{O_2} \rightleftharpoons \mathrm{MbO_2} \tag{B.15}$$

Using the law of mass action and following a similar derivation pathway to that of hemoglobin, the following relations are arrived at:

$$R^{Mb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Mb} = k_{-1}^{Mb}[Mb]\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(\frac{c}{c_{50\%}^{Mb}}\right)\right) \tag{B.16}$$

$$\frac{\partial S^{Mb}}{\partial t} + \left(\vec{v}\cdot\nabla\right)S^{Mb} = D^{Mb}\nabla^2 S^{Mb} - \frac{R^{Mb}}{[Mb]} \tag{B.17}$$

where D^{Mb} is the diffusivity of myoglobin. In previous works (see [#],[#],[#]), given that the oxymyoglobin dissociation rate constant k_{-1}^{Mb} is sufficiently larger than that

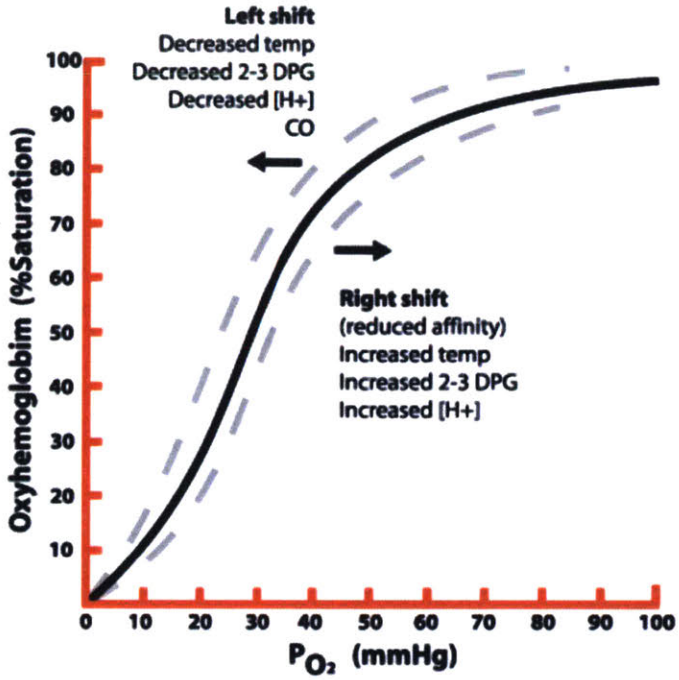


Figure B-1: Oxyhemoglobin curve as a function of partial pressure of oxygen. Image courtesy of *www.anaesthesiauk.com*.

of oxyhemoglobin, k_{-1}^{Hb}, an assumption of local equilibrium between oxygen and myoglobin was employed. This work chooses to neglect this assumption to help reduce non-linearities in the numerical discretization and scheme, aid with nondimensionalization, and to use as a verification for the previous assumption.

The tracking of oxygen concentration throughout the system is also desired, both because the oxygenation of tissue is a desired output and because the reaction rates of oxyhemoglobin and oxymyglobin depend on the relative concentration. To capture the transport of oxygen, the advective-diffusion equation is once again invoked:

$$\frac{\partial c}{\partial t} + (\vec{v} \cdot \nabla)\, c = D\nabla^2 c + R^{\cdot} + M \tag{B.18}$$

where $()^{\cdot}$ is a placeholder for either oxyhemoglobin or oxymyoglobin, which is domain dependent. The diffusivity coefficient D is that of oxygen and the hypothesized oxygen consumption rate constant M is domain dependent quantities as the material properties change across the domains.

Invoking Henry's law, the concentration of oxygen c can be expressed in terms of its partial pressure, which also has the property of being continuous across infinitely thin membranes:

$$c = \alpha p_{O_2} \tag{B.19}$$

where α is the solubility constant of oxygen and p_{O_2} is the partial pressure of oxygen. Upon substitution:

$$\frac{\partial p_{O_2}}{\partial t} + (\vec{v} \cdot \nabla) p_{O_2} = D_{O_2} \nabla^2 p_{O_2} + \frac{R^{\cdot}}{\alpha} + \frac{M}{\alpha} \tag{B.20}$$

and

$$R^{Hb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Hb} = k_{-1}^{Hb}[Hb]\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{c}{c_{50\%}^{Hb}}\right)^n\right) \tag{B.14}$$

$$R^{Mb} = \left.\frac{\partial c}{\partial t}\right|_{chem}^{Mb} = k_{-1}^{Mb}[Mb]\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(\frac{c}{c_{50\%}^{Mb}}\right)\right) \tag{B.16}$$

Once again, the oxygen consumption rate M is a hypothesized value and is set prior to computation start-up.

With a final substitution, the working form of the oxygen partial pressure transport equations are as follows:

$$\alpha\left(\frac{\partial p_{O_2}}{\partial t} + (\vec{v} \cdot \nabla) p_{O_2}\right) = \alpha D \nabla^2 p_{O_2} + k_{-1}^{Hb}[Hb]\left(S^{Hb} - \left(1 - S^{Hb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Hb}}\right)^n\right) + M \tag{B.21}$$

$$\alpha\left(\frac{\partial p_{O_2}}{\partial t} + (\vec{v} \cdot \nabla) p_{O_2}\right) = \alpha D \nabla^2 p_{O_2} + k_{-1}^{Mb}[Mb]\left(S^{Mb} - \left(1 - S^{Mb}\right)\left(\frac{p_{O_2}}{p_{O_2 50\%}^{Mb}}\right)\right) + M \tag{B.22}$$

The most general form of equations (B.21) and (B.22) is equation (B.20). Thus, in terms of solver implementation, only equation (B.20) is implemented, but the form

of its rate term $R^{\cdot}$ is adjusted for the particular domain.

The final step in this derivation process is nondimensionalizing the two governing equations. In order to be thorough with the nondimensionalization, equations (B.13), (B.17), (B.21), and (B.22) will be shown explicitly for clarity. The nondimensional variables are defined as follows:

$$x^* \equiv \frac{x}{L} \qquad y^* \equiv \frac{y}{L} \qquad t^* \equiv \frac{tU}{L} \tag{B.22-B.27}$$

$$p^*_{O_2} \equiv \frac{p_{O_2}}{p^{Hb}_{O_2 50\%}} \qquad S^{Hb*} \equiv S^{Hb} \qquad S^{Mb*} \equiv S^{Mb}$$

The length and timescale variables, x^*, y^*, and t^* are identical to those defined in Appendix A. The oxygen partial pressure p_{O_2} is nondimensionalized by the reference quantity of oxygen partial pressure at at 50% saturation in that of oxyhemoglobin, $p^{Hb}_{O_2 50\%}$. The saturation quantities, S^{Hb} and S^{Mb}, are nondimensional by definition, thus no special manipulation is required.

Upon the substitution of the above definitions into equations (B.13), (B.17), (B.21), and (B.22), their nondimensional forms are:

$$\frac{\partial S^{Hb}}{\partial t^*} + (\nabla^*_\perp \Psi^*) \cdot \nabla^* S^{Hb} = \frac{1}{Pe_{Hb}} \nabla^{*2} S^{Hb} - Da_{Hb} \left(S^{Hb} - \left(1 - S^{Hb}\right) \left(p^*_{O_2}\right)^n \right) \tag{B.23}$$

$$\frac{\partial S^{Mb}}{\partial t^*} + (\nabla^*_\perp \Psi^*) \cdot \nabla^* S^{Mb} = \frac{1}{Pe_{Mb}} \nabla^{*2} S^{Mb} - Da_{Mb} \left(S^{Mb} - \left(1 - S^{Mb}\right) \left(r p^*_{O_2}\right) \right) \tag{B.24}$$

$$\frac{\partial p^*_{O_2}}{\partial t^*}+(\nabla^*_\perp \Psi^*)\cdot\nabla^* p^*_{O_2} = \frac{1}{Pe_{O_2}}\nabla^{*2} p^*_{O_2}+Da_{Hb}\left(\frac{[Hb]}{\alpha p^{Hb}_{O_2 50\%}}\right)\left(S^{Hb}-\left(1-S^{Hb}\right)\left(p^*_{O_2}\right)^n\right)+M^* \tag{B.25}$$

$$\frac{\partial p^*_{O_2}}{\partial t^*}+(\nabla^*_\perp \Psi^*)\cdot\nabla^* p^*_{O_2} = \frac{1}{Pe_{O_2}}\nabla^{*2} p^*_{O_2}+Da_{Mb}\left(\frac{[Mb]}{\alpha p^{Mb}_{O_2 50\%}}\right) r\left(S^{Mb}-\left(1-S^{Mb}\right)\left(r p^*_{O_2}\right)\right)+rM^* \tag{B.26}$$

where r represents the ratio of equilibrium partial pressures $r \equiv \frac{p^{Mb}_{O_2 50\%}}{p^{Hb}_{O_2 50\%}}$.

Here, eight nondimensional parameters are immediately evident. The first three are the Péclet numbers Pe_{Hb}, Pe_{Mb}, and Pe_{O_2}. The Péclet number measures the ratio of advective transport to diffusive transport and is defined as follows:

$$Pe_{\cdot} \equiv \frac{UL}{D_{\cdot}} \tag{B.27}$$

where $()_{\cdot}$ refers to either the diffusivity of oxyhemoglobin, oxymyoglobin, or pure oxygen.

The next two nondimensional parameters of importance are the Damkhöler numbers Da_{Hb} and Da_{Mb}. The Damkhöler number measures the reaction rate of a substance versus its advective transport rate and is defined as follows:

$$Da_{\cdot} \equiv \frac{k_{\cdot} L}{U} \tag{B.28}$$

where $()_{\cdot}$ refers to the dissociation rate of either oxyhemoglobin or oxymyoglobin. In the context of this problem, this measures nondimensionally how quickly oxygen binds or unbinds to either oxyhemoglobin or oxymyoglobin relative to how quickly the blood plasma moves through the length of the capillary (how quickly the oxygen is advected along).

The sixth and seventh nondimensional parameters, $\frac{[Hb]}{\alpha p^{Hb}_{O_2 50\%}}$ and $\frac{[Mb]}{\alpha p^{Mb}_{O_2 50\%}}$, correspond to the oxygen concentration present in either oxyhemoglobin or oxymyoglobin, $[Hb]$ or $[Mb]$, and its diffusive rate. If the product of these parameters with their

associated Damkhöler number is $O(1)$ or greater, they are non-negligible and must be included in the analysis.

The eighth nondimensional number M^* is associated with the hypothesized constant oxygen consumption rate M. It takes the following form:

$$M^* \equiv \frac{M\,(L/U)}{\alpha p^{Hb}_{O_2 50\%}} \tag{B.29}$$

Here, M^* measures the rate at which oxygen is consumed relative to advective timescales.

The nondimensional parameters have been clearly defined and identified, thus completing the derivation for oxygen transport in the blood microcirculation system.

Appendix C

Derivation of Equations of Motion - Membrane Mechanics

For a two-dimensional, extensible (moveable and deformable) membrane, a differential line element analysis of tensile and bending stresses is required as well as a derivation of the necessary constituent relations, which are used to capture the accurate behavior of the mechanics of the membrane.

Taking a two-dimensional differential line element of length ds (see Figure C-1), drawing the appropriate forces per unit length, dF_n and dF_t, and moment per unit length, dM_z, and balancing them yields:

$$dF_n = -t_s(s+ds)\sin\left(\phi(s+ds)-\phi(s)\right) + q_s(s) - q_s(s+ds)\cos\left(\phi(s+ds)-\phi(s)\right) \tag{C.1}$$

$$dF_t = -t_s(s) + t_s(s+ds)\cos\left(\phi(s+ds)-\phi(s)\right) - q_s(s+ds)\sin\left(\phi(s+ds)-\phi(s)\right) \tag{C.2}$$

$$dM_z = -m_s(s) + m_s(s+ds) - q_s(s+ds)ds \tag{C.3}$$

where $t_s(s)$ denotes the tension along the s direction, $q_s(s)$ denotes the shear stress along the s direction, and $m_s(s)$ denotes the local moments per unit length along the s direction. It should also be noted that the total bending moment $dM_z = 0$ as the

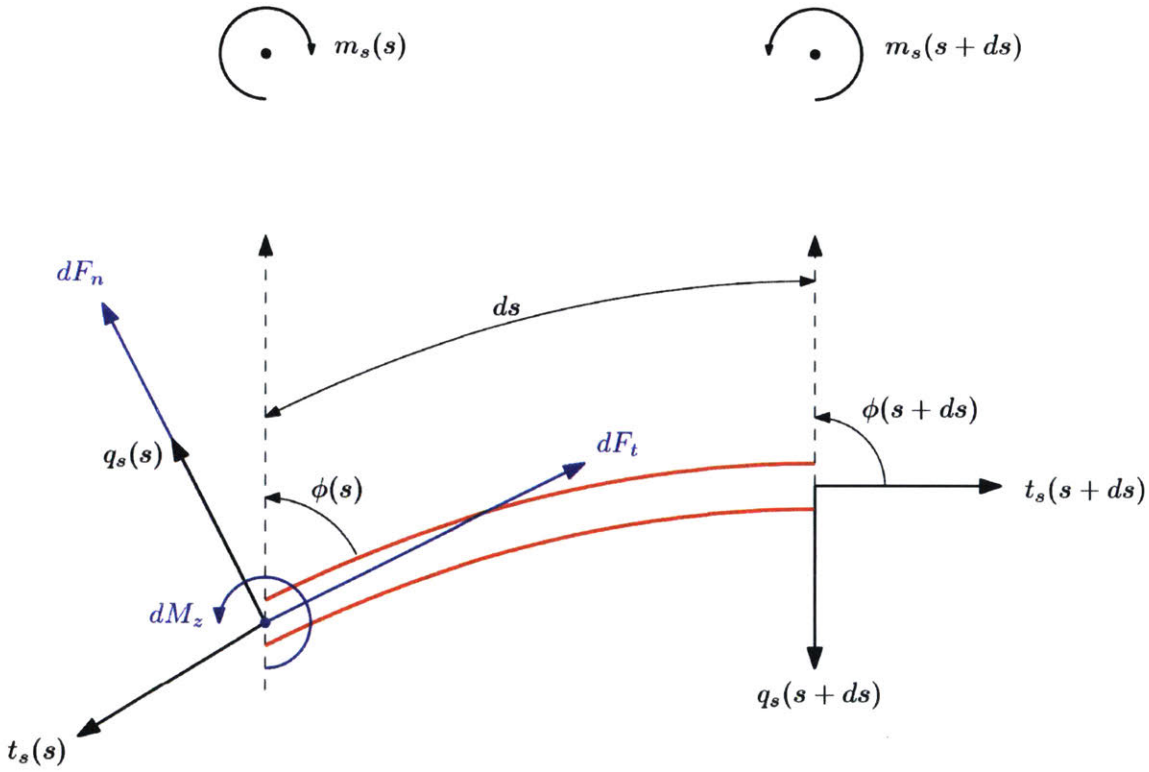


Figure C-1: Membrane tension, stress, and moments on a differential line element of length ds.

red blood cell is unanchored, thus no external moments are applied.

The differential line element is considered sufficiently small so that the difference between angles $\phi(s+ds) - \phi(s) \ll 1$, thus a small angle approximation for ϕ can be used. Considering that the forces and moment are derived as forces per unit length, a division by the differential length ds is necessary to make them compatible with the body force term in the equations of motion for the blood plasma and cytoplasm. Dividing equations (C.1), (C.2), and (C.3) by ds gives the following:

$$\frac{dF_t}{dA} = \frac{dt_s}{ds} - q_s\frac{d\phi}{ds} \tag{C.4}$$

$$\frac{dF_n}{dA} = -t_s\frac{d\phi}{ds} - \frac{dq_s}{ds} \tag{C.5}$$

$$\frac{dM_z}{dA} = \frac{dm_s}{ds} - q_s \tag{C.6}$$

In order to properly substitute these relations into the fluid equations of motion (see Appendix A), the body force vector $\vec{f} = \langle f_x, f_y \rangle$ must be derived from the local geometry at a particular point on the cellular membrane:

$$f_x = \frac{dF_t}{dA}\sin\phi - \frac{dF_n}{dA}\cos\phi \tag{C.7}$$

$$f_y = \frac{dF_t}{dA}\cos\phi + \frac{dF_n}{dA}\sin\phi \tag{C.8}$$

where $\frac{dF_t}{dA}$ and $\frac{dF_n}{dA}$ are the same as equations (C.4) and (C.5), respectively. This completes the derivation for the body forces.

However, constituent relations must be developed in order to compute t_s and q_s acting on the cellular membrane differential line segment. In order to accomplish this task, a two-dimensional leaflet body is constructed, as seen in Figure C-2 (a more general derivation is presented by Secomb[12]).

The areas per unit lengths A^- and A^+ can be approximated to first order by the local curvature k_x as $A^+ \approx A^-(1-k_x h)$. Similarly, in the membrane's unstressed state, $A_0^+ \approx A_0^-(1 - k_0 h)$, where k_0 is the unstressed membrane curvature. The isotropic

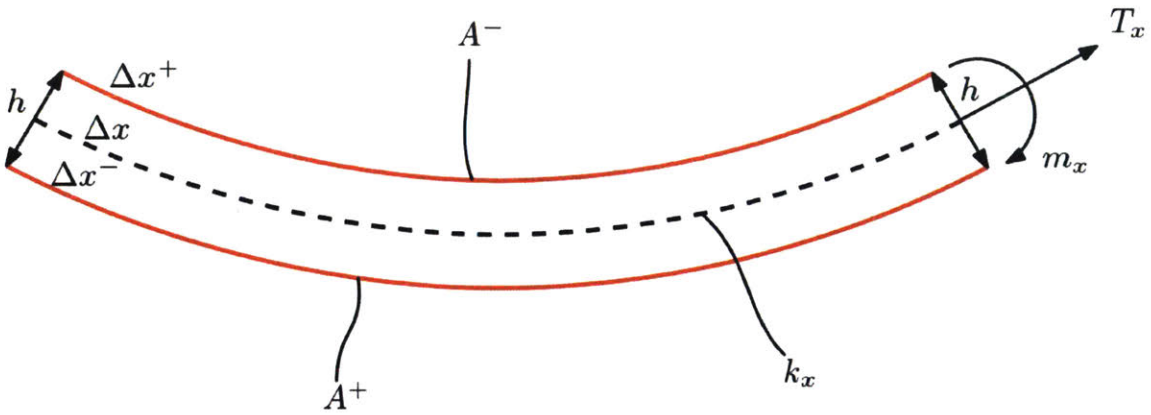


Figure C-2: Two-dimensional leaflet used for the analysis of membrane bending.

tensions on the upper and lower surfaces can be expressed by $T^+ = K(A^+/A_0^+ - 1)/2$ and $T^- = K(A^-/A_0^- - 1)/2$, respectively.

In order to include bending stresses in the membrane constituent relations, the bending moments must be analyzed about the z axis (point out of the page). Given that m_z is the bending moment in the z axis, then

$$m_z \Delta x = \frac{h}{2}\left(T^+ \Delta x^+ - T^- \Delta x^-\right) \tag{C.9}$$

which, upon substitution gives

$$m_z \approx -B\left(k_x - k_0\right) \tag{C.10}$$

where $B = Kh^2/4$ is the membrane bending modulus. Higher order terms have been neglected under, and as a result the bending moments are also isotropic.

Now, analyzing the tension force acting along the x-axis and using the fact that bending moments are isotropic yields the following relation for tension T_x:

$$T_x \Delta x = T^+ \Delta x^+ + T^- \Delta x^- \tag{C.11}$$

and so

$$T_x = T_m - Bk_x\left(k_x - k_0\right) \tag{C.12}$$

and

$$T_m = \sigma_0 + \left(\frac{\partial T_m}{\partial \alpha}\right)_T \Bigg|_{\alpha=0} \alpha = \sigma_0 + K\alpha \tag{C.13}$$

where σ_0 is the initial isotropic tension in the membrane, K is the isothermal area compressibility modulus, and α is the area change with respect to the reference state, where

$$\alpha = \lambda_x\lambda_z - 1 = \frac{dA}{dA_0} - 1 \tag{C.14}$$

and λ is the membrane extension ratio in the indicated direction. An extensive derivation of the mean tension T_m can be found in Evans and Shalak[5].

The effects of shear resistance in the membrane can also be accounted for, as also seen in Evans and Shalak[5]. This additional term T_x^s should be included in the component T_x, where

$$T_x^s = \frac{\kappa}{2}\left(\lambda_x^2 - \lambda_z^2\right) \tag{C.15}$$

where κ is the shear modulus. The principal axes of shear are aligned with the bending-induced tension according to the assumptions, thus allowing a shear in x to be approximately equal to that in z. Along with the large shear modulus, κ, this fact ensure that $\lambda_x\lambda_z \approx 1$, even though λ_z is a constant in this model. Upon substituting, T_x is found to be:

$$T_x = \sigma_0 + K\alpha - Bk_x\left(k_x - k_0\right) + \frac{\kappa}{2}\left(\lambda_x^2 - \lambda_x^{-2}\right) \tag{C.16}$$

Equation (C.16) is equivalent to equation (4.4.5) in Evans and Shalak[5], except tension due to bending has now also been accounted for. Given this relation, x can be taken as the direction of the membrane surface coordinate, s, leading to the basic subscript change and substitutions:

$$T_s = \sigma_0 + K\left(\lambda_s\lambda_s^{-1} - 1\right) - Bk_s\left(k_s - k_0\right) + \frac{\kappa}{2}\left(\lambda_s^2 - \lambda_s^{-2}\right) \tag{C.17}$$

The extension ratios and local radius of curvature can also be expressed based on local membrane geometry. Keep in mind that the membrane exists a distance R from the axis, and is R_0 away from the axis in its initial resting state. The extension ratios can be expressed as

$$\lambda_s = \frac{ds}{ds_0} \tag{C.18}$$

$$\lambda_s^{-1} = \frac{R}{R_0} \tag{C.19}$$

and the membrane local radius of curvature is

$$k_s = \frac{d\phi}{ds} \tag{C.20}$$

and substituted into equation (C.17), while noticing that $t_s = T_s$:

$$t_s = \sigma_0 + K\left(\frac{ds}{ds_0}\frac{R}{R_0} - 1\right) - B\frac{d\phi}{ds}\left(\frac{d\phi}{ds} - k_0\right) + \frac{\kappa}{2}\left(\left(\frac{ds}{ds_0}\right)^2 - \left(\frac{R}{R_0}\right)^2\right) \tag{C.21}$$

As for q_s, recall that $\frac{dM_z}{dA} = 0$, and now $m_s = m_z$ from from previous assertions, thus:

$$q_s = \frac{dm_s}{ds} = -B\frac{d^2\phi}{ds^2} \tag{C.22}$$

Equations (C.21) and (C.22) are the working constituent relations which complete the model.

The equations should be nondimensionalized so that they can be used in the body force formulation of the fluid equations of motion. Using the following nondimensional definitions

$$s^* \equiv \frac{s}{L} \qquad R^* \equiv \frac{R}{L} \qquad k_0^* \equiv Lk_0$$
$$t_s^* \equiv \frac{t_s}{\rho U^2 L} \qquad q_s^* \equiv \frac{q_s}{\rho U^2 L} \qquad m_s^* \equiv \frac{m_s}{\rho U^2 L^2}$$

and substituting into equations (C.7) and (C.8) yields

$$\boxed{f_x^* \equiv \frac{f_x}{\rho U^2} = \left(\frac{dt_s^*}{ds^*} - q_s^* \frac{d\phi}{ds^*}\right)\sin(\phi) - \left(-t_s^* \frac{d\phi}{ds^*} - \frac{dq_s^*}{ds^*}\right)\cos(\phi)} \tag{C.23}$$

$$\boxed{f_y^* \equiv \frac{f_y}{\rho U^2} = \left(\frac{dt_s^*}{ds^*} - q_s^* \frac{d\phi}{ds^*}\right)\cos(\phi) + \left(-t_s^* \frac{d\phi}{ds^*} - \frac{dq_s^*}{ds^*}\right)\sin(\phi)} \tag{C.24}$$

where

$$\boxed{t_s^* = \sigma_0^* + K^*\left(\frac{ds^*}{ds_0^*}\frac{R^*}{R_0^*} - 1\right) - B^* \frac{d\phi}{ds^*}\left(\frac{d\phi}{ds^*} - k_0^*\right) + \frac{\kappa^*}{2}\left(\left(\frac{ds^*}{ds_0^*}\right)^2 - \left(\frac{R^*}{R_0^*}\right)^2\right)} \tag{C.25}$$

and

$$\boxed{q_s^* = -B^* \frac{d^2\phi}{ds^{*2}}} \tag{C.26}$$

with the nondimensional variants of the material constants defined as follows:

$$\sigma_0^* \equiv \frac{\sigma_0}{\rho U^2 L} \qquad K^* \equiv \frac{K}{\rho U^2 L} \tag{C.26-C.27}$$

$$B^* \equiv \frac{B}{\rho U^2 L^3} \qquad \kappa^* \equiv \frac{\kappa}{\rho U^2 L} \tag{C.28-C.29}$$

The nondimentional parameters (C.26) - (C.29) describe the initial isotropic tension in the membrane, the isothermal area compressibility modulus, the membrane bending modulus, and the membrane shear modulus, respectively.

Appendix D

Non-Uniform Differentiation via Lagrange Basis Polynomials

In order to minimize the effects of numerical boundary layers, a non-uniform mesh may be used near the domain boundaries for increased mesh resolution and better gradient approximations. However, non-uniform meshing results in different finite difference weights for the derivative approximations as the distance between the node locations is no longer constant. Thus, a non-uniform stencil must be developed, and one way to accomplish this task is using Lagrange basis polynomials.

The finite difference approximation is a local approximation method (i.e. the solution is dependent only on the local neighboring points). A function at location x_i, $f(x_i)$, can be approximated locally by a polynomial:

$$\begin{aligned} f(x_i) \approx L(x_i) &= f(x_1)l_1(x) + ... + f(x_i)l_i(x) + ... + f(x_N)l_{x_N}(x) \\ &= \sum_{i=1}^{N} f(x_i)l_i(x) \end{aligned} \tag{D.1}$$

where $l_i(x)$ is the Lagrange basis polynomial:

$$l_i(x) = \frac{x - x_1}{x_i - x_1} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_N}{x_i - x_N}$$
$$= \prod_{\substack{m=1 \\ m \neq i}}^{N} \frac{x - x_m}{x_i - x_m} \tag{D.2}$$

In order to use this approximation, the polynomial is analyzed at the desired point location x_i. Neighboring points are used to generate the desired approximation. In order to generate the desired second order difference scheme, three points are used. As an example, for the first-derivative centered difference scheme (also known as the midpoint formula), points x_{i-1}, x_i, and x_{i+1} are used:

$$f'(x_i) \approx l'_{i-1}(x_i) f(x_{i-1}) + l'_i(x_i) f(x_i) + l'_{i+1}(x_i) f(x_{i+1}) \tag{D.3}$$

In order to find $l'_m(x_i) = \partial l_m(x_i)/\partial x$, the Lagrange basis polynomial must be evaluated over the points $m = \{i-1, i, i+1\}$ at x before being differentiated:

$$l_{i-1}(x) = \frac{x - x_i}{x_{i-1} - x_i} \frac{x - x_{i+1}}{x_{i-1} - x_{i+1}} = \frac{x^2 - x(x_i + x_{i+1}) + x_i x_{i+1}}{\Delta x_1 (\Delta x_1 + \Delta x_2)}$$
$$l_i(x) = \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} = -\frac{x^2 - x(x_{i-1} + x_{i+1}) + x_{i-1} x_{i+1}}{\Delta x_1 \Delta x_2}$$
$$l_{i+1}(x) = \frac{x - x_{i-1}}{x_{i+1} - x_{i-1}} \frac{x - x_i}{x_{i+1} - x_i} = \frac{x^2 - x(x_{i-1} + x_i) + x_{i-1} x_i}{\Delta x_2 (\Delta x_1 + \Delta x_2)}$$

where $\Delta x_1 = x_i - x_{i-1}$ and $\Delta x_2 = x_{i+1} - x_i$. Now, upon differentiating with respect to x:

$$l'_{i-1}(x) = \frac{2x - (x_i + x_{i+1})}{\Delta x_1 (\Delta x_1 + \Delta x_2)}$$
$$l'_i(x) = -\frac{2x - (x_{i-1} + x_{i+1})}{\Delta x_1 \Delta x_2}$$
$$l'_{i+1}(x) = \frac{2x - (x_{i-1} + x_i)}{\Delta x_2 (\Delta x_1 + \Delta x_2)}$$

Evaluating the derivatives at point $x = x_i$ yields:

$$
\begin{aligned}
l'_{i-1}(x_i) &= \frac{2x_i - (x_i + x_{i+1})}{\Delta x_1(\Delta x_1 + \Delta x_2)} = -\frac{\Delta x_2}{\Delta x_1(\Delta x_1 + \Delta x_2)} \\
l'_i(x_i) &= -\frac{2x_i - (x_{i-1} + x_{i+1})}{\Delta x_1 \Delta x_2} = \frac{\Delta x_2 - \Delta x_1}{\Delta x_1 \Delta x_2} \\
l'_{i+1}(x_i) &= \frac{2x_i - (x_{i-1} + x_i)}{\Delta x_2(\Delta x_1 + \Delta x_2)} = \frac{\Delta x_1}{\Delta x_2(\Delta x_1 + \Delta x_2)}
\end{aligned}
$$

Upon substituting these values into equation (D.3), the new approximation is:

$$
f'(x_i) \approx -\frac{\Delta x_2}{\Delta x_1(\Delta x_1 + \Delta x_2)} f(x_{i-1}) + \frac{\Delta x_2 - \Delta x_1}{\Delta x_1 \Delta x_2} f(x_i) + \frac{\Delta x_1}{\Delta x_2(\Delta x_1 + \Delta x_2)} f(x_{i+1}) \tag{D.4}
$$

For a uniform mesh where $\Delta x_1 = \Delta x_2 = \Delta x$, equation (D.4) simplifies down to the common midpoint formula:

$$
f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} \tag{D.5}
$$

where Δx corresponds to the uniform grid spacing.

For a second-derivative, the methodology is identical. Taking a centered approximation as an example, $l_m(x)$ is differentiated twice with respect to x:

$$
\begin{aligned}
l''_{i-1}(x) &= \frac{2}{\Delta x_1(\Delta x_1 + \Delta x_2)} \\
l''_i(x) &= -\frac{2}{\Delta x_1 \Delta x_2} \\
l''_{i+1}(x) &= \frac{2}{\Delta x_2(\Delta x_1 + \Delta x_2)}
\end{aligned}
$$

Upon substitution into a second order centered approximation, the result is:

$$f''(x_i) \approx l''_{i-1}(x_i)f(x_{i-1}) + l''_i(x_i)f(x_i) + l''_{i+1}(x_i)f(x_{i+1})$$
$$\approx \frac{2f(x_{i-1})}{\Delta x_1(\Delta x_1 + \Delta x_2)} - \frac{2f(x_i)}{\Delta x_1 \Delta x_2} + \frac{2f(x_{i+1})}{\Delta x_2(\Delta x_1 + \Delta x_2)} \tag{D.6}$$

Once again, in the case of a uniform mesh where $\Delta x_1 = \Delta x_2 = \Delta x$, the common second-order finite difference stencil is recovered:

$$f''(x_i) \approx \frac{f(x_{i-1}) + -2f(x_i) + f(x_{i+1})}{\Delta x^2} \tag{D.6}$$

where Δx similarly corresponds to the uniform grid spacing listed previously.

This Lagrange basis polynomial approach can be carried out for any desired stencil, including forward and backward differencing, for any order derivative, provided enough points are used in the approximation. A follow-up stability analysis should be performed for less common stencils that are not well studied in order to ensure that the approximation will not cause numerical divergence.

For convenience, the additional forward and backward first derivative stencils implemented in the code are listed below. The second-order accurate, first derivative forward difference is

$$\boxed{f'(x_i) \approx -\frac{2\Delta x_1 + \Delta x_2}{\Delta x_1 (\Delta x_1 + \Delta x_2)} f(x_i) + \frac{\Delta x_1 + \Delta x_2}{\Delta x_1 \Delta x_2} f(x_{i+1}) - \frac{\Delta x_1}{\Delta x_2 (\Delta x_1 + \Delta x_2)} f(x_{i+2})} \tag{D.7}$$

where $\Delta x_1 = x_{i+1} - x_{i+2}$ and $\Delta x_2 = x_{i+2} - x_{i+1}$. The second-order accurate, first derivative backward difference is

$$\boxed{f'(x_i) \approx \frac{\Delta x_2}{\Delta x_1 (\Delta x_1 + \Delta x_2)} f(x_{i-2}) - \frac{\Delta x_1 + \Delta x_2}{\Delta x_1 \Delta x_2} f(x_{i-1}) + \frac{\Delta x_1 + 2\Delta x_2}{\Delta x_2 (\Delta x_1 + \Delta x_2)} f(x_i)} \tag{D.8}$$

where $\Delta x_1 = x_{i-1} - x_{i-2}$ and $\Delta x_2 = x_i - x_{i-1}$.

The central, forward, and backward second derivative stencils implemented in

this code are also listed below. The second-order accurate, second derivative central difference is

$$\boxed{f''(x_i) \approx \frac{2}{\Delta x_1 (\Delta x_1 + \Delta x_2)} f(x_{i-1}) - \frac{2}{\Delta x_1 \Delta x_2} f(x_i) + \frac{2}{\Delta x_2 (\Delta x_1 + \Delta x_2)} f(x_{i+1})} \tag{D.9}$$

where $\Delta x_1 = x_i - x_{i-1}$ and $\Delta x_2 = x_{i+1} - x_i$. The second-order accurate, second derivative forward difference is

$$\boxed{\begin{aligned} f''(x_i) \approx & \frac{6\Delta x_1 + 4\Delta x_2 + 2\Delta x_3}{\Delta x_1 (\Delta x_1 + \Delta x_2)(\Delta x_1 + \Delta x_2 + \Delta x_3)} f(x_i) \\ & - \frac{4(\Delta x_2 + \Delta x_2) + 2\Delta x_3}{\Delta x_1 \Delta x_2 (\Delta x_2 + \Delta x_3)} f(x_{i+1}) \\ & + \frac{\Delta x_1 + 2(\Delta x_2 + \Delta x_3)}{(\Delta x_1 + \Delta x_2)\Delta x_2 \Delta x_3} f(x_{i+2}) \\ & - \frac{4\Delta x_1 + 2\Delta x_2}{(\Delta x_1 + \Delta x_2 + \Delta x_3)(\Delta x_2 + \Delta x_3)\Delta x_3} f(x_{i+3}) \end{aligned}} \tag{D.10}$$

where $\Delta x_1 = x_{i+1} - x_i$, $\Delta x_2 = x_{i+2} - x_{i+1}$, and $\Delta x_3 = x_{i+3} - x_{i+2}$. The second-order accurate, second derivative backward difference is

$$\boxed{\begin{aligned} f''(x_i) \approx & -\frac{2\Delta x_2 + 4\Delta x_3}{\Delta x_1 (\Delta x_1 + \Delta x_2)(\Delta x_2 + \Delta x_2 + \Delta x_3)} f(x_{i-3}) \\ & + \frac{2(\Delta x_1 + \Delta x_2) + 4\Delta x_3}{\Delta x_1 \Delta x_2 (\Delta x_2 + \Delta x_3)} f(x_{i-2}) \\ & - \frac{2\Delta x_1 + 4(\Delta x_2 + \Delta x_3)}{(\Delta x_1 + \Delta x_2)\Delta x_2 \Delta x_3} f(x_{i-1}) \\ & + \frac{2\Delta x_1 + 4\Delta x_2 + 6\Delta x_3}{(\Delta x_1 + \Delta x_2 + \Delta x_3)(\Delta x_2 + \Delta x_3)\Delta x_3} f(x_i) \end{aligned}} \tag{D.11}$$

where $\Delta x_1 = x_{i-2} - x_{i-3}$, $\Delta x_2 = x_{i-1} - x_{i-2}$, and $\Delta x_3 = x_i - x_{i-1}$.

Appendix E

MATLAB Code

Capillary_Driver.m

```
clear

% ----------------
% Input Parameters
% ----------------

% Time loop controls
unsteady = true;            % steady vs unsteady
Nt = 1500;                    % Number of time-steps
dt = 0.0005;                  % time-step dt
ssb = 1;                    % steady-state break; 0 == false, 1 == true

% Newton loop controls
maxIter = 15;
tol = 1e-8;

% Plotter Controls
plotType = 2;       % (1 for Surface, 2 for Contours)
contourLines = 51;
resCheck = 0;       % (0 == off, 1 == check between Newton iterations)

% -------------------------
% Nondimensional Parameters
% -------------------------
% Reynolds Number (1 for simple convergence tests)
Re_rbc = 1;
Re_p = 1;

% Membrane Constituent Relation Constants (1 for simple convergence tests)
sigmap = 1;
K_bup = 1;
K_shp = 1;
Bp = 1;

% Spring test constant (bf-model)
kappa = 0;
```

```
% ------------------------------
% Grid Parameters and Generation
% ------------------------------

% X inputs
Lx = 2;
nx = 80;
Lxli = 0;
Lxri = Lx;
xextr = 0.2;
xextn = 9;
[xc,hx] = GridExt(nx,0,Lx,Lxli,Lxri,xextr,xextn);
% xp1 = 1;
% xp2 = 3;
% xppts = 100;
% [xc,hx] = GridPack(xc,xp1,xp2,xppts);
nxc = nx;

% Y (radius) inputs
% Capillary
R_cap = 1;
nycap = 40;
Lyli = 0;
Lyri = R_cap-0.1;
yextr = 0.2;
yextn = 17;
[yc_cap,hy_cap] = GridExt(nycap,0,R_cap,Lyli,Lyri,yextr,yextn);
% yp1 = 0;
% yp2 = Ly-0.25;
% yppts = 50;
% [yc,hy] = GridPack(yc,yp1,yp2,yppts);
% yc = yc(2:end-1);
% hy = hy(2:end-1);
nyc_cap = length(yc_cap);
nt_cap = nyc_cap*nxc;

% Lagrangian Mesh (RBC)
lpts = 100;
RL = 0.75;
k0p = RL;
theta = (0:2*pi/(lpts):pi)';
XL = RL.*cos(theta)+1;
YL = RL.*sin(theta);
XLe = XL;
YLe = YL;
dXL = diff([XL;XL(1)]);
dYL = diff([YL;YL(1)]);
dSL = sqrt(dXL.^2+dYL.^2);

% DO NOT TOUCH
xcd = xc;
ycd = yc_cap(1:end);
nyc = length(ycd);
nt = nxc*nyc;
[coords_cap, ~, ~] = GridGenX(xcd',yc_cap');
x_cap = coords_cap(:,1);
y_cap = coords_cap(:,2);

% Boundary Node Index Points (lexicographic in x)
% Capillary
```

```
bccap1 = 1:nxc:(nyc_cap-1)*nxc+1;
bccap2 = 1:nxc;
bccap3 = nxc:nxc:nxc*nyc_cap;
bccap4 = (nyc_cap-1)*nxc+1:nxc*nyc_cap;

figure(1); clf;
hold on
plot(x_cap,y_cap,'ok')
plot(XL,YL,'-or')
xlabel('x');
ylabel('y');
axis([0 Lx 0 R_cap])
hold off

% ------------------------
% Boundary Condition Input
% ------------------------
% Wall 1:
vnorm1 = 1;
vtan1 = 0;
% Wall 2:
sfc2 = 0;
vtan2 = 0;
% Wall 3:
vnorm3 = 0;
vtan3 = 0;
% Wall 4:
sfc4 = R_cap;
vtan4 = 0;

% ------------------------
% Initial Condition Input
% ------------------------
Psi = zeros(nxc*nyc_cap,1);
w = zeros(nxc*nyc_cap,1);

% -------------
% Initial Plots
% -------------
xr=reshape(x_cap,nxc,nyc);
yr=reshape(y_cap,nxc,nyc);
xr_cap=reshape(coords_cap(:,1),nxc,nyc_cap);
yr_cap=reshape(coords_cap(:,2),nxc,nyc_cap);
Psir=reshape(Psi,nxc,nyc_cap);
wr=reshape(w,nxc,nyc_cap);

% Surface Plots
if plotType == 1
figure(2); clf;
hold on
xlabel('x');
ylabel('y');
surf(xr,yr,Psir)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
view(45,30)
hold off

figure(3); clf;
```

```
hold on
xlabel('x');
ylabel('y');
surf(xr,yr,wr)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
view(45,30)
hold off
% Contour Plots
elseif plotType == 2
figure(2); clf;
hold on
xlabel('x');
ylabel('y');
contour(xr_cap,yr_cap,Psir,contourLines,'linewidth',2)
plot([XLe;XLe(1)],[YLe;YLe(1)],'-k','linewidth',2)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
hold off

figure(3); clf;
hold on
xlabel('x');
ylabel('y');
contour(xr_cap,yr_cap,wr,contourLines,'linewidth',2)
plot([XLe;XLe(1)],[YLe;YLe(1)],'-k','linewidth',2)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
hold off
end

fprintf(1,'%s \n','Press any key to continue with calculation.');
pause

% Set up membrane variables
% Coordinates
skx0 = XL;
sky0 = YL;
skx = skx0;
sky = sky0;
% Total mesh points
nskx = length(skx);
nsky = length(sky);
nskt = nskx+nsky;

% Construct residual differentiation matrices
fprintf(1,'%s \n','Constructing Residual Differentiation Matrices.');
[Dx,Dy] = Diff1Matrices_Cap(xc,yc_cap);
D = speye(length(XL),nxc*nyc_cap);
Dsmear = D';
L = speye(nskt+2*nt_cap);
Res = zeros(nskt+2*nt_cap,1);

% Capillary edges (symmetry line and capillary wall)
L = Matrix_CapillaryEdge(L,bccap2,2,dt,hx,hy_cap);
```

```
L = Matrix_CapillaryEdge(L,bccap4,4,dt,hx,hy_cap);

% ------------------
% Start of time loop
% ------------------
fprintf(1,'%s \n','Starting time-loop.');

tic
for t = 1:Nt
% --------------------------------
% Lagrangian Body Force Calculation
% --------------------------------
% Constituent relationships go here
[tss,dtss,qss,dqss,dsk,phik,dphik] = ...
RBCMeshQuants(skx0,sky0,skx,sky,sigmap,K_bup,K_shp,Bp,k0p);
fx = (dtss-qss.*dphik).*sin(phik)+(tss.*dphik-dqss).*cos(phik);
fy = (dtss-qss.*dphik).*cos(phik)-(tss.*dphik+dqss).*sin(phik);
%     Fx = kappa.*(XLe - XL);
%     Fy = kappa.*(YLe - YL);

% ----------------------------
% Eularian Body Force Smearing
% ----------------------------
[D,Dsmear] = DdsMat(D,Dsmear,xc,yc_cap,skx,sky,dsk,hx,hy_cap);
Dx_til = D*Dx;
Dy_til = D*Dy;
Bx = Dsmear*fx;
By = Dsmear*fy;
%     [dBxdy,dBydx] = DDUV(Bx,By,hx,hy_cap);
dBxdy = Dy*Bx;
dBydx = Dx*By;
curlB = dBydx-dBxdy;
%     fx = Dsmear*Fx;
%     fy = Dsmear*Fy;
%     [dfxdy,dfydx] = DDUV(fx,fy,hx,hy_cap);
%     curlf = dfydx-dfxdy;

% --------------------
% Start of Newton loop
% --------------------
% Update previous time-step values
w_old = w;
%     p_old = p;
%     S_old = S;
u_old = Dy*Psi;
v_old = -Dx*Psi;
%     [u_old,v_old] = SFtoVelX(Psi,hx,hy_cap);
k = 1;      % Newton loop counter initialization
while k <= maxIter
% Update Lagrangian Mesh for Implicit Membrane Feedback
[D,Dsmear] = DdsMat(D,Dsmear,xc,yc_cap,skx,sky,dsk,hx,hy_cap);
Dx_til = D*Dx;
Dy_til = D*Dy;

% Find Eularian points inside RBC Membrane
[in_rbc,dsk,~,skx,sky] = inPolyPeriodic(x_cap,y_cap,skx,sky,0);

% Update Differentiation Matrix for Implicit Feedback
L = Matrix_Capillary_Updater(L,skx,sky,Re_rbc,Re_p,dt,hx,hy_cap,Dx_til,Dy_til,in_rbc);

% Construct Residuals
```

```
% Capillary
Res = Residual_Capillary_SFV(Res,Psi,w,w_old,curlf,us,vs,...
skx,sky,Re_rbc,Re_pl,dt,hx,hy_cap,Dx_til,Dy_til,in_rbc);
Res = Residual_CapillaryEdge(Res,Psi,w,sfc,bccap2,2,...
dt,hx,hy_cap);
Res = Residual_CapillaryEdge(Res,Psi,w,sfc,bccap4,4,...
dt,hx,hy_cap);
Res = Residual_CapillaryInletOutlet(Res);
Res = Residual_CapillaryInletOutlet(Res);

% Calculate and reassign solution deltas
dU = L\-Res;
dskx = reshape(dU(1:2:2*nskx-1),nskx,1);
dsky = reshape(dU(2:2:2*nsky),nsky,1);
dPsi = reshape(dU(nskt+1:2:2*nt_cap-1),nt_cap,1);
dw = reshape(dU(nskt+2:2:2*nt_cap),nt_cap,1);

% Reconstruct individual residual vectors
Rskx = reshape(abs(Res(1:2:2*nskx-1)),nskx,1);
Rsky = reshape(abs(Res(2:2:2*nsky)),nsky,1);
RPsi = reshape(abs(Res(nskt+1:2:2*nt_cap-1,1)),nxc,nyc_cap);
Rw = reshape(abs(Res(nskt+2:2:2*nt_cap,1)),nxc,nyc_cap);

% Check max residuals
Rskxmax = full(max(abs(Rskx)));
Rskymax = full(max(abs(Rsky)));
RPsimax = full(max(abs(Res(nskt+1:2:2*nt_cap-1,1))));
Rwmax = full(max(abs(Res(nskt+2:2:2*nt_cap,1))));

% Calculate rms of deltas for adequate convergence
drms = sqrt(norm(dskx)^2 + norm(dsky)^2 + norm(dPsi)^2 + norm(dw)^2);

% Print intermediate result checks
fprintf(1,strcat('iter=%i    Rskxmax=%9.3e    Rskymax=%9.3e    ',...
'RPsimax=%9.3e    Rwmax=%9.3e    drms=%9.3e \n'), ...
k,Rskxmax,Rskymax,RPsimax,Rwmax,drms);

% Graphical check of entire domain's residuals
if resCheck == 1
figure(1); clf;
hold on
subplot(1,2,1)
surf(xr_cap,yr_cap,RPsi)
xlabel('x');
ylabel('y');
subplot(1,2,2)
surf(xr_cap,yr_cap,Rw)
xlabel('x');
ylabel('y');
hold off

pause
end

% Update solution values for next iteration
skx = skx + dskx;
sky = sky + dsky;
Psi = Psi + dPsi;
w = w + dw;

% Break look if rms of deltas is within specified tolerance
```

```
if drms <= tol
fprintf(1,'%s %d %s %d \n','Iteration count = ',k,', drms = ',drms);
break
end

% Update Newton loop iteration counter
k=k+1;
end

% ------------------
% End of Newton loop
% ------------------

fprintf(1,'%s %d \n','Time counter = ',t);

% -------
% Plotter
% -------
Psir=reshape(Psi,nxc,nyc_cap);
wr=reshape(w,nxc,nyc_cap);
ur=reshape(us,nxc,nyc_cap);
vr=reshape(vs,nxc,nyc_cap);

if unsteady == true
if plotType == 1
figure(2); clf;
hold on
xlabel('x');
ylabel('y');
surf(xr,yr,Psir)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
view(45,30)
hold off

figure(3); clf;
hold on
xlabel('x');
ylabel('y');
surf(xr,yr,wr)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
view(45,30)
hold off

drawnow;
elseif plotType == 2
figure(2); clf;
hold on
xlabel('x');
ylabel('y');
contour(xr_cap,yr_cap,Psir,contourLines,'linewidth',2)
plot([XLe;XLe(1)],[YLe;YLe(1)],'-k','linewidth',2)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
```

```
hold off

figure(3); clf;
hold on
xlabel('x');
ylabel('y');
contour(xr_cap,yr_cap,wr,contourLines,'linewidth',2)
plot([XLe;XLe(1)],[YLe;YLe(1)],'-k','linewidth',2)
colormap(jet)
colorbar
grid on
axis([0 Lx 0 R_cap]);
hold off

drawnow;
end

figure(6); clf;
hold on
plot(ur(1,:),yr_cap(1,:),ur(floor(nxc/2),:),yr_cap(floor(nxc/2),:),...
ur(end,:),yr_cap(end,:),'linewidth',2)
xlabel('U');
ylabel('y/R');
grid on
axis([0 1.5 0 1]);
hold off

if k > maxIter
fprintf(1,'%s \n','Maximum iteration exceeded.');
break
elseif k == 1 && ssb == 1
fprintf(1,'%s \n','Steady-state reached, proceeding to next time-step.');
break
end
end
end
toc

% ----------------
% End of time loop
% ----------------
```

GridExt.m

```
function [pts,hpts] = GridExt(nt,Linit,Ldom,Lext0,Lext1,ratio,nr)
%UNTITLED8 Summary of this function goes here
%   Detailed explanation goes here

dpts_new = (Lext1-Lext0)/(nt-1);
pts_new = Lext0:dpts_new:Lext1;

if Lext0 == Linit && Lext1 == Ldom
pts = pts_new;
hpts = diff(pts);
return;
end

if nr == 1 && Lext1 == Ldom
pts = [Linit pts_new];
return;
elseif nr == 1 && Lext0 == Linit
pts = [pts_new Ldom];
return;
end

if(ratio==1)
alpha  = 1.0;
factor = 1.0/nr;
else
texp = 1/(nr-1);
alpha = ratio^texp;
factor = (1.0-alpha)/(1.0-alpha^nr);
end

if Lext1 ~= Ldom
delta=(Ldom-Lext1)*factor;
for i=2:1:nr+1
pts_new = [pts_new pts_new(end)+delta];
delta = delta*alpha;
end
end
if Lext0 ~= Linit
delta=(Lext0-Linit)*factor;
for i=2:1:nr+1
pts_new = [pts_new(1)-delta pts_new];
delta = delta*alpha;
end
end
pts = pts_new;
hpts = diff(pts);
end
```

GridGenX.m

```
function [coords nodevals bcnodes] = GridGenX(xc,yc)
%UNTITLED8 Summary of this function goes here
%   Detailed explanation goes here

nx = length(xc);
ny = length(yc);

xgrid = [];
ygrid = [];
nodevals = [];
bcnodes = [];

k=1;
for j = 1:1:ny
xgrid = [xgrid xc'];
for i = 1:1:nx
ygrid = [ygrid yc(j)];
nodevals = [nodevals k];
if j == 1 || j == ny || i == 1 || i == nx
bcnodes = [bcnodes k];
end
k=k+1;
end
end

coords = [xgrid' ygrid'];

end
```

Diff1Matrices_Cap.m

```
function [Dx,Dy] = Diff1Matrices_Cap(xcap,ycap)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

nx = length(xcap);
ny = length(ycap);
hx = diff(xcap);
hy = diff(ycap);

% Initialize Dx,Dy
Dx = zeros(nx*ny);
Dy = zeros(nx*ny);

% First Derivative Weights (central)
a1x = -hx(2:end)./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
b1x = (hx(2:end)-hx(1:end-1))./(hx(1:end-1).*hx(2:end));
c1x = hx(1:end-1)./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a1y = -hy(2:end)./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
b1y = (hy(2:end)-hy(1:end-1))./(hy(1:end-1).*hy(2:end));
c1y = hy(1:end-1)./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% First Derivative Weights (forwards)
%     a1xf = -(hx(2:end)+2.*hx(1:end-1))./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
%     b1xf = (hx(2:end)+hx(1:end-1))./(hx(1:end-1).*hx(2:end));
%     c1xf = -hx(1:end-1)./(hx(2:end).*(hx(1:end-1)+hx(2:end)));
%     a1yf = -(hy(2:end)+2.*hy(1:end-1))./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
%     b1yf = (hy(2:end)+hy(1:end-1))./(hy(1:end-1).*hy(2:end));
%     c1yf = -hy(1:end-1)./(hy(2:end).*(hy(1:end-1)+hy(2:end)));

% First Derivative Weights (backwards)
%     a1xb = hx(2:end)./(hx(1:end-1).*(hx(1:end-1)+hx(2:end)));
%     b1xb = -(hx(2:end)+hx(1:end-1))./(hx(1:end-1).*hx(2:end));
%     c1xb = (2.*hx(2:end)+hx(1:end-1))./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a1yb = hy(2:end)./(hy(1:end-1).*(hy(1:end-1)+hy(2:end)));
b1yb = -(hy(2:end)+hy(1:end-1))./(hy(1:end-1).*hy(2:end));
c1yb = (2.*hy(2:end)+hy(1:end-1))./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% Dx
for i = 1:1:nx
for j = 1:1:ny
if i==1 % periodic
Dx((j-1)*nx+i,(j)*nx) = a1x(1);
Dx((j-1)*nx+i,(j-1)*nx+i) = b1x(1);
Dx((j-1)*nx+i,(j-1)*nx+i+1) = c1x(1);
elseif i==nx % periodic
Dx((j-1)*nx+i,(j-1)*nx+i-1) = a1x(end);
Dx((j-1)*nx+i,(j-1)*nx+i) = b1x(end);
Dx((j-1)*nx+i,(j-1)*nx+1) = c1x(end);
else % typical central appx
Dx((j-1)*nx+i,(j-1)*nx+i-1) = a1x(i-1);
Dx((j-1)*nx+i,(j-1)*nx+i) = b1x(i-1);
Dx((j-1)*nx+i,(j-1)*nx+i+1) = c1x(i-1);
end
end
end
```

```
% Dy
for i = 1:1:nx
for j = 1:1:ny
if j==1 % use axisymmetry
Dy((j-1)*nx+i,(j)*nx+i) = a1y(1)+c1y(1);
Dy((j-1)*nx+i,(j-1)*nx+i) = b1y(1);
elseif j==ny % need backward diff
Dy((j-1)*nx+i,(j-1)*nx+i) = c1yb(end);
Dy((j-1)*nx+i,(j-2)*nx+i) = b1yb(end);
Dy((j-1)*nx+i,(j-3)*nx+i) = a1yb(end);
else % typical central appx
Dy((j-1)*nx+i,(j-2)*nx+i) = a1y(j-1);
Dy((j-1)*nx+i,(j-1)*nx+i) = b1y(j-1);
Dy((j-1)*nx+i,(j)*nx+i) = c1y(j-1);
end
end
end

Dx = sparse(Dx);
Dy = sparse(Dy);
end
```

RBCMeshQuants.m

```
function [tss,dtss,qss,dqss,dsk,phik,dphik] = ...
RBCMeshQuants(skx0,sky0,skx,sky,sigma,K_bu,K_sh,B,k0)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

%% Geometry Relations
% derivatives of elementary quantities
dsk5 = sqrt((skx(2:end)-skx(1:end-1)).^2 + (sky(2:end)-sky(1:end-1)).^2);
phik5 = acos((sky(2:end)-sky(1:end-1))./dsk5(1:end));

dsk5e = [dsk5(1); dsk5; dsk5(end)];
phik5e = [-phik5(1); phik5; phik5(end)];

dsk = (dsk5e(1:end-1)+dsk5e(2:end))./2;

phik = (phik5e(2:end).*dsk5e(1:end-1)+phik5e(1:end-1).*dsk5e(2:end))./...
(dsk5e(1:end-1)+dsk5e(2:end));

dphik = (phik5e(2:end)-phik5e(1:end-1))./((dsk5e(1:end-1)+dsk5e(2:end))./2);

ddphik5 = (dphik(2:end)-dphik(1:end-1))./dsk5;
ddphik5e = [ddphik5(1); ddphik5; ddphik5(end)];    % may require - sign at start/end
ddphik = (ddphik5e(2:end).*dsk5e(1:end-1)+ddphik5e(1:end-1).*dsk5e(2:end))./...
(dsk5e(1:end-1)+dsk5e(2:end));

dddphik = (ddphik5e(2:end)-ddphik5e(1:end-1))./((dsk5e(1:end-1)+dsk5e(2:end))./2);


% derivatives of non-elementary quantities
skye = [sky(2); sky; sky(end-1)];
sky0e = [sky0(2); sky0; sky0(end-1)];

sky5e = skye(1:end-1)+skye(2:end);
sky05e = sky0e(1:end-1)+sky0e(2:end);
sky505er = sky5e./sky05e;

dsk05 = sqrt((skx0(2:end)-skx0(1:end-1)).^2 + (sky0(2:end)-sky0(1:end-1)).^2);
dsk05e = [dsk05(1); dsk05; dsk05(end)];
dsds0e = dsk5e./dsk05e;

dsds0 = (dsds0e(2:end).*dsk5e(1:end-1)+dsds0e(1:end-1).*dsk5e(2:end))./...
(dsk5e(1:end-1)+dsk5e(2:end));
RR0 = sky./sky0;

% first deriv
ddsds0 = (dsds0e(2:end)-dsds0e(1:end-1))./((dsk5e(2:end)+dsk5e(1:end-1))./2);
dRR0 = (sky505er(2:end)-sky505er(1:end-1))./((dsk5e(1:end-1)+dsk5e(2:end))./2);
% squared deriv
ddsds02 = (dsds0e(2:end).^2-dsds0e(1:end-1).^2)./((dsk5e(2:end)+dsk5e(1:end-1))./2);
dRR02 = (sky505er(2:end).^2-sky505er(1:end-1).^2)./((dsk5e(1:end-1)+dsk5e(2:end))./2);

%% Membrane Constituent Relations
tss = sigma+K_bu.*(dsds0.*RR0-1)+K_sh./2.*(dsds0.^2+RR0.^2)-B.*...
dphik.*(dphik-k0);
dtss = K_bu.*(ddsds0.*RR0+dsds0.*dRR0)+K_sh./2.*(ddsds02-dRR02)-...
B.*ddphik.*(2.*dphik-k0);
qss = -B.*ddphik;
```

```
dqss = -B.*dddphik;

end
```

DdsMat.m

```
% function [D,D2,Ds,Dsmear] = DdsMat(D,D2,Ds,Dsmear,xc,yc,XL,YL,ds,hx,hy)
function [D,Dsmear] = DdsMat(D,Dsmear,xc,yc,XL,YL,ds,hx,hy)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
L = length(XL);
nx = length(hx)+1;
ny = length(hy)+1;

for l = 1:1:L
for i = 1:1:nx
for j = 1:1:ny
if i == nx && j == ny
D(l,((j-1)*nx+i)) = dfr((xc(i)-XL(l)),hx(end)).*dfr((yc(j)-YL(l)),hy(end));
Dsmear(((j-1)*nx+i),l) = (1/(hx(end)*hy(end))).*...
dfr((xc(i)-XL(l)),hx(end)).*dfr((yc(j)-YL(l)),hy(end)).*ds(l);
elseif i == nx
D(l,((j-1)*nx+i)) = dfr((xc(i)-XL(l)),hx(end)).*dfr((yc(j)-YL(l)),hy(j));
Dsmear(((j-1)*nx+i),l) = (1/(hx(end)*hy(j))).*...
dfr((xc(i)-XL(l)),hx(end)).*dfr((yc(j)-YL(l)),hy(j)).*ds(l);
elseif j == ny
D(l,((j-1)*nx+i)) = dfr((xc(i)-XL(l)),hx(i)).*dfr((yc(j)-YL(l)),hy(end));
Dsmear(((j-1)*nx+i),l) = (1/(hx(i)*hy(end))).*...
dfr((xc(i)-XL(l)),hx(i)).*dfr((yc(j)-YL(l)),hy(end)).*ds(l);
else
D(l,((j-1)*nx+i)) = dfr((xc(i)-XL(l)),hx(i)).*dfr((yc(j)-YL(l)),hy(j));
Dsmear(((j-1)*nx+i),l) = (1/(hx(i)*hy(j))).*...
dfr((xc(i)-XL(l)),hx(i)).*dfr((yc(j)-YL(l)),hy(j)).*ds(l);
end
end
end
end

end
```

dfr.m

```
function [d] = dfr(r,h)
% dfr(r,h)
%
% r: input (X-x) or (Y-y); (Lagrangian - Eulerian) value
% h: input local dx or dy; local Eulerian grid displacement value
if abs(r/h) <= 1
d = (1/8)*(3-2*abs(r/h)+sqrt(1+4*abs(r/h)-4*abs(r/h)^2));
elseif abs(r/h) > 1 && abs(r/h) <= 2
d = (1/8)*(5-2*abs(r/h)-sqrt(-7+12*abs(r/h)-4*abs(r/h)^2));
elseif abs(r/h) > 2
d = 0;
end
end
```

inPolyPeriodic.m

```
function [intPntMat,dsk,phik,skx,sky] = inPolyPeriodic(xcap_mat,ycap_mat,...
skx,sky,plotter)
% [intPntMat,dsk,phik] = inPolyPeriodic(xcap_mat,ycap_mat,skx,sky)
%
% Finds points inside periodically wrapped curve for capillary domain.
%
% INPUTS:
% xcap_mat: x-coordinates of Eulerian mesh
% ycap_mat: y-coordinates of Eulerian mesh
% skx: x-coordinates of Lagrangian mesh
% sky: y-coordinates of Lagrangian mesh
% plotter: 0 or 1, off or on (recommended to keep at 0 unless this function
% is used for periodic testing)
%
% OUTPUTS:
% intPntMat: matrix of 0 and 1 values indicating if value inside curve skx,
% sky
% dsk: corrected distance between curve points for periodic wrapping
% phik: corrected phi angle at curve points for periodic wrapping

xMax = max(xcap_mat);

for i=1:1:length(skx)
if skx(i)>xMax
skx(i)=skx(i)-xMax;
end
end

dskx = diff(skx);
dsky = diff(sky);

for i=1:1:length(dskx)
if abs(dskx(i))>xMax/2
dskx(i) = skx(i+1)-(skx(i)+xMax);
end
end

dsk5 = sqrt((dskx).^2 + (dsky).^2);
phik5 = acos((sky(2:end)-sky(1:end-1))./dsk5(1:end));

dsk5e = [dsk5(1); dsk5; dsk5(end)];
phik5e = [-phik5(1); phik5; phik5(end)];

dsk = (dsk5e(1:end-1)+dsk5e(2:end))./2;
phik = (phik5e(2:end).*dsk5e(1:end-1)+phik5e(1:end-1).*dsk5e(2:end))./...
(dsk5e(1:end-1)+dsk5e(2:end));

[~,index] = min(skx);

if index~=length(skx)
skx_t1 = [skx(1:index); skx(index+1)-xMax; skx(index+1)-xMax; skx(1); skx(1)];
sky_t1 = [sky(1:index); sky(index+1); -0.1; -0.1; sky(1)];
skx_t2 = [skx(index+1:end); skx(end); skx(index)+xMax; skx(index)+xMax; skx(index+1)];
sky_t2 = [sky(index+1:end); -0.1; -0.1; sky(index); sky(index+1)];

if plotter==1
figure(1); clf;
```

```
hold on
plot(skx_t1,sky_t1,'-or',skx_t2,sky_t2,'-or','linewidth',2);
end

in1 = inpolygon(xcap_mat,ycap_mat,skx_t1,sky_t1);
in2 = inpolygon(xcap_mat,ycap_mat,skx_t2,sky_t2);

intPntMat = logical(in1+in2);
else
skx_t1 = [skx(1:end); skx(end); skx(1); skx(1)];
sky_t1 = [sky(1:end); -0.1; - 0.1; sky(1)];

if plotter==1
figure(1); clf;
hold on
plot(skx_t1,sky_t1,'-or','linewidth',2);
end

in = inpolygon(xcap_mat,ycap_mat,skx_t1,sky_t1);
intPntMat = logical(in);
end
end
```

Matrix_Capillary_Updater.m

```
function [L] = Matrix_Capillary_Updater(L,skx,sky,Re_rbc,Re_pl,dt,hx,hy,Dx_til,Dy_til,in_rbc)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Total Grid Points
nx = length(hx);
ny = length(hy);
% number of Lagrangian X points
nskx = length(skx);
% number of Lagrangian Y points
nsky = length(sky);
% total number of Lagrangian (X,Y) points * 2
nskt = nskx+nsky;
% nDxr == Dx rows, corresponds to Lagrangian points
% nDxc == Dx columns, corresponds to Eulerian points
[nDxr,nDxc] = size(Dx_til);

% Non-dimensional Vectors
Re_vec = zeros(nx*ny,1);

% Second Derivative Weights (centered)
a2x = 2./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
b2x = -2./(hx(1:end-1).*hx(2:end));
c2x = 2./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a2y = 2./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
b2y = -2./(hy(1:end-1).*hy(2:end));
c2y = 2./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% Second Derivative Weights (forwards)
a2xf = (6.*hx(1:end-2)+4.*hx(2:end-1)+2.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xf = -(4.*(hx(1:end-2)+hx(2:end-1))+2.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xf = (hx(1:end-2)+2.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xf = -(4.*hx(1:end-2)+2.*hx(2:end-1))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yf = (6.*hy(1:end-2)+4.*hy(2:end-1)+2.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
b2yf = -(4.*(hy(1:end-2)+hy(2:end-1))+2.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yf = (hy(1:end-2)+2.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
d2yf = -(4.*hy(1:end-2)+2.*hy(2:end-1))./...
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% Second Derivative Weights (backwards)
a2xb = -(2.*hx(2:end-1)+4.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xb = (2.*(hx(1:end-2)+hx(2:end-1))+4.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xb = -(2.*hx(1:end-2)+4.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xb = (2.*hx(1:end-2)+4.*hx(2:end-1)+6.*hx(3:end))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yb = -(2.*hy(2:end-1)+4.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
```

```
b2yb = (2.*(hy(1:end-2)+hy(2:end-1))+4.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yb = -(2.*hy(1:end-2)+4.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
d2yb = (2.*hy(1:end-2)+4.*hy(2:end-1)+6.*hy(3:end))./...
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% Assign domains' Reynolds numbers into vector format
for i=1:1:nx
for j=1:1:ny
if in_rbc((j-1)*nx+i)==1
Re_vec((j-1)*nx+i) = Re_rbc;
else
Re_vec((j-1)*nx+i) = Re_pl;
end
end
end

% Lagrangian Mesh Implicit Feedback Derivatives
for l = 1:1:nDxr
L(2*l-1,2*l-1) = 1;
L(2*l,2*l) = 1;
for k = 1:1:nDxc
L(2*l-1,nskt+4*k-3) = -dt.*Dy_til(l,k);
L(2*l,nskt+4*k-3) = dt.*Dx_til(l,k);
end
end

% Lexicographical Ordering: x followed by y, reduce bandwidth
for i = 2:1:nx-1
for j = 2:1:nx-1
%% Residual 1
% Check if points are inside or outside cell, determines if
% discretization scheme needs to be adjusted for second
% derivative approximations.

% Point lies completely inside/outside cell s.t. centered
% approximation for second derivative is physically valid
if (in_rbc((j-1)*nx+i+1)==1 && in_rbc((j-1)*nx+i-1)==1 && ...
in_rbc((j-2)*nx+i)==1 && in_rbc((j)*nx+i)==1) || ...
(in_rbc((j-1)*nx+i+1)==0 && in_rbc((j-1)*nx+i-1)==0 && ...
in_rbc((j-2)*nx+i)==0 && in_rbc((j)*nx+i)==0)

% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = (b2x(1) + b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = c2x(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = a2x(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(b2x(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2x(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2x(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
```

```
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);

% Point lies to left of cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-1)*nx+i+1))
% Point lies to right of cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-1)*nx+i-1))
break;
% Point lies below cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j)*nx+i))
% Point lies above cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
break;
% Residual
else
% No periodic overlap required for second derivative
if (((j-1)*nx+i)-3)-((j-1)*nx+1)>=0
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-2)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-3)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-1
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-2)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
```

```
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-2
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-1)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-3
% Residual 1
```

```
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-1)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-2)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

end

end
% Point lies above cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
% No periodic overlap required for second derivative
if (((j-1)*nx+i)-3)-((j-1)*nx+1)>=0
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-2)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-3)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
```

```
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-1
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-2)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-2
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-1)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
```

```
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-3
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-1)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-2)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

end

% Residual
else
% Periodic accounting for left of cell edge -
% first-derivative must be upwinded/downwinded
```

```
% accordingly

% No periodic overlap required for second derivative
if (((j-1)*nx+i)-3)-((j-1)*nx+1)>=0
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-2)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-3)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-1
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-2)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
```

```
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-2
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-1)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-3
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx)-1) = ...
c2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-1)-1) = ...
b2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*(j*nx-2)-1) = ...
a2xb(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(d2xb(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*(j*nx-2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2xb(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
```

```
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);

end
end
% Point lies to right of cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-3)*nx+i-1))
% Point lies below cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j)*nx+i))
% Point lies above cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
break;
% Residual
else
% No periodic overlap required for second derivative
if (((j-1)*nx+i)+3)-(j*nx)<=0
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+3)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==1
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
d2xf(1);
```

```
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==2
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+2)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==3
```

```
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+3)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

end
end
% Point lies above cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
% No periodic overlap required for second derivative
if (((j-1)*nx+i)+3)-(j*nx)<=0
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+3)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
```

```
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==1
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==2
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+2)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
```

```
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==3
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+3)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

end

% Residual
else
% No periodic overlap required for second derivative
if (((j-1)*nx+i)+3)-(j*nx)<=0
```

```
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+3)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==1
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);

% Periodic overlap required for first/second derivative
```

```
elseif (((j-1)*nx+i)+3)-(j*nx)==2
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+2)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==3
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+1)-1) = ...
b2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+2)-1) = ...
c2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+3)-1) = ...
d2xf(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
c2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
a2y(j-1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(a2xf(1)+b2y(j-1));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+2)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+3)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2xf(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2y(j-1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2y(j-1);
```

```
end
end
% Point lies below cell edge
elseif  (in_rbc((j-1)*nx+i)~=in_rbc((j)*nx+i))
% Point lies above cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
break;
% Residual
else
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(b2x(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
c2x(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
a2x(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-2)*nx+i)-1) = ...
c2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-3)*nx+i)-1) = ...
b2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-4)*nx+i)-1) = ...
a2yb(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(b2x(1)+d2yb(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2x(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2x(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-3)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yb(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-4)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2yb(j-2);

end
% Point lies above cell edge
% Residual
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
% Residual 1
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)-1) = ...
(b2x(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i+1)-1) = ...
c2x(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i-1)-1) = ...
a2x(1);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j)*nx+i)-1) = ...
b2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+1)*nx+i)-1) = ...
c2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j+2)*nx+i)-1) = ...
d2yf(j-2);
L(nskt+2*((j-1)*nx+i)-1,nskt+2*((j-1)*nx+i)) = 1;

% Residual 2
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i)) = ...
1-(dt/Re_vec((j-1)*nx+i)).*(b2x(1)+a2yf(j-2));
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i+1)) = ...
```

```
-(dt/Re_vec((j-1)*nx+i)).*c2x(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j-1)*nx+i-1)) = ...
-(dt/Re_vec((j-1)*nx+i)).*a2x(1);
L(nskt+2*((j-1)*nx+i),nskt+2*((j)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*b2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+1)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*c2yf(j-2);
L(nskt+2*((j-1)*nx+i),nskt+2*((j+2)*nx+i)) = ...
-(dt/Re_vec((j-1)*nx+i)).*d2yf(j-2);

end
end
end
end
```

Matrix_CapillaryEdge.m

```
function [L] = Matrix_CapillaryEdge(L,nodes,side,dt,hx,hy_cap)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Total Grid Points
nx = length(hx)+1;
lbc = length(nodes);

if side == 2
% Lexicographical Ordering: x followed by y, reduce bandwidth
for i = 1:1:lbc
% Psi
L(nodes(i).*2-1,nodes(i).*2-1) = 1;

% Omega
L(nodes(i).*2,nodes(i).*2) = 1;
end
elseif side == 4
% Lexicographical Ordering: x followed by y, reduce bandwidth
for i = 1:1:lbc
% Psi
L(nodes(i).*2-1,nodes(i).*2-1) = 1;

% Omega
L(nodes(i).*2,nodes(i).*2) = 1;
L(nodes(i).*2,nodes(i).*2-1) = -2/hy_cap(end)^2;
L(nodes(i).*2,(nodes(i)-nx).*2-1) = 2/hy_cap(end)^2;
end
end
end
```

Matrix_CapillaryInletOutlet.m

```
function [L] = Matrix_CapillaryInletOutlet(L,skx,sky,Re_rbc,Re_pl,dt,hx,hy,Dx_til,Dy_til,in_rbc)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Total Grid Points
nx = length(hx);
ny = length(hy);
% number of Lagrangian X points
nskx = length(skx);
% number of Lagrangian Y points
nsky = length(sky);
% total number of Lagrangian (X,Y) points * 2
nskt = nskx+nsky;
% nDxr == Dx rows, corresponds to Lagrangian points
% nDxc == Dx columns, corresponds to Eulerian points
[nDxr,nDxc] = size(Dx_til);

% Non-dimensional Vectors
Re_vec = zeros(nx*ny,1);

% Second Derivative Weights (centered)
a2x = 2./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
b2x = -2./(hx(1:end-1).*hx(2:end));
c2x = 2./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a2y = 2./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
b2y = -2./(hy(1:end-1).*hy(2:end));
c2y = 2./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% Second Derivative Weights (forwards)
a2xf = (6.*hx(1:end-2)+4.*hx(2:end-1)+2.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xf = -(4.*(hx(1:end-2)+hx(2:end-1))+2.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xf = (hx(1:end-2)+2.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xf = -(4.*hx(1:end-2)+2.*hx(2:end-1))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yf = (6.*hy(1:end-2)+4.*hy(2:end-1)+2.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
b2yf = -(4.*(hy(1:end-2)+hy(2:end-1))+2.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yf = (hy(1:end-2)+2.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
d2yf = -(4.*hy(1:end-2)+2.*hy(2:end-1))./...
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% Second Derivative Weights (backwards)
a2xb = -(2.*hx(2:end-1)+4.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xb = (2.*(hx(1:end-2)+hx(2:end-1))+4.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xb = -(2.*hx(1:end-2)+4.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xb = (2.*hx(1:end-2)+4.*hx(2:end-1)+6.*hx(3:end))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yb = -(2.*hy(2:end-1)+4.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
```

```
b2yb = (2.*(hy(1:end-2)+hy(2:end-1))+4.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yb = -(2.*hy(1:end-2)+4.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
d2yb = (2.*hy(1:end-2)+4.*hy(2:end-1)+6.*hy(3:end))./...
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% Assign domains' Reynolds numbers into vector format
for i=1:1:nx
for j=1:1:ny
if in_rbc(bcnodes(i))==1
Re_vec(bcnodes(i)) = Re_rbc;
else
Re_vec(bcnodes(i)) = Re_pl;
end
end
end

% Lagrangian Mesh Implicit Feedback Derivatives
for l = 1:1:nDxr
L(2*l-1,2*l-1) = 1;
L(2*l,2*l) = 1;
for k = 1:1:nDxc
L(2*l-1,nskt+4*k-3) = -dt.*Dy_til(l,k);
L(2*l,nskt+4*k-3) = dt.*Dx_til(l,k);
end
end

% Lexicographical Ordering: x followed by y, reduce bandwidth
for i = 2:1:nx-1
for j = 2:1:nx-1
%% Residual 1
% Check if points are inside or outside cell, determines if
% discretization scheme needs to be adjusted for second
% derivative approximations.

% Point lies completely inside/outside cell s.t. centered
% approximation for second derivative is physically valid
if (in_rbc(bcnodes(i)+1)==1 && in_rbc(bcnodes(i)+nx-1)==1 && ...
in_rbc(bcnodes(i)-nx)==1 && in_rbc(bcnodes(i)+nx)==1) || ...
(in_rbc(bcnodes(i)+1)==0 && in_rbc(bcnodes(i)+nx-1)==0 && ...
in_rbc(bcnodes(i)-nx)==0 && in_rbc(bcnodes(i)+nx)==0)

% Residual 1
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i))-1) = (b2x(1) + b2y(j-1));
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+1)-1) = c2x(1);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+nx-1)-1) = a2x(1);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+nx)-1) = a2y(j-1);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)-nx)-1) = c2y(j-1);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i))) = 1;

% Residual 2
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i))) = ...
1-(dt/Re_vec(bcnodes(i))).*(b2x(1)+b2y(j-1));
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+1)) = ...
-(dt/Re_vec(bcnodes(i))).*c2x(1);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+nx-1)) = ...
-(dt/Re_vec(bcnodes(i))).*a2x(1);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+nx)) = ...
-(dt/Re_vec(bcnodes(i))).*a2y(j-1);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)-nx)) = ...
```

```
-(dt/Re_vec(bcnodes(i))).*c2y(j-1);

% Point lies to left of cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+1))
% Point lies to right of cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+nx-1))
break;
% Point lies below cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+nx))
% Point lies above cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-nx))
break;
% Residual
else


end
% Point lies above cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-nx))


% Residual
else


end
% Point lies to right of cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-2*nx-1))
% Point lies below cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+nx))
% Point lies above cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-nx))
break;
% Residual
else


end
% Point lies above cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-nx))


% Residual
else


end
% Point lies below cell edge
elseif  (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+nx))
% Point lies above cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-nx))
break;
% Residual
else
% Residual 1
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i))-1) = ...
(b2x(1)+d2yb(j-2));
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+1)-1) = ...
c2x(1);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+nx-1)-1) = ...
a2x(1);
```

```
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)-nx)-1) = ...
c2yb(j-2);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)-2*nx)-1) = ...
b2yb(j-2);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)-3*nx)-1) = ...
a2yb(j-2);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i))) = 1;

% Residual 2
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i))) = ...
1-(dt/Re_vec(bcnodes(i))).*(b2x(1)+d2yb(j-2));
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+1)) = ...
-(dt/Re_vec(bcnodes(i))).*c2x(1);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+nx-1)) = ...
-(dt/Re_vec(bcnodes(i))).*a2x(1);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)-nx)) = ...
-(dt/Re_vec(bcnodes(i))).*c2yb(j-2);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)-2*nx)) = ...
-(dt/Re_vec(bcnodes(i))).*b2yb(j-2);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)-3*nx)) = ...
-(dt/Re_vec(bcnodes(i))).*a2yb(j-2);

end
% Point lies above cell edge
% Residual
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-nx))
% Residual 1
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i))-1) = ...
(b2x(1)+a2yf(j-2));
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+1)-1) = ...
c2x(1);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+nx-1)-1) = ...
a2x(1);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+nx)-1) = ...
b2yf(j-2);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+2*nx)-1) = ...
c2yf(j-2);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i)+3*nx)-1) = ...
d2yf(j-2);
L(nskt+2*(bcnodes(i))-1,nskt+2*(bcnodes(i))) = 1;

% Residual 2
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i))) = ...
1-(dt/Re_vec(bcnodes(i))).*(b2x(1)+a2yf(j-2));
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+1)) = ...
-(dt/Re_vec(bcnodes(i))).*c2x(1);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+nx-1)) = ...
-(dt/Re_vec(bcnodes(i))).*a2x(1);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+nx)) = ...
-(dt/Re_vec(bcnodes(i))).*b2yf(j-2);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+2*nx)) = ...
-(dt/Re_vec(bcnodes(i))).*c2yf(j-2);
L(nskt+2*(bcnodes(i)),nskt+2*(bcnodes(i)+3*nx)) = ...
-(dt/Re_vec(bcnodes(i))).*d2yf(j-2);

end
end
end
end
```

Residual_Capillary_SFV.m

```
function [Res] = Residual_Capillary_SFV(Res,Psi,w,w_old,curlf,us,vs,...
skx,sky,Re_rbc,Re_pl,dt,hx,hy,Dx_til,Dy_til,in_rbc)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Total Grid Points
% Eulerian
nx = length(hx)+1;
ny = length(hy)+1;
% Lagrangian
nskx = length(skx);
nsky = length(sky);
nskt = nskx+nsky;

% Non-dimensional Vectors
Re_vec = zeros(nx*ny,1);

% First Derivative Weights (forwards)
a1xf = -(hx(2:end)+2.*hx(1:end-1))./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
b1xf = (hx(2:end)+hx(1:end-1))./(hx(1:end-1).*hx(2:end));
c1xf = -hx(1:end-1)./(hx(2:end).*(hx(1:end-1)+hx(2:end)));
a1yf = -(hy(2:end)+2.*hy(1:end-1))./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
b1yf = (hy(2:end)+hy(1:end-1))./(hy(1:end-1).*hy(2:end));
c1yf = -hy(1:end-1)./(hy(2:end).*(hy(1:end-1)+hy(2:end)));

% First Derivative Weights (backwards)
a1xb = hx(2:end)./(hx(1:end-1).*(hx(1:end-1)+hx(2:end)));
b1xb = -(hx(2:end)+hx(1:end-1))./(hx(1:end-1).*hx(2:end));
c1xb = (2.*hx(2:end)+hx(1:end-1))./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a1yb = hy(2:end)./(hy(1:end-1).*(hy(1:end-1)+hy(2:end)));
b1yb = -(hy(2:end)+hy(1:end-1))./(hy(1:end-1).*hy(2:end));
c1yb = (2.*hy(2:end)+hy(1:end-1))./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% Second Derivative Weights (centered)
a2x = 2./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
b2x = -2./(hx(1:end-1).*hx(2:end));
c2x = 2./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a2y = 2./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
b2y = -2./(hy(1:end-1).*hy(2:end));
c2y = 2./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% Second Derivative Weights (forwards)
a2xf = (6.*hx(1:end-2)+4.*hx(2:end-1)+2.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xf = -(4.*(hx(1:end-2)+hx(2:end-1))+2.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xf = (hx(1:end-2)+2.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xf = -(4.*hx(1:end-2)+2.*hx(2:end-1))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yf = (6.*hy(1:end-2)+4.*hy(2:end-1)+2.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
b2yf = -(4.*(hy(1:end-2)+hy(2:end-1))+2.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yf = (hy(1:end-2)+2.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
d2yf = -(4.*hy(1:end-2)+2.*hy(2:end-1))./...
```

```
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% Second Derivative Weights (backwards)
a2xb = -(2.*hx(2:end-1)+4.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xb = (2.*(hx(1:end-2)+hx(2:end-1))+4.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xb = -(2.*hx(1:end-2)+4.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xb = (2.*hx(1:end-2)+4.*hx(2:end-1)+6.*hx(3:end))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yb = -(2.*hy(2:end-1)+4.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
b2yb = (2.*(hy(1:end-2)+hy(2:end-1))+4.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yb = -(2.*hy(1:end-2)+4.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
d2yb = (2.*hy(1:end-2)+4.*hy(2:end-1)+6.*hy(3:end))./...
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% u+, u-, v+, v-
up = max(us,0);
vp = max(vs,0);
um = min(us,0);
vm = min(vs,0);

% Reynolds number vector
for i=1:1:nx
for j=1:1:ny
if in_rbc((j-1)*nx+i)==1
Re_vec((j-1)*nx+i) = Re_rbc;
else
Re_vec((j-1)*nx+i) = Re_pl;
end
end
end

% Lagrnagian (finish comment)
for l = 1:1:nskx
Res(2*l-1,1) = skx(l)-dt.*Dy_til(l,:)*Psi;
Res(2*l,1) = sky(l)+dt.*Dx_til(l,:)*Psi;
end

% Lexicographical Ordering: y followed by x, reduce bandwidth
for i = 2:1:nx-1
for j = 2:1:ny-1
%% Residual 1
% Check if points are inside or outside cell, determines if
% discretization scheme needs to be adjusted for second
% derivative approximations.

% Point lies completely inside/outside cell s.t. centered
% approximation for second derivative is physically valid
if (in_rbc((j-1)*nx+i+1)==1 && in_rbc((j-1)*nx+i-1)==1 && ...
in_rbc((j-2)*nx+i)==1 && in_rbc((j)*nx+i)==1) || ...
(in_rbc((j-1)*nx+i+1)==0 && in_rbc((j-1)*nx+i-1)==0 && ...
in_rbc((j-2)*nx+i)==0 && in_rbc((j)*nx+i)==0)

% Residual 1
Res(nskt+2*((j-1)*nx+i)-1,1) = (b2x(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
```

```
c2x(1).*Psi((j-1)*nx+i+1) + a2x(1).*Psi((j-1)*nx+i-1) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+2*((j-1)*nx+i),1) = w((j-1)*nx+i) + dt.*(up((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
um((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2))) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((b2x(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
c2x(1).*w((j-1)*nx+i+1) + a2x(1).*w((j-1)*nx+i-1)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Point lies to left of cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-1)*nx+i+1))
% Point lies to right of cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-1)*nx+i-1))
break;
% Point lies below cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j)*nx+i))
% Point lies above cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
break;
% Residual
else
% No periodic overlap required for second derivative
if (((j-1)*nx+i)-3)-((j-1)*nx+1)>=0
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + ...
a2yb(j-2).*Psi((j-4)*nx+i) + c2xb(1).*Psi((j-1)*nx+i-1) + ...
b2xb(1).*Psi((j-1)*nx+i-2) + a2xb(1).*Psi((j-1)*nx+i-3) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + ...
a2yb(j-2).*w((j-4)*nx+i) + c2xb(1).*w((j-1)*nx+i-1) + ...
b2xb(1).*w((j-1)*nx+i-2) + a2xb(1).*w((j-1)*nx+i-3)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-1
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + ...
a2yb(j-2).*Psi((j-4)*nx+i) + c2xb(1).*Psi((j-1)*nx+i-1) + ...
b2xb(1).*Psi((j-1)*nx+i-2) + a2xb(1).*Psi(j*nx) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + ...
a2yb(j-2).*w((j-4)*nx+i) + c2xb(1).*w((j-1)*nx+i-1) + ...
```

```
b2xb(1).*w((j-1)*nx+i-2) + a2xb(1).*w(j*nx)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-2
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + ...
a2yb(j-2).*Psi((j-4)*nx+i) + c2xb(1).*Psi((j-1)*nx+i-1) + ...
b2xb(1).*Psi(j*nx) + a2xb(1).*Psi(j*nx-1) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old(j*nx)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + ...
a2yb(j-2).*w((j-4)*nx+i) + c2xb(1).*w((j-1)*nx+i-1) + ...
b2xb(1).*w(j*nx) + a2xb(1).*w(j*nx-1)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-3
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + ...
a2yb(j-2).*Psi((j-4)*nx+i) + c2xb(1).*Psi(j*nx) + ...
b2xb(1).*Psi(j*nx-1) + a2xb(1).*Psi(j*nx-2) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old(j*nx-1)+b1xb(i-1).*w_old(j*nx)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + ...
a2yb(j-2).*w((j-4)*nx+i) + c2xb(1).*w(j*nx) + ...
b2xb(1).*w(j*nx-1) + a2xb(1).*w(j*nx-2)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

end

end
% Point lies above cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
% No periodic overlap required for second derivative
if (((j-1)*nx+i)-3)-((j-1)*nx+1)>=0
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+1)*nx+i) + ...
d2yf(j-2).*Psi((j+2)*nx+i) + c2xb(1).*Psi((j-1)*nx+i-1) + ...
b2xb(1).*Psi((j-1)*nx+i-2) + a2xb(1).*Psi((j-1)*nx+i-3) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+2*((j-1)*nx+i),1) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+1)*nx+i) + ...
```

```
d2yf(j-2).*w((j+2)*nx+i) + c2xb(1).*w((j-1)*nx+i-1) + ...
b2xb(1).*w((j-1)*nx+i-2) + a2xb(1).*w((j-1)*nx+i-3)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-1
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+1)*nx+i) + ...
d2yf(j-2).*Psi((j+2)*nx+i) + c2xb(1).*Psi((j-1)*nx+i-1) + ...
b2xb(1).*Psi((j-1)*nx+i-2) + a2xb(1).*Psi(j*nx) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+2*((j-1)*nx+i),1) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+1)*nx+i) + ...
d2yf(j-2).*w((j+2)*nx+i) + c2xb(1).*w((j-1)*nx+i-1) + ...
b2xb(1).*w((j-1)*nx+i-2) + a2xb(1).*w(j*nx)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-2
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+1)*nx+i) + ...
d2yf(j-2).*Psi((j+2)*nx+i) + c2xb(1).*Psi((j-1)*nx+i-1) + ...
b2xb(1).*Psi(j*nx) + a2xb(1).*Psi(j*nx-1) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+2*((j-1)*nx+i),1) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old(j*nx)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+1)*nx+i) + ...
d2yf(j-2).*w((j+2)*nx+i) + c2xb(1).*w((j-1)*nx+i-1) + ...
b2xb(1).*w(j*nx) + a2xb(1).*w(j*nx-1)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-3
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1)+a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+1)*nx+i) + ...
d2yf(j-2).*Psi((j+2)*nx+i) + c2xb(1).*Psi(j*nx) + ...
b2xb(1).*Psi(j*nx-1) + a2xb(1).*Psi(j*nx-2) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+2*((j-1)*nx+i),1) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old(j*nx-1)+b1xb(i-1).*w_old(j*nx)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1)+a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+1)*nx+i) + ...
d2yf(j-2).*w((j+2)*nx+i) + c2xb(1).*w(j*nx) + ...
b2xb(1).*w(j*nx-1) + a2xb(1).*w(j*nx-2)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);
```

```
end

% Residual
else
% Periodic accounting for left of cell edge -
% first-derivative must be upwinded/downwinded
% accordingly

% No periodic overlap required for second derivative
if (((j-1)*nx+i)-3)-((j-1)*nx+1)>=0
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
c2xb(1).*Psi((j-1)*nx+i-1) + b2xb(1).*Psi((j-1)*nx+i-2) + ...
a2xb(1).*Psi((j-1)*nx+i-3) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
c2xb(1).*w((j-1)*nx+i-1) + b2xb(1).*w((j-1)*nx+i-2) + ...
a2xb(1).*w((j-1)*nx+i-3)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-1
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
c2xb(1).*Psi((j-1)*nx+i-1) + b2xb(1).*Psi((j-1)*nx+i-2) + ...
a2xb(1).*Psi(j*nx) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
c2xb(1).*w((j-1)*nx+i-1) + b2xb(1).*w((j-1)*nx+i-2) + ...
a2xb(1).*w(j*nx)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-2
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
c2xb(1).*Psi((j-1)*nx+i-1) + b2xb(1).*Psi(j*nx) + ...
a2xb(1).*Psi(j*nx-1) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old(j*nx)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
c2xb(1).*w((j-1)*nx+i-1) + b2xb(1).*w(j*nx) + ...
a2xb(1).*w(j*nx-1)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);
```

```
% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)-3)-((j-1)*nx+1)==-3
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (d2xb(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
c2xb(1).*Psi(j*nx) + b2xb(1).*Psi(j*nx-1) + ...
a2xb(1).*Psi(j*nx-2) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*us((j-1)*nx+i).*...
(a1xb(i-1).*w_old(j*nx-1)+b1xb(i-1).*w_old(j*nx)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((d2xb(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
c2xb(1).*w(j*nx) + b2xb(1).*w(j*nx-1) + ...
a2xb(1).*w(j*nx-2)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

end
end
% Point lies to right of cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-3)*nx+i-1))
% Point lies below cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j)*nx+i))
% Point lies above cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
break;
% Residual
else
% No periodic overlap required for second derivative
if (((j-1)*nx+i)+3)-(j*nx)<=0
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + a2yb(j-2).*Psi((j-4)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+i+2) + ...
d2xf(1).*Psi((j-1)*nx+i+3) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2)) + ...
dt.*(vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + a2yb(j-2).*w((j-4)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+i+2) + ...
d2xf(1).*w((j-1)*nx+i+3)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==1
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + a2yb(j-2).*Psi((j-4)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+i+2) + ...
d2xf(1).*Psi((j-1)*nx+1) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2)) + ...
dt.*(vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + a2yb(j-2).*w((j-4)*nx+i) + ...
```

```
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+i+2) + ...
d2xf(1).*w((j-1)*nx+1)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==2
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + a2yb(j-2).*Psi((j-4)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+1) + ...
d2xf(1).*Psi((j-1)*nx+2) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+1)) + ...
dt.*(vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i))) - ..
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + a2yb(j-2).*w((j-4)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+1) + ...
d2xf(1).*w((j-1)*nx+2)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==3
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + a2yb(j-2).*Psi((j-4)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+1) + c2xf(1).*Psi((j-1)*nx+2) + ...
d2xf(1).*Psi((j-1)*nx+3) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+1)+c1xf(i).*w_old((j-1)*nx+2)) + ...
dt.*(vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i))) - ..
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + a2yb(j-2).*w((j-4)*nx+i) + ...
b2xf(1).*w((j-1)*nx+1) + c2xf(1).*w((j-1)*nx+2) + ...
d2xf(1).*w((j-1)*nx+3)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

end
end
% Point lies above cell edge
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
% No periodic overlap required for second derivative
if (((j-1)*nx+i)+3)-(j*nx)<=0
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+2)*nx+i) + d2yf(j-2).*Psi((j+2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+i+2) + ...
d2xf(1).*Psi((j-1)*nx+i+3) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+2)*nx+i) + d2yf(j-2).*w((j+2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+i+2) + ...
d2xf(1).*w((j-1)*nx+i+3)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==1
% Residual 1
```

```
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+2)*nx+i) + d2yf(j-2).*Psi((j+2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+i+2) + ...
d2xf(1).*Psi((j-1)*nx+1) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+2)*nx+i) + d2yf(j-2).*w((j+2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+i+2) + ...
d2xf(1).*w((j-1)*nx+1)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==2
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+2)*nx+i) + d2yf(j-2).*Psi((j+2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+1) + ...
d2xf(1).*Psi((j-1)*nx+2) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+1)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+2)*nx+i) + d2yf(j-2).*w((j+2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+1) + ...
d2xf(1).*w((j-1)*nx+2)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==3
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+2)*nx+i) + d2yf(j-2).*Psi((j+2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+1) + c2xf(1).*Psi((j-1)*nx+2) + ...
d2xf(1).*Psi((j-1)*nx+3) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+1)+c1xf(i).*w_old((j-1)*nx+2)) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+2)*nx+i) + d2yf(j-2).*w((j+2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+1) + c2xf(1).*w((j-1)*nx+2) + ...
d2xf(1).*w((j-1)*nx+3)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

end

% Residual
else
% No periodic overlap required for second derivative
if (((j-1)*nx+i)+3)-(j*nx)<=0
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+i+2) + ...
d2xf(1).*Psi((j-1)*nx+i+3) + w((j-1)*nx+i);

% Residual 2
```

```
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+i+2) + ...
d2xf(1).*w((j-1)*nx+i+3)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==1
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+i+2) + ...
d2xf(1).*Psi((j-1)*nx+1) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+i+2) + ...
d2xf(1).*w((j-1)*nx+1)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==2
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+i+1) + c2xf(1).*Psi((j-1)*nx+1) + ...
d2xf(1).*Psi((j-1)*nx+2) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+1)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+i+1) + c2xf(1).*w((j-1)*nx+1) + ...
d2xf(1).*w((j-1)*nx+2)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

% Periodic overlap required for first/second derivative
elseif (((j-1)*nx+i)+3)-(j*nx)==3
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (a2xf(1) + b2y(j-1)).*Psi((j-1)*nx+i) + ...
c2y(j-1).*Psi((j)*nx+i) + a2y(j-1).*Psi((j-2)*nx+i) + ...
b2xf(1).*Psi((j-1)*nx+1) + c2xf(1).*Psi((j-1)*nx+2) + ...
d2xf(1).*Psi((j-1)*nx+3) + w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*...
us((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+1)+c1xf(i).*w_old((j-1)*nx+2)) + ...
dt.*(vp((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i)) + ...
vm((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((a2xf(1) + b2y(j-1)).*w((j-1)*nx+i) + ...
c2y(j-1).*w((j)*nx+i) + a2y(j-1).*w((j-2)*nx+i) + ...
b2xf(1).*w((j-1)*nx+1) + c2xf(1).*w((j-1)*nx+2) + ...
```

```
d2xf(1).*w((j-1)*nx+3)) - w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

end
end
% Point lies below cell edge
elseif  (in_rbc((j-1)*nx+i)~=in_rbc((j)*nx+i))
% Point lies above cell edge
if (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
break;
% Residual
else
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (b2x(1) + d2yb(j-2)).*Psi((j-1)*nx+i) + ...
c2yb(j-2).*Psi((j-2)*nx+i) + b2yb(j-2).*Psi((j-3)*nx+i) + a2yb(j-2).*Psi((j-4)*nx+i) +...
c2x(1).*Psi((j-1)*nx+i+1) + a2x(1).*Psi((j-1)*nx+i-1) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*(up((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
um((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2))) + ...
dt.*(vs((j-1)*nx+i).*(a1yb(j-1).*w_old((j-3)*nx+i)+b1yb(j-1).*w_old((j-2)*nx+i)+c1yb(j-1).*w_old((j-1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((b2x(1) + d2yb(j-2)).*w((j-1)*nx+i) + ...
c2yb(j-2).*w((j-2)*nx+i) + b2yb(j-2).*w((j-3)*nx+i) + a2yb(j-2).*w((j-4)*nx+i) +...
c2x(1).*w((j-1)*nx+i+1) + a2x(1).*w((j-1)*nx+i-1)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

end
% Point lies above cell edge
% Residual
elseif (in_rbc((j-1)*nx+i)~=in_rbc((j-2)*nx+i))
% Residual 1
Res(nskt+((j-1)*nx+i)*2-1) = (b2x(1) + a2yf(j-2)).*Psi((j-1)*nx+i) + ...
b2yf(j-2).*Psi((j)*nx+i) + c2yf(j-2).*Psi((j+1)*nx+i) + d2yf(j-2).*Psi((j+2)*nx+i) +...
c2x(1).*Psi((j-1)*nx+i+1) + a2x(1).*Psi((j-1)*nx+i-1) + ...
w((j-1)*nx+i);

% Residual 2
Res(nskt+((j-1)*nx+i)*2) = w((j-1)*nx+i) + dt.*(up((j-1)*nx+i).*...
(a1xb(i-1).*w_old((j-1)*nx+i-2)+b1xb(i-1).*w_old((j-1)*nx+i-1)+c1xb(i-1).*w_old((j-1)*nx+i)) + ...
um((j-1)*nx+i).*(a1xf(i).*w_old((j-1)*nx+i)+b1xf(i).*w_old((j-1)*nx+i+1)+c1xf(i).*w_old((j-1)*nx+i+2))) + ...
dt.*(vs((j-1)*nx+i).*(a1yf(j).*w_old((j-1)*nx+i)+b1yf(j).*w_old((j)*nx+i)+c1yf(j).*w_old((j+1)*nx+i))) - ...
(dt/Re_vec((j-1)*nx+i)).*((b2x(1) + a2yf(j-2)).*w((j-1)*nx+i) + ...
b2yf(j-2).*w((j)*nx+i) + c2yf(j-2).*w((j+1)*nx+i) + d2yf(j-2).*w((j+2)*nx+i) +...
c2x(1).*w((j-1)*nx+i+1) + a2x(1).*w((j-1)*nx+i-1)) - ...
w_old((j-1)*nx+i) - dt.*curlf((j-1)*nx+i);

end
end
end
end
```

Residual_CapillaryEdge.m

```
function [Res] = Residual_CapillaryEdge(Res,Psi,w,sfc,nodes,side,...
dt,hx,hy_cap)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Total Grid Points
nx = length(hx)+1;
lbc = length(nodes);

if side == 2
for i = 1:1:lbc
% Psi
Res(nodes(i).*2-1) = Psi(nodes(i)) - sfc;

% Omega
Res(nodes(i).*2) = w(nodes(i));
end
elseif side == 4
% Lexicographical Ordering: y followed by x, reduce bandwidth
for i = 1:1:lbc
% Psi
Res(nodes(i).*2-1) = Psi(nodes(i)) - sfc;

% Omega
Res(nodes(i).*2) = w(nodes(i)) + ...
2./(hy_cap(end).^2).*(Psi(nodes(i)-nx) - Psi(nodes(i)));
end
end
end
```

Residual_CapillaryInletOutlet.m

```
function [Res] = Residual_CapillaryInletOutlet(Res,Psi,w,w_old,curlf,us,vs,...
Re_rbc,Re_pl,bcnodes,side,dt,hx,hy)
%UNTITLED9 Summary of this function goes here
%   Detailed explanation goes here

% Total Grid Points
% Eulerian
nx = length(hx)+1;
ny = length(hy)+1;
lbc = length(bcnodes);
% Lagrangian
nskx = length(skx);
nsky = length(sky);
nskt = nskx+nsky;

% Non-dimensional Vectors
Re_vec = zeros(nx*ny,1);

% First Derivative Weights (forwards)
a1xf = -(hx(2:end)+2.*hx(1:end-1))./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
b1xf = (hx(2:end)+hx(1:end-1))./(hx(1:end-1).*hx(2:end));
c1xf = -hx(1:end-1)./(hx(2:end).*(hx(1:end-1)+hx(2:end)));
a1yf = -(hy(2:end)+2.*hy(1:end-1))./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
b1yf = (hy(2:end)+hy(1:end-1))./(hy(1:end-1).*hy(2:end));
c1yf = -hy(1:end-1)./(hy(2:end).*(hy(1:end-1)+hy(2:end)));

% First Derivative Weights (backwards)
a1xb = hx(2:end)./(hx(1:end-1).*(hx(1:end-1)+hx(2:end)));
b1xb = -(hx(2:end)+hx(1:end-1))./(hx(1:end-1).*hx(2:end));
c1xb = (2.*hx(2:end)+hx(1:end-1))./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a1yb = hy(2:end)./(hy(1:end-1).*(hy(1:end-1)+hy(2:end)));
b1yb = -(hy(2:end)+hy(1:end-1))./(hy(1:end-1).*hy(2:end));
c1yb = (2.*hy(2:end)+hy(1:end-1))./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% Second Derivative Weights (centered)
a2x = 2./(hx(1:end-1).*(hx(2:end)+hx(1:end-1)));
b2x = -2./(hx(1:end-1).*hx(2:end));
c2x = 2./(hx(2:end).*(hx(2:end)+hx(1:end-1)));
a2y = 2./(hy(1:end-1).*(hy(2:end)+hy(1:end-1)));
b2y = -2./(hy(1:end-1).*hy(2:end));
c2y = 2./(hy(2:end).*(hy(2:end)+hy(1:end-1)));

% Second Derivative Weights (forwards)
a2xf = (6.*hx(1:end-2)+4.*hx(2:end-1)+2.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xf = -(4.*(hx(1:end-2)+hx(2:end-1))+2.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xf = (hx(1:end-2)+2.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xf = -(4.*hx(1:end-2)+2.*hx(2:end-1))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yf = (6.*hy(1:end-2)+4.*hy(2:end-1)+2.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
b2yf = -(4.*(hy(1:end-2)+hy(2:end-1))+2.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yf = (hy(1:end-2)+2.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
```

```
d2yf = -(4.*hy(1:end-2)+2.*hy(2:end-1))./...
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% Second Derivative Weights (backwards)
a2xb = -(2.*hx(2:end-1)+4.*hx(3:end))./...
(hx(1:end-2).*(hx(1:end-2)+hx(2:end-1)).*(hx(1:end-2)+hx(2:end-1)+hx(3:end)));
b2xb = (2.*(hx(1:end-2)+hx(2:end-1))+4.*hx(3:end))./...
(hx(1:end-2).*hx(2:end-1).*(hx(2:end-1)+hx(3:end)));
c2xb = -(2.*hx(1:end-2)+4.*(hx(2:end-1)+hx(3:end)))./...
((hx(1:end-2)+hx(2:end-1)).*hx(2:end-1).*hx(3:end));
d2xb = (2.*hx(1:end-2)+4.*hx(2:end-1)+6.*hx(3:end))./...
((hx(1:end-2)+hx(2:end-1)+hx(3:end)).*(hx(2:end-1)+hx(3:end)).*hx(3:end));
a2yb = -(2.*hy(2:end-1)+4.*hy(3:end))./...
(hy(1:end-2).*(hy(1:end-2)+hy(2:end-1)).*(hy(1:end-2)+hy(2:end-1)+hy(3:end)));
b2yb = (2.*(hy(1:end-2)+hy(2:end-1))+4.*hy(3:end))./...
(hy(1:end-2).*hy(2:end-1).*(hy(2:end-1)+hy(3:end)));
c2yb = -(2.*hy(1:end-2)+4.*(hy(2:end-1)+hy(3:end)))./...
((hy(1:end-2)+hy(2:end-1)).*hy(2:end-1).*hy(3:end));
d2yb = (2.*hy(1:end-2)+4.*hy(2:end-1)+6.*hy(3:end))./...
((hy(1:end-2)+hy(2:end-1)+hy(3:end)).*(hy(2:end-1)+hy(3:end)).*hy(3:end));

% u+, u-, v+, v-
up = max(us,0);
vp = max(vs,0);
um = min(us,0);
vm = min(vs,0);

% Reynolds number vector
for i=1:1:nx
for j=1:1:ny
if in_rbc(bcnodes(i))==1
Re_vec(bcnodes(i)) = Re_rbc;
else
Re_vec(bcnodes(i)) = Re_pl;
end
end
end

if side == 1
for i = 1:1:lbc
%% Residual 1
% Check if points are inside or outside cell, determines if
% discretization scheme needs to be adjusted for second
% derivative approximations.

% Point lies completely inside/outside cell s.t. centered
% approximation for second derivative is physically valid
if (in_rbc(bcnodes(i)+1)==1 && in_rbc((bcnodes(i)+nx-1)==1 && ...
in_rbc(bcnodes(i)-ny)==1 && in_rbc(bcnodes(i)+ny)==1)) || ...
(in_rbc(bcnodes(i)+1)==0 && in_rbc(bcnodes(i)+nx-1)==0 && ...
in_rbc(bcnodes(i)-ny)==0 && in_rbc(bcnodes(i)+ny)==0)

% Residual 1
Res(nskt+2*(bcnodes(i))-1,1) = (b2x(1) + b2y(j-1)).*Psi(bcnodes(i)) + ...
c2y(j-1).*Psi(bcnodes(i)+nx) + a2y(j-1).*Psi(bcnodes(i)-nx) + ...
c2x(1).*Psi(bcnodes(i)+1) + a2x(1).*Psi(bcnodes(i)+nx-1) + ...
w(bcnodes(i));

% Residual 2
Res(nskt+2*(bcnodes(i)),1) = w(bcnodes(i)) + dt.*(up(bcnodes(i)).*...
(a1xb(i-1).*w_old(bcnodes(i)+nx-2)+b1xb(i-1).*w_old(bcnodes(i)+nx-1)+c1xb(i-1).*w_old(bcnodes(i))) + ...
```

```
um(bcnodes(i)).*(a1xf(i).*w_old(bcnodes(i))+b1xf(i).*w_old(bcnodes(i)+1)+c1xf(i).*w_old(bcnodes(i)+2))) + ...
dt.*(vp(bcnodes(i)).*(a1yb(j-1).*w_old(bcnodes(i)-2*ny)+...
b1yb(j-1).*w_old(bcnodes(i)-ny)+c1yb(j-1).*w_old(bcnodes(i))) + ...
vm(bcnodes(i)).*(a1yf(j).*w_old(bcnodes(i))+b1yf(j).*w_old(bcnodes(i)+ny)+c1yf(j).*w_old(bcnodes(i)+2*ny))) - ...
(dt/Re_vec(bcnodes(i))).*((b2x(1) + b2y(j-1)).*w(bcnodes(i)) + ...
c2y(j-1).*w(bcnodes(i)+ny) + a2y(j-1).*w(bcnodes(i)-ny) + ...
c2x(1).*w(bcnodes(i)+1) + a2x(1).*w(bcnodes(i)+nx-1)) - ...
w_old(bcnodes(i)) - dt.*curlf(bcnodes(i));

% Point lies to left of cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+1))
% Point lies to right of cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+nx-1))
break;
% Point lies below cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+ny))
% Point lies above cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-ny))
break;
% Residual
else

end
% Point lies above cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-ny))
% Residual


% Residual
else

end
% Point lies to right of cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-2*ny-1))
% Point lies below cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+ny))
% Point lies above cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-ny))
break;
% Residual
else

end
% Point lies above cell edge
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-ny))


% Residual
else

end
% Point lies below cell edge
elseif  (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)+ny))
% Point lies above cell edge
if (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-ny))
break;
% Residual
else
% Residual 1
Res(nskt+(bcnodes(i))*2-1) = (b2x(1) + d2yb(j-2)).*Psi(bcnodes(i)) + ...
c2yb(j-2).*Psi(bcnodes(i)-ny) + b2yb(j-2).*Psi(bcnodes(i)-2*ny) + a2yb(j-2).*Psi(bcnodes(i)-3*ny) +...
```

```
c2x(1).*Psi(bcnodes(i)+1) + a2x(1).*Psi(bcnodes(i)+nx-1) + ...
w(bcnodes(i));

% Residual 2
Res(nskt+(bcnodes(i))*2) = w(bcnodes(i)) + dt.*(up(bcnodes(i)).*...
(a1xb(i-1).*w_old(bcnodes(i)+nx-2)+b1xb(i-1).*w_old(bcnodes(i)+nx-1)+c1xb(i-1).*w_old(bcnodes(i))) + ...
um(bcnodes(i)).*(a1xf(i).*w_old(bcnodes(i))+b1xf(i).*w_old(bcnodes(i)+1)+c1xf(i).*w_old(bcnodes(i)+2))) + ...
dt.*(vs(bcnodes(i)).*(a1yb(j-1).*w_old(bcnodes(i)-2*ny)+...
b1yb(j-1).*w_old(bcnodes(i)-ny)+c1yb(j-1).*w_old(bcnodes(i)))) - ...
(dt/Re_vec(bcnodes(i))).*((b2x(1) + d2yb(j-2)).*w(bcnodes(i)) + ...
c2yb(j-2).*w(bcnodes(i)-ny) + b2yb(j-2).*w(bcnodes(i)-2*ny) + a2yb(j-2).*w(bcnodes(i)-3*ny) +...
c2x(1).*w(bcnodes(i)+1) + a2x(1).*w(bcnodes(i)+nx-1)) - ...
w_old(bcnodes(i)) - dt.*curlf(bcnodes(i));

end
% Point lies above cell edge
% Residual
elseif (in_rbc(bcnodes(i))~=in_rbc(bcnodes(i)-ny))
% Residual 1
Res(nskt+(bcnodes(i))*2-1) = (b2x(1) + a2yf(j-2)).*Psi(bcnodes(i)) + ...
b2yf(j-2).*Psi(bcnodes(i)+ny) + c2yf(j-2).*Psi(bcnodes(i)+2*ny) + d2yf(j-2).*Psi(bcnodes(i)+3*ny) +...
c2x(1).*Psi(bcnodes(i)+1) + a2x(1).*Psi(bcnodes(i)+nx-1) + ...
w(bcnodes(i));

% Residual 2
Res(nskt+(bcnodes(i))*2) = w(bcnodes(i)) + dt.*(up(bcnodes(i)).*...
(a1xb(i-1).*w_old(bcnodes(i)+nx-2)+b1xb(i-1).*w_old(bcnodes(i)+nx-1)+c1xb(i-1).*w_old(bcnodes(i))) + ...
um(bcnodes(i)).*(a1xf(i).*w_old(bcnodes(i))+b1xf(i).*w_old(bcnodes(i)+1)+c1xf(i).*w_old(bcnodes(i)+2))) + ...
dt.*(vs(bcnodes(i)).*(a1yf(j).*w_old(bcnodes(i))+...
b1yf(j).*w_old(bcnodes(i)+ny)+c1yf(j).*w_old(bcnodes(i)+2*ny))) - ...
(dt/Re_vec(bcnodes(i))).*((b2x(1) + a2yf(j-2)).*w(bcnodes(i)) + ...
b2yf(j-2).*w(bcnodes(i)+ny) + c2yf(j-2).*w(bcnodes(i)+2*ny) + d2yf(j-2).*w(bcnodes(i)+3*ny) +...
c2x(1).*w(bcnodes(i)+1) + a2x(1).*w(bcnodes(i)+nx-1)) - ...
w_old(bcnodes(i)) - dt.*curlf(bcnodes(i));

end
end
elseif side == 3
for i = 1:1:lbc
% Psi
Res(bcnodes(i)*4-3) = Psi(bcnodes(i)) - Psi((i-1)*nx+1);

% Omega
Res(bcnodes(i)*4-2) = w(bcnodes(i)) - w((i-1)*nx+1);
end
end
end
```

Other Functions:

All other functions not listed here that are used **Capillary_Driver.m** are existing MATLAB functions with documentation available through MathWorks.

Bibliography

[1] **Gompper, G., Fedosov, D.A.**(2015) *Modeling microcirculatory blood flow: current state and future perspectives.* WIREs Syst Biol Med 2016, 8:157–168. doi: 10.1002/wsbm.1326

[2] **Le Floch-Yin, F.T.** (2010). *Design of a Numerical Model for Simulation of Blood Microcirculation and Study of Sickle Cell Disease.* Doctoral Thesis, Massachusetts Institute of Technology.

[3] **Bueno, G., Harris, W.L.** (2016). *A Two-Dimensional Spectral Model of Blood Plasma Flow with Oxygen Transport and Blood Cell Membrane Deformation.* Ninth International Conference on Computational Fluid Dynamics (ICCFD9).

[4] **Piel, F.B., Patil, A.P., Howes, R.E., Nyangiri, O.A., Gething, P.W., Williams, T.N., Weatherall, D.J., Hay, S.I.** (2010). *Global distribution of the sickle cell gene and geographical confirmation of the malaria hypothesis.* Nature Communications, 1:104, DOI: 10.1038/ncomms1104

[5] **Evans, E. A., and Skalak, R.** (1980). *Mechanics and Thermodynamics of Biomembranes.* CRC Press, Inc., Boca Raton, Florida.

[6] **Dullemond, C.P., and Kuiper, R.** (2008). *Lectures on Numerical Fluid Dynamics, Chapter 5.* Online lecture notes, University of Heidelberg.

[7] **Vadapalli, A., Goldman, D., and Popel, A.S.** (2002). *Calculations of Oxygen Transport by Red Blood Cells and Hemoglobin Solutions in Capillaries.* Art. Cells, Blood Subs., and Immob. Biotech., **30**(3), 157-188.

[8] **Halpern, D., Secomb, T.W.** (1989). *The squeezing of red blood cells through capillaries with near-minimal diameters.* J. Fluid Mechanics, **203**, 381-400.

[9] **Brust, M., Schaefer, C., Doerr, R., Pan, L., Garcia, M., Arratia, P.E., and Wagner, C.** (2013). *Rheology of Human Blood Plasma: Viscoelastic Versus Newtonian Behavior.* American Physical Society, PRL **110**, 078305.

[10] **Zohuri, B.** (2015). *Dimensional Analysis and Self-Similarity Methods for Engineers and Scientists.* Springer International Publishing Switzerland, **Chapter 2**.

[11] **Peskin, C.S.** (2002) *The immersed boundary method.* Acta Numerica, pp. 479-517, DOI: 10.1017/S0962492902000077.

[12] **Secomb, T.W.** (1988). *Interaction Between Tension and Bending Forces in Bilayer Membranes.* Biophys. J., Biophysics Society, Volume 54, **743-746**.

[13] **Secomb, T.W., Skalak, R., Ozkaya, N., and Gross, J.F.** (1986) *Flow of Axisymmetric Red Blood Cells in Narrow Capillaries.* Journal of Fluid Mechanics, Volume 163, **405-423**.

[14] **Sousa, P.C., Pinho, F.T., Oliviera, M.S.N., and Alves, M.A.** (2011). *Extensional Flow of Blood Analog Solutions in Microfluidic Devices.* Biomicrofluidics **5**, 014108.

[15] **Tözeren, A., Skalak, R., Sung, K.P., and Chien, S.** (1982) *Viscoelastic behavior of erythrocyte membrane.* Biophys. J., **39**, 23-32.

[16] **Tözeren, A., Skalak, R. Fedorciw, R., Sung, K.P., and Chien, S.**(1984) *Constitutive equations of erythrocyte membrane incorporating evolving preferred configuration.* Biophys. J., **45**, 541-549.

[17] **Lai, M.C., Peskin, C.S.**(2000) *An Immersed Boundary Method with Formal Second-Order Accuracy and Reduced Numerical Viscosity.* Journal of Computational Physics **160**, 705–719.

[18] **Peraire, J., Harris, W.L.** (2017) *Personal communications.* Massachusetts Institute of Technology.