PARTICLE IMAGING VELOCIMETRY DATA ASSIMILATION USING LEAST – SQUARE FINITE ELEMENT METHODS

by

Prathish Kumar Rajaraman

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

of

Doctor of Philosophy

in

Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2016

DEDICATION

This thesis is dedicated to my family and friends who have supported me since my undergraduate days thru my Ph.D.

ACKNOWLEDGEMENTS

I begin by thanking my Ph.D. advisor Dr. Jeffery Heys for giving me this opportunity to work at the computational biofluids lab. I also would like to thank all the members of the computational biofluids group, past, present and the future. My secondary Ph.D. advisor is Dr. Tianyu Zhang from the Math Department at Montana State University, thank you for all the support and encouragements. Thank you to both Drs. Jeffery Heys and Tianyu Zhang for answering all my questions in the past five years, even though saying something knowingfully well both of them are correct, I would ask them, why not do this or try that? And they were willing to explain to me why I am wrong. That is how I learnt about applied math and CFD, thank you.

I would like to thank all my committee member who took the time to read this thesis and provided me with advice and suggestions. Thank you for the help for the past three year.

Special thanks to Drs. Tom Manteuffel, Steve McCormick and John Ruge from the Computational Math group at Boulder Colorado for their guidance and advice regarding this Ph.D. project.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

Recent advancements in the field of echocardiography have introduced various methods to image blood flow in the heart. Of particular interest is the left ventricle of the heart, which pumps oxygenated blood from the lungs out through the aorta. One method for imaging blood flow is injecting FDA-approved micro-bubbles into the left ventricle, and then, using the motion of the microbubbles and the frame rate of the ultrasound scan, the blood velocity can be calculated. In addition to blood velocity, echocardiologists are also interested in calculating pressure gradients and other flow properties, but this is not currently possible because the velocity data obtained is two-dimensional and contains noise. In order to realize the full potential of microbubbles as a tool for determining the pumping efficiency and health of the LV, three-dimensional velocity data is required. Our goal is to assimilate two-dimensional velocity data from ultrasound experiments into a three-dimensional computer model. In order to achieve this objective, a numerical method is needed that can approximate the solution of a system of differential equations and assimilate an arbitrary number of noisy experimental values at arbitrary points within the domain of interests to provide a "most probable" approximate solution that is accordingly influenced by the experimental data. In this thesis we present two different approaches for data assimilation, the first approach is more computationally expensive, but requires only a single step. The second approach uses a two stage data assimilation technique but is computationally less expensive. The motivation for using the least-squares finite element method approach is that it provides many advantages such as the ability to match the numerical solution more closely to more accurate data and less closely to the less accurate data.

CHAPTER 1

1.1. GENERAL INTRODUCTION

The efficient operation of the heart results in blood being circulated throughout the body, which is critical for overall health [1-6]. In recent years, new measurement devices and imaging techniques for blood flow in the human heart have provided researchers and scientists with new data on various blood flow properties [7-10]. The data from these techniques are useful for evaluating the health of the heart, but they are also insufficient in many cases because some of the desired flow properties cannot be measured, e.g., viscous energy loss, local pressure gradients, etc. A more comprehensive set of left ventricle blood flow properties is essential for validating the overall health of a patient.

One potential solution that would allow a more complete set of blood flow properties to be obtained is integration of the experimental data provided by medical devices into a numerical model thus creating a patient specific result from the numerical model. This approach of combining additional experimental data into a numerical model is often called data assimilation. Often, the data provided by medical devices is either noisy or spatially limited (e.g., to a lower dimensional space such as a plane or a line). Many attempts have been made to remove the noise in the experimental data [11], and there have been many studies on data assimilation of velocity data into numerical models, but these methods are computationally expensive and also increases programming complexity.

The overall goal of the work described in this thesis is the development of computational tools that can combine noisy, experimental velocity data with a numerical model. The approaches that have been developed have the following novel properties:

1. computationally inexpensive so that the computational costs are only slightly greater than solving the numerical model without any data assimilation,
2. robust enough to handle experimental data with highly variable accuracy and spatially limited to strict sub regions of the overall domain, and
3. the overall accuracy of the numerical model is normally improved by the experimental data, which may require largely ignoring low accuracy experimental data.

The weighted least-squares finite element methods (WLSFEM) described in this thesis are novel and the only known methods with the properties described above. As is described in this thesis, the approaches have been extensively tested on a wide range of test problems, including blood flow in the left ventricle of the heart.

1.2. Anatomy of the Heart

In Figure 1.1, the circulatory system of the heart is shown along with other parts of the heart. The heart can be divided into two halves: the right heart and the left heart, and each half is divided into two more parts: an atrium and a ventricle [1, 12]. The cardiac rhythmicity causes heart contractions, which pumps the blood out of the chamber that it is occupying. The atrium is a weak pump where the blood is initially held before it enters into the ventricle [1].

The heart's primary job is to distribute blood through the circulatory system [2]. The human heart can be divided into two ventricles: the right and left ventricle. The left ventricle is particularly important as it is the largest pumping chamber in the heart, and it is where the blood is held before being pumped to the rest of the body [1, 13]. The left ventricle of the heart pumps the oxygenated blood from the lungs out through the aorta and distributes it to the rest of the body [14-16]. The right ventricle on the other hand receives the deoxygenated blood from the superior vena cava and the inferior vena cava and pumps it to the lungs for oxygenation [1]. In this thesis our focus will be only on the left ventricle since the available blood velocity data is in the left ventricle. Modeling blood flow in the human heart poses numerous challenges because the complex shape of the heart and the motion of the heart walls during expansion and contraction [9]. In this thesis we created a simple left ventricle geometry based on measurements given in previous studies [6].

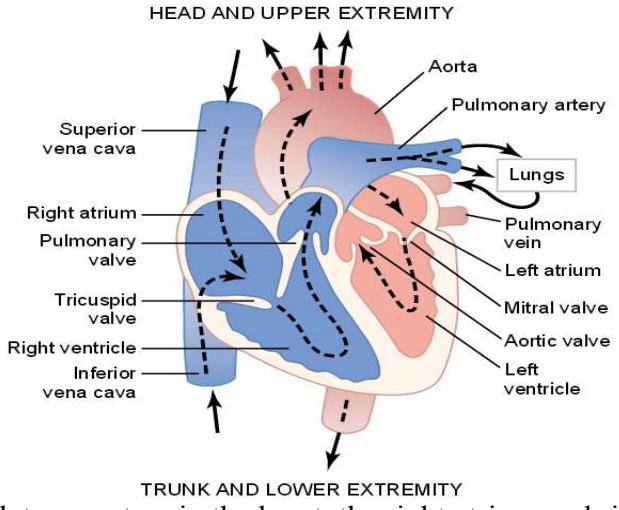


Figure 1.1. The circulatory system in the heart, the right atrium and right ventricle are shown in blue and in red is the left atrium and left ventricle (used without permission) [1].

1.3. Echocardiographic Particle Imaging Velocimetry

In recent years, echocardiographic particle imaging velocimetry (echo-PIV) has been an active area of research due to the noninvasive nature of the approach and high temporal resolution [7, 8, 17, 18]. The fluid velocities measured by the echo-PIV method are calculated after seeding the fluid with ultrasound contrast particles such as microbubbles [7]. The microbubbles are typically small enough that they do not impact flow and will follow the flow dynamics [2, 9]. The microbubbles are commonly made of perfluoropropane, which is FDA approved [6, 19]. The ultrasound contrast particles are used to calculate velocity information using a particle tracking algorithm to estimate the displacement of a microbubble between ultrasound image frames of known temporal separation [8, 9]. In this thesis we use an open source package called PIVlab that can calculate the velocity data from echo-PIV images [20].

The open source PIVlab software is written in MATLAB, and the software requires the input of ultrasound images with the seeded microbubbles visible. PIVlab uses pairs of sequential images for calculating the direction and magnitude of the fluid flow [9, 20]. The velocity is calculated by tracking the displacement of the microbubbles in the second image, based on the positon of the particle in the first image [6]. Once the displacement is estimated, the velocities can be calculated using the scan rate of the ultrasound probe (e.g., results presented in this thesis are fixed at a constant $150\ \frac{frames}{s}$ [9]).

Figure 1.2 shows the time averaged velocity of the left ventricle LV blood flow generated from PIV analysis [4]. This analysis has some weaknesses since time averaging will lead to loss of information about the blood flow. While the time averaging method

leads to lower noise, the smaller flow features that only exist for a short time period will disappear as a result of the time averaging [2, 9].

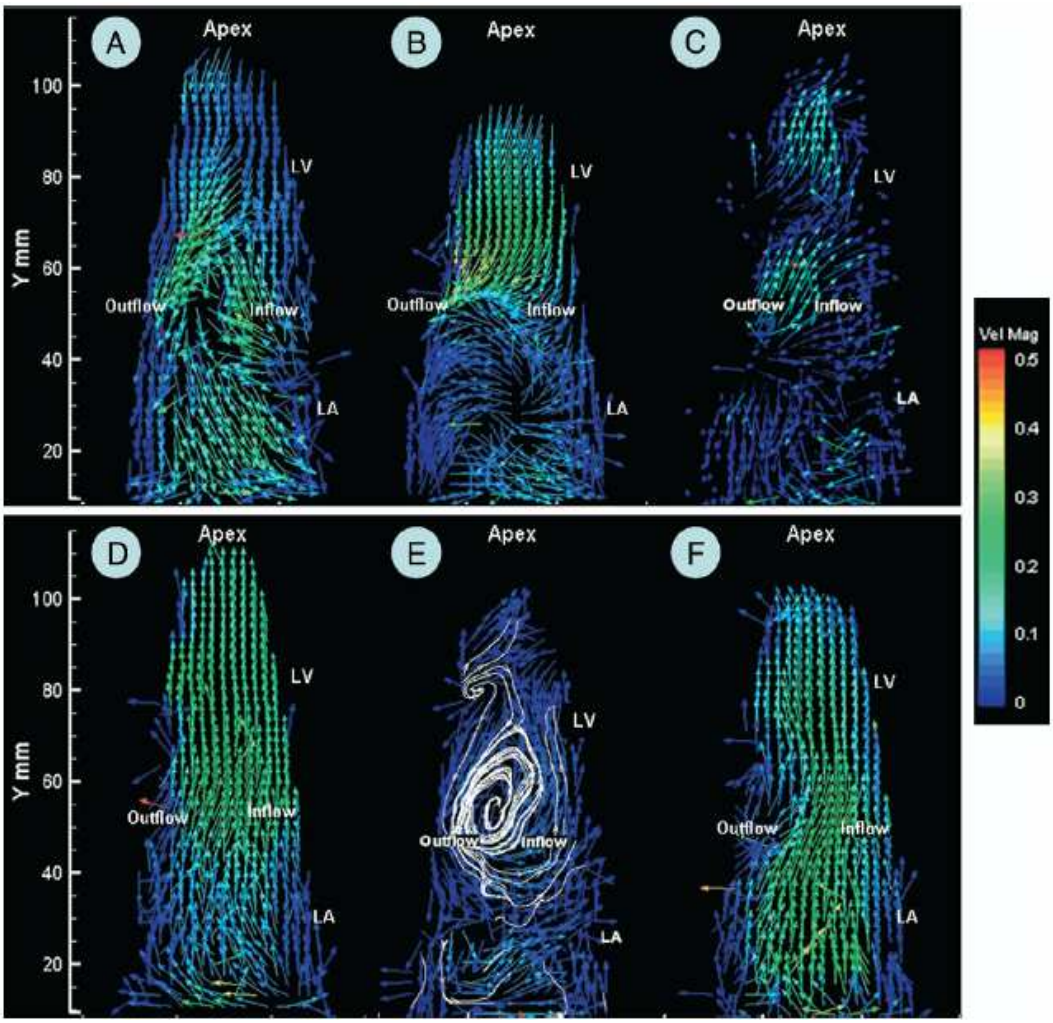


Figure 1.2. PIV analysis shows the velocity magnitudes at different phases of the cardiac cycle. (A) Pre-ejection to (F) late diastole (used without permission) [4].

1.4. Modeling Blood Flow

Blood flow in the LV is frequently modeled as an isothermal and Newtonian viscous fluid with constant material properties. In reality, blood is a non-Newtonian fluid, but the non-Newtonian fluid effects are negligible in the LV [9, 11, 13, 21]. The blood entering the LV from the left atrium is in the form of a pulsatile flow, and the unsteady Navier-Stokes equations are solved in order to capture the blood flow accurately [6, 9, 22]. The momentum and the mass conservation equations for modeling blood flow are given by:

$$\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}\right) = -\nabla P + \frac{1}{Re}\nabla^2 \boldsymbol{u} \quad in\ \Omega,$$
$$\nabla \cdot \boldsymbol{u} = 0 \quad in\ \Omega, \tag{1.1}$$

where $\boldsymbol{u}$ is the dimensionless velocity and P is the dimensionless pressure deviation from hydrostatic so no gravitational body force term is added to equation (1.1) [23, 24]. The Reynolds number (Re) in equation (1.1) is defined as $\frac{\rho u_{0L}}{\mu}$, where ρ is the density, u_0 is the characteristic velocity, L is the characteristic lenght and μ is the viscosity [9]. There are many other possible forms for writing the Navier-Stokes equation, but we maintain the form in the primitive variables $(\boldsymbol{u} - P)$ for now. Later, the Navier-Stokes equations will be written in alternative forms becuase the numerical approach introduced below requires rewriting the Navier-Stokes equation as a first-order system of equations.

There are various approaches to solve (1.1) in order to obtain an approximate numerical solution. The most popular approach is to solve (1.1) using the finite volume method (FVM) [24]. Another popular approach is to use the Galerkin finite element

method (FEM) [23-25]. The problem we want to address in this thesis is data assimilation. In the section below we discuss the least-squares finite element method (LSFEM), which allows straightforward data assimilation compared to the FVM or FEM.

1.5. Least-Squares Finite Element Method

The LSFEM framework can be summarized by its two important features, the defined PDEs (1.1) are rewritten as a first-order differential system and the variation problem is obtained by minimizing the norm of the residual equations [26-30]. Rewriting the original PDEs into a first-order system reduces the condition number, since a large condition number will slow down algebraic solvers such as multigrid [22, 31-34].

There are many ways of rewriting equation (1.1) into a first-order system, and we opt to use a first-order system presented in a previous study that provides improved mass conservation properties [34, 35]. This first-order system is written by introduction of two new vector variables: $\boldsymbol{\omega}$ and $\boldsymbol{r}$, which are the vorticity and the gradient of the total pressure [6, 9, 34, 35]. The vorticity is defined by taking the negative curl of the velocity. The definitions of both variables are shown below:

$$\boldsymbol{\omega} = -\nabla \times \boldsymbol{u}, \tag{1.2}$$

$$\boldsymbol{r} = \sqrt{Re}\nabla P + \frac{\sqrt{Re}}{2}\nabla|u|^2 = \nabla\left(\frac{\sqrt{Re}}{2}|u|^2 + P\right). \tag{1.3}$$

In equation (1.3), P is the kinematic pressure and $\frac{\sqrt{Re}}{2}|u|^2$is the dynamic pressure (kinetic energy) [6, 9, 34, 35]. In order to rewrite the vector laplacian $\nabla^2\boldsymbol{u}$ as a first-order term we use the vector identity defined below:

$$\nabla^2 \boldsymbol{u} = \nabla(\nabla \cdot \boldsymbol{u}) - \nabla \times (\nabla \times \boldsymbol{u}) \tag{1.4}$$

First we simplify identity (1.4) using the mass conservation equation $(\nabla \cdot \boldsymbol{u} = 0)$, then substituting into the momentum equation gives:

$$\sqrt{Re}\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}\right) = -\sqrt{Re}\nabla P - \frac{1}{\sqrt{Re}}\nabla \times \boldsymbol{\omega} \tag{1.5}$$

Note that we have scaled the equation (1.1) with $\sqrt{Re}$, in order to get efficient multigrid performance. Since we have defined the new variable $\boldsymbol{\omega}$, the nonlinear term $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$ in equation (1.5) can be rewritten in term of the new variable $\boldsymbol{\omega}$ using the vector identity shown below:

$$\boldsymbol{u} \cdot \nabla \boldsymbol{u} = \frac{1}{2}\nabla(\boldsymbol{u} \cdot \boldsymbol{u}) - \boldsymbol{u} \times \nabla \times \boldsymbol{u}, \tag{1.6}$$

Noting that the term $\frac{1}{2}\nabla(\boldsymbol{u} \cdot \boldsymbol{u})$ is the kinetic energy, which is defined earlier in equation (3).

$$\nabla \times \boldsymbol{u} + \boldsymbol{\omega} = 0 \ \ in\ \Omega,$$

$$\alpha \nabla \cdot \boldsymbol{u} = 0 \ \ \ in\ \Omega,$$

$$\frac{1}{\sqrt{Re}}\nabla \times \boldsymbol{\omega} - \boldsymbol{r} - \sqrt{Re}\left(\boldsymbol{u} \times \boldsymbol{\omega} + \frac{\partial \boldsymbol{u}}{\partial t}\right) = 0 \ \ \ in\ \Omega, \tag{1.7}$$

$$\beta \nabla \cdot \boldsymbol{\omega} = 0 \ \ \ in\ \Omega,$$

$$\nabla \times \boldsymbol{r} = 0 \ \ \ in\ \Omega,$$

$$\nabla \cdot \boldsymbol{r} - \sqrt{Re}(\boldsymbol{\omega} \cdot \boldsymbol{\omega}) - Re(\boldsymbol{u} \cdot \boldsymbol{r}) = 0 \ \ \ in\ \Omega.$$

The full first-order system with the two new variables is shown above in system (1.7). The first equation in system (1.7) is the definition of the new variable $\boldsymbol{\omega}$ defined earlier in equation (1.2), while the second and third equations in system (1.7) are mass conversation and momentum conservation as a first-order equations in terms of the new variables $\boldsymbol{\omega}$ and $\boldsymbol{r}$. The fourth equation is obtained by taking the divergence of equation (1.2), which is the conservation of vorticity. The fifth equation in system (1.7) is the consequence of the Helmholtz vorticity theorem where the vorticity is divergence-free (no sources or sinks of vorticity within the fluid) [24]. This fifth equation in system (1.7) is obtained by taking the curl of equation (1.3). The final equation in system (1.7) is derived by taking the divergence of the momentum equation and simplifying. Note that the final equation in system (1.7) is a variation of a first-order equation of the pressure Poisson equation, which is another form of rewriting equation (1.1) by taking its divergence. The two constants present in system (1.7), α and β, are included to potentially improve mass conservation and solver performance [9]. In order to create a data assimilation framework for LSFEM, a functional that minimizes the norm of the residual must be defined, and we outline that process in the section below.

1.6. Data Assimilation using Least-Squares Finite Element Method

The first-order equations in system (1.7), are cast into an unconstrained optimization problem. This is important to create a framework for data assimilation where the experimental data will be incorporated into the numerical model via the functional. The functional for system (1.7) is shown below:

$$
\begin{aligned}
F(\boldsymbol{u},\boldsymbol{\omega},\boldsymbol{r}:0) = {} & \|\nabla\times\boldsymbol{u}+\boldsymbol{\omega}\|_{0,\Omega}^2+\alpha\|\nabla\cdot\boldsymbol{u}\|_{0,\Omega}^2 \\
& + \left\|\frac{1}{\sqrt{Re}}\nabla\times\boldsymbol{\omega}-\boldsymbol{r}-\sqrt{Re}\left(\boldsymbol{u}\times\boldsymbol{\omega}+\frac{\partial\boldsymbol{u}}{\partial t}\right)\right\|_{0,\Omega}^2+\beta\|\nabla\cdot\boldsymbol{\omega}\|_{0,\Omega}^2 \\
& + \|\nabla\times\boldsymbol{r}\|_{0,\Omega}^2+\left\|\nabla\cdot\boldsymbol{r}-\sqrt{Re}(\boldsymbol{\omega}\cdot\boldsymbol{\omega})-Re(\boldsymbol{u}\cdot\boldsymbol{r})\right\|_{0,\Omega}^2 \\
& + \frac{w_\Gamma}{h}\|\boldsymbol{u}-g_1\|_{0,\Gamma}^2+\frac{w_\Gamma}{h}\|\boldsymbol{\omega}-g_2\|_{0,\Gamma}^2+\frac{w_\Gamma}{h}\|\boldsymbol{r}-g_3\|_{0,\Gamma}^2 \\
& + \frac{w_{PIV}}{h}\|\boldsymbol{u}-g_{PIV}\|_{0,\Gamma_{PIV}}^2.
\end{aligned}
\tag{1.8}
$$

In order to obtain functional (1.8), the L^2-norm of each equation in system (1.7) is computed on the 3D domain (Ω) and a weighted L^2-norm on the 2D boundary surface (Γ) [2, 6, 9, 34, 35]. The weighted L^2-norm that is used on the boundary is an approximate $H^{\frac{1}{2}}-norm$. In functional (1.8), h is the mesh size and $\left(w_\Gamma, w_{piv}\right)$ are the corresponding weight chosen to indicate how strongly the terms needed to be imposed. The boundary terms added in functional (1.8) result in the boundary conditions being imposed weakly, while a strong boundary condition is imposed by restricting the finite element space [26, 27, 29, 30, 35, 36].

The experimental data is assimilated into the numerical model by adding the term, $\frac{w_{PIV}}{h}\|\boldsymbol{u}-g_{PIV}\|_{0,\Gamma_{PIV}}^2$ into the functional, where $\boldsymbol{u}$ is the approximate numerical solution calculated at the location of the experimental data. Note the location of the data may not be at the computational nodes of the mesh. In chapter 3 we will present an ad-hoc method for calculating w_{piv} and the variable g_{PIV} is the experimental data. The data assimilation

term minimizes the distance between the numerical solution and the experimental data [9, 37-39] in the functional norm.

Functional (1.8) has several potential draw backs, including the fact that computationally solving functional (1.8) is very expensive since there are 9 unknowns per node and also severe mass loss can be observed when $\alpha = 1.0$. Setting $\alpha > 1.0$ does reduce mass loss but the errors are pushed into the conservation of momentum equation and optimal multigrid performance is lost [9]. Using $\alpha > 1.0$ empasized the role of the divergence of velocity, i.e., emphasizes conservation of mass, but a large emphasis on mass conservation can have unintended consequences including the fact that the error in $\frac{w_{PIV}}{h}\|\boldsymbol{u} - g_{PIV}\|^2_{0,\Gamma_{PIV}}$ will not be reduced significantly. In chapter 5 we present a new numerical model for data assimilation that utilizes the full potential of WLSFEM without requiring a larger α and is computationally cheaper compared to functional (1.8). The section below will discuss the scope of this thesis in detail.

1.7. Scope of the Thesis

The main focus of this thesis is the assimilation of noisy patient specific blood flow data in the left ventricle of the heart into numerical model. This thesis will focus on creating a numerical framework for assimilating the noisy 2-dimensional velocity data obtained from echo-PIV method. In some cases, the FEM framework is used to solve the Navier-Stokes equations and the data assimilation problem was solved using the WLSFEM framework. In all cases, the algorithms developed support distributed memory parallel

computations. The linear matrix problems are solved using either the PETSC library or the hypre library.

Several techniques have been used successfully for data assimilation problems where there is a need to incorporate experimental data into numerical models. These techniques have two significant limitations. First, methods based on the use of the very popular ensemble Kalman filter require an ensemble of approximate solutions [6, 9, 19, 40]. In terms of computational time, this approach would lead to a very long computational time for large-scale simulation [40] because it essentially requires a large number of independent simulations. Second, assimilating an arbitrary number of experimental values located at arbitrary locations in space requires smoothing and interpolation. This approach of smoothing and interpolation of the data can severely impact the accuracy of the data beyond the original error.

Using the least-squares finite method (LSFEM), we have developed multiple frameworks for the assimilation of experimental data into the approximate numerical solution of partial differential equations [9]. The framework has the ability to assimilate data of many different types, including concentration, velocity, and temperature. In this thesis our focus is on the assimilation of velocity data into the numerical solution of the Navier-Stokes equations.

The outline of this work is as follows. Chapter 3 will discuss the motivation for using the LSFEM approach and its advantages, including the ability to match the numerical solution more closely to the more accurate data and match less closely to the less accurate data. Another attractive advantage of LSFEM framework is that it leads to a symmetric

positive definite matrix, which can be solved in an efficient and scalable manner using variational multigrid [6, 19, 22, 31-34, 41].

In Chapter 4, a comparison between two different numerical methods was tested in terms of accuracy and computational time. The focus of Chapter 4 is to provide a better numerical approach to be used for the study in Chapter 5. Next, Chapter 5 will present another novel numerical method for data assimilation that has a number of potential advantages over the methods presented in earlier chapters. Chapter 6 will discuss the conclusion and future research along with a summary of goals reached in this current work.

CHAPTER 2

2.1. LITERATURE REVIEW

The focus of this chapter is to discuss work previously published by other researchers on blood flow in the left ventricle and data assimilation techniques. This chapter will begin by discussing the immersed boundary method followed by variational data assimilation approaches for the Navier-Stokes equations. There are two approaches used by researchers extensively when modeling blood flow. The first is to use a numerical approach based on imaging the left ventricle and reconstructing the three-dimensional geometry for use in the numerical model. The immersed boundary approach provides a good framework to solve this problem. The second approach focuses on obtaining patient specific velocity data and incorporating the data into the numerical model. The current approaches for the immersed boundary method do not have a framework for incorporating patient specific data into the numerical solution.

2.2. Immersed Boundary Method

The immersed boundary method was pioneered by Peskin and collaborators for modeling ventricular flow, and they used a simplified 3-dimensional model of the left ventricle with lower temporal and spatial resolution [42-44]. This early study of the left ventricle was a stepping stone for future studies in blood flow modeling [45-48]. The immersed boundary method was also applied to modelling heart valves by Griffith *et al*. Including the heart valves in the numerical model increases the computational cost

dramatically, and the computational models in this thesis do not include the heart valves. Disregarding the heart valves results in a mathematical model that does not capture all the physics of the blood flow in the ventricle, but this missing information can be partially recovered using data assimilation approaches [9].

The immersed boundary method has been used extensively for fluid structure interaction problems, and the method provides a framework for coupling an Eulerian description of the fluid with a Lagrangian description of the structure [44, 45]. The fluid and structure variables in the numerical model are defined by integral transforms with Dirac delta functions [42, 43]. This method provides a solution to the computational challenges associated with the large deformations of the human heart [47]. The equations for the fluid are defined previously in equation (1.1), which are the incompressible Navier-Stokes equations. The elastic forces and displacements between the fluid and the structure are defined below in equations (2.1) and (2.2).

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_{\Omega} \boldsymbol{F}(q,r,s,t)\, \delta\big(\boldsymbol{x} - \boldsymbol{X}(q,r,s,t)\big)\, dq\, dr\, ds, \tag{2.1}$$

$$\frac{\partial \boldsymbol{X}}{\partial t}(q,r,s,t) = \int_{U} \boldsymbol{u}(\boldsymbol{x},t)\, \delta\big(\boldsymbol{x} - \boldsymbol{X}(q,r,s,t)\big)\, d\boldsymbol{x}. \tag{2.2}$$

In equation (2.1) and (2.2) $\boldsymbol{X}(q,r,s,t)$ are the physical position of the material (solid) points (q,r,s) at time (t). In equation (2.2), $\boldsymbol{u}(\boldsymbol{x},t)$ is the velocity of the fluid and δ is the three-dimensional Dirac delta function, which is used to convert between Eulerian and Lagrangian quantities [42, 43, 48]. In equation (2.1), $\boldsymbol{f}(\boldsymbol{x},t)$ is the Eulerian description of the elastic force in terms of the Cartesian coordinates. The description here of the

immersed boundary method is brief, but further information about the background and implementation can be found in [42-48].

The fluid problem in the immersed boundary method can be solved with various numerical methods such as the finite volume or finite element methods. The immersed boundary approach for modeling blood flow does not incorporate any patient data. In Chapter 5, the new data assimilation introduced provides a seamless path for integration with existing numerical models. It should be possible to integrate the new data assimilation approach into the immersed boundary method, thus creating a data assimilation framework for the immersed boundary method. This new area of research will be the focus of future work.

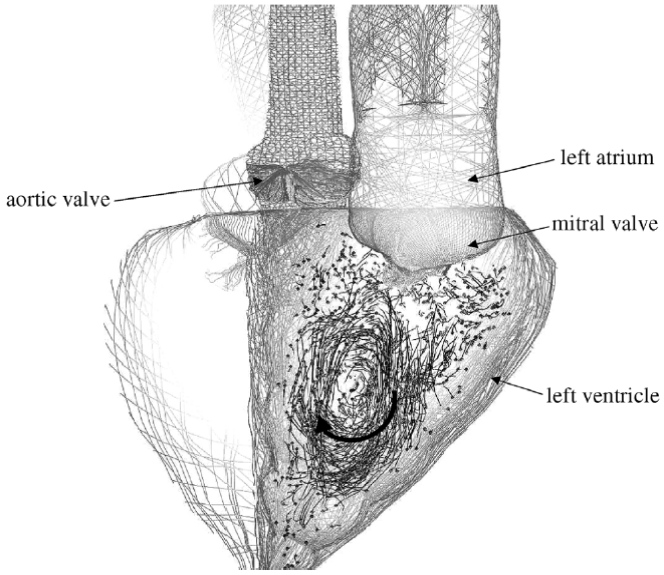


Figure 2.1. Slice of left ventricle along with the mitral valves and the simulation result show the streamlines during the filling stage (used without permission) [44].

In Figure 2.1, the immersed boundary simulation results of blood flow in the left ventricle are shown. The opening of the mitral valve allows the blood to flow into the ventricle and the closed aortic valve prevents the blood from exiting the ventricle [44]. During the filing phase, the walls of the ventricle are moved based on the force matching condition between the fluid and the immersed solid. The formation of vortex rings can be observed, and the accuracy of the velocity flow field can be improved further by solving the model on a finer mesh [24, 49]. The accuracy cannot be improved by using a higher-order discretization scheme due to the use of a discrete delta function.

2.3. Variational Data Assimilation for Navier-Stokes Equation

Many studies have used traditional Computational Fluid Dynamics (CFD) methods for modeling blood flow in left ventricle, but there are limited studied for assimilating data with CFD in ventricular flow, the focus of this thesis. In this section we present an alternate data assimilation framework for assimilating noisy blood flow data obtained using Magnetic Resonance Imaging (MRI) [13, 50]. The blood flow data was obtained in the ascending aorta, where the noisy data can be assimilating into a numerical model to improve accuracy and reliability of the numerical solution [11, 21].

We begin this section by discussing the mathematical framework for the data assimilation. The formulation for this technique is based upon the variational control approach, and this method is somewhat similar to the approach presented in Chapter 3 where the control problem minimizes the distance between the data and the numerical solution [9, 49, 51, 52]. In [13, 53, 54], the steady Navier-Stokes equation are used as the

model problem, the nonlinear term is linearized, and the equations are discretized using the FEM. The Navier-Stokes equations are formulated as a constrained minimization problem (an approach sometimes used to solve inverse problems) as shown below [13]:

$$\min_{\boldsymbol{h}} \mathcal{J}(\boldsymbol{u}, \boldsymbol{h}) = dist(f(\boldsymbol{u}), \boldsymbol{d}) + \mathcal{R}(\boldsymbol{h}) \tag{2.3}$$

$$s.t. \begin{cases} -\upsilon \nabla \cdot (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \nabla p = \boldsymbol{s} \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases}$$

In system (2.3), $dist(.,.)$ is the term where the incompatibility between the numerical model $(\boldsymbol{u}, p)$ and data $(\boldsymbol{d})$ is minimized and $\mathcal{R}$ is the regularization term, which is necessary to prevent an ill-conditioned problem [39]. System (2.3) must also include boundary conditions associated with the model problem (further details on boundary can be found in [2]). In [13, 39, 54], the nonlinear linear term $(\boldsymbol{u} \cdot \nabla \boldsymbol{u})$ is substituted with $(\boldsymbol{\beta} \cdot \nabla \boldsymbol{u})$, where $\boldsymbol{\beta}$ is the known convective field, thus linearizing the equations and avoiding the need for Newton's method or some other non-linear iteration method.

The model described above is discretized by the FEM, using a $P_1 - bubble, P_1$ spatial discretization. The data assimilation problem of interest in [13] is blood flow in the aorta. Other validations of the numerical method are also presented elsewhere, but the focus of [2] is only on the aorta. The noisy data for assimilation was generated by adding random Gaussian noise at the sites where the assimilation was done, this approach allows the user to inspect the noise filtering capability of the proposed method [13]. The fundamental reason for doing data assimilation is to provide valuable information to a medical doctor on the quantity of interest, which in most cases is the wall shear stress $\boldsymbol{\tau}$, an indicator for

possible rupture of the vessel wall and the formation of stenosis [11, 13, 21, 50]. Equation (2.4) below defines the tangential wall stress, where $\boldsymbol{n}$ is the normal vector [13].

$$\boldsymbol{\tau} = \frac{1}{Re}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)\boldsymbol{n} - \frac{1}{Re}\Big(\big((\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)\boldsymbol{n}\big)\Big) \cdot \boldsymbol{n}\Big)\boldsymbol{n} \tag{2.4}$$

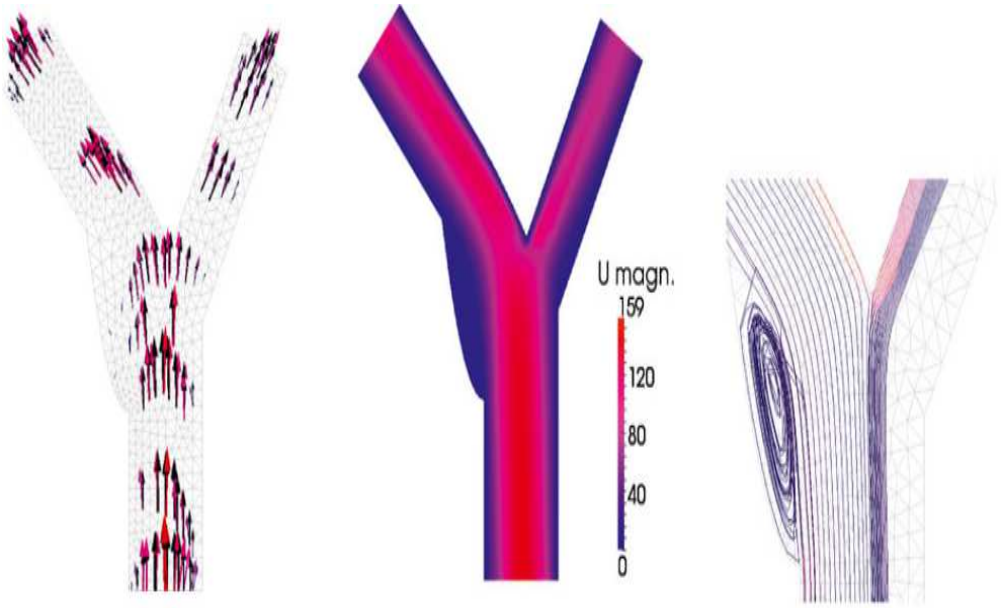


Figure 2.2. The first figure from left shows the data used for the data assimilation model, the black data is the original data, while the red is the interpolated data. The center figure shows the magnitude of velocity and finally the figure to the right show the streamline (used without permission) [13].

In Figure 2.2, the results presented in [13] are shown. The geometry of the aorta was obtained from MRI scans. The left image in Figure 2.2, shows the data with and without noise. In order to solve the numerical model with data assimilation, the data is interpolated to the mesh nodes. While this study [13] uses artificially generated data, this

thesis will present data obtained from real experiments, which will be presented in Chapter 3 and Chapter 5.

The work presented in [11, 13, 21, 39, 50, 53], shows the method employed for data assimilation is an efficient method for integrating noisy data into the numerical solution, but this approach is computational expensive. This is one of the primary motivations for the methods developed and described in this thesis. The numerical method presented in Chapter 5 presents an approach for data assimilation that is computationally less expensive. The computational time for each data assimilation step costs as much as solving a Poisson problem.

2.4. Uncertainty Quantification for Data Assimilation

In this section we discuss briefly an uncertainty quantification approach for numerical modeling with data assimilation present in [54]. The method presented here is closely tied with the Bayesian approach presented in the section above for inclusion of data into a numerical model by minimizing the distance between the model and the data. The uncertainty quantification approach is applied here in order to improve the reliability of the numerical model [54]. In [54], the goal is to quantify the uncertainty due to the noise in the data and misfit between the numerical model and data. The numerical model solved here is the Navier-Stokes equation presented in system (2.3).

When employing control theory approaches for data assimilation problems, the common practice is to use an adjoint method, which is used to impose the model equation shown in equation (2.3) as a constraint to the numerical solution [49, 51, 52, 54, 55]. The

numerical model can be split into two stages, where the adjoint problem and the forward problem can be solved monolithically or in an optimization framework where the adjoint provides the gradients [52, 54]. This approach adds significant computational time since the simulation are solved several times in an iterative manner [54].

The method was applied to a similar cardiovascular problem as in the previous section, although in this work the statistical properties of the velocity field are estimated [54]. Incorporating additional information regarding measurement process and statistical information (data correlation function) improves the accuracy by 30% with respect to solution presented in previous section [54]. The new data assimilation method presented in this thesis does not quantify uncertainty, but this will be the focus area for future work. The new data assimilation framework does provide a computationally cheaper approach compared to other existing methods presented in this thesis.

2.5. A Bayesian Perspective on Data Assimilation using WLSFEM

Many studies have been done in the area of data assimilation for combining data into numerical models. There have been only a handful of studies using WLSFEM for data assimilation applications [6, 9, 19, 40]. In this section we will focus on an alternate perspective for data assimilation using WLSFEM. In [40], the WLSFEM method for data assimilation is viewed as finding a maximum *a posteriori* (MAP) *estimator*. This can be viewed as the model equations (Navier-Stokes) described in Chapter 1 as a prior belief when no data exist in the model ($\boldsymbol{w}$), and this is considered to be the best guess for the

actual physics [40]. When there are observational data ($\boldsymbol{d}$), performing an assimilation of data into the simulation corrects the prior belief based on the trust of the data [40].

$$\underset{\boldsymbol{w} \in W}{argmax}\ \pi^{\boldsymbol{d}}(\boldsymbol{w}) = \underset{\boldsymbol{w} \in W}{argmin} \left\{ \frac{1}{2} \|\boldsymbol{d} - G(\boldsymbol{w}, \boldsymbol{x})\|^2_{\Sigma_d} + \frac{1}{2} \mathcal{J}(\boldsymbol{w}; \boldsymbol{f}, \boldsymbol{g}) \right\} \tag{2.5}$$

Equation (2.5) defines the MAP *estimator*, where $G(\boldsymbol{w}, \boldsymbol{x})$ is the numerical solution at the data location and $\mathcal{J}(\boldsymbol{w}; \boldsymbol{f}, \boldsymbol{g})$ is the numerical model. Analyzing the WLSFEM framework as MAP *estimator*, shows that in the WLSFEM model, equations (1.7) play the role of the prior, while the data plays the role of correcting likelihood [40]. This frameworks is very favorable since the prior can be bested when experimental data are assimilated into the numerical model [9, 40]. According to the authors, this feature of WLSFEM for data assimilation application is consistent with modern scientific philosophy when the model equations are treated as a Bayesian prior [40].

Solving for the MAP *estimator* using the WLSFEM framework does not increase the computational cost significantly compared to solving of the model equation without any data assimilated. The other methods presented in this chapter involve solving the model equation in an iterative manner, which increases the computational cost significantly. The study presented here concludes that the WLSFEM framework demonstrates favorable properties compared to other methods in terms of computational cost and the ability to handle data with noise without requiring any regularization terms [13, 40, 52, 54]. In the case of more experimental data available, model equations plays a weaker role when obtaining numerical solution [40].

CHAPTER 3

ECHOCARDIOGRAPIC PARTICLE IMAGING VELOCIMETRY DATA ASSIMILATION WITH LEAST SQUARE FINITE ELEMENT METHODS

Contribution of Authors and Co-Authors

Manuscript in Chapter 3

Author: Prathish K. Rajaraman

Contributions: Programing computational model, data collection and analysis manuscript drafting and editing.

Co-Author: T. A. Manteuffel

Contributions: Conception and critical analysis of least squares finite element method.

Co-Author: M. Belohlavek

Contributions: Echo-PIV experiment and data collection.

Co-Author: E. McMahon

Contributions: Echo-PIV experiment and data collection.

Co-Author: Jeffrey J. Heys

Contributions: Conception and critical analysis of article, manuscript drafting and editing.

Manuscript Information Page

Prathish K. Rajaraman [a], T.A. Manteuffel [b], M. Belohlavek [c], E. McMahon [c], and Jeffrey J. Heys [a]

Journal Name: Computers & Mathematics with Applications

Status of Manuscript:

____ Prepared for submission to a peer-reviewed journal

____ Officially submitted to a peer-review journal

____ Accepted by a peer-reviewed journal

_X_ Published in a peer-reviewed journal

Computers & Mathematics with Applications

Published in CMA Volume 68, Issue 11, December 2014

[a] Chemical and Biological Engineering Department, Montana State University, Bozeman, MT 59717

[b] Department of Applied Mathematics, University of Colorado, Boulder, CO 80309

[c] Mayo Clinic Arizona, Scottsdale, AZ 85259

3.1. Abstract

Recent developments in the field of echocardiography have introduced various noninvasive methods to image blood flow within the heart chambers. FDA-approved microbubbles can be used for intracardiac blood flow imaging and determining the velocity of the blood based on the displacement of the bubbles and the frame rate of the ultrasound scan. A limitation of this approach is that the velocity field information is only two-dimensional and inevitably contains noise. A weighted least square finite element method (WLSFEM) was developed to assimilate noisy, two-dimensional data from echocardiographic particle imaging velocimetry (echo-PIV) into a three-dimensional Navier-Stokes numerical model so that additional flow properties such as the stress and pressure gradient can be determined from the full velocity and pressure fields. The flexibility of the WLSFEM framework allows for matching the noisy echo-PIV data weakly and using the weighted least square functional as an indicator of how well the echo-PIV data are satisfying the numerical model. Results from the current framework demonstrate the ability of the approach to more closely match the more accurate echo-PIV data and less closely match the noisy data. The positive impact of assimilating the echo-PIV data is demonstrated: compared to a conventional computational fluid dynamic approach, echo-PIV data assimilation potentially enables a more accurate flow model.

3.2. Introduction

The efficient flow and pumping of blood in the left ventricle of the heart is critical for overall health [1-3]. In recent years, many researchers and scientist have made significant progress towards a better understanding of efficient blood flow in the left ventricle, including the development of imaging techniques and computational fluid dynamics (CFD) models [1, 4-6]. Each method, imaging and simulation, carries its own advantages and disadvantages. For example, magnetic resonance imaging (MRI) is expensive and confines the patient to a limited space and immobile position to obtain an imaging sequence of the flow in the heart, whereas ultrasound imaging, in comparison, is less expensive and restrictive, but provides anatomic images with lesser fidelity although with higher temporal resolution. The common advantage of imaging techniques, compared to CFD models, is obtaining patient-specific data [2, 3, 5, 6]. The advantage of a CFD model is the ability to provide a virtual environment in which ranges of fluid parameters (e.g., blood viscosity) and geometric features can be explored without jeopardizing patient safety thus facilitating better understanding of the physics of cardiac functioning as a pump. However, CFD models can include physical assumptions and are rarely patient-specific. Another limitation of CFD models is that they are never perfect in the sense that they do not capture all of the physics, chemistry, and biology of a real heart.

Many investigators and clinicians are in need of a framework that combines the strengths of both a CFD model and patient-specific imaging. This framework would allow assimilation of patient-specific data into a CFD model to enable realistic replication of cardiac fluid dynamic properties and, thus, further improvement in diagnosis and prognosis

of cardiac disease [1]. Methods have been developed that enable data obtained from experimental measurements to be assimilated (or integrated) into a computational model [1, 2, 7]. Challenges associated with previously developed methods, however, make those methods impractical for the problem of interest here. For example, many data assimilation approaches use the Kalman filter that requires an ensemble of approximate solutions, which increases the computational cost dramatically [2, 8]. Other approaches incorporate experimental data points at arbitrary spatial locations by smoothing and interpolating the experimental data to computational nodes, but the interpolation can compromise the accuracy of the experimental data [1, 2, 7]. The approach described here is based on the weighted least-square finite element method (WLSFEM), which reduces computational costs relative to ensemble approaches and is able to assimilate an arbitrary number of data points into the simulation from arbitrary spatial locations without any interpolation or smoothing of the data [2].

The WLSFEM framework allows for the simulation of blood flow in the cardiac left ventricle and assimilates experimental data obtained using echocardiographic particle imaging velocimetry (echo-PIV). The echo-PIV method utilizes FDA approved microbubbles injected into the blood stream. Imaging of the left ventricle by cardiac ultrasonography determines microbubble location [1, 2, 7]. In recent years, the echo-PIV technique has been studied extensively in both experimental and clinical settings [9]. The images obtained by ultrasonography can be assessed by a number of available PIV analysis algorithms. PIVlab, an open source software algorithm for calculating particle (i.e.,

microbubble) velocity based on its displacement over a time period between successive image frames [10], is used here to obtain flow velocity vectors inside the left ventricle.

Figure 3.1(a) depicts the left ventricle during an experimental study with microbubbles (appearing as white spots). The ultrasound probe scans from the apex of the

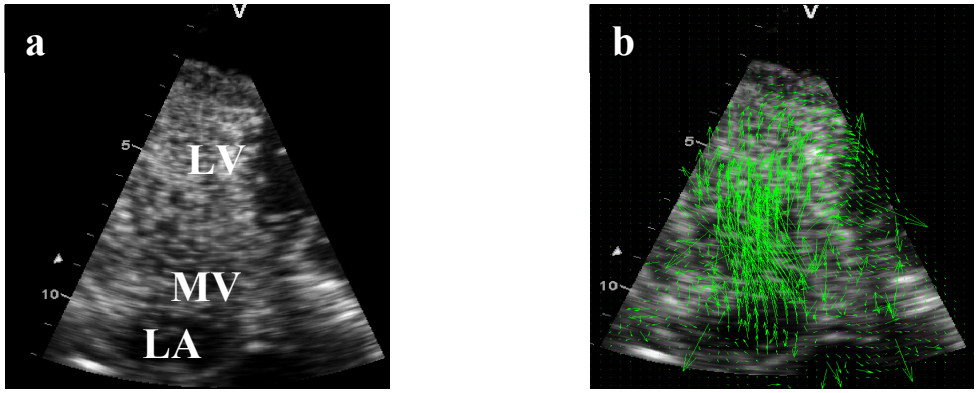


Figure 3.1. An ultrasonogram of the left ventricle (LV), mitral valve (MV), and left atrium (LA) after injecting microbubbles (a). The corresponding echo-PIV data (velocity vectors) (b).

heart (i.e., top of the sector image). Figure 3.1(b) shows a velocity vector field reconstructed from the corresponding echo-PIV, and we can identify misdirected, i.e., noisy, velocity vectors. Because sufficient temporal resolution (i.e., >100 frames/s) is currently technologically limited to 2-dimensional scans, PIV analysis only provides two components of 3-dimensional velocity vectors (the two components are tangential to the plane of the original image). These are still valuable blood flow velocity data, but additional flow properties, such as the pressure gradient and viscous energy losses, would be of a great clinical value but cannot be computed from the 2-dimensional vector fields [1, 2]. The desired flow information, especially pressure gradients and stresses, requires that the full 3-dimensional velocity fields be known. At this point, the reader may ask why

not use other methods to obtain the full three dimensional velocity field, such as an MRI approach? MRI machines are expensive and patients need to be sedated before the scan in order to prevent them from moving, which often leads to side effects such as headaches and nausea [9]. Further, MRI imaging often lacks the temporal accuracy required for some of the desired flow properties. The 3-dimensional data obtained from MRI are time-averaged over several cardiac cycles, which prevent the capture of a real-time flow pattern [9]. The echo-PIV method is attractive because it is inexpensive, broadly available, has a high temporal resolution, and the microbubble injection procedure is minimally invasive [1, 2, 9].

The current paper builds upon our previous publication [2], but focuses on assimilation of data from a significantly larger number of time steps and a higher temporal resolution (a more than one order of magnitude increase in both time and space). In the current paper, we have also employed a much more realistic geometry for the left ventricle, which is more demanding on meshing and has required implementation of higher-order tetrahedral elements [11]. In the previous paper, the weighting of the echo-PIV data was mostly arbitrary, because it was based on the accuracy of a different PIV approach – optical PIV. Here, we demonstrate a novel method for weighting the echo-PIV data in a consistent manner and show that data assimilation allows the model to potentially capture physics that would be missed without the experimental data, due to simplifying assumptions in the numerical model.

3.3. Model Problem

3.3.1 Navier-Stokes

The Navier-Stokes equations model many physical phenomena in fluid mechanics, including both turbulent and laminar fluid flow [11, 12]. Many previous studies have shown that using the incompressible Navier-Stokes equations is appropriate when modeling blood flow in the heart [6, 13, 14]. The incompressible Navier-Stokes equations can be written as:

$$\begin{aligned} \sqrt{Re}\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) &= -\nabla P + \frac{1}{\sqrt{Re}}\nabla^2\boldsymbol{u} \quad in\ \Omega, \\ \nabla\cdot\boldsymbol{u} &= 0 \quad in\ \Omega, \end{aligned} \tag{3.1}$$

where $\boldsymbol{u}$ is the dimensionless velocity, P is the dimensionless pressure, and Re is the Reynolds number $\left(Re = \frac{LU\rho}{\mu}\right)$. There are many possible approaches for scaling the dimensionless Navier-Stokes equations. Here both the viscous term and convective term are scaled with $\frac{1}{\sqrt{Re}}$ and $\sqrt{Re}$, respectively. In order to solve the Navier-Stokes equations in the WLSFEM setting, new variables are defined so that the original, second-order equations can be rewritten as a first-order system [15, 16]. There are a number of different first-order systems of equations proposed by researchers for reformulating the Navier-Stokes equations, and each of these systems has advantages and disadvantages [17]. The first-order system used here has two new variables and was developed to improve mass conservation and multilevel solver performance when using C^0 finite elements [11, 15, 16, 18, 19].

The first-order system is derived using the vorticity,$\boldsymbol{\omega}$ as a new unknown vector, equal to the negative curl of velocity as shown below in equation (3.2) [11, 15, 16]. Equation (3.3) defines another new variable, $\boldsymbol{r}$, which is often called the gradient of the total pressure [2, 18]. In equation (3.3), P is the kinematic pressure and $\frac{\sqrt{Re}}{2}\nabla|u|^2$ is the dynamic pressure. The new variable, $\boldsymbol{r}$, helps to strengthen the coupling between velocity and pressure, and a weak velocity-pressure coupling has been previously identified as leading to poor mass conservation when using least-squares finite element methods [1, 20].

$$\boldsymbol{\omega} = -\nabla \times \boldsymbol{u}, \tag{3.2}$$

$$\boldsymbol{r} = \nabla P + \frac{\sqrt{Re}}{2}\nabla|u|^2 = \nabla\left(\frac{\sqrt{Re}}{2}|u|^2 + P\right). \tag{3.3}$$

The full first-order system with the two new variables is shown below in system (3.4). The second equation in system (3.4) is the continuity equation and the variable α is included to improve mass conservation properties on coarser meshes, for the results presented here, $\alpha = 80.0$ was used [15, 18-20]. The third equation in system (3.4) is the momentum equation in terms of the variables $\boldsymbol{u}, \boldsymbol{r}$ and $\boldsymbol{\omega}$. The fourth equation in system (3.4) is the conservation of vorticity and the variable β can be set to $\beta = \frac{1}{\sqrt{Re}}$ to improve solver performance, but this can decrease the accuracy of the solution. For the results presented here, $\beta = 1.0$ was used. The first-order system with variables of velocity $(\boldsymbol{u})$, vorticity $(\boldsymbol{\omega})$ and gradient of total pressure $(\boldsymbol{r})$ in system (3.4) leads to a system with curl-div blocks for each variable $(\boldsymbol{u}, \boldsymbol{\omega},$ and $\boldsymbol{r})$, and this provides in favorable multigrid performance if sufficient boundary conditions are used. For further details of this system the reader should refer to [19].

$$\nabla \times \boldsymbol{u} + \boldsymbol{\omega} = 0 \ \ in\ \Omega,$$

$$\alpha \nabla \cdot \boldsymbol{u} = 0 \quad in\ \Omega,$$

$$\frac{1}{\sqrt{Re}} \nabla \times \boldsymbol{\omega} - \boldsymbol{r} - \sqrt{Re}\left(\boldsymbol{u} \times \boldsymbol{\omega} + \frac{\partial \boldsymbol{u}}{\partial t}\right) = 0 \quad in\ \Omega, \tag{3.4}$$

$$\beta \nabla \cdot \boldsymbol{\omega} = 0 \quad in\ \Omega,$$

$$\nabla \times \boldsymbol{r} = 0 \quad in\ \Omega,$$

$$\nabla \cdot \boldsymbol{r} - \sqrt{Re}(\boldsymbol{\omega} \cdot \boldsymbol{\omega}) - Re(\boldsymbol{u} \cdot \boldsymbol{r}) = 0 \quad in\ \Omega.$$

The next step is to cast the equations in system (3.4) into an optimization problem. The corresponding functional for system (3.4) is shown below in equation (3.5), and we can further analyze the functional in equation (3.5) to show it is continuous and coercive in the $\left(H(div) \cap H(curl)\right)^3 \times H^1$ norm using a standard compactness argument [19, 20]. In [19], the reader can find more details on using appropriate functional and boundary condition for system (3.4). The functional is a sum of the $L^2 - norm$ on the 3D domain (Ω) and a weighted $L^2 - norm$ on the 2D boundary surface (Γ), more details below. The local functional in equation (3.5) has been studied extensively by many authors as a sharp error indicator, which can be used for adaptive refinement [21, 22]. In the results section, we will demonstrate how we have used the functional as an indicator of the quality of the numerical solution to the Navier-Stokes equations after data assimilation, which is unique to this paper. Once the functional and the first-order system are defined, we discretized the equation onto a finite element space in order to obtain the numerical solution [15].

In [7], the reader can find a different perspective on the WLSFEM framework for data assimilation in terms of Bayesian inference, and a demonstration of the approach to consistently incorporate noisy experimental data into simulations without regularization terms. The WLSFEM framework allows the simulation without any echo-PIV data as a prior and, using the available data, is able to correct the simulation result to a more likely result based on Gaussian observational noise (error-free data matched exactly). In a scenario with more echo-PIV data available, the simulation plays a weaker role (and the data play a stronger role) when obtaining the numerical solution. The WSLEM framework treats the simulation as a Bayesian prior, which is considered consistent with modern scientific philosophy [7].

$$
\begin{aligned}
F(\boldsymbol{u},\boldsymbol{\omega},\boldsymbol{r}:0) = {} & \|\nabla\times\boldsymbol{u}+\boldsymbol{\omega}\|_{0,\Omega}^2+\alpha\|\nabla\cdot\boldsymbol{u}\|_{0,\Omega}^2 \\
& + \left\|\frac{1}{\sqrt{Re}}\nabla\times\boldsymbol{\omega}-\boldsymbol{r}-\sqrt{Re}\left(\boldsymbol{u}\times\boldsymbol{\omega}+\frac{\partial\boldsymbol{u}}{\partial t}\right)\right\|_{0,\Omega}^2 \\
& + \beta\|\nabla\cdot\boldsymbol{\omega}\|_{0,\Omega}^2+\|\nabla\times\boldsymbol{r}\|_{0,\Omega}^2 \\
& + \left\|\nabla\cdot\boldsymbol{r}-\sqrt{Re}(\boldsymbol{\omega}\cdot\boldsymbol{\omega})-Re(\boldsymbol{u}\cdot\boldsymbol{r})\right\|_{0,\Omega}^2+\frac{w_\Gamma}{h}\|\boldsymbol{u}-g_1\|_{0,\Gamma}^2 \\
& + \frac{w_\Gamma}{h}\|\boldsymbol{\omega}-g_2\|_{0,\Gamma}^2+\frac{w_\Gamma}{h}\|\boldsymbol{r}-g_3\|_{0,\Gamma}^2 \\
& + \frac{w_{PIV}}{h}\|\boldsymbol{u}-g_{PIV}\|_{0,\Gamma_{PIV}}^2.
\end{aligned}
\tag{3.5}
$$

In functional (3.5), the terms on the right side weighted by w_Γ are used to incorporate the weak boundary conditions, which are given in the next section. The final term, weighted by w_{PIV} is used for the assimilation of the PIV data as described in next section. Thus, minimizing this functional requires finding an approximate solution that

balances satisfying the: (1) Navier-Stokes equations rewritten as a first-order system, (2) the weak boundary conditions, and (3) the assimilated PIV data. The balance between these three requirements is determined by the weights as described in section below.

3.3.2 Boundary Conditions

In the WLSFEM framework, we have the option of enforcing boundary conditions strongly, i.e., enforcing them directly on the finite element space, or weakly, i.e., adding boundary terms to the functional [19, 20]. One attractive feature of the WLSFEM framework is the ability to weight the weak boundary conditions in a comprehensive, overall minimization framework [15, 27]. The computation of the boundary functional is based on approximating the $H^{\frac{1}{2}} - norm$ using a weighted $L^2 - norm$, which leads to the $\frac{1}{h}$ terms in functional (3.5) [19]. In the functional (3.5), Γ_{PIV} is the boundary where the assimilated echo-PIV data are weakly matched. The reason for not matching the echo-PIV data strongly is due to the errors in the experimental data that would cause the entire approximate solution to be contaminated if the data were matched exactly [2, 28]. The echo-PIV data that we are matching weakly are the two tangential components of the velocity vector along the ultrasound plane, and the velocity data can be at arbitrary locations within the computational domain.

Table (3.1) summarizes the boundary condition used for simulating blood flow in the left ventricle. Note that some of the boundary conditions vary according to the time step. In table (3.1), the variable $g(t)$ indicates the specified velocity on the boundary (when it is known), which varies for each time step. The normal velocity on both the inlet and outlet is not specified with a velocity boundary condition, but the numerical model

determines the normal velocity based on the motion of the walls. This strategy was chosen because specifying the normal velocity strongly will diminish any influence of the echo-PIV data near the inlet and outlet. In [19], it is established that setting weak boundary conditions on $\boldsymbol{\omega}$ and $\boldsymbol{r}$ will create an operator that is continuous and coercive in $\big(H(div) \cap H(curl)\big)^3 \times H^1$ and leads to optimal multigrid performance. Unfortunately, boundary conditions on $\boldsymbol{r}$ are not available on every boundary of the left ventricle, so the functional is not fully coercive here and optimal multigrid convergence cannot be proven. See [18, 19] and the reference therein.

Table 3.1. Summary of boundary conditions used for blood flow in left ventricle.

Boundary	Condition	Enforcement	Time Step
Inlet	$n \times \boldsymbol{u} = 0$	Weak	≤ 59 (filling)
	$\boldsymbol{u} = 0$	Strong	≥ 60
	$n \cdot \boldsymbol{\omega} = 0$	Weak	*all Time Steps*
Outlet	$\boldsymbol{u} = 0$	Strong	≤ 74
	$n \cdot \boldsymbol{r} = 0$	Weak	≤ 74
	$n \cdot \boldsymbol{\omega} = 0$	Weak	*all Time Steps*
	$n \times \boldsymbol{u} = 0$	Weak	≥ 75 (ejection)
	$\boldsymbol{u} = 0$	Strong	$= 134$
Wall	$n \cdot \boldsymbol{\omega} = 0$	Weak	*all Time Steps*
	$\boldsymbol{u} = g(t)$	Strong	*all Time Steps*
PIV-Plane	$n \times \boldsymbol{u} = g(t)$	Weak	*all Time Steps*

The motion of the upper surface of the left ventricle was fixed near the inlet and outlet surface in order to keep the cross sectional area of both surfaces constant [23]. The tangential velocity at the inlet and outlet was set to zero, which was imposed weakly. In [2], the reader can find a strong boundary condition at the inlet and outlet boundary that is a rough approximation of the true inflow and outflow velocity, but this option does not utilize the full potential of data assimilation of the echo-PIV data into the simulation. Along the ventricle wall, the velocity was specified to equal the wall displacement rate. Using the echo-PIV data we calculated the Reynolds number to be $Re = 560$.

The echo-PIV weight (w_{PIV}) should be calculated such that the more accurate velocity data is matched more accurately by the numerical solution and the numerical solution should not match the echo-PIV data when the data is less accurate. In [28], an optimal method of calculating w_{PIV} was given (equation 3.6), but that approach requires that the standard deviation (σ) of the experimental data is known.

$$w_{PIV} = \frac{1}{\sigma^2}. \tag{3.6}$$

An alternative approach is required for assimilating the echo-PIV data in the left ventricle because the standard deviation is not known and varies with time and microbubble concentration. An approach tested here is to estimate the weight using the temporal change in the PIV velocity vectors at the same spatial location and adjacent time steps. Here, w_{PIV} is calculated using:

$$w_{PIV} = \frac{u \cdot v}{0.5 \cdot \|u\| + 0.5 \cdot \|v\|}, \tag{3.7}$$

where u and v are the PIV velocity vectors at the same spatial location and adjacent time steps. The implication of this equation is that small temporal changes in the velocity are considered to be an indication of 'good' or more accurate data, supporting the use of a larger weight, and large temporal changes in the velocity are believed to be evidence for erroneous data, indicative of the need for a small weight. If the computed value is negative at any node, the data is categorized as inconsistent and $w_{PIV} = 0.0$ for that point. Equation (3.7) can also be used to calculate w_{PIV} locally on the PIV plane, a global approach was used here since the spatial accuracy of the ultrasound probe does not vary much. Once we have calculated w_{PIV} using equation (3.7) we simply compute the average w_{PIV} for the entire PIV plane. Even though none of the boundary data is known exactly (e.g., in many cases the exact location of the boundary is not known), the value for w_{Γ}, which is the weight for weak external boundary conditions on $\boldsymbol{u}, \boldsymbol{\omega}$, and $\boldsymbol{r}$, is set to 1.0 based on fact that larger errors are present in the PIV data than along the external boundaries.

3.3.3 Moving Mesh

In order to accurately model blood flow in the left ventricle, a moving mesh method is utilized to account for the expansion and contraction of the left ventricle and the changing shape of the fluid domain. There are several methods available to handle the moving domain problem [23]. A pseudo-solid domain mapping technique is used here and is based on solving a compressible elasticity equation to move the fluid mesh [24]. Equation (3.8) is the linear, compressible elasticity equation, where $\boldsymbol{s}$ is the displacement and μ is a Lamé coefficient. Based on previous studies, $\mu = 1.0$ is used throughout the simulations shown in the results section [2].

$$\mu\nabla(\nabla\cdot\boldsymbol{s})+\nabla^2\boldsymbol{s}=0\ \ in\ \Omega. \tag{3.8}$$

The WLSFEM framework is used, once again. To rewrite equation (3.8) into a first-order system, a new variable, U, equal to the gradient of displacement, $\boldsymbol{s}$, is defined. Rewriting the elasticity equation as a first-order system with the new variable gives:

$$\begin{gathered} U-\nabla\boldsymbol{s}=0\ \ in\ \Omega, \\ \mu\nabla\big(tr(U)\big)+\nabla\cdot U=0\ \ \ in\ \Omega, \\ \nabla\times U=0\ \ \ in\ \Omega. \end{gathered} \tag{3.9}$$

In system (3.9), U is the gradient of the displacement, $\boldsymbol{s}$, and $tr(U)=U_{11}+U_{22}+U_{33}$. The corresponding functional for system (3.9) is:

$$\begin{aligned} F_s(s,U{:}\,0)=\|\mathrm{U}-\nabla\mathbf{s}\|_{0,\Omega}^2+\left\|\mu\nabla\big(tr(U)\big)+\nabla\cdot U\right\|_{0,\Omega}^2+\|\nabla\times\mathrm{U}\|_{0,\Omega}^2 \\ +\frac{1}{h}\|\mathbf{s}-\mathrm{g}_s\|_{0,\Gamma}^2, \end{aligned} \tag{3.10}$$

where g_s is the given displacement on the surface (Γ). Using a moving mesh for the fluid domain creates artificial advection due to mesh displacement, and the effects of mesh motion are subtracted from the advective velocity in the Navier-Stokes equation [2, 17, 24]. The functional associated with the elasticity equation is minimized and discretized using the same approach (and typically the same finite element space) as the Navier-Stokes equation described above [14, 25, 26].

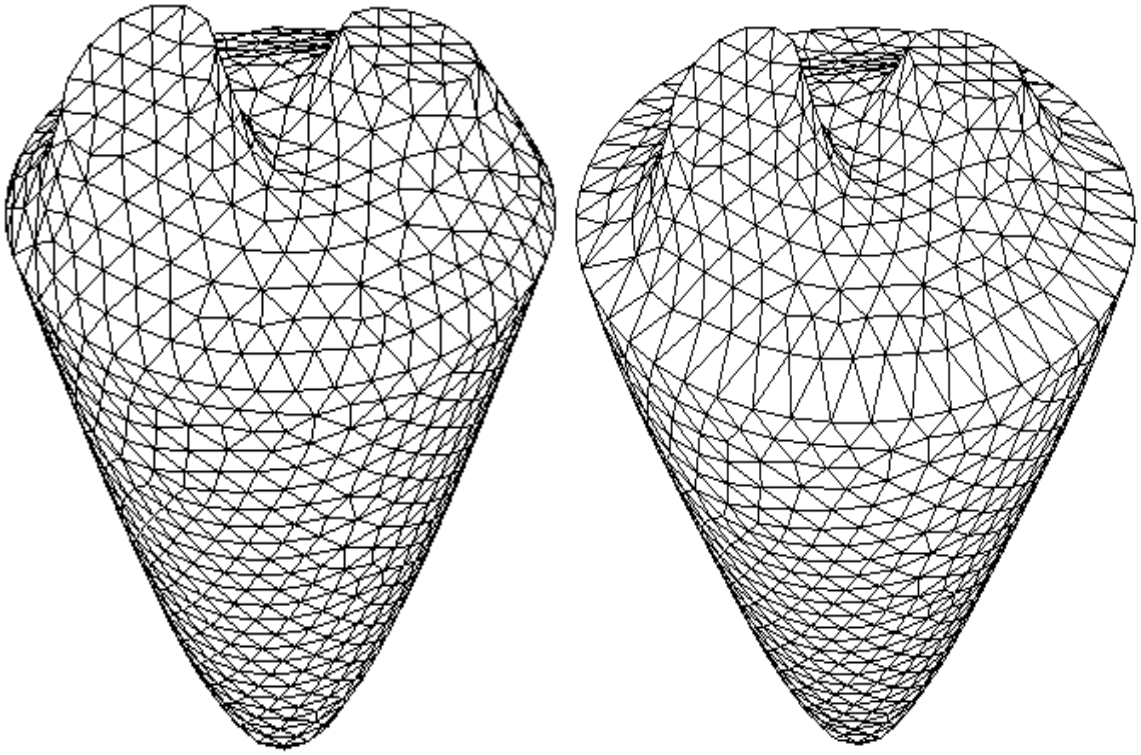
3.3.4 Implementation

In order to solve the WLSFEM problem for blood flow in the left ventricle, the equations in both systems (3.4) and (3.9) must be linearized using the Gauss-Newton method. The finite elements that we have used in all our simulations are 10-node

tetrahedron with quadratic basis functions. When solving the Navier-Stokes equations in a Galerkin finite element setting, the finite element spaces for each variable must be chosen so that the inf-sup condition is satisfied [11, 12, 15, 16, 27]. This means an equal order test functions for velocity and pressure will cause the Galerkin finite element method to be unstable [11, 26, 29]. The WLSFEM framework allows equal order test functions for all variables, including both the velocity and pressure approximation for the Navier-Stokes equations [15, 16]. This feature is an attractive advantage for the WLSFEM framework since is reduces programing complexity. The temporal discretization is handled using a 2^{nd}-order backward difference formula (BDF-2), which is unconditionally stable.

An in-house C code called ParaFOS was developed, which imports tetrahedral meshes from the Cubit (14.0) mesh generation package, developed and maintained by Sandia National Laboratory. An example of the mesh generated by Cubit is shown in Figure 3.2. ParaFOS was developed to run on distributed memory clusters using MPI, and the mesh is spatially partitioned using Metis [30]. This allows the ParaFOS code to be scalable to thousands of processors. The linear matrix problem was solved using a preconditioned CG (Conjugate Gradient) algorithm preconditioned with BoomerAMG, a parallel algebraic multigrid solver developed by Lawrence Livermore National Laboratory and part of the *hypre* algorithm library [31]. The convergence criteria for both CG iterations and Gauss-Newton iteration were fixed to 10^{-6}, which is a fraction of the discretization error [32]. The relative change of functional (3.5) was used as the criteria for Newton convergence. The computational time for the blood flow simulation varied with the total number of elements in mesh and the finest mesh, which has approximately 5400 total

elements, required about 70-100 hours of computational time to simulate 135 times steps, with a step size of 0.019 (dimensionless time). The simulation were performed on a Dell workstation T7500 with dual 6-core Xeon E5645.



(a) (b)

Figure 3.2. A tetrahedral mesh of the left ventricle at minimum volume (a). The same mesh at maximum volume after the final stage of diastole (b).

3.4. Results & Discussion

Our primary interest in this paper is to evaluate the assimilation of echo-PIV data using the WLSFEM method. The echo-PIV data was obtained from blood flow in the left ventricle of a pig, using ultrasound imaging with a temporal resolution of 150 frames/s [9]. In order to accurately capture the physics of blood flow in the left ventricle, the walls are moved (i.e., the wall displacement boundary condition is specified) based on the ejection fraction and the wall location observed in the ultrasound images. The geometry of the left ventricle for the simulation was constructed using the ultrasound images, which are limited to a few projection planes of the heart, and also based on the geometry used in previous CFD studies [33]. Even though the geometry was based on a human heart, while the echo-PIV was conducted on a pig, both have a similar size and morphology [34].

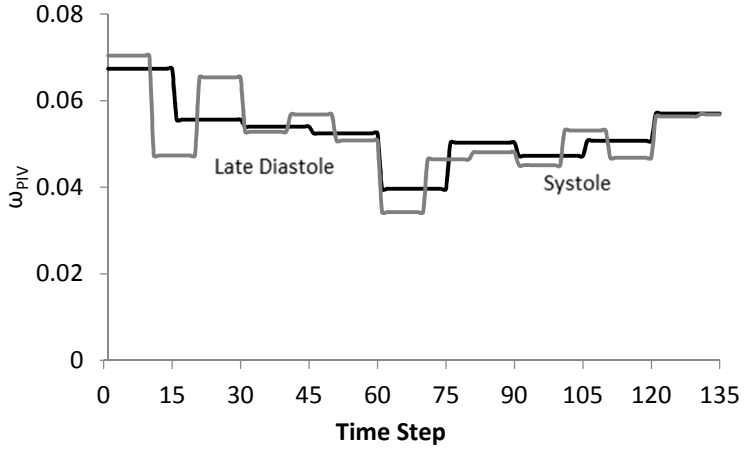
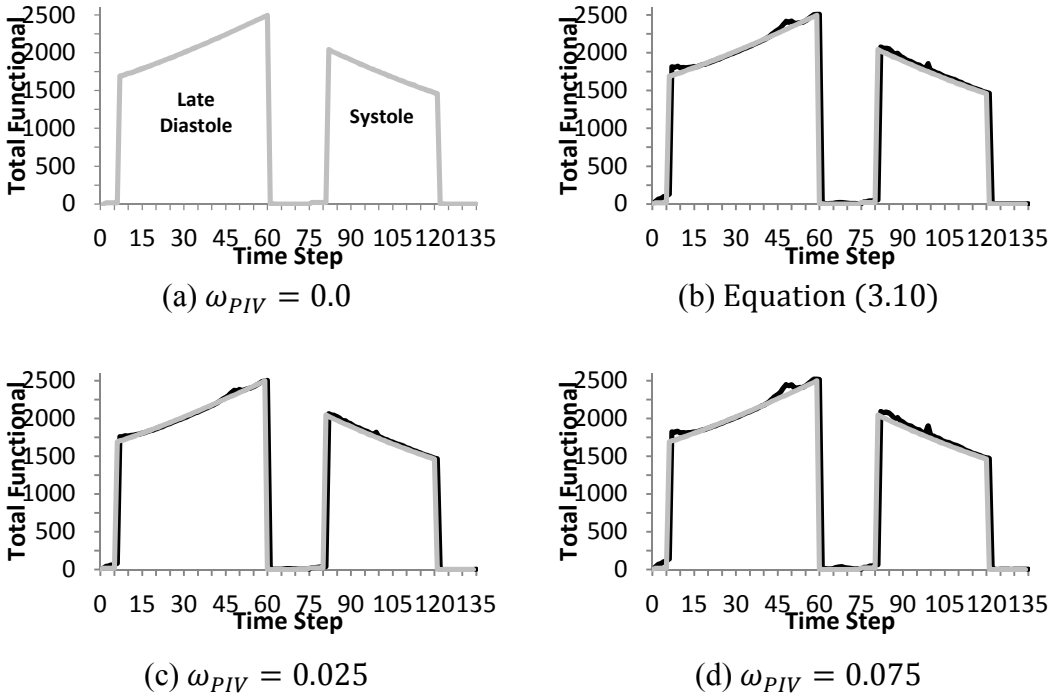


Figure 3.3.Values of ω_{PIV} calculate using equation (3.7) at every time step for a total 135 time steps.

In Figure 3.2, the finite element mesh for the left ventricle is shown at 2 different time points: (a) after ejection, when the left ventricle contains a minimal volume of blood, and (b) at late diastole after filling, when the ventricle has a maximum volume. The mesh shown in Figure 3.2 contains approximately 16,000 quadratic (10 node) elements. In order to quantify the impact of the echo-PIV data on the numerical solution, we have used various weightings, ω_{PIV}, for the assimilated echo-PIV data . Because the boundary data are believed to be more accurate than the echo-PIV data (but obviously not exact) our focus in this paper is limited to $\omega_{PIV} < \omega_{\Gamma}$. In Figure 3.3, the values calculated for ω_{PIV} using equation (3.7) are plotted with respect to the time-step for the simulation. The analysis, using equation (3.7), was done in sets of 15 time steps (black) and 10 time steps (gray), and the total number of frames was 135. The accuracy of the echo-PIV data does not appear to change significantly over these short time spans. In Figure 3.3, we can observe a mild 'U'-shaped curve for the calculated values of ω_{PIV} with a lower weighting (i.e., less temporally consistent and, thus, probably less accurate data) in very late diastole and very early systole (i.e., approximately time step 55-75). It appears that the time steps with the stronger, more directional flows – filling and ejection – lead to higher ω_{PIV} values. The transition time period (time steps 55-75) when the ventricle is fully expanded and the data is least temporally consistent due to the result of more significant out of plane motion for the microbubbles, and this may lead to the smaller ω_{PIV} values that are calculated for those time steps.

Regardless of the cause for the variability in the temporal consistence of the echo-PIV data, the combination of WLSFEM and equation (3.10) for estimating ω_{PIV} provides

a unique approach in data assimilation to identify and match the data that is probably more accurate more closely with the numerical solution and the less accurate data, less closely.



(a) $\omega_{PIV} = 0.0$ (b) Equation (3.10)

(c) $\omega_{PIV} = 0.025$ (d) $\omega_{PIV} = 0.075$

Figure 3.4. Impact of data assimilation on the functional at each time step for mesh with approximately 918 element. The gray line represents the base case in, which the simulation was run without any echo-PIV data.

In order to quantify the effects of the assimilated data on the numerical solution, we examine the functional value, $F(\boldsymbol{u}, \boldsymbol{\omega}, \boldsymbol{r}: 0)$ defined earlier in equation (3.5), at each time step. The functional is a globally reliable measure of the error in the numerical solution for solving the model equations. Hence, a large functional value reflects a numerical solution that does not solve the model equations (i.e., the Navier-Stokes equations) as well as a numerical solution that gives a smaller functional value. When $\omega_{PIV} = 0.0$ (Gray line throughout Figure 3.4) there is no data being assimilated into the numerical solution, and

this case will be used as the basis for the evaluating the impact of the assimilated data on the numerical solution. Throughout Figure 3.4, the functional values increases during the late diastole stage (i.e., filling) and decrease during the systole stage (i.e., ejection). This is partially due to the mesh motion causing the mesh size (h) to increase, and the more rapid increases and decreases in the functional are due to the more dynamic and complex nature of the flow. During systole in Figure 3.4, the functional value decrease, since the numerical model is better able to satisfy the model equations and the mesh size has decreased with the reduced volume of the left ventricle. The transition period from late diastole to systole has a much smaller functional value due to the reduced blood velocity, which makes it easier for the numerical solution to capture the physics of the blood flow in left ventricle. The transitional period is when there is an isovolumic period where no blood is coming in or going out, and at the end of this period is where the ejection phase starts (systole).

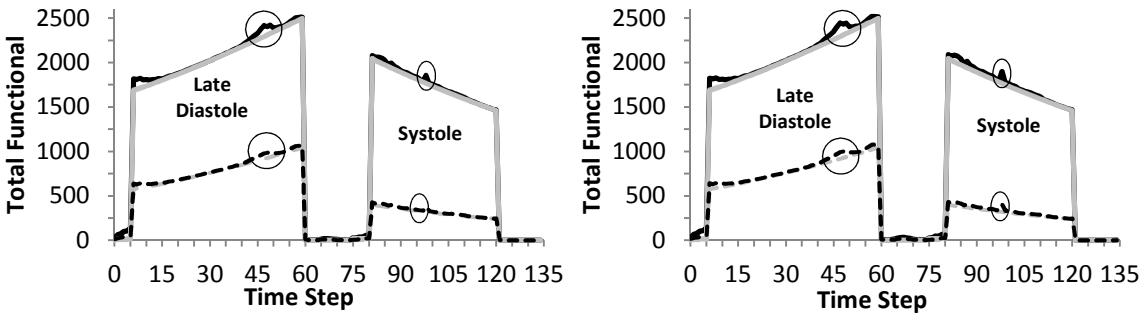
For the base case, where $\omega_{PIV} = 0.0$, the Navier-Stokes equations does not capture the exact physics of blood flow in the left ventricle because the numerical model does not incorporated a number of small features that impact blood flow, most significantly, the mitral valve [35]. Simplifying the model and leaving out some of these small features is essential for reducing the computational cost and the complexity of implementing the numerical model. Data assimilation provides an avenue for, at least partially recovering the information that is lost due to these modeling simplifications.

Figure 3.4 (b) shows the functional value at every time step for the case with (black line) and without assimilated echo-PIV data (gray line), when the value of ω_{PIV} was calculated using equation (3.10) at each time step. The average value of ω_{PIV} over all time

steps using equation (3.10) was 0.052. In addition to calculating ω_{PIV} using equation (3.10), the impact of a fixed value for ω_{PIV} at every time step was also tested using, $\omega_{PIV} = 0.025$ and $\omega_{PIV} = 0.075$. The results of these two cases are shown in Figure 3.4 (c) and Figure 3.4 (d), respectively.

For all cases with data assimilation, Figure 3.4 (b, c, and d), we observe an increase in the functional values when data are assimilated relative to the case without assimilation both near the beginning and near the end of diastole. Whenever we observe an increase in the functional with data assimilation relative to the base case, we know that these are time points where the assimilated data is not consistent with the solution to the model equations and the assimilated data is having the greatest impact on the numerical solution. This increase in the impact of the echo-PIV data could possibly be due to larger errors in the assimilated data, but that effect should be addressed through use of a variable value for ω_{PIV}. The more likely possibility is that these are the time points where the simplifications in the numerical model (e.g., the lack of a mitral valve or an aortic valve in the model) are most significant. It can also be seen in Figure 3.4 that larger values for ω_{PIV} lead to larger increases in the functional relative to the base model without data assimilation. Comparing the results from Figure 3.4 (b) and (c), the increase in the functional value is not as high for the late diastolic stage. For $\omega_{PIV} = 0.025$, (Figure 3.4(c)) shows closer agreement to the base case $\omega_{PIV} = 0.0$, but, of course, the echo-PIV data has less influence on the numerical solution due to the smaller weight, which may not be desirable if the data is helping to recover some physical effects that are not captured by the mathematical model. For the case $\omega_{PIV} = 0.025$, using a very small weight implies that

the model solution is more important compared to the data that are being assimilated, and this may lead to the loss of important information that is only available from the echo-PIV data (i.e., features that are not included in the model). During the systole stage we observe similar trends when $\omega_{PIV} = 0.025$. Figure 3.4 (d) when $\omega_{PIV} = 0.075$ shows a distinct increase in the functional value relative to the base case without data assimilation, indicating that the echo-PIV data are either noisy, i.e., temporally inconsistent, or it contains flow features not captured by the numerical model. The increase in the function value in the systole region is much smaller when equation (3.10) is used to calculate ω_{PIV}, and this demonstrates the ability of equation (3.10) to calculate a useful value for ω_{PIV}.



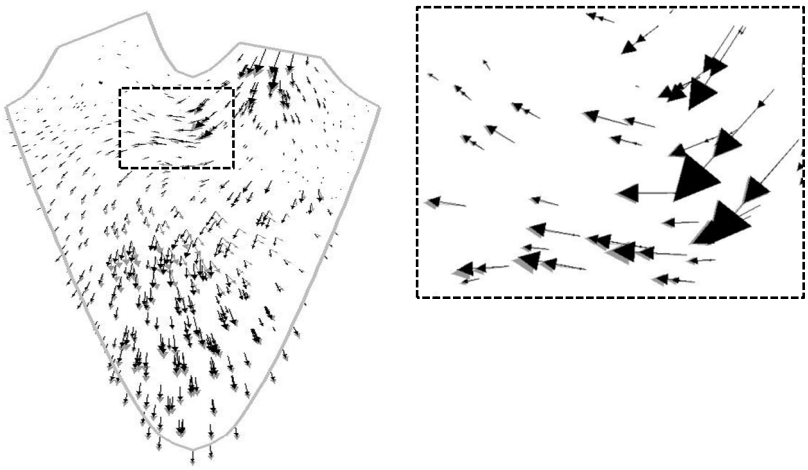
(a) Equation (3.10) (b) $\omega_{PIV} = 0.075$

Figure 3.5. Impact of mesh refinement on the functional at each time step for mesh with approximately 918 elements (solid lines) and approximately 5400 elements (dotted lines), the gray line represents the base case where the simulation was ran without any echo-PIV data.

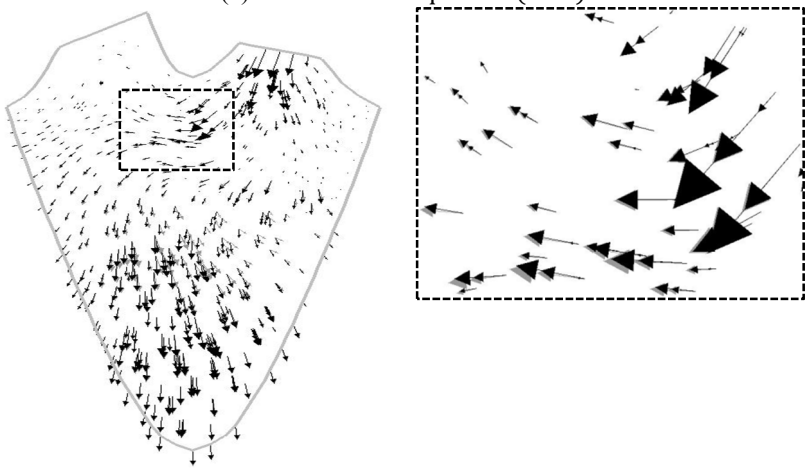
The effects of mesh resolution are shown in Figure 3.5, for cases with and without echo-PIV data assimilation. Increasing the mesh resolution provides a significant decrease in the functional value, which is consistent with finite element theory [15, 16]. The increase

in the number of elements also increases the computational time 3-4 times. In Figure 3.5 (a), during the late diastole phase, the impacts of mesh refinement are shown when calculating ω_{PIV} using equation (3.10). The circle area in Figure 3.5(a) shows that for the coarser mesh, the functional relative to the base case is much higher than at the same time step for the finer mesh. The increase in number of nodes in the finer mesh improves the numerical solution even when noisy data is being assimilated. Similar observation can be made during the systole phase (see the circled region in Figure 3.5 (a) where the relative increase in the functional is reduced when the mesh is refined). Keeping ω_{PIV} constant is less effective because, even with mesh refinement, there is still a significant increase in the functional relative to the base case at certain time points.

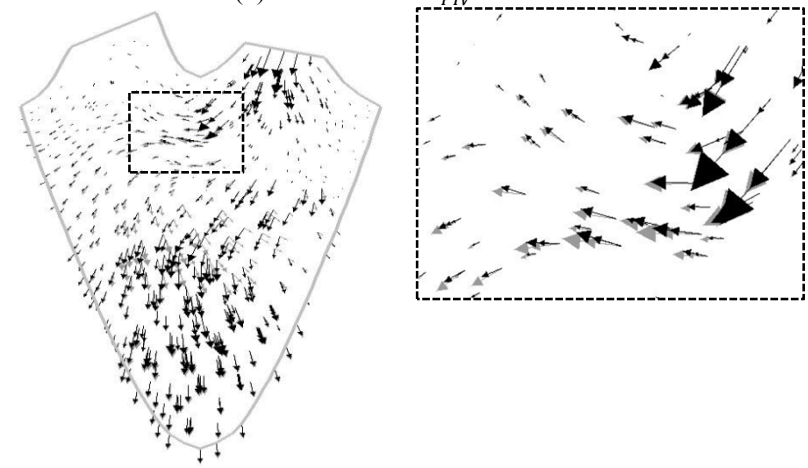
Using the functional defined in equation (3.5) we were able to evaluate the impact of data assimilation on the numerical solution in terms of satisfying the Navier-Stokes equations. In order to quantify the effect of the assimilated data on the blood velocity prediction at specific spatial locations, the change in the blood flow velocity with and without echo-PIV data is compared. The reason for focusing the analysis on the echo-PIV plane within the mesh is because the impact of the data will be greatest on this plane, and moving away from this plane reduced the effects of the assimilated echo-PIV data.



(a) Late Diastole Equation (3.10)

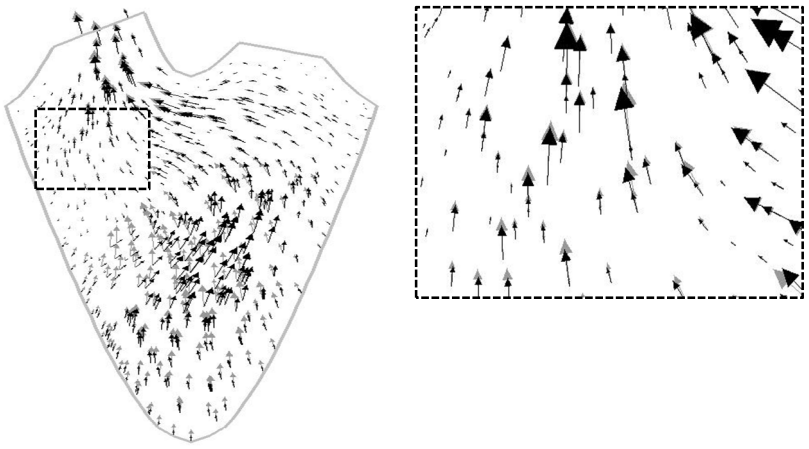


(b) Late Diastole $\omega_{PIV} = 0.025$

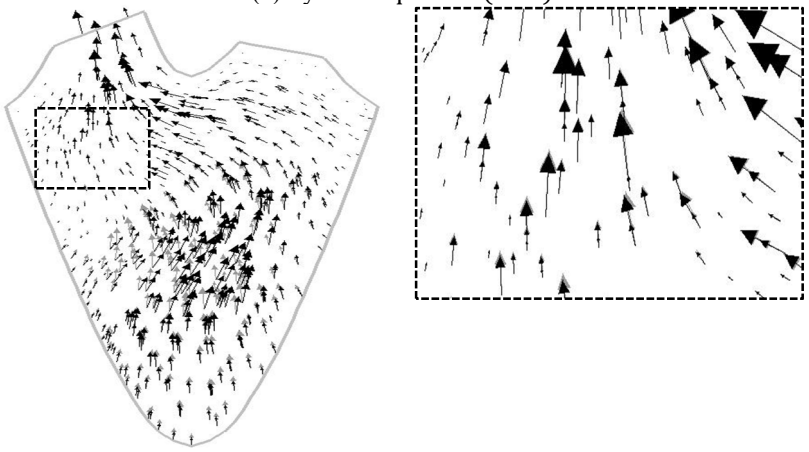


(c) Late Diastole $\omega_{PIV} = 0.075$

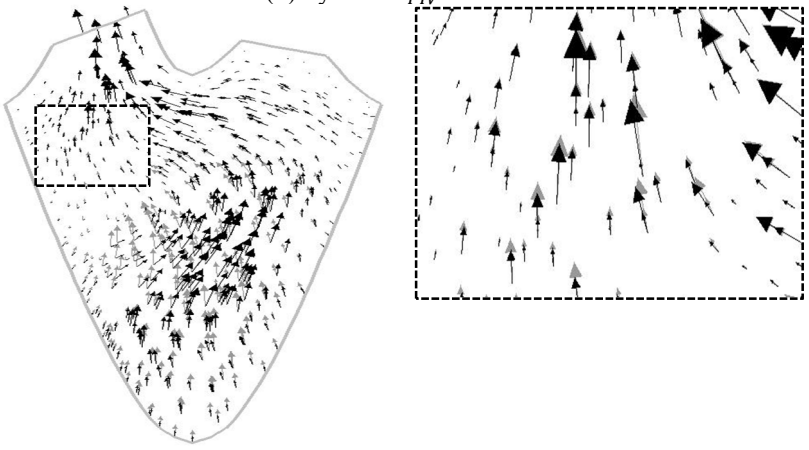
Figure 3.6. Comparison of velocity field with (black) and without (gray) echo-PIV data during late diastole ("filling") with different cases of ω_{PIV} and the boxed figure shows the focused region.



(a) Systole Equation (3.10)



(b) Systole $\omega_{PIV} = 0.025$

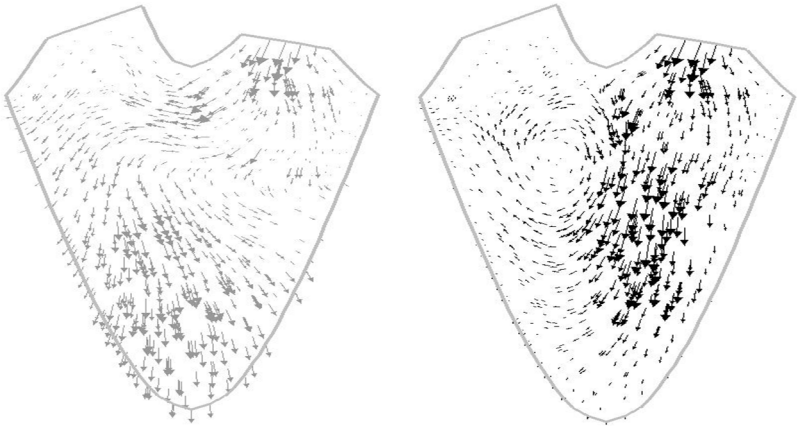


(c) Systole $\omega_{PIV} = 0.075$

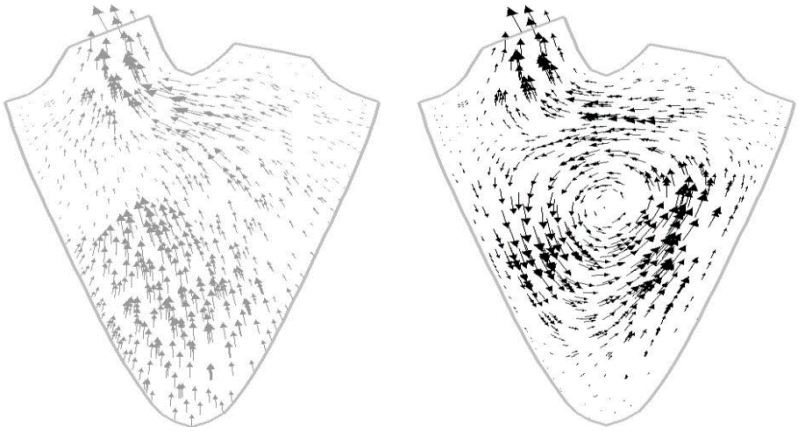
Figure 3.7. Comparison of velocity field with (black) and without (gray) echo-PIV data during systole ("ejection") with different cases of ω_{PIV} and the boxed figure shows the focused region.

In Figure 3.6, the PIV plane is shown with the velocity at each point (mesh node) both with (black) and without (grey) echo-PIV data assimilated during late diastole (i.e., filling). In Figure 3.6, the boxed area was chosen to be closer to the inlet so that the impact of data assimilation can be quantified. In Figure 3.6 (a), the effects of calculating ω_{PIV} using equation (3.10) are shown with the middle of the plane showing the largest impact from the assimilated the echo-PIV data. Comparing Figure 3.6 (b) and (c), the results show the expected larger impact from the echo-PIV data with the larger weight, ω_{PIV}. It should also be noted that the simulations with a constant value for ω_{PIV} (i.e., not using equation (3.10)) , showed an increase in computational time due to an increase in the number of CG and Newton iterations.

Throughout Figure 3.7, a similar analysis as in Figure 3.6 was conducted but focusing on the systole (ejection) phase. The middle of the planes shown in Figure 3.7 has the largest difference in the velocity field. The boxed region closer to the outlet is expanded and shown in more detail in the adjacent image. In Figure 3.7 (a) the boxed region shows slight changes to the velocity field as the fluid approaches the outlet and the impact of the boundary condition used for the numerical solution becomes dominant. In Figure 3.7 (b) using the smaller constant weight shows hardly any influence due to data assimilation in the boxed region. In contrast, Figure 3.7(c) shows a slightly more significant impact to the velocity field due to the assimilated data.



(a) Late Diastole $\omega_{PIV} = 0.0$ (b) Late Diastole Equation (3.10)



(c) Systole $\omega_{PIV} = 0.0$ (d) Systole Equation (3.10)

Figure 3.8. Comparison of velocity field with (black) and without (gray) echo-PIV data during late diastole ("Filing") and systole ("Ejection").

Figure 3.6 and Figure 3.7 highlight some representative time steps during the filling and early ejection phases of the simulation. For these times steps, as well as most other time steps, the assimilated echo-PIV data have an impact on the numerical solution, but the changes to the numerical solution due to the assimilated data are relatively small. This is exactly what is expected if the model is reasonable and the data do not contain significant amounts of error. Figure 3.8, in contrast, highlights two time steps where the assimilated

data have a larger significant impact on the numerical solution and, in fact, adds a significant vertical structure to the flow.

In Figure 3.8, the velocity field without any echo-PIV data assimilated (a) is compared to the velocity with data assimilation (b), and significant differences in the velocity field can be seen. When data are being assimilated we can clearly see the formation of a secondary flow vortex, probably due to the mitral valve, and the numerical model is capturing these effects with the help of the echo-PIV data even though the model does not include any mitral valve. Similar observation can be made by comparing, Figure 3.8 (c) and Figure 3.8 (d) we can see the vortex formation in the middle of the plane in Figure 3.8 (d). Where else Figure 3.8 (c) shows no such secondary flow occur within the model, this comparison shows the importance of data assimilation when making simplification to the numerical model. The comparison also demonstrates the WLSFEM framework as a powerful method for assimilating data into a numerical model.

The use of the WLSFEM for data assimilation has a number of advantages for the problem of interest described here, specifically, the assimilation of echo-PIV data into a model of flow in the left ventricle. However, it is important to emphasize that WLSFEM for data assimilation (and even the idea of data assimilation) is not beneficial for some categories of problems. If the data to be assimilated contains a significant amount of noise or other errors, it is unlikely to be beneficial and should probably not be assimilated. The technique described here for estimating ω_{PIV} is helpful in identifying situations where the data is of such low quality that it should not be assimilated, and if the WLSFEM method is used correctly, the small ω_{PIV} value will result in the assimilated data having a negligible

impact on the numerical solution. Similarly, if a highly accurate mathematical model is available for simulating the physical problem, it is unlikely that assimilating experimental data will provide any benefit. See [7, 28] and the reference therein. Whenever the assimilated data has almost no effect globally on the numerical solution, it is likely that the mathematical model is highly accurate and data assimilation is unnecessary. In previous work [28], the WLSFEM method was used to assimilate optical PIV data for flow through a straight, rigid cylinder. For this simple problem, a very accurate mathematical model is available (even an analytical solution is model is known since the solution is in the finite element space) and the assimilated data did not significantly impact the numerical solution if it was properly weighted. The ideal problem for data assimilation by the WLSFEM is one in which the mathematical model is 'good' (i.e., the numerical solution is generally quantitatively similar to the experimental data) but still contains some simplifying assumptions or inaccurate boundary conditions. For this situation, as demonstrated here, sufficiently accurate assimilated data can beneficially impact the numerical solution.

3.5. Conclusion

The assimilation of experimental data into the numerical solution of the Navier-Stokes equation remains an active area of research and is of interest to many investigators. The problem focused on here was to incorporate echo-PIV data into the numerical simulation of blood flow in the left ventricle. A particularly challenging aspect to this problem is that the accuracy (i.e., the standard deviation) of the echo-PIV data is not known and probably varies with time. The WLSFEM framework demonstrated here provides many advantages for assimilating data into numerical model, since the data can be matched based on the estimated accuracy or reliability of the data and, significantly, the computational costs is negligibly higher than a single CFD simulation. We have demonstrated the impact of using temporal changes in the data (i.e., using equation (3.10)) as an estimator of the accuracy of the data, when standard deviation is not available. The impact of the assimilated echo-PIV data was assessed through the WLSFEM functional and by examining the velocity a various points in the numerical approximation.

3.6. References

1. Borazjani, I., et al., *Left Ventricular Flow Analysis: Recent Advances in Numerical Methods and Applications in Cardiac Ultrasound.* Computational and Mathematical Methods in Medicine, 2013.

2. Wei, F., et al., *Weighted Least-Squares Finite Element Method for Cardiac Blood Flow Simulation with Echocardiographic Data.* Computational and Mathematical Methods in Medicine, 2012.

3. Westerdale, J., et al., *Flow Velocity Vector Fields by Ultrasound Particle Imaging Velocimetry In Vitro Comparison With Optical Flow Velocimetry.* Journal of Ultrasound in Medicine, 2011. **30**(2): p. 187-195.

4. Jiamsripong, P., et al., *Impact of Acute Moderate Elevation in Left Ventricular Afterload on Diastolic Transmitral Flow Efficiency: Analysis by Vortex Formation Time.* Journal of the American Society of Echocardiography, 2009. **22**(4): p. 427-431.

5. Sengupta, P.P., et al., *Left ventricular form and function revisited: Applied translational science to cardiovascular ultrasound imaging.* Journal of the American Society of Echocardiography, 2007. **20**(5): p. 539-551.

6. Sengupta, P.P., et al., *Left ventricular structure and function - Basic science for cardiac imaging.* Journal of the American College of Cardiology, 2006. **48**(10): p. 1988-2001.

7. Dwight, R.P., *Bayesian Inference for Data Assimilation using Least-Squares Finite Element Methods.* 9th World Congress on Computational Mechanics and 4th Asian Pacific Congress on Computational Mechanics, 2010. **10**.

8. D'Elia, M., M. Perego, and A. Veneziani, *A Variational Data Assimilation Procedure for the Incompressible Navier-Stokes Equations in Hemodynamics.* Journal of Scientific Computing, 2012. **52**(2): p. 340-359.

9. Sengupta, P.P., et al., *Emerging Trends in CV Flow Visualization.* Jacc-Cardiovascular Imaging, 2012. **5**(3): p. 305-316.

10. Thielicke, W. and E. Stamhuis, *PIVlab–time-resolved digital particle image velocimetry tool for matlab.* Published under the BSD license, programmed with MATLAB, 2010. **7**(0.246): p. R14.

11. Gresho, P.M., R.L. Sani, and M.S. Engelman, *Incompressible flow and the finite element method*2000, Chichester England ; New York: Wiley.

12. Donéa, J. and A. Huerta, *Finite element methods for flow problems*2003, Chichester ; Hoboken, NJ: Wiley. xi, 350 p.

13. Griffith, B.E., et al., *An adaptive, formally second order accurate version of the immersed boundary method.* Journal of Computational Physics, 2007. **223**(1): p. 10-49.

14. Heys, J.J., et al., *Modeling 3-D compliant blood flow with fosls.* Biomedical Sciences Instrumentation, Vol 40, 2004. **449**: p. 193-199.

15. Bochev, P.B. and M.D. Gunzburger, *Least-Squares Finite Element Methods.* Least-Squares Finite Element Methods, 2009. **166**: p. 1-660.

16. Jiang, B.N., *A Least-Squares Finite-Element Method for Incompressible Navier-Stokes Problems.* International Journal for Numerical Methods in Fluids, 1992. **14**(7): p. 843-859.

17. Heys, J.J., et al., *First-order system least-squares (FOSLS) for modeling blood flow.* Medical Engineering & Physics, 2006. **28**(6): p. 495-503.

18. Heys, J.J., et al., *An alternative least-squares formulation of the Navier-Stokes equations with improved mass conservation.* Journal of Computational Physics, 2007. **226**(1): p. 994-1006.

19. Heys, J.J., et al., *On mass-conserving least-squares methods.* Siam Journal on Scientific Computing, 2006. **28**(5): p. 1675-1693.

20. Heys, J.J., et al., *Enhanced Mass Conservation in Least-Squares Methods for Navier-Stokes Equations.* Siam Journal on Scientific Computing, 2009. **31**(3): p. 2303-2321.

21. Manteuffel, T., et al., *Further results on error estimators for local refinement with first-order system least squares (FOSLS).* Numerical Linear Algebra with Applications, 2010. **17**(2-3): p. 387-413.

22. Adler, J., et al., *An Efficiency-Based Adaptive Refinement Scheme Applied to Incompressible, Resistive Magnetohydrodynamics.* Large-Scale Scientific Computing, 2010. **5910**: p. 1-13.

23. Bazilevs, Y., K. Takizawa, and T.E. Tezduyar, *Challenges and Directions in Computational Fluid-Structure Interaction.* Mathematical Models & Methods in Applied Sciences, 2013. **23**(2).

24. Sackinger, P.A., P.R. Schunk, and R.R. Rao, *A newton-raphson pseudo-solid domain mapping technique for free and moving boundary problems: A finite element implementation.* Journal of Computational Physics, 1996. **125**(1): p. 83-103.

25. Heys, J.J., et al., *First-order system least squares (FOSLS) for coupled fluid-elastic problems.* Journal of Computational Physics, 2004. **195**(2): p. 560-575.

26. Heys, J.J., et al., *First-order system least squares for elastohydrodynamics with application to flow in compliant blood vessels.* Biomedical Sciences Instrumentation, Vol 38, 2002. **38**: p. 277-282.

27. Bochev, P.B., *Analysis of least-squares finite element methods for the Navier-Stokes equations.* Siam Journal on Numerical Analysis, 1997. **34**(5): p. 1817-1844.

28. Heys, J.J., et al., *Weighted least-squares finite elements based on particle imaging velocimetry data.* Journal of Computational Physics, 2010. **229**(1): p. 107-118.

29. Rajaraman, P. and J.J. Heys, *Simulation of nanoparticle transport in airways using Petrov–Galerkin finite element methods.* International Journal for Numerical Methods in Biomedical Engineering, 2014. **30**(1): p. 103-116.

30. Karypis, G. and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs.* Siam Journal on Scientific Computing, 1998. **20**(1): p. 359-392.

31. Falgout, R.D. and U.M. Yang, *hypre: A library of high performance preconditioners.* Computational Science-Iccs 2002, Pt Iii, Proceedings, 2002. **2331**: p. 632-641.

32. Greenbaum, A., *Iterative methods for solving linear systems*. Frontiers in applied mathematics1997, Philadelphia, PA: Society for Industrial and Applied Mathematics. xiii, 220 p.

33. Cheng, Y.G., H. Oertel, and T. Schenkel, *Fluid-structure coupled CFD simulation of the left ventricular flow during filling phase.* Annals of Biomedical Engineering, 2005. **33**(5): p. 567-576.

34. Niekrasz, M., et al., *The Pig as Organ Donor for Man.* Transplantation Proceedings, 1992. **24**(2): p. 625-626.

35. Griffith, B.E., *Immersed boundary model of aortic heart valve dynamics with physiological driving and loading conditions.* International Journal for Numerical Methods in Biomedical Engineering, 2012. **28**(3): p. 317-345.

CHAPTER 4

COMPARISON OF CONTINUOUS AND DISCONTINUOUS FINITE ELEMENT METHODS FOR PARABOLIC DIFFERENTIAL EQAUTIONS EMPLOYING IMPLICIT TIME INTERGATION

Contribution of Authors and Co-Authors

Manuscript in Chapter 4

Author: Prathish K. Rajaraman

Contributions: Programing computational model of continuous FEM, data collection and analysis manuscript drafting and editing.

Co-Author: G. D. Vo

Contributions: Programing computational model of discontinuous FEM, data collection and analysis manuscript drafting and editing.

Co-Author: G. Hansen

Contributions: Conception and critical analysis of article, manuscript drafting and editing.

Co-Author: Jeffrey J. Heys

Contributions: Conception and critical analysis of article, manuscript drafting and editing.

Manuscript Information Page

Prathish K. Rajaraman [a], G.D. Vo [a], G. Hansen [b], and Jeffrey J. Heys [a]

Journal Name: International Journal of Numerical Methods for Heat & Fluid Flow

Status of Manuscript:

____ Prepared for submission to a peer-reviewed journal

_X_ Officially submitted to a peer-review journal

____ Accepted by a peer-reviewed journal

___ Published in a peer-reviewed journal

Emerald Group Publishing

Submitted in October 2015

[a] Chemical and Biological Engineering Department, Montana State University, Bozeman, MT 59717

[b] Sandia National Laboratories, Albuquerque, NM 87185-1321

4.1. Abstract

This paper compares the computational cost, stability, and accuracy (when possible) of continuous and discontinuous Galerkin Finite Element Method (GFEM) for different test problems including the advection-diffusion equation, viscous Burgers' equation, and Turing pattern formation equation system. The results show that, for implicit time integration, the continuous GFEM is typically 5-20 times less computationally expensive than the discontinuous GFEM using the same finite element mesh and element order. However, the discontinuous GFEM is significantly more stable than the continuous GFEM for advection dominated problems and is able to obtain accurate approximate solutions for cases where the classic, un-stabilized continuous GFEM fails.

4.2. Introduction

Parabolic differential equations arise in many applications in science and engineering, including the study of mass transport, heat transport, and fluid dynamics (Gresho and Sani 1998, Donea and Huerta 2003, Reddy and Gartling 2010, Gunzburger 2012). A simple, characteristic example is the heat equation:

$$\frac{\partial u}{\partial t} = k\nabla^2 u \tag{4.1}$$

where $u(\mathbf{x}, t)$ is the temperature and k is the constant thermal conductivity. Parabolic equations frequently have second-order derivatives with respect to space and first-order derivatives with respect to time. Discretization in time can be achieved using a number of different explicit or implicit methods, but the restrictive CFL condition on the size of an

explicit time step motivates the choice here of focusing on implicit time integration methods (Courant, Friedrichs et al. 1967). Spatial discretization approaches can largely be separated into finite difference methods and variational methods (Braess 2007). The Galerkin finite element method (GFEM) belongs to the later category and is the focus here.

The finite element method provides a formalism for generating discrete algorithms for approximating the solution of a differential equation (Brenner and Scott 2008). The basic idea of the method is to view a given domain as an composition of simple geometric shapes for which it is possible to generate the approximate solution of the differential equations by a variational or weighted-residual method (Reddy and Gartling 2010). The strengths of this approach include geometric flexibility and a straightforward method for obtaining high-order spatial accuracy (Deville, Fischer et al. 2002). However, there are also two well-known disadvantages of the classical continuous Galerkin finite element approach (Gresho and Sani 1998, Donea and Huerta 2003, Hesthaven and Warburton 2007). First, when using explicit time integration, the globally defined mass matrix must be inverted or lumped. Second, for problems in which information flows in a specific direction (e.g., in advection dominated problems), the approximate solution can suffer from instability. This instability is especially common when solving hyperbolic differential equations, but as demonstrated in the sequel, is also common in parabolic problems when the first-order advection term is large relative to the second-order term in the differential equation (Donea and Huerta 2003). In finite difference and finite volume methods, this instability is addressed using upwinding (Gresho and Sani 1998, Donea and Huerta 2003). The instability is often addressed in finite element methods through the use of one of the

many different Petrov-Galerkin methods, which typically adds artificial diffusion in some way to control the instability (Gresho and Sani 1998, Donea and Huerta 2003). The focus here is on classical continuous and discontinuous GFEM, not Petrov-Galerkin approaches.

The globally defined basis and test functions of the continuous GFEM leads to a mass matrix with off-diagonal structure and can display stability problems with advection or directionally dominated equations (Gresho and Sani 1998, Gunzburger 2012). Finite volume methods tend to not suffer from these two limitations, but they are rather difficult to extend to higher-order discrete spatial approximations when unstructured meshes are used (Gresho and Sani 1998). One approach to combine the strengths of the continuous GFEM with those of the finite volume method is through the use of the discontinuous GFEM (Cockburn, Nguyen et al. 2010, Cockburn, Gopalakrishnan et al. 2011). This approach retains the definition of elements, but the basis and test functions are only defined locally on each element. As a result, higher-order basis and test functions are usually straightforward to build, the mass matrix consists of local blocks for each element, and a numerical flux, which combines information from adjacent elements, can lead to stable upwinding schemes for problems with highly directional information flow. Unfortunately, these advantages for the discontinuous GFEM come at the price of an increase in the total number of degrees of freedom (or, equivalently, in the case of nodal basis functions, an increase in the number of nodes). A second potential disadvantage is the challenge of determining a proper form for the numerical flux for parabolic and elliptic differential equations (Rivière 2008).

Since the strengths of discontinuous GFEM relative to classical continuous GFEM include enhanced stability through the numerical flux and efficient explicit time integration due to the local block nature of the mass matrix, much of the discontinuous GFEM development has focused on hyperbolic problems employing explicit time integration (Cockburn and Shu 2001, Arnold, Brezzi et al. 2002, Hesthaven and Warburton 2007). This class of problems is well suited to approximation by the discontinuous GFEM, as demonstrated in numerous studies (e.g.,(Hesthaven and Warburton 2007, Gabard, Gamallo et al. 2011)). For parabolic problems, there appears to be substantial variety in discontinuous GFEM (and hybrid discontinuous GFEM) approaches that have been considered (Werder, Gerdes et al. 2001, Cockburn, Nguyen et al. 2010, Wirasaet, Tanaka et al. 2010). Further, some studies have focused on discontinuous Galerkin time discretization methods and compared them to continuous GFEM or continuous Galerkin-Petrov finite element methods. For example, (Hussain, Schieweck et al. 2011) compared computational cost and accuracy for the parabolic heat equation using a geometric multigrid solver. Also, (Kubatko, Bunya et al. 2009) compared continuous and discontinuous finite element methods for shallow water models using explicit time stepping. We are not aware of existing work that has quantitatively compared the advantages and disadvantages of discontinuous GFEM with continuous GFEM for a range of parabolic problems employing implicit time integration.

For parabolic differential equations with implicit time integration, the stability advantages of using a numerical flux are only realized for highly advective problems. In fact, the proper form for the numerical flux can often be challenging to determine. The

local block mass matrix is less of an advantage because the solving of a large linear matrix is unavoidable. Finally, the increased number of degrees of freedom can be a greater liability because a linear matrix problem must be solved. In many ways, parabolic differential equations with implicit time integration are not ideally suited for approximation using discontinuous GFEM. However, the question we seek to answer is when, if ever, discontinuous GFEM should be considered for solving parabolic differential equations with implicit time integration. We seek to quantitatively answer this question by comparing discontinuous and continuous GFEM in terms of accuracy and computational time for a number of common parabolic equations. There are, of course, a number of limitations when trying to perform a quantitative comparison of two different numerical algorithms. The biggest limitation is implementing both algorithms in a fair way.

It is easy to make one approach look better than another approach by simply implementing one method more carefully. We have made every effort to avoid this situation by having the two algorithm share as much code as possible, but we believe it is impossible to achieve a perfectly fair comparison. Certainly, experts could examine the thousands of lines of code in each algorithm and identify small improvements. A second major limitation is that every problem, every parabolic differential equation is different, and it is practically impossible to perform a comparison for all possible parabolic equations. We have selected 3 characteristic parabolic equations (and one elliptic equation) that form a foundation from which the performance for other parabolic equations can be estimated.

4.3. Methods

The objective in developing the computational code for this comparison was to have as much shared code as practically possible between the continuous and discontinuous GFEM algorithms. To simplify the comparison, the code was written for serial execution in a single thread, and all code developed in-house was written in C++. The importing and exporting of the mesh description file and the writing of the final approximate solution all utilized the Exodus II library, written in C (Schoof and Yarberry 1994). The finite element meshes were all generated using the Cubit software (Sandia National Laboratory), and the geometries are all 2-dimensional and meshed with 3 or 6 node triangles using the TriMesh algorithm.

Both the continuous and discontinuous GFEM algorithms utilized some of the core packages from the Trilinos library (Heroux, Bartlett et al. 2003, Heroux and Willenbring 2003). Specifically, all global operators and vectors were built and stored using the Epetra package. The objective in selecting algorithms to solve the linear matrix problem was to utilize the most popular and robust methods available. Therefore, to compare performance using a direct matrix solver, the Amesos package from Trilinos is used (Sala, Stanley et al. 2007), and, to compare performance using an iterative matrix solver, the AztecOO GMRES algorithm with as many default settings as possible was used (Heroux and Willenbring 2003). Obviously, the computational performance could be improved through the use of more recent, specialized matrix solvers such as an algebraic multigrid solver. The problem with using these types of solvers in a comparison is that they often have a number of adjustable parameters that can significantly impact solver performance. It is difficult (or

impossible) to determine the optimal solver parameters for a wide range of problems and discretizations, and this makes it difficult to develop a fair comparison. If a uniform set of solver parameters is used in the comparison, it will almost always favor one discretization over another. The other issue with utilizing a wide range of matrix solvers is that, inevitably, some solvers will be left out and those might be very computationally efficient with one discretization. To minimize all these issues, every effort has been made to focus on just the most common linear solvers with very few adjustable parameters.

Since common libraries are used for mesh import and export, operator storage, linear operations, and linear solvers, the main sections of the algorithms that are not shared involve the building of basis functions and the construction of element level operators. For the continuous GFEM code, standard quadratic (i.e., 6-node triangles) nodal basis functions were generated, and the local element operators were based on the well-known Galerkin weak form (Gresho and Sani 1998, Donea and Huerta 2003, Brenner and Scott 2008, Reddy and Gartling 2010, Gunzburger 2012). In all cases, time integration was accomplished using the second-order accurate trapezoidal rule (i.e., Crank-Nicholson method). Due to concerns about the accuracy and stability of the trapezoid rule, time integration using the second-order Backward Difference Formula (BDF2) was also tested, and it gave results nearly identical to the trapezoid rule in every case tested.

While the continuous GFEM has a relatively standard weak form, the same cannot be said for the discontinuous GFEM when the differential equation contains a second-order term. A number of different forms for the numerical flux have been developed and tested for application to second-order terms, leading to methods such as the local discontinuous

Galerkin flux, internal penalty (IP) flux, and central stabilized flux (Arnold 1982, Hesthaven and Warburton 2007, Rivière 2008). In addition to choosing a form for the numerical flux, it is often necessary to select values for parameters in the numerical flux. We have chosen to focus on the internal penalty method here for three reasons: (1) it is probably the most popular choice, especially when one considers that the parameters for the local discontinuous Galerkin flux are often chosen so that it simplifies to the IP flux (Arnold, Brezzi et al. 2002), (2) it results in greater sparsity, and (3) the appropriate value for the penalty parameter is fairly well established (Arnold 1982, Hesthaven and Warburton 2007). The derivation of the IP weak form is outlined below for the simple case of the conductive heat equation.

The conductive heat equation, (4.1), on Ω can be rewritten as a first-order system:

$$\frac{\partial u}{\partial t} = \nabla \cdot q, \tag{4.2}$$

$$q = \nabla u. \tag{4.3}$$

Multiplying by a test function and integrating by parts over the K elements gives:

$$-(q_h, \nabla \phi_h)_\Omega + \sum_K (n \cdot q_h^*)_\Gamma = \left(\frac{\partial u}{\partial t}, \phi_h\right)_\Omega, \tag{4.4}$$

$$(q_h, \psi_h)_\Omega + \sum_K \left(u_h^* n \cdot \psi_h\right)_\Gamma = (\nabla \cdot \psi_h, u_h)_\Omega, \tag{4.5}$$

where q_h^* and u_h^* are the numerical flux terms on the boundary of every element (to be defined later). Defining $\{u\} = \frac{u^- + u^+}{2}$, where '$+$' and '$-$', signify the interior and exterior (or neighboring) element, respectively, and $[u] = n^- \cdot u^- + n^+ \cdot u^+$, then it can easily be shown that $\sum_K (n \cdot u, \phi)_\Gamma = \oint \{u\} \cdot [\phi]\, dx + \oint_{\Gamma_i} \{\phi\}[u]\, dx$, where Γ_i represents the set of

purely internal edges. This identity allows the previous system, (4.4) and (4.5), to be simplified:

$$-(q_h, \nabla\phi_h)_\Omega + \oint_\Gamma \{q_h^*\}[\phi_h]\,dx + \oint_{\Gamma_i} [q_h^*]\{\phi_h\}\,dx = \left(\frac{\partial u}{\partial t}, \phi_h\right)_\Omega, \tag{4.6}$$

$$(q_h, \psi_h)_\Omega + \oint_\Gamma [u_h^*]\{\psi_h\}\,dx + \oint_{\Gamma_i} \{u_h^*\}[\psi_h]\,dx = (\nabla\cdot\psi_h, u_h)_\Omega. \tag{4.7}$$

Finally, integrating by parts (again) gives:

$$(q_h, \psi_h)_\Omega = (\nabla u_h, \psi_h)_\Omega + \oint_\Gamma [u_h^* - u_h]\{\psi_h\}\,dx - \oint_{\Gamma_i} \{u_h^* - u_h\}[\psi_h]\,dx, \tag{4.8}$$

which is typically written as:

$$q_h = \nabla u_h + \mathcal{L}(u_h). \tag{4.9}$$

Assuming that the numerical flux is a single value and utilizing the IP form for the numerical flux gives: $[u_h^*] = 0, [q_h^*] = 0, \{u_h^*\} = u_h^* = \{u_h\}$, and $\{q_h^*\} = q_h^* = \{\nabla u_h\} - \tau[u_h]$ where τ is the penalty parameter. Substituting (9) and the IP numerical fluxes into (4.6) gives:

$$\left(\frac{\partial u}{\partial t}, \phi_h\right)_\Omega = -(\nabla u_h, \nabla\phi_h)_\Omega + \oint_\Gamma [u_h]\{\nabla\phi_h\} + \{\nabla u_h\}[\phi_h]\,dx - \oint_\Gamma \tau[u_h][\phi_h]\,dx. \tag{4.10}$$

One factor in comparing the continuous to discontinuous GFEM is the complexity of implementation, and this factor is very difficult to quantify. We will simply note that the IP weak form used here for the discontinuous GFEM is more complex to derive and implement than the associated continuous GFEM case. The discontinuous GFEM

algorithm used here was based on the algorithms presented in (Hesthaven and Warburton 2007). Specifically, the nodal basis functions and nodal element operators were based on algorithms presented in chapter 6 of Hesthaven and Warburton's book (Hesthaven and Warburton 2007), and careful comparisons were made between the C++ code developed here and the MATLAB code provided by Hesthaven and Warburton to help support a proper and correct implementation.

In order to simplify and clarify the comparison between continuous and discontinuous GFEM for parabolic equations, the test problems used in the comparison are described in the Results section so that the results can be closely connected to the relevant test problem.

4.4. Results

Four different test problems were selected for comparing the continuous and discontinuous GFEM. The first problem is the steady-state heat equation, or Poisson's equation, which is obviously not a parabolic equation, but has the advantages of sharing a number of properties with a simple parabolic equation, an analytical solution, and a focus on spatial error, not temporal error. The second problem is the advection-diffusion equation with decreasing diffusivity values. The third problem is the viscous Burger's equation, which is similar to the second problem, but is nonlinear. The final problem is the Turing pattern formation problem, which was included because it is a system of equations. We have deliberately avoided the Stokes and Navier-Stokes equations because they are the

subject of so much ongoing research in the field of continuous and discontinuous GFEM and there is not general agreement on the best approach.

4.4.1. Poisson's Equation

The first test problem is the 2-dimensional Poisson equation:

$$\Delta u = -2\pi^2 sin(\pi x) sin(\pi y), \tag{4.11}$$

with the righthand side chosen so that it has homogenous Dirichlet boundary conditions on the unit square and an analytical solution. Table 4.1 summarizes the computational time and error in the L^∞-norm for the the continuous and discontinuous GFEM.

Table 4.1: The error in the L^∞-norm and the total computational time (in seconds) for three different solvers for solving (4.11) using the continuous and discontinuous GFEM with various mesh resolution. Amesos is a direct solver, Aztec is an ILU-preconditioned GMRES solver, and ML is an algebraic multigrid solver.

Number	Continuous GFEM				Discontinuous GFEM			
Elements	Error	Amesos	Aztec	ML	Error	Amesos	Aztec	ML
666	2.80E-05	0.043	0.056	0.08	3.10E-05	0.52	0.66	1.3
2664	2.70E-06	0.2	0.25	0.29	3.00E-06	4.5	4.4	5.5
4448	1.50E-06	0.42	0.39	0.57	1.50E-06	8.3	10.5	10.6
10656	3.70E-07	1.3	1.1	1.8	3.90E-07	23.2	39.6	28.6

A number of important observations can be made from the results shown in Table 4.1. First, for this problem possessing a very smooth solution, both methods have practically equal accuracy in the L^∞-norm. This is not surprising since both approaches used the same 6-node, triangular elements and have the same order of accuracy. The second observation that can be made is that the computational cost for the discontinuous GFEM

approach is consistently higher than for the continuous case. For Amesos, a sparse, direct solver, the computational cost is approximately a factor of 10 higher for the discontinuous algorithm relative to the continuous, and the solver displays surprisingly good scalability in both cases. This trend continues with the Aztec solver, an ILU-preconditioned GMRES algorithm, but here the solver does not scale as well in the discontinuous case (order n^2) compared to the continuous case (order $n^{3/2}$). The algebraic multigrid solver, ML, gives similar scalability in either case, but this problem is really too small to take advantage of benefits of a multilevel solver. The performance of the ML solver could be improved by adjusting some of the solver parameters or using specialized aggregation schemes (Olson and Schroder 2011), but the default parameters were used for both cases here to keep the comparison as fair as possible.

The higher computational cost of the discontinuous algorithm is explained by the fact that the discontinuous approach has many more degrees-of-freedom relative to the continuous case. The 6 nodes of every triangular element lie either on the edge of the element or the vertex, so the unknown associated with each and every node is duplicated at least once in the discontinuous algorithm. As a result, the discontinuous approach has approximately 3 times as many unknowns as the continuous approach, and those additional unknowns significantly increase the computational costs. For this particular Poisson problem, there is not a compelling reason to use a discontinuous GFEM approach. However, when we add advection in the next example, the case for using discontinuous GFEM approaches becomes stronger.

4.4.2. Advection-Diffusion Equation

The second test problem is based on the advection-diffusion equation:

$$\frac{\partial u}{\partial t} + \boldsymbol{v} \cdot \nabla u = D\Delta u, \tag{4.12}$$

where u is an unknown field undergoing advection and diffusion, D is the diffusivity, and $\boldsymbol{v}$ is a given velocity field. The issue of primary interest here is the performance of the continuous and discontinuous GFEM when D is small (i.e., $D << 1$). The domain and boundary conditions used in this test problem are shown in Figure 4.1, and $g(x,t) = 4\sin(\pi t)(x-1)(2-x)$ for $sin(\pi t) > 0$ else $g(x,t) = 0$. The test problem is basically a pulsing plate (e.g., a plate with an oscillating temperature) and the pulses are advected downstream to the right. For this channel flow problem, the analytical solution to the Navier-Stokes equations is available and used to specify the velocity in (4.12). The lower velocities near the walls cause the pulses to advect more slowly when the diffusivity is reduced because the pulses stay closer to the walls. Table 4.2 shows a numerical solution for both the continuous and discontinuous GFEM simulations when $D = 0.01$, and the solutions are quite similar at this diffusivity value. However, if the diffusivity is further reduced to $D = 0.0005$, the numerical solution using a continuous GFEM discretization shows significant instability because the discretization does not including upwinding. The discontinuous GFEM solution is stable because upwinding is incorporated into the discretization through the numerical flux.

The instability in the continuous GFEM can be addressed through a number of different mechanisms including the use of a Petrov-Galerkin finite element method. Figure 4.3 shows the approximate solution using an SUPG finite element method with $D =$

0.0005 (Tezduyar 1992, Donea and Huerta 2003, Akin and Tezduyar 2004). The computational cost of the SUPG approximation was typically 5-10% higher than the continuous GFEM approximation, but it was still considerably lower than the discontinuous GFEM case. The SUPG finite element method introduces some numerical diffusion in the approximate solution, and for this test problem the SUPG solution was more diffusive than the discontinuous GFEM solution. It is important to emphasize that there are a number of different Petrov-Galerkin methods and many of them contain adjustable parameters. We are currently performing a detailed comparison of various Petrov-Galerkin approaches and discontinuous GFEM in terms of accuracy, stability, and computational cost for this class of problems, but that comparison is large and beyond the scope of this work.

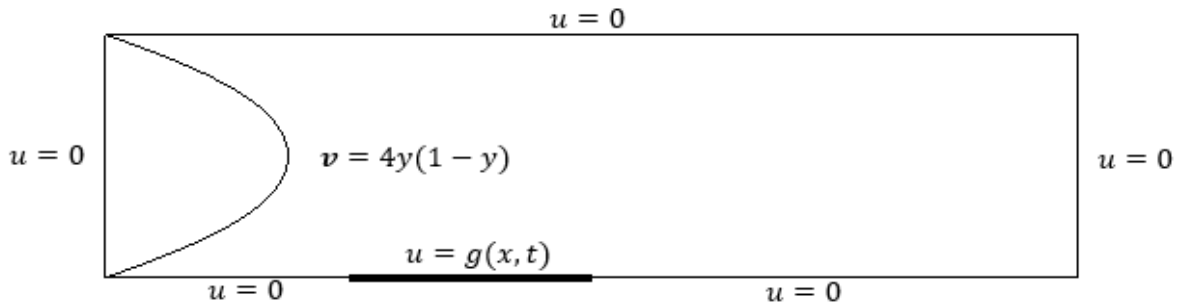


Figure 4.1.The domain and boundary conditions for both the advection-diffusion test problem and the viscous Burgers' equation test problem. The height of the domain is 1.0, and the width is 4.0.

Table 4.2. The total computational time (in seconds) required for solving (4.12) with 800 time steps using the continuous and discontinuous GFEM with two different meshes, two different values for D, and two different solvers (Amesos = direct solver, Aztec = ILU-preconditioned GMRES solver). For the small values of D, the numerical solution using the continuous GFEM can become unstable and contain large errors. Unstable solutions are indicated by a '*'.

D	Number Elements	Continuous GFEM		Discontinuous GFEM	
		Amesos	Aztec	Amesos	Aztec
0.01	586	14.7	23	48.5	62.9
0.00005	586	14.4*	23.3*	48.4	58.5
0.01	2025	57.2	84.8	240.1	237.3
0.00005	2025	57.2*	84.9*	240.9	220.9

The computational costs of the continuous and discontinuous GFEM for 800 times steps of the advection-diffusion test problem are summarized in Table 4.2. The first observation that can be made is that for this small test problem, a sparse direct solver is faster than an iterative solver. The trend in the rates of increase of solution time suggests that the iterative solvers scale better than the direct solver, which should be the case. The computational cost is also largely independent of D. Finally, the computational cost of the continuous GFEM is typical a factor of 5 lower than the discontinuous GFEM due to the small number of degrees of freedom in the continuous case, but the continuous case is also much more susceptible to large instability errors at smaller values of D.

(a) Continuous GFEM.

(b) Discontinuous GFEM.

(c) Continuous GFEM with SUPG stabilization

Figure 4.3. The advection-diffusion test problem result for $D = 0.0005$ at $t = 5.6\ s$. The continuous GFEM approximation is unstable and contains significant error due to the lack of upwinding in the discretization. The discontinuous GFEM is stable because the numerical flux enables upwinding. The continuous GFEM with SUPG stabilization is also stable but contains more numerical diffusion than the discontinuous GFEM case.

4.4.3. Viscous Burger's Equation

The third test problem is the 2-dimensional viscous Burgers' equation:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \mu \Delta \boldsymbol{u}, \tag{4.13}$$

which is similar to the advection-diffusion except the advection term is now nonlinear. The domain and boundary conditions for this problem are the same as those used for the advection diffusion equation, shown in Figure 4.1, but in this case there is no external velocity field that is advecting the unknown quantity. The nonlinearity was handled in both algorithms using Newton's method which requires the construction of a residual vector and a Jacobian matrix. For the comparison, two different viscosities μ, were tested in both continuous and discontinuous GFEM. The results for $\mu = 0.01$ using both the continuous and discontinuous GFEM are shown in Figure 4.4.

The first observation that can be made is that the pulses from the plate along the bottom of the domain are advected at an angle 45^oaway from the plate, and the velocity of the advection is (u, u). As a result, regions where $\boldsymbol{u}$ is larger move faster than regions where $\boldsymbol{u}$ is smaller. As the viscosity decreases, the faster moving regions tend to run into the slower moving regions and a large gradient that approaches a shock (but does not become a shock for $\mu > 0$) forms along the front. The moving region is addressed well by either approach as long as μ is sufficiently large. However, as shown in Figure 4.5, when μ is reduced to a value less than 0.001, large instabilities begin to develop when using the continuous GFEM for the meshes tested. In fact, these instabilities lead to divergence of the Newton solution method at approximately $t = 1.0\,\mathrm{s}$ in several of the problems

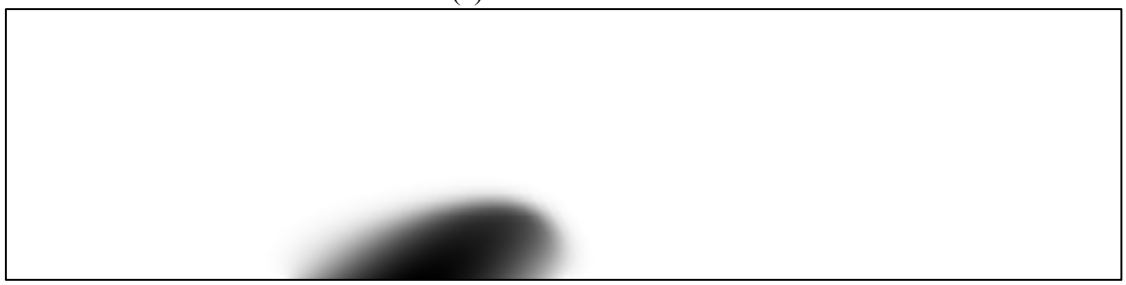
considered. Very small oscillations can also form using the discontinuous GFEM, but do not appear to affect the results significantly over the range of μ values considered in this study.

The computational time required to simulate 800 time steps for both the continuous and discontinuous GFEM using different values for μ and different finite element meshes are summarized in Table 4.3. Even though the same meshes are used as in the advection-diffusion test problem, multiple iterations are required by Newton's method each time step, and this increases the overall computational cost. For small values of μ, the continuous GFEM diverged and a solution was not obtained. These simulations are indicated by '–' in Table 4.3. When the continuous GFEM method did converge, it was significantly faster than the discontinuous GFEM, often by a factor of 20 or more.

Table 4.3. The total computational time (in seconds) for solving the viscous Burger's equation with 800 time steps and using the continuous and discontinuous GFEM with two different meshes, two different values for μ, instabilities led to Newton's method failing to converge using the continuous GFEM, and there cases are shown indicated with a '-'.

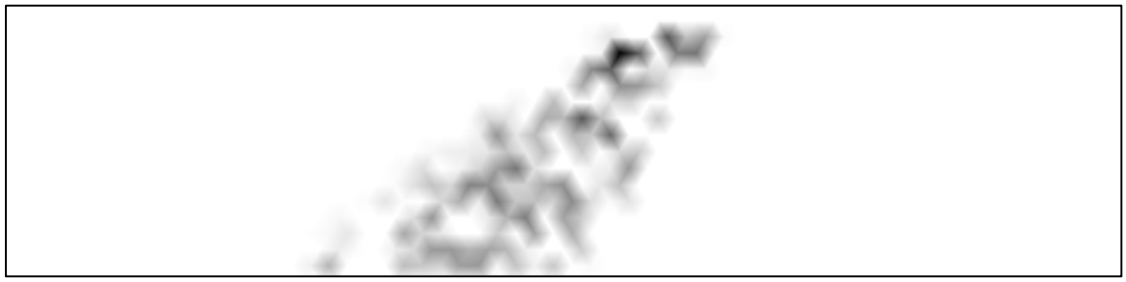
		Continuous GFEM		Discontinuous GFEM	
μ	Elements	Amesos	Aztec	Amesos	Aztec
0.01	586	23.7	37.6	171.9	222.6
0.00005	586	-	-	185.9	221.7
0.01	2025	104.5	156.2	844.1	858.3
0.00005	2025	-	-	953	891.8

(a) Continuous GFEM

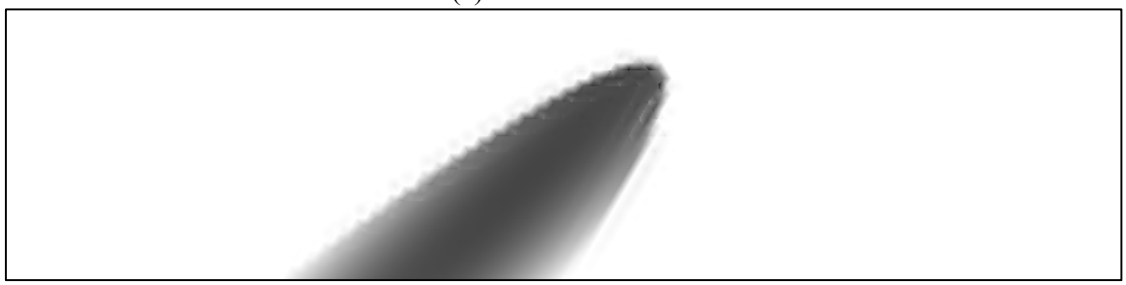


(b) Discontinuous GFEM

Figure 4.4. The viscous Burger's equation test problem result for $\mu = 0.01$ at $t = 0.06s$. The approximate solutions are similar for the continuous and discontinuous GFEM.



(a) Continuous GFEM



(b) Discontinuous GFEM

Figure 4.5. The viscous Burger's equation test problem result for $\mu = 0.000001$ at $t = 0.15s$. The continuous GFEM approximation is unstable and contains significant error that ultimately leads to convergence issues a later point in the simulation.

4.4.4. Turing Pattern Equation

The final test problem is the Turing pattern formation problem (also known as the reaction-diffusion model), which is used to model the formation of biological patterns (e.g., zebra stripes). Simple Turing models include two biological morphogens; one morphogen is responsible for short-range positive feedback and the other is responsible for long-range negative feedback. In these systems, a random initial condition will lead to a stationary pattern given the proper choice of parameters (Kondo and Miura 2010). This test problem was chosen because it has multiple equations and multiple unknowns. The following system of equations and model parameters were chosen because they are known to have a stationary pattern as a solution (Kondo and Miura 2010):

$$\frac{\partial u}{\partial t} = F(u,v) - 0.03u + 1.2 * 10^{-6}\Delta u, \tag{4.14}$$

$$\frac{\partial v}{\partial t} = G(u,v) - 0.08v + 3.1 * 10^{-5}\Delta v, \tag{4.15}$$

where

$$F(u,v) = 0.08u - 0.08v + 0.04,$$

$$G(u,v) = 0.1u - 0.15,$$

and the reaction rates are limited to the range of $0 \leq F(u,v) \leq 0.2$ and $0 \leq G(u,v) \leq 0.5$. The limits on the reaction rate make the problem nonlinear and are addressed using Newton's method. The model domain was the unit square with Neumann boundary conditions on all sides. The final pattern was always based on a random initial guess so the final stationary solution was never the same. For all finite element meshes tested here, both the continuous and discontinuous GFEM gave very accurate results and there was never a

noticeable difference in accuracy between the methods. A typical solution is shown in Figure 4.6.

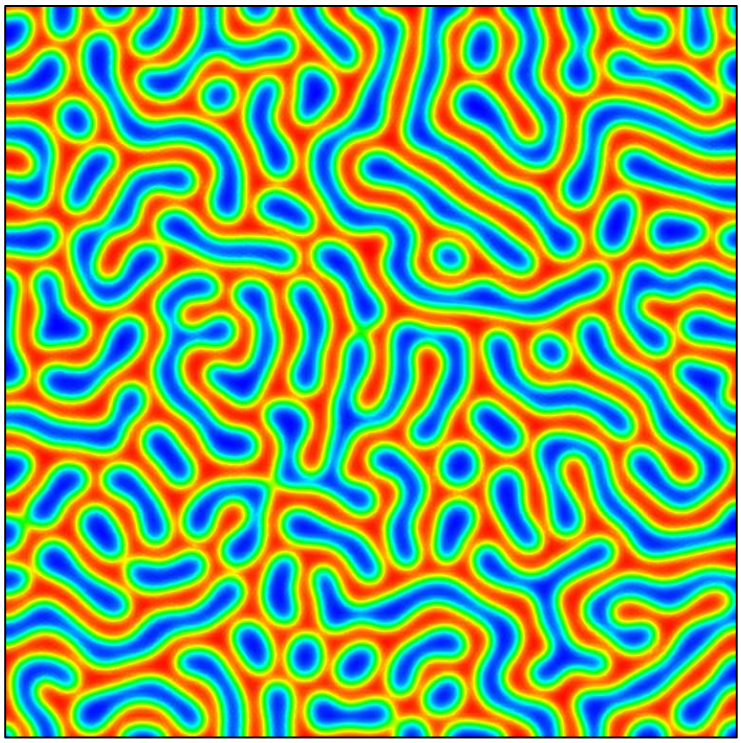


Figure 4.6. A typical solution for the concentration u in the Turing pattern formation problem, and using the given parameters. The solution is based on a random initial pattern and the final, stationary pattern does depend somewhat on the initial conditions.

The computational cost associated with solving (4.14) and (4.15) for 1400 time steps is summarized in Table 4.4. Again, considering only computational cost, the data leads to the conclusion that the continuous GFEM with its reduced number of degrees of freedom is significantly less expensive. For both the continuous and discontinuous GFEM, the problems are large enough that the ILU-precondition GMRES solver, Aztec, is significantly faster than the direct solver, Amesos, due to the better scalability of the

iterative solver. This problem can also be solved very efficiently using a multilevel solver such as ML, but the basic conclusion is still the same. The continuous GFEM method is significantly faster because it has fewer degrees of freedom. Without the presence of strong advection, it is difficult to justify the additional computational cost associated with the discontinuous GFEM.

Table 4.4. The total computational time in seconds for solving (4.14) and (4.15) for 1400 time steps using continuous and discontinuous GFEM and various mesh resolutions.

Number Elements	Continuous GFEM		Discontinuous GFEM	
	Amesos	Aztec	Amesos	Aztec
2664	985	988	13050	5568
4488	1998	1654	33538	10234
10656	6209	4043	140691	27539

4.5. Conclusions

Both continuous and discontinuous GFEM are useful tools for approximating the solution to parabolic differential equation. The best choice, of course, is highly problem dependent. For problems with relatively smooth solutions and a sufficiently large second-order term, the choice will usually be the continuous GFEM due to the significantly reduced number of degrees-of-freedom. For the test problems explored here, the continuous GFEM method was typically 5-20 times less computationally expensive relative to the discontinuous GFEM. For higher-order elements with more interior nodes, this difference may be less. Future work would include considering higher-order methods. For problems with large advection terms or, equivalently, small second-order terms in the equation, the instabilities that arise when using continuous GFEM can lead to these methods not being acceptable. In these cases, the discontinuous GFEM can be an excellent choice because it displays much better stability without the addition of stabilization. It is important to note that the instabilities that plague the continuous GFEM cannot be resolved with a moderate amount of mesh refinement for the examples shown here. Indeed, refining the mesh to the point that the continuous and discontinuous GFEM had the same computational cost did not lead to stability of the continuous method. In closing, for the advection dominated problems examined here, the discontinuous GFEM was the most effective option.

A number important question remains with regards to the choice between the various finite element methods. The biggest question is with regards to the comparison between stabilized continuous GFEM (i.e., Petrov-Galerkin methods) and discontinuous

GFEM for advection dominated parabolic differential equations. It is well known that continuous finite elements can be stabilized using Petrov-Galerkin methods, but these methods typically introduce artificial viscosity. Similarly, in discontinuous GFEM, there are many approaches to forming the inter-element flux terms that also can introduce artificial viscosity (Zhu, Qiu et al. 2011). The choice between Petrov-Galerkin methods and discontinuous GFEM is extremely complex due to the number of different Petrov-Galerkin methods and the fact that the performance and accuracy of Petrov-Galerkin methods can be highly problem and geometry dependent.

Ultimately, the choice between Petrov-Galerkin methods and discontinuous GFEM will depend upon the unique problem and the accuracy requirements of the particular application. A second important question is the choice of linear system solver. Established solvers have been well tested and, to some extent, optimized for classical continuous GFEM. As experience is gained with discontinuous GFEM, it is reasonable to expect that linear solver performance and range of choices will both improve. A final question with regards to the comparison of continuous and discontinuous methods is performance in a parallel computing setting. Discontinuous GFEM can be more straightforward to implement for parallel computing environments because only the elements need to be partitioned among the different processes used in the parallel computation. In closing, we believe it is important to answer all these questions using a carefully controlled set of quantitative comparisons. Our hope is that the comparison presented here can begin to answer some of these questions, but it is, of course, on the beginning.

4.6. References

Akin, J. E. and T. E. Tezduyar (2004). "Calculation of the advective limit of the SUPG stabilization parameter for linear and higher-order elements." Computer methods in applied mechanics and engineering **193**(21): 1909-1922.

Arnold, D. N. (1982). "An interior penalty finite element method with discontinuous elements." SIAM journal on numerical analysis **19**(4): 742-760.

Arnold, D. N., et al. (2002). "Unified analysis of discontinuous Galerkin methods for elliptic problems." SIAM journal on numerical analysis **39**(5): 1749-1779.

Braess, D. (2007). Finite elements: Theory, fast solvers, and applications in solid mechanics, Cambridge University Press.

Brenner, S. C. and R. Scott (2008). The mathematical theory of finite element methods, Springer Science & Business Media.

Cockburn, B., et al. (2011). "Analysis of HDG methods for Stokes flow." Mathematics of Computation **80**(274): 723-760.

Cockburn, B., et al. (2010). "A comparison of HDG methods for Stokes flow." Journal of scientific computing **45**(1-3): 215-237.

Cockburn, B. and C.-W. Shu (2001). "Runge–Kutta discontinuous Galerkin methods for convection-dominated problems." Journal of scientific computing **16**(3): 173-261.

Courant, R., et al. (1967). "On the partial difference equations of mathematical physics." IBM journal of Research and Development **11**(2): 215-234.

Deville, M. O., et al. (2002). High-order methods for incompressible fluid flow, Cambridge University Press.

Donea, J. and A. Huerta (2003). Finite element methods for flow problems, John Wiley & Sons.

Gabard, G., et al. (2011). "A comparison of wave-based discontinuous Galerkin, ultra-weak and least-square methods for wave problems." International journal for numerical methods in engineering **85**(3): 380-402.

Gresho, P. M. and R. L. Sani (1998). "Incompressible flow and the finite element method. Volume 1: Advection-diffusion and isothermal laminar flow."

Gunzburger, M. D. (2012). Finite Element Methods for Viscous Incompressible Flows: A guide to theory, practice, and algorithms, Elsevier.

Heroux, M., et al. (2003). An overview of Trilinos, Citeseer.

Heroux, M. A. and J. M. Willenbring (2003). Trilinos users guide, Citeseer.

Hesthaven, J. S. and T. Warburton (2007). Nodal discontinuous Galerkin methods: algorithms, analysis, and applications, Springer Science & Business Media.

Hussain, S., et al. (2011). "Higher order Galerkin time discretizations and fast multigrid solvers for the heat equation." Journal of Numerical Mathematics **19**(1): 41-61.

Kondo, S. and T. Miura (2010). "Reaction-diffusion model as a framework for understanding biological pattern formation." science **329**(5999): 1616-1620.

Kubatko, E. J., et al. (2009). "A performance comparison of continuous and discontinuous finite element shallow water models." Journal of scientific computing **40**(1-3): 315-339.

Olson, L. N. and J. B. Schroder (2011). "Smoothed aggregation multigrid solvers for high-order discontinuous Galerkin methods for elliptic problems." Journal of Computational Physics **230**(18): 6959-6976.

Reddy, J. N. and D. K. Gartling (2010). The finite element method in heat transfer and fluid dynamics, CRC press.

Rivière, B. (2008). Discontinuous Galerkin methods for solving elliptic and parabolic equations: theory and implementation, Society for Industrial and Applied Mathematics.

Sala, M., et al. (2007). Amesos: A set of general interfaces to sparse direct solver libraries. Applied Parallel Computing. State of the Art in Scientific Computing, Springer**:** 976-985.

Schoof, L. A. and V. R. Yarberry (1994). EXODUS II: a finite element data model, Sandia National Labs., Albuquerque, NM (United States).

Tezduyar, T. E. (1992). Stabilized finite element formulations for incompressible flow computations, Academic Press.

Werder, T., et al. (2001). "hp-discontinuous Galerkin time stepping for parabolic problems." Computer methods in applied mechanics and engineering **190**(49): 6685-6708.

Wirasaet, D., et al. (2010). "A performance comparison of nodal discontinuous Galerkin methods on triangles and quadrilaterals." International Journal for Numerical Methods in Fluids **64**(10-12): 1336-1362.

Zhu, J., et al. (2011). "RKDG methods with WENO type limiters and conservative interfacial procedure for one-dimensional compressible multi-medium flow simulations." Applied Numerical Mathematics **61**(4): 554-580.

CHAPTER 5

COMBINING EXISTING NUMERICAL MODELS WITH DATA ASSIMILATION USING WEIGHTED LEAST-SQUARES FINITE ELEMENT METHODS

Contribution of Authors and Co-Authors

Manuscript in Chapter 5

Author: Prathish K. Rajaraman

Contributions: Programing computational model, data collection and analysis manuscript drafting and editing.

Co-Author: T. A. Manteuffel

Contributions: Conception and critical analysis of least squares finite element method.

Co-Author: M. Belohlavek

Contributions: Echo-PIV experiment and data collection.

Co-Author: Jeffrey J. Heys

Contributions: Conception and critical analysis of article, manuscript drafting and editing.

Manuscript Information Page

Prathish K. Rajaraman [a], T.A. Manteuffel [b], M. Belohlavek [c], and Jeffrey J. Heys [a]

Journal Name: International Journal of Numerical Methods in Biomedical Engineering

Status of Manuscript:

____ Prepared for submission to a peer-reviewed journal

_X_ Officially submitted to a peer-review journal

____ Accepted by a peer-reviewed journal

___ Published in a peer-reviewed journal

John Wiley & Sons

Submitted in October 2015

[a] Chemical and Biological Engineering Department, Montana State University, Bozeman, MT 59717

[b] Department of Applied Mathematics, University of Colorado, Boulder, CO 80309

[c] Mayo Clinic Arizona, Scottsdale, AZ 85259

5.1. Summary

We discuss in this article a new approach for combining and enhancing the results from an existing computational fluid dynamics model with experimental data using the weighted least squares finite element method (WLSFEM). The development of the approach was motivated by the existence of both limited experimental blood flow data in the left ventricle (LV) and inexact numerical models of the same flow. The limitations of the experimental data are measurement noise and, in the case of high temporal resolution ultrasound imaging with microbubbles, having data only along a two-dimensional plane. Most numerical modeling approaches do not provide the flexibility to assimilate noisy experimental data in way that can positively influence, but not contaminate, the numerical solution. We previously developed an approach that could assimilate experimental data into the process of numerically solving the Navier-Stokes equations, but the approach was limited because it required the use of specific finite element methods for solving all model equations and did not support alternative approximation methods. The new approach presented here allows virtually any numerical method to be used for approximately solving the Navier-Stokes equations, and then the WLSFEM is used to combine the experimental data with the numerical solution of the model equations in a final step. The approach dynamically adjusts the influence of the experimental data on the numerical solution so that more accurate data is more closely matched by the final solution and less accurate data is not closely matched. The new approach is demonstrated on different test problems and provides significantly reduced computational costs relative to other most other methods for data assimilation.

5.2. Introduction

In recent years, numerical models of blood flow in the left ventricle (LV) have been used as a tool by a number of researchers and scientists [1-5]. Many imaging methods have also been developed in order to better understand the blood flow in the left ventricle with the goal of assessing the overall health of the heart [6-8]. One example of an imaging method for blood flow in the left ventricle is using magnetic resonances imaging (MRI), but this approach is expensive, confines patients to a limited space and restrict patients to an immobile position in order to obtain blood flow velocity information within the heart [5, 9, 10]. An alternate approach for imaging blood flow in the heart is using ultrasound in combination with a microbubble contrast agent added to the blood. This approach is less expensive and less restrictive; unfortunately the images have a lower spatial resolution, are currently limited to a single spatial plane if a high frame rate is desired (or cross-section), but do have the potential for high temporal resolution [9-12].

One advantage of using *in vivo* measurements is to obtain patient specific information regarding blood flow in LV [3]. In contrast, *in silico* methods have the advantage of predicting physiological flow properties that cannot be measured via existing imaging methods and that potentially provide insights into the health of the heart [1, 6, 9]. In the past, researchers have used the data provided by *in vivo* methods as a benchmark and validation tool for *in silico* methods [3]. In order to complement the *in silico* approach, researchers have recently combined both *in vivo* and *in silico* approaches to better understand the physical process and create a patient specific numerical model, which can potentially be used for diagnostic and prognostic information [3]. This combination of

patient specific information from imaging techniques with a numerical model is one form of data assimilation, and this process requires the combining of noisy experimental data into a numerical model that is based on physical and constitutive laws (e.g., Navier-Stokes).

Data assimilation techniques have the potential to improve the quality of an existing numerical model through the incorporation of actual, physically accurate, patient specific data. For a numerical model of blood flow in the heart, the Navier-Stokes equations are typically the foundation for the numerical model. There are two approaches used extensively to assimilate data into a numerical model: variation methods and stochastic methods. Stochastic methods uses Kalman filters that require an ensemble of approximate solutions [3, 9, 12-14]. This is a popular approach in metrology, but also increases computational cost dramatically [11, 14]. Other approaches assimilate experimental data, which often lies at arbitrary spatial locations, by interpolating the data to the computational nodes and then requiring the numerical solution to exactly match the experimental data. Any error in the data pollutes the numerical solution, and, in some cases, the interpolation process can add even more error to the experimental data [15]. A new approach is presented in this paper that improves upon the previously developed weighted least-squares finite element method (WLSFEM). The new approach has several advantages compared to the previous WLSFEM approach, including: a significantly reduction (2-3 times) in computational cost, the ability to continue using existing numerical model algorithms, and potentially improved accuracy, especially with respect to the mass loss that has been previously observed [9, 12, 14-16].

The new approach is initially tested on Poiseuille flow through a straight cylinder, but the majority of the testing is on the assimilation of experimental data from echocardiographic particle imaging velocimetry (echo-PIV) measurements. The echo-PIV method works by injecting FDA approved microbubbles into the blood stream and using cardiac ultrasonography with a temporal resolution of $150\,\frac{frames}{s}$ to determine the location of the bubbles as a function of time [6, 10-12]. Further processing with PIV software on the images from ultrasonography calculates the velocity of the microbubbles based on the displacement and image frame rate information [6, 10-12]. PIVlab, an open source software package, is used here to calculate the 2-dimensional velocity field in the left ventricle from ultrasound images [17]. The technology of cardiac ultrasonography at sufficiently high frame rates for PIV analysis is limited to 2-dimenional scans. In Figure 5.1, the resulting 2-dimensional velocity generated by PIVlab is shown. Flow properties, such as viscous energy loss and pressure gradients, cannot be calculated from this limited (2-dimensional) velocity information. In order to determine flow properties of interest, a 3-dimensional velocity field is required, and the approach described here has the ability to assimilate noisy, 2-dimensional echo-PIV data into the numerical solution of a 3-dimensional simulation to produce a 3-dimensional velocity field that is influenced by the experimental data and can be used to compute flow properties that cannot be determined from the experimental data alone.

The new approach also provides flexibility in terms of integrating with almost any existing numerical algorithm for solving the Navier-Stokes equations, and the approach can potentially integrate with a range of numerical models that are based on a partial

differential equation(s). Most existing numerical methods for the Navier-Stokes equations or other partial differential equation, e.g., the Galerkin finite element, finite volume or finite difference methods, do not allow simple, inexpensive assimilation of experimental data [14]. Here we demonstrate a flexible, new approach that can combine experimental data with an existing numerical solution to a PDE-based model into a final solution using the weighted least-squares finite element framework.

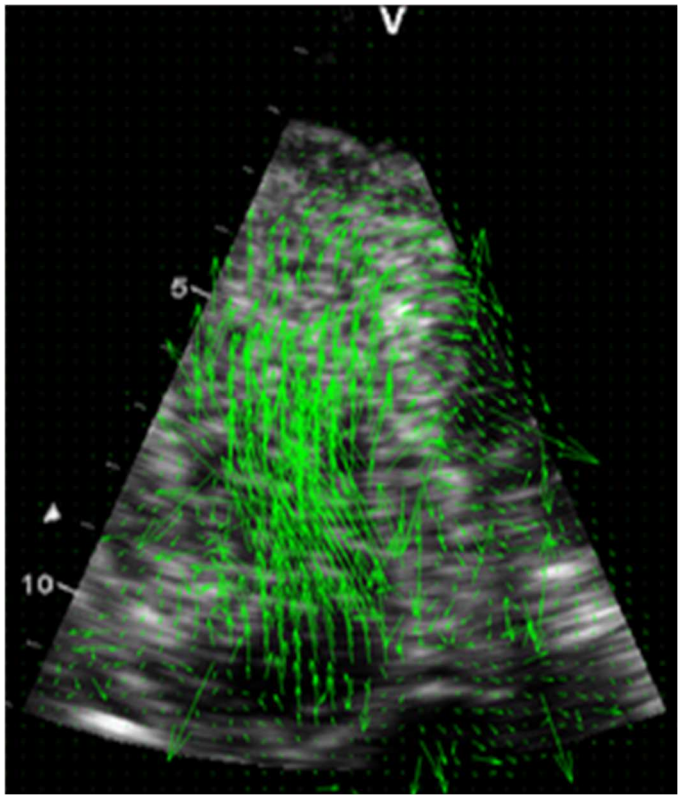


Figure 5.1. Ultrasound image containing microbubbles and the corresponding velocity data generated using PIVlab.

5.3. Methods

The data assimilation approach described here begins with a numerical model of the physical problem that is based on the Navier-Stokes equations. The numerical model is outlined first in the methods section. Once an approximate solution to the numerical model has been obtained, the WLSFEM approach is used to combine experimental data with the numerical solution to obtain a new approximate solution that better reflects the physical problem of interest, provided the experimental data is sufficiently accurate.

5.3.1. Navier-Stokes

In order to model blood flow in the left ventricle, blood is assumed to be an incompressible, Newtonian fluid that can be described by the Navier-Stokes equations and can be written in dimensionless form as [18-20]:

$$\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \boldsymbol{f}\right) = -\nabla P + \frac{1}{Re}\nabla^2 \boldsymbol{u} \quad in\ \Omega,$$
$$\nabla \cdot \boldsymbol{u} = 0 \quad in\ \Omega, \tag{5.1}$$

where $\boldsymbol{u}$ is the dimensionless velocity, P is the dimensionless pressure and $\boldsymbol{f}$ is the source terms. Re is the Reynolds number, defined as $Re = \left(\frac{LU\rho}{\mu}\right)$, where L is the characteristic length, U is the characteristic velocity, ρ is the density of the fluid, and μ is the viscosity. Blood flow in the left ventricle is simulated here using $Re = 560$ based on previous measurements of density and viscosity for blood, and using ultrasound images and echo-PIV data to estimate the characteristic length and velocity. The first equation in (5.1) is the conservation of momentum for a Newtonian fluid and the second equation describes conservation of mass for an incompressible fluid [18-23]. In order to solve equation (5.1)

for a fluid contained in Ω, the Galerkin finite element method was used to rewrite the partial differential equation into a variational form. The equations in (5.1) are multiplied by test functions $(\boldsymbol{w}, q)$ and integrated (by parts for second-order terms) over the domain to give:

$$\int_{\Omega} \boldsymbol{w}\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} - \boldsymbol{f}\right) d\Omega + \frac{1}{RE}\int_{\Omega} \nabla\boldsymbol{w}\cdot\nabla\boldsymbol{u}\; d\Omega - \int_{\Omega} \nabla\boldsymbol{w}\cdot p\boldsymbol{I}\; d\Omega = 0$$
$$\int_{\Omega} (\boldsymbol{\nabla}\cdot\boldsymbol{u}) q\; d\Omega = 0\,. \tag{5.2}$$

Many previous studies have shown that the Galerkin finite element method can suffer from spurious oscillations for convection dominated problems (i.e.$Re > 100$) unless the mesh is highly refined [19, 20, 23]. A standard approach for avoiding oscillations while retaining a computationally feasible mesh is it to introduce stabilization terms to the Navier-Stokes weak form. There are many choices for the stabilization method, and the stabilization method used in this paper is the continuous interior penalty finite element method (IPFEM) (for further information the reader can refer to [24, 25]). Each of the test problems described below has a unique domain, initial conditions, and boundary conditions, which are also given below. Based on the domain, initial conditions and boundary conditions, the IPFEM approach is used to obtain a velocity field for every time step. The approximate velocity solution from IPFEM (or any other numerical discretization approach) is then used to compute the vorticity field for a given mesh by taking the negative curl of the velocity field. This approximate vorticity, which is often discontinuous, is used as an input to the new data assimilation approach that is outlined in the section below.

5.3.2. Data Assimilation

A numerical method that approximately solves the Navier-Stokes equations provides the vorticity without any impact from the echo-PIV data, and the next step is to assimilate the experimental data to obtain a new numerical solution that is influenced by the patient specific data. The data is assimilated into a new model solution by solving an overdetermined system of first-order differential equations using a weighted least-square finite element method approach (WLSFEM) [9, 12, 15, 16, 26, 27]. This approach was inspired by previous WLSFEM approaches, which have been used successfully for data assimilation [9]. Unlike the previous approaches, the current approach only requires an approximate numerical solution from almost any discretization method for solving the Navier-Stokes equations. Given a numerical approximation for the vorticity, $\boldsymbol{\omega}$, obtained by taking the curl of the velocity solution, a first-order system of equations are solved to combine the experimental data and the numerical solution to determine the new, experimentally influenced velocity field. The WLSFEM equations are:

$$\nabla \times \boldsymbol{v} + \boldsymbol{\omega} = 0 \ \ in\ \Omega, \tag{5.3}$$

$$\nabla \cdot \boldsymbol{v} = 0 \ \ \ in\ \Omega.$$

where $\boldsymbol{v}$ is the new, unknown velocity that is influenced by the experimental data. Typically, both the IPFEM step for Navier-Stokes and the WLSFEM assimilation step are solved on the same mesh and use the same boundary conditions for $\boldsymbol{u}$ and $\boldsymbol{v}$. It is also possible to solve Navier-Stokes for the vorticity on a different mesh from the mesh used for the WLSFEM assimilation step, but having different meshes requires a projection of the vorticity to the WLSFEM mesh. It is also important to note that the WLSFEM does not

place any requirements on the finite element space or smoothness for the given vorticity field. Previous studies have shown that when solving the Navier-Stokes equations with many common least-squares finite element methods, the standard LSFEM approach (or even a WLSFEM approach) can lead to an approximate solution with poor mass conservation depending on the method used to rewrite the momentum balance as a system of first-order equations and depending on the finite element approximation space that is used [21, 26, 28, 29]. A number of approaches have been developed to alleviate poor mass conservation, including the use of higher-order polynomial basis functions, but these can lead to higher computational costs for some problems [16, 18, 22, 26]. The approach demonstrated here largely avoids the mass conservation challenge because the Navier-Stokes equations are solved using alternative discretization methods to WLSFEM. The new approach is also computationally cheaper compared to previous WLSFEM approaches used for data assimilation problems [9, 12, 15].

The next step in the WLSFEM framework is to cast system (5.3) into an unconstrained optimization problem and this is achieved by defining a functional for system (5.3) by taking the L^2-norm of the equations on the 3D domain (Ω) and summing the equations:

$$F(\boldsymbol{v}; 0) = \|\nabla \times \boldsymbol{v} + \boldsymbol{\omega}\|_{0,\Omega}^2 + \|\nabla \cdot \boldsymbol{v}\|_{0,\Omega}^2 + \frac{1}{h}\|\boldsymbol{v} - \boldsymbol{g}\|_{0,\Gamma}^2 \tag{5.4}$$

$$+ \frac{w_{PIV}}{h}\|\boldsymbol{v} - v_{PIV}\|_{0,\Gamma_{PIV}}^2.$$

In the WLSFEM framework there are two options when imposing boundary conditions: strong or weak. Imposing weak boundary conditions is done by adding the boundary condition terms into the functional, while strong boundary conditions are

enforced by restricting the finite element space [18, 22]. In functional (5.4), w_{PIV} is the weight used to assimilate the echo-PIV data. The goal is to set the weight for the echo-PIV data so that accurate data is more closely match by the final numerical solution (i.e., a larger weight), and less accurate data is not matched as closely by the final solution (i.e., a smaller weight). In [9, 15], the reader can find more details on how the weight on the echo-PIV data, w_{PIV}, is optimally calculated, and in [9], a method is described for automatically estimating the weight for a given set of echo-PIV data based on estimates of the temporal accuracy of the data. The approach allows the 2-dimensional echo-PIV velocity, $\boldsymbol{v_{piv}}$, at any spatial location to be weakly matched by the final solution, $\boldsymbol{v}$ [9, 15]. The functional term for the weak boundary conditions, given by $\boldsymbol{g}$, is scaled by $\frac{1}{h}$ in order to approximate the $H^{\frac{1}{2}}$–norm with a weighted L_2-norm on the boundary. Functional (5.4) can also be used as a sharp measure of the error in the approximate solution to the first-order system of equations [30]. A WLSFEM data assimilation approach was also studied in [2, 3, 14, 31], where the data assimilation approach is interpreted as finding a *maximum a posteriori* (MAP) estimator in a Bayesian approach.

The new method for data assimilation is summarized in Figure 5.2. The new approach is divided into two stages, the first stage is solving the model problem without any direct assimilation of experimental data and using any choice of numerical approximation method for the model equations. Stage 2 is where the data assimilation step is performed, and this step requires the vorticity field calculated from the numerical solution provided in stage 1.

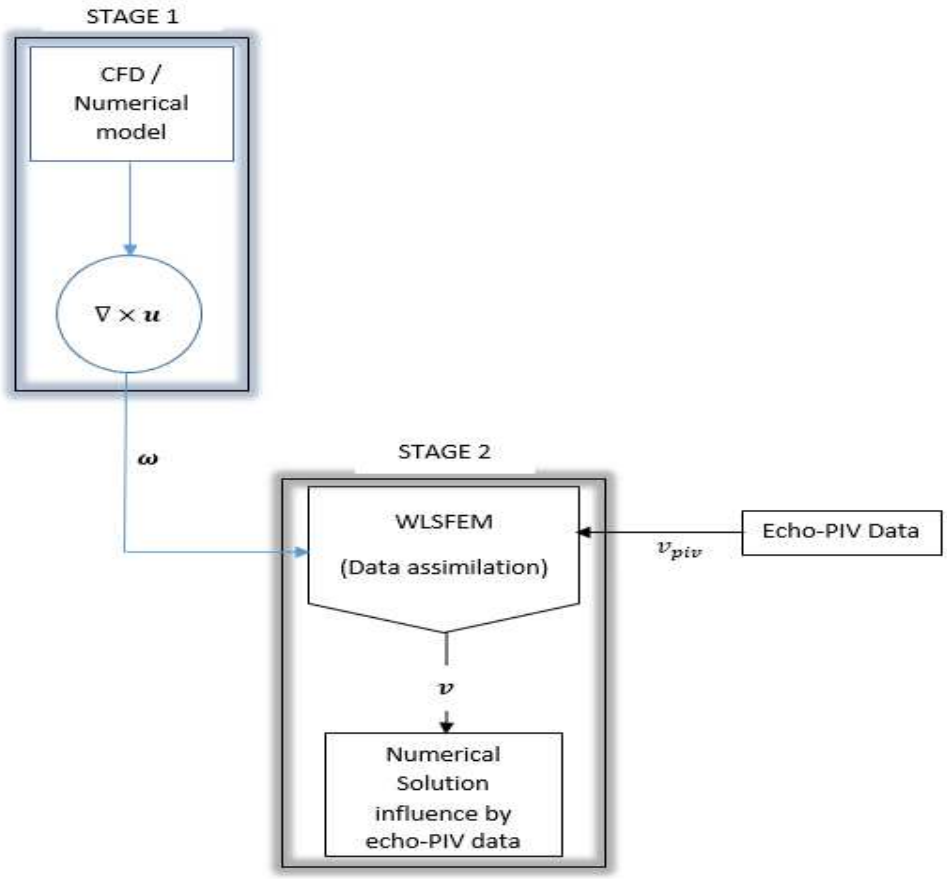


Figure 5.2. Workflow for new data assimilation approach showing stage 1: solution of the numerical model, and stage 2: assimilation of experiment data to obtain a new numerical solution that is influence and improved by the assimilated data.

5.3.3. Test Problem I

In order to test the new data assimilation approach, a test case was studied where the exact solution, $\boldsymbol{v}$, and the exact vorticity, $\boldsymbol{\omega}$, are known. For the flow in a straight cylinder, i.e., Poiseuille flow, artificial data was generated by adding Gaussian white noise to the exact velocity and this artificial data was assimilated to obtain a final, numerical solution. The domain has a dimensionless radius of 0.5 and a dimensionless length of 5.0, and a tetrahedral finite element mesh was generated for the cylindrical geometry. Table 5.1

lists the boundary conditions used in the simulation and the type of enforcement. The vorticity, $\boldsymbol{\omega}$, input for system (5.3) was calculated based on the exact velocity solution (i.e., stage 1 in Figure 5.2 was replaced with the known, exact vorticity solution). In Table 5.1, the variable $\boldsymbol{f_{data}}$ is the artificially generated 2-dimensional velocity data at arbitrary spatial locations that is assimilated to obtain a final numerical solution. Since $\boldsymbol{\omega}$ is known everywhere in the domain, i.e., for this example the exact vorticity solution is used, the impact of the error in the artificial data on the final velocity, $\boldsymbol{v}$ can be quantified. It should be noted that if, in practice, the numerical model is perfect (exact), one would never assimilate experimental data, but this test problem allows for the isolation of error from the experimental data on the final solution. The artificial data used for this test case was generated on a plane $(y = 0)$, using the exact velocity solution and adding random Gaussian noise. Since the standard deviation of the artificial noise is known for this test problem, the weight on the data assimilation term in the functional (5.4) was set to $w_{PIV} = 2.0$. The data was assimilated weakly for the two tangential components of the velocity vector along the $(y = 0)$ plane.

Table 5.1. Boundary conditions used for flow in cylinder problem.

Boundary	Condition	Enforcement
Inlet	$\boldsymbol{n} \times \boldsymbol{v} = 0$	Strong
	$\boldsymbol{n} \cdot \boldsymbol{v} = -1.0 + \left(\frac{x^2 + y^2}{0.25}\right)$	Strong
Outlet	$\boldsymbol{n} \times \boldsymbol{v} = 0$	Strong
	$\boldsymbol{n} \cdot \boldsymbol{v} = -1.0 + \left(\frac{x^2 + y^2}{0.25}\right)$	Strong
Wall	$\boldsymbol{v} = 0$	Strong
Data-Plane	$\boldsymbol{n} \times \boldsymbol{v} = \boldsymbol{f_{data}}$	Weak

5.3.4 Test Problem II

In Test Problem I, the goal was to evaluate the data assimilation scheme when an exact solution from the numerical model is available (i.e., the exact vorticity is known). The goal of the second test problem is to test the new data assimilation scheme when the vorticity field that is obtained from the numerical algorithm used to solve the Navier-Stokes equations also contains error. In this test problem, the same geometry and finite element mesh was used as in the Test Problem I, but in this test, stage 1 used the IPFEM algorithm to approximately solve the Navier-Stokes equations and obtain an approximate vorticity for input into stage 2.

The approximate velocity obtained using IPFEM was still highly accurate for this simple problem relative to the more complex problems that are normally solved using computational fluid dynamics, so quantifiable random noise was added, in some cases, to the approximate velocity solution from the IPFEM method (i.e., stage 1 in Figure 5.2) before moving to the data assimilation step (stage 2). The amount of noise added into the solution was proportional to a variable τ. As a result, when $\tau = 0.0$ there is no noise added to the numerical solution and for $\tau > 0.0$, varying amounts of error are added to the stage 1 solution to the Navier-Stokes equations.

5.3.5. Left Ventricle Blood Flow

The new approach was also applied to the left ventricle data assimilation problem, and Table 5.2 lists the boundary conditions and the type of enforcement (i.e., strong or weak). In the blood flow simulation, the boundary conditions change with time depending on the stage of the cardiac cycle -- filling (late diastole) or ejection (systole) of the blood

in the left ventricle. In Table 5.2, $g_1(t)$ and $\boldsymbol{v_{piv}(t)}$ are the specified velocity of the left ventricle wall and the echo-PIV data on the PIV-plane, and both were obtained from ultrasound measurements. The normal velocity at the inlet during late diastole and outlet during systole are not specified. The boundary conditions in Table 5.2 were implemented in order to obtain a numerical solution with maximum influence of the echo-PIV data near the inlet and outlet. At the inlet and outlet the tangential velocity was set to zero and was enforced weakly. Note, the boundary conditions listed (Inlet, Outlet and Wall) in Table 5.2 were also used for the stage 1 IPFEM simulation.

Table 5.2. Boundary conditions used for data assimilation left ventricle simulation.

Boundary	Condition	Enforcement	Time Step
Inlet	$\boldsymbol{n} \times \boldsymbol{v} = 0$	Weak	≤ 59 (diastole)
	$\boldsymbol{v} = 0$	Strong	≥ 60
Outlet	$\boldsymbol{v} = 0$	Strong	≤ 74
	$\boldsymbol{n} \times \boldsymbol{v} = 0$	Weak	≥ 75 (systole)
	$\boldsymbol{v} = 0$	Strong	$= 134$
Wall	$\boldsymbol{v} = g_1(t)$	Strong	*all Time Steps*
PIV-Plane	$\boldsymbol{n} \times \boldsymbol{v} = \boldsymbol{v_{piv}(t)}$	Weak	*all Time Steps*

The echo-PIV weight, w_{piv}, must be set so that the final solution from stage 2, $\boldsymbol{v}$, is able to match the more accurate velocity data more closely and less accurately velocity data should be matched loosely [9]. When solving for the flow in a cylinder, the standard deviation of the error in the assimilated data (σ) is known, but for the left ventricle case, the error in the assimilated experimental data is not available and the value varies with time

and concentration of microbubbles [9, 15]. In [9], an alternative approach for calculating w_{piv} was developed that estimated the accuracy of the experimental data based on the temporal consistency of the data. Large changes to the velocity data over short time intervals result in w_{piv} being set to a smaller value due to the error in the data [9].

5.3.6. Implementation

The Navier-Stokes equations for stage 1 were solved using the open source package FENICS [32]. Solving the Navier-Stokes equations in a Galerkin finite element setting requires the selection of finite element spaces that satisfy the inf-sup condition. A quadratic basis was used for velocity and a linear basis was used for pressure, i.e., P_2P_1 tetrahedral elements [19, 20, 23]. A 2nd-order backward difference formula (BDF-2), which is unconditionally stable, was used for the temporal discretization with a dimensionless time step size of 0.019, the same time step that was available for the ultrasound frame rate that generated the echo-PIV data [19, 20]. In order to capture the blood flow in the left ventricle accurately, the walls of the domain are moved using displacements from the ultrasound images, and the mesh is moved using the FENICS mesh move functionality [32].

ParaFOS is an in-house C code and was developed to solve the second stage using the WLSFEM. The code can import the vorticity $\boldsymbol{\omega}$ from the stage 1 (IPFEM solution), import the echo-PIV data, and minimize the stage 2 WLSFEM functional. Tetrahedral meshes were used in both stage 1 and stage 2 and are generated using Cubit (13.2), developed by Sandia National Laboratory. An example of mesh generated using Cubit with 52000 tetrahedral shown in Figure 5.3. Stage 1 was solved using incomplete LU factorization (ILU), while the linear matrix problem in the stage 2 was solved using an

algebraic multigrid (AMG) preconditioner for a conjugate gradient (CG) iteration, using two V-cycles for every CG iteration [33]. The convergence criteria for the CG iteration was fixed to a residual norm reduction of a factor of 10^{-6}, which is a fraction of the discretization error [34]. Simulations for both flow in a cylinder and left ventricle blood flow were performed with meshes varying in total number of elements. The left ventricle blood flow simulation used approximately 52,000 total elements, requiring about 170 h of computational time on a single processor. The simulations were performed on a Dell Workstation T5610 with dual 8-core Xeon E5-2650 and with 128 GB RAM.

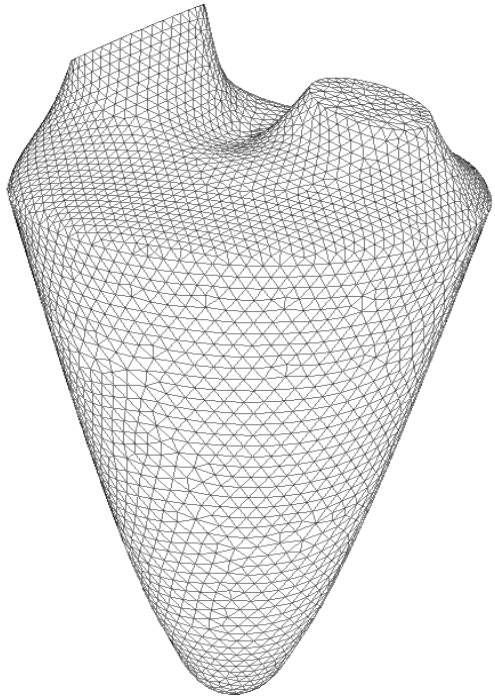


Figure 5.3. A tetrahedral mesh with approximately 52000 elements of the left ventricle generated using Cubit.

5.4. Results

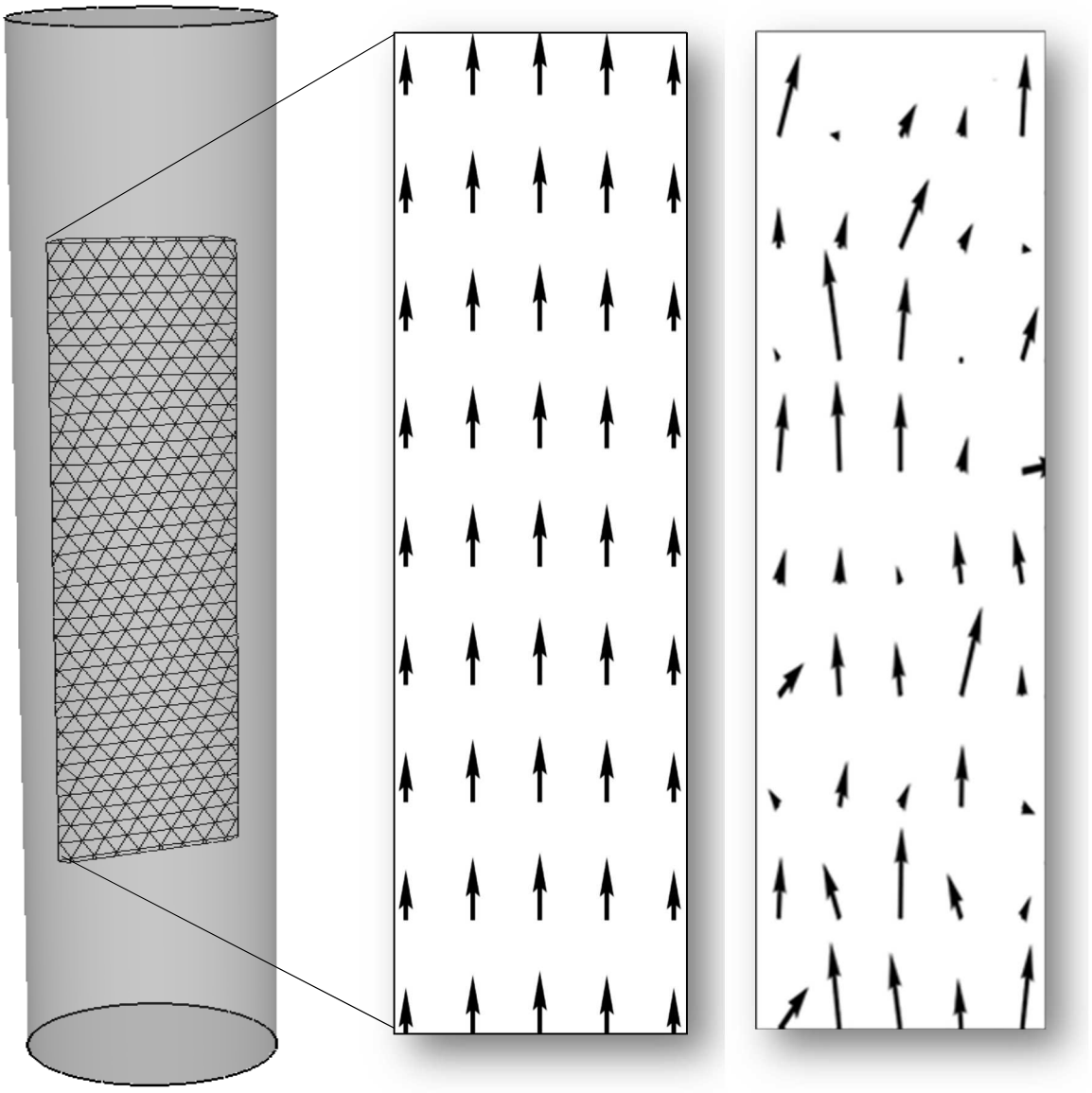
The main goal of this paper is to examine the performance of the new data assimilation approach. The first and second test problem uses artificially generated data and not actual experimental data. The final test problem uses echo-PIV data obtained from the left ventricle of a pig [35].

5.4.1. Results and Discussion (Test Problem I)

Figure 5.4(a), shows the cylinder geometry with the plane where assimilated data is available highlighted in the middle of the cylinder. The exact flow solution for the model problem is fully developed and laminar, and the exact solution is $\boldsymbol{v} = \left(0,0,-1+(\frac{x^2+y^2}{R^2})\right)$, where $R = 0.5$ is the radius of the tube is. Functional (5.4) can be used as a sharp measure of the error, and larger values for the functional indicate the approximate solution does not satisfy the first-order system of equations (5.3). Figure 5.4(b) shows the assimilated data without any noise added, and Figure 5.4(c) shows one example of the artificial data that was assimilated into the simulation (Test Problem I). The artificial data is generated using the exact solution and adding Gaussian noise with $(\sigma = 0.70711)$. The data being assimilated into the numerical model is in the $y = 0$ plane and the assimilated data is located only in the mid-region of the plane, away from boundaries, as shown in Figure 5.4(a).

The results of the cylindrical flow test problem (Test Problem I), with and without data assimilation, are shown in Figure 5.5 using a mesh with approximately 90,000 tetrahedral elements. Figure 5.5 shows the simulation without any data assimilated into the

numerical model (gray arrows). The simulation with the data from Figure 5.4(c) assimilated (black arrows) are overlaid on top at every mesh node so that the influence of the assimilated data can be observed. The black arrows in the boxed region are slightly different from the gray arrows, showing that the assimilated data has a small impact on the numerical solution. It would be considered a failure if the data assimilation method showed no difference between the two solutions (with and without data assimilation) and it would be a failure if the data assimilation method showed a large difference between the two solutions because that would indicate that the artificial error in the assimilated data was being allowed to contaminate the final numerical solution. At the boundaries of the domain, the simulations with and without noisy assimilated data show more consistent results because the assimilated data does not extend all the way to the wall. Global mass conservation can be computed by calculating the $L_2 - norm$ of the continuity equation $\|\nabla \cdot \boldsymbol{v}\|_{0,\Omega}^2$, for the simulations with and without assimilated data. It is interesting to note that improved mass conservation (i.e., less mass loss) is observed when data is assimilated.



(a) (b) (c)

Figure 5.4. Cylindrical geometry with an aspect ratio of 1:1:5 with the velocity data plane from 25%-75% of the cylinder. The assimilated velocity data is shown on the right of the figure.

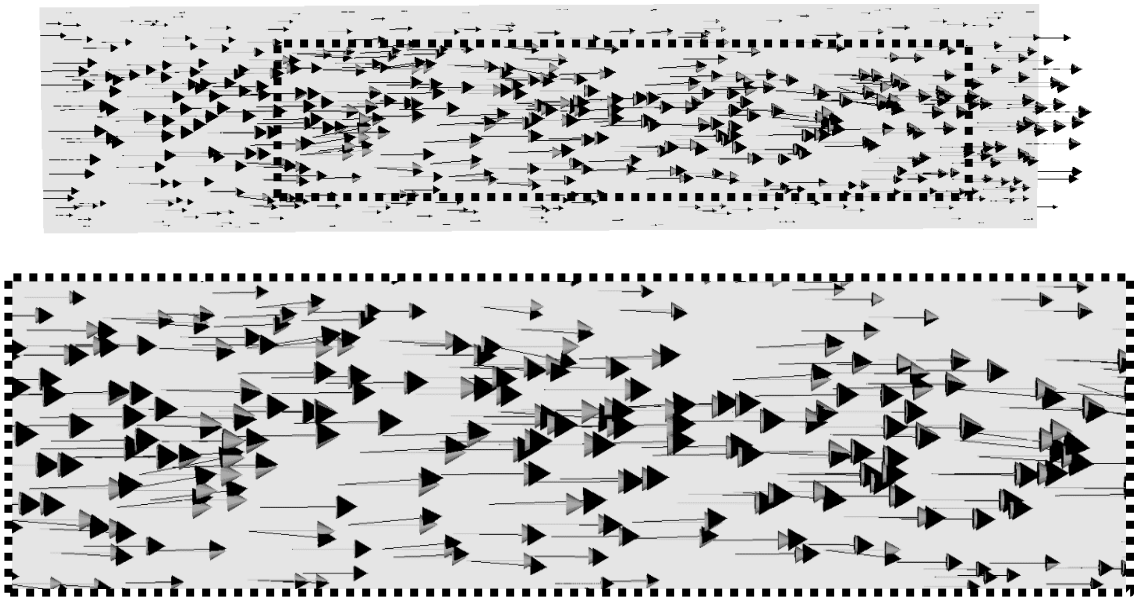
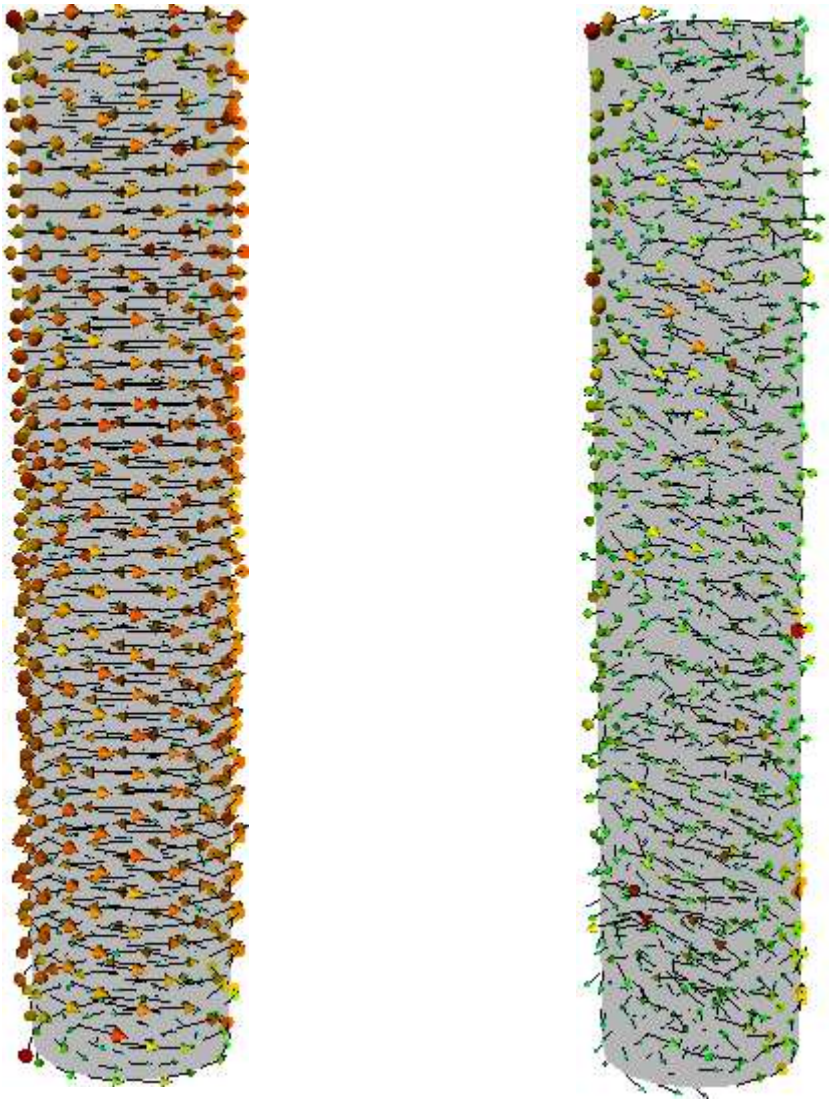


Figure 5.5. Comparison of velocity with (black) and without (gray) data for mesh with approximately 90000 elements.

5.4.2. Results and Discussion (Test Problem II)

The vorticity that is input into the WLSFEM for the second test case is shown in Figure 5.6 both without ((a), $\tau = 0.0$) and with ((b), $\tau = 0.025$) artificial noise being added to the stage 1 numerical solution. Recall that the goal of this test problem was to examine the impact of numerical model error from stage 1 on the final result. The initial result was obtained using the IPFEM and variable amounts of random noise (i.e., known approximation error) were added to the velocity field. In order to obtain the vorticity field, the negative curl of the noisy velocity is computed. The noisy vorticity is then used as an input to stage 2 to perform the data assimilation step. The $L_2 - norm$ calculated is the difference between the final numerical solution,$\boldsymbol{v}$, and the exact solution for the flow in the cylinder can be calculated after stage 2.

Figure 5.7 shows the error in the final numerical solution, $\boldsymbol{v}$, after the second stage for different mesh resolutions and varying accuracy for the vorticity field from stage 1. The x-axis reflects the accuracy of the numerical solution from stage one and the y-axis reflects the overall accuracy of the final numerical solution after the second stage. The black points show the solution with the influence of the assimilated data (Figure 5.4(b)) and the gray points are not influenced by assimilated data. Comparing the gray points to the black points for a given mesh shows that the assimilation of high quality data with proper weighting always improves the accuracy of the final result for this test problem (i.e., the lower L_2-norm of the difference between the approximate and exact solution). This is exactly what one hopes will happen when experimental data (in this case artificial experimental data) is assimilated into an approximate numerical solution – a more accurate approximate solution is obtained. As the mesh is refined (e.g., the 38,000 (tetrahedral) and 60,000 (tetrahedral) element mesh) the approximate numerical solution becomes more accurate and there is less benefit associated with the assimilation of the experimental data. Again, this is exactly what one would desire. An exact or nearly exact numerical model does not need the benefits that are associated with assimilating experimental data.



(a) (b)

Figure 5.6. Vorticity field generated from stage 1: (a) $\tau = 0.0$, and (b) $\tau = 0.025$.

In reality, there is a broad range of accuracy among different numerical models that could be used for the first stage and some models will benefit more than others from the assimilation of experimental data. These results also shows the robustness of the new data assimilation approach to compensate for any inadequacies in the numerical model (i.e., the stage 1 solution) through the addition of data to obtain a final numerical solution (i.e., the

stage 2 solution) obtained better represents the physical system being modeled. It is important to emphasize that the artificial error is added to each node so the finer meshes effectively have more error included. This was an intentional choice as the overall accuracy of all numerical methods used for stage 1 are well established. Using a Galerkin finite element approximation for the solution of the Navier-Stokes equations leads to improved mass conservation compared to previous approaches that fully use WLSFEM approximations for both stage 1 and stage 2 of the data assimilation process.

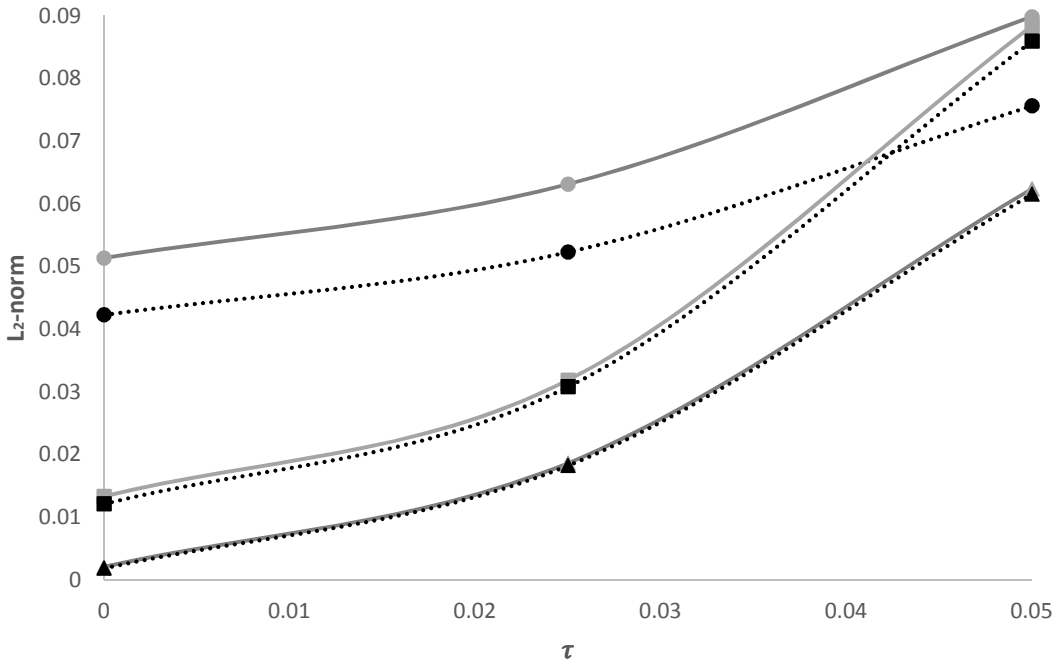


Figure 5.7. The error in the final numerical solution (after the second stage) as a function of the error in the initial numerical solution from stage 1. The gray points have no assimilated data and the black points include assimilated data. Circle points are an approximately 4800 element mesh, square points are a 38000 element mesh, and triangles are a 60000 element mesh.

5.4.3. Results and Discussion (Test Problem III)

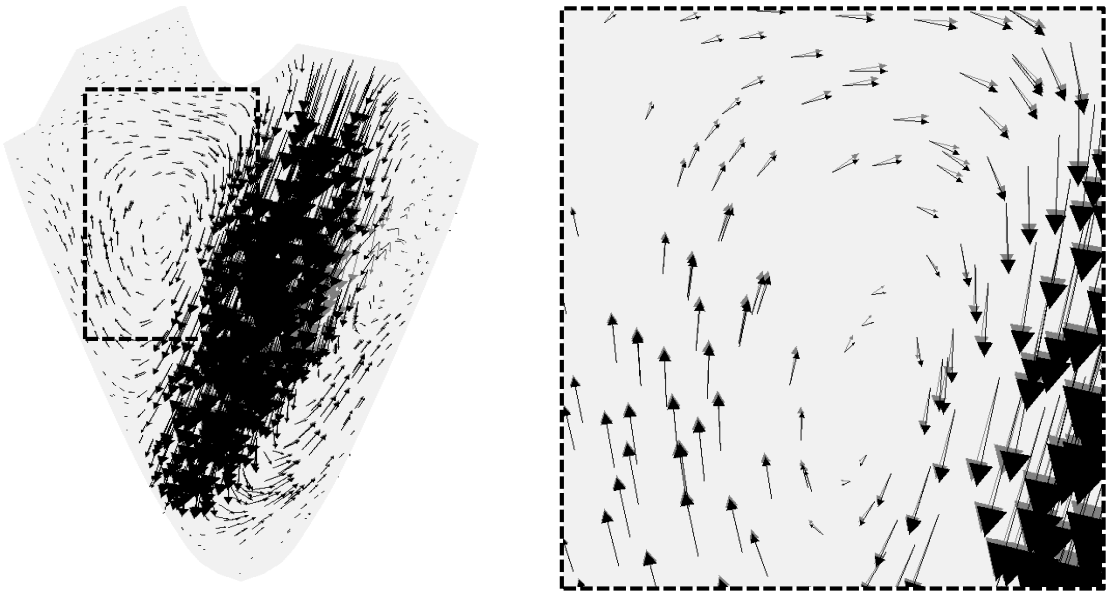
The new approach was also tested for the blood flow in the left ventricle. The numerical model and the resulting numerical solution from stage 1 without data assimilation do not capture all the physical influences on blood flow in the left ventricle. Many simplifications and assumptions have been made in deriving the mathematical model. For example, the current model does not include the mitral valve. Omitting the mitral valve and other, hopefully minor, physical phenomenon, significant reduces the computation time and programing complexity. Data assimilation provides a framework to partially recover the lost physical phenomenon due to model simplifications.

In Figure 5.8, the velocity field on the PIV plane is shown at each mesh node for the numerical solution influenced by the echo-PIV data (black) and without the influence of the data (gray). The impact of the echo-PIV data is highlighted on the PIV plane because this where the impact is greatest. Figure 5.8(a) highlights a temporal snapshot during the filling phase and Figure 5.8(b) is a sample time point during the ejection phase. Both plots shown in Figure 5.8 are a snapshot of a single time step of an entire simulation, and the assimilated echo-PIV data improves the numerical solution even though the echo-PIV data is noisy (i.e., contains errors). The impact of the assimilated data on the numerical solution is relatively small, which is anticipated since the numerical model agrees well with the experimental echo-PIV data at the time steps shown in Figure 8.

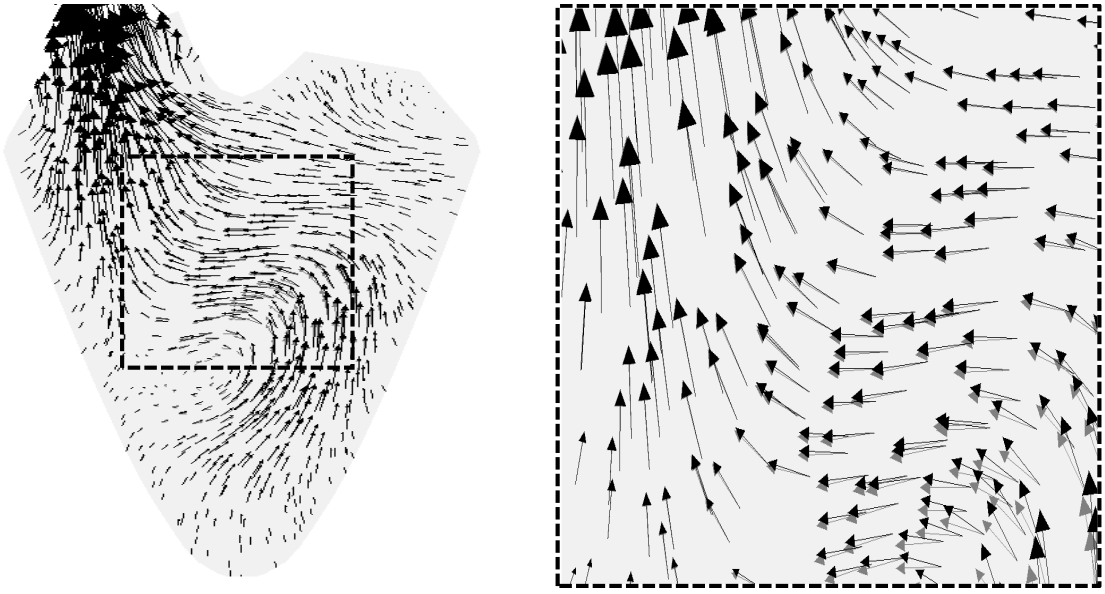
The boxed areas in Figure 5.8 were chosen to highlight interesting physics (i.e., vortices) observed during the late diastole (filling) and systole (ejection) phases. In Figure 5.8(a) the boxed area focused on the recirculation near the outlet and the simulations with

and without echo-PIV are consistent, which shows that when the echo-PIV data is consistent with the numerical model, only small changes are observed. In Figure 5.8(b) the boxed region is placed near the outlet during the systole phase, and the gray and black arrows show good agreement near the outlet. This agreement is due to the impact of the boundary conditions at the outlet that dominated the flow field relative the echo-PIV data. This observed solution is reasonable if the model is accurate enough and the data does not contain significant amounts of error.

Figure 5.9 is a snap shot of the numerical solution during the final stages of the late diastole phase where there is minimal flow into the domain. The numerical solution with echo-PIV data is shown in Figure 5.9(a), and numerical solution without data can be seen in Figure 5.9(b). The numerical model tends to be less accurate during these lower flow rate time steps due to the underlining assumptions made for the numerical model, which include neglecting the effects of the mitral valve. The contour lines in Figure 5.9(a) show higher velocities in the lower right region of the left ventricle compared to the contours shown in Figure 5.9(b). These higher velocities are due to a vortex (i.e., a secondary recirculation of the fluid and higher velocity) generated by the mitral valve. This vortex and locally higher velocity are either not observed or are much weaker when there is no echo-PIV data incorporated into the numerical model because the model does not include the mitral valve. Without the mitral valve, only a very weak vortex is observed in the numerical model at this stage of the cardiac cycle, even with a highly refined mesh.

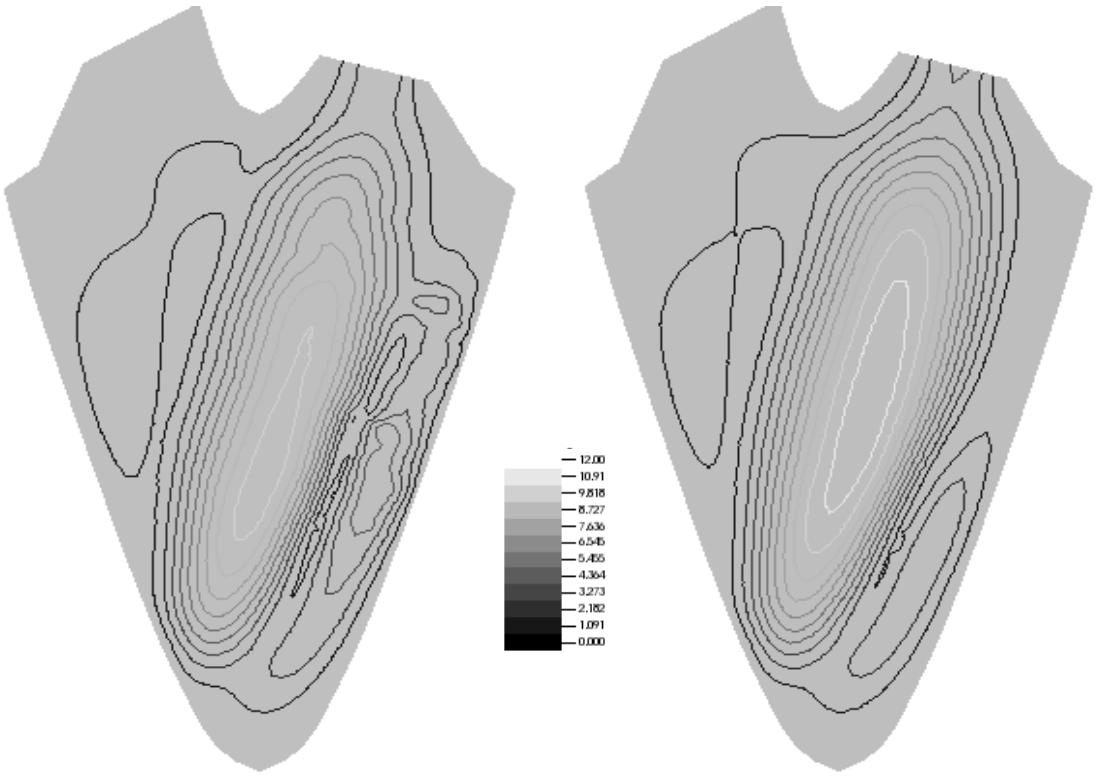


(a) Filling Phase



(b) Ejection Phase

Figure 5.8. Velocity field with assimilated echo-PIV data (black) and without data assimilation (gray) during: (a) late diastole, and (b) systole. The boxed region is shown on the right.

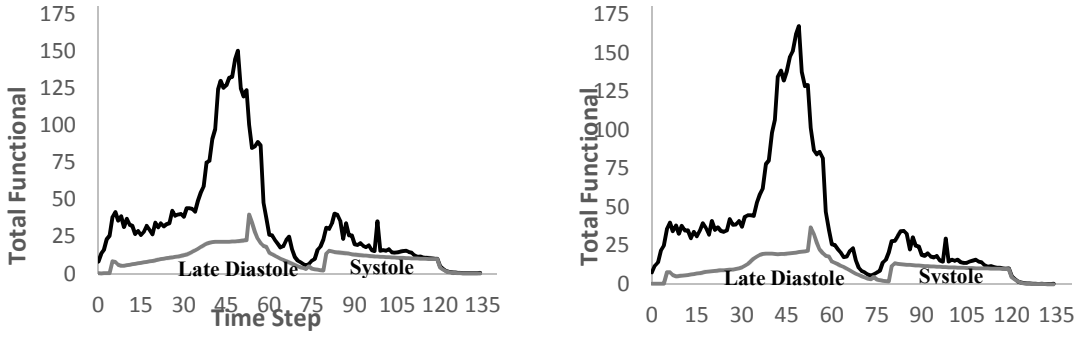


(a) (b)

Figure 5.9. Velocity contour field with echo-PIV data (a) and without data (b) during late diastole.

In order to quantify the effects of the assimilated echo-PIV data, the value of the functional (5.4) was calculated at every time step. The functional is a measure of the error in the numerical solution relative to the exact solution to the second stage equations (i.e., system (5.3)). It is important to note that the functional is not a measure of the error between the true, physical solution and the approximate solution, only a measure of the error for the second stage solution. Hence, a large functional value indicates the incompatibility between the equations defined in system (5.3) and the echo-PIV data and a smaller value indicated "good" agreement between the model and the data.

In Figure 5.10, the gray lines are the functional values without data assimilation while the black lines are the functional values with data assimilation calculated at every time step. The WLSFEM minimizes the value of the functional over the finite element approximation space, so the functional value must increase when additional data is assimilated and influences the approximate solution. The functional value increases during the filling phase (late diastole) and decreases during the ejection phase (systole) due to both the change in the mesh size, h, (the expanding domain increases the size of the elements and makes the numerical model less accurate), and the changes in the accuracy of the echo-PIV data (higher velocities can make the tracking of microbubbles more difficult for the PIV software). Comparing both parts of Figure 5.10, the functional shows a mild decrease when the mesh is refinement from approximately 40000 elements (Figure 5.10(a)) to 52000 elements (Figure 5.10(b)), consistent with finite element approximation theory. During the late filling (diastole) phase, the functional increases significantly when the echo-PIV data is assimilated due to the noisy data generated by tracking all the microbubbles.



(a) (b)

Figure 5.10. Functional (4) calculated at each time step with echo-PIV data (black) and without data (gray). Mesh with approximately 40000 (a) elements and 52000 (b) elements.

5.5. Conclusion

We have presented a new numerical approach for assimilating experimental data into an existing numerical model solution through a two-stage process: (1) solving the original numerical model, and (2) assimilating experimental data using the WLSFEM approach. The new approach was used to assess the impact of error in the assimilated experimental data (Test Problem 1) and to assess the impact of error in the stage 1 numerical model (Test Problem 2). Finally, the new approach for data assimilation was demonstrated through the assimilation of echo-PIV data into a numerical model of blood flow in the left ventricle (Test Problem 3). The advantages of the new approach include: the ability to integrate with existing CFD codes (or many other numerical modeling codes based on PDE models), and the computational cost can be significantly lower compared to using least-squares finite element methods for all aspects of the numerical model.

5.6. References

1. Belohlavek, M., *Vortex formation time: an emerging echocardiographic index of left ventricular filling efficiency?* European Heart Journal-Cardiovascular Imaging, 2012. **13**(5): p. 367-369.

2. D'Elia, M., *Assimilation of velocity data into fluid dynamic simulations, an application to computational hemodynamics*. 2011, Emory University.

3. D'Elia, M., et al., *Applications of variational data assimilation in computational hemodynamics*, in *Modeling of Physiological Flows*. 2012, Springer. p. 363-394.

4. Sengupta, P.P., et al., *Left ventricular form and function revisited: applied translational science to cardiovascular ultrasound imaging.* Journal of the American Society of Echocardiography, 2007. **20**(5): p. 539-551.

5. Westerdale, J., et al., *Flow velocity vector fields by ultrasound particle imaging velocimetry in vitro comparison with optical flow velocimetry.* Journal of Ultrasound in Medicine, 2011. **30**(2): p. 187-195.

6. Agati, L., et al., *Quantitative analysis of intraventricular blood flow dynamics by echocardiographic particle image velocimetry in patients with acute myocardial infarction at different stages of left ventricular dysfunction.* European Heart Journal-Cardiovascular Imaging, 2014: p. jeu106.

7. Bertagna, L., et al., *Data Assimilation in Cardiovascular Fluid–Structure Interaction Problems: An Introduction*, in *Fluid-Structure Interaction and Biomedical Applications*. 2014, Springer. p. 395-481.

8. Gao, H., et al., *How to optimize intracardiac blood flow tracking by echocardiographic particle image velocimetry? Exploring the influence of data acquisition using computer-generated data sets.* European Heart Journal-Cardiovascular Imaging, 2011: p. jer285.

9. Rajaraman, P.K., et al., *Echocardiographic particle imaging velocimetry data assimilation with least square finite element methods.* Computers & Mathematics with Applications, 2014. **68**(11): p. 1569-1580.

10. Sengupta, P.P., et al., *Emerging trends in CV flow visualization.* JACC: Cardiovascular Imaging, 2012. **5**(3): p. 305-316.

11. Borazjani, I., et al., *Left ventricular flow analysis: recent advances in numerical methods and applications in cardiac ultrasound.* Computational and mathematical methods in medicine, 2013. **2013**.

12. Wei, F., et al., *Weighted least-squares finite element method for cardiac blood flow simulation with echocardiographic data.* Computational and mathematical methods in medicine, 2012. **2012**.

13. D'Elia, M., M. Perego, and A. Veneziani, *A variational data assimilation procedure for the incompressible Navier-Stokes equations in hemodynamics*. Journal of Scientific Computing, 2012. **52**(2): p. 340-359.

14. Dwight, R.P. *Bayesian inference for data assimilation using least-squares finite element methods*. in *IOP Conference Series: Materials Science and Engineering*. 2010. IOP Publishing.

15. Heys, J.J., et al., *Weighted least-squares finite elements based on particle imaging velocimetry data.* Journal of Computational Physics, 2010. **229**(1): p. 107-118.

16. Heys, J.J., et al., *Enhanced mass conservation in least-squares methods for Navier-Stokes equations.* SIAM Journal on Scientific Computing, 2009. **31**(3): p. 2303-2321.

17. Thielicke, W. and E.J. Stamhuis, *PIVlab–Towards user-friendly, affordable and accurate digital particle image velocimetry in MATLAB*. Journal of Open Research Software, 2014. **2**(1): p. e30.

18. Bochev, P.B. and M.D. Gunzburger, *Least-squares finite element methods*. Vol. 166. 2009: Springer Science & Business Media.

19. Donea, J. and A. Huerta, *Finite element methods for flow problems*. 2003: John Wiley & Sons.

20. Gresho, P., R. Sani, and M. Engelman, *Incompressible flow and the finite element method: advection-diffusion and isothermal laminar flow. 1998*. John Wiley& Sons.

21. Bochev, P.B., *Analysis of Least-Squares Finite Element Methods for the Navier--Stokes Equations.* SIAM Journal on Numerical Analysis, 1997. **34**(5): p. 1817-1844.

22. Jiang, B.N., *A least-squares finite element method for incompressible Navier-Stokes problems*. International Journal for Numerical Methods in Fluids, 1992. **14**(7): p. 843-859.

23. Rajaraman, P. and J.J. Heys, *Simulation of nanoparticle transport in airways using Petrov–Galerkin finite element methods.* International journal for numerical methods in biomedical engineering, 2014. **30**(1): p. 103-116.

24. Burman, E. and M.A. Fernández, *Continuous interior penalty finite element method for the time-dependent Navier–Stokes equations: space discretization and convergence.* Numerische Mathematik, 2007. **107**(1): p. 39-77.

25. Burman, E. and P. Hansbo, *Edge stabilization for the generalized Stokes problem: a continuous interior penalty method.* Computer methods in applied mechanics and engineering, 2006. **195**(19): p. 2393-2410.

26. Heys, J.J., et al., *On mass-conserving least-squares methods.* SIAM Journal on Scientific Computing, 2006. **28**(5): p. 1675-1693.

27. Heys, J.J., et al., *An alternative least-squares formulation of the Navier–Stokes equations with improved mass conservation.* Journal of Computational Physics, 2007. **226**(1): p. 994-1006.

28. Adler, J. and P. Vassilevski, *Improving Conservation for First-Order System Least-Squares Finite-Element Methods*, in *Numerical Solution of Partial Differential Equations: Theory, Algorithms, and Their Applications*. 2013, Springer. p. 1-19.

29. Heys, J., et al., *First-order system least-squares (FOSLS) for modeling blood flow.* Medical engineering & physics, 2006. **28**(6): p. 495-503.

30. Manteuffel, T., et al., *Further results on error estimators for local refinement with first-order system least squares (FOSLS).* Numerical Linear Algebra with Applications, 2010. **17**(2-3): p. 387-413.

31. D'Elia, M. and A. Veneziani, *Uncertainty quantification for data assimilation in a steady incompressible Navier-Stokes problem.* ESAIM: Mathematical Modelling and Numerical Analysis, 2013. **47**(04): p. 1037-1057.

32. Logg, A., K.-A. Mardal, and G. Wells, *Automated solution of differential equations by the finite element method: The FEniCS book*. Vol. 84. 2012: Springer Science & Business Media.

33. Falgout, R. and U. Yang, *hypre: A library of high performance preconditioners.* Computational Science—ICCS 2002, 2002: p. 632-641.

34. Greenbaum, A., *Iterative methods for solving linear systems*. Vol. 17. 1997: Siam.

35. Niekrasz, M., et al. *The pig as organ donor for man*. in *Transplantation proceedings*. 1992.

CHAPTER 6

6.1. CONCLUSION AND FUTURE WORK

The main focus of this thesis is to demonstrate a data assimilation framework for echo-PIV data that is computationally less expensive than existing methods that are typically based on an ensemble of solutions or the solving of a constrained optimization problem. Even though the assimilation of echo-PIV data into numerical models was very much the focus, other aspect of the model such as the weighting of the data, the geometry of the left ventricle and mass conservation were also studied in this thesis. There are many issues associated with assimilating data into a numerical model, including error analysis and special weighting of the data based on accuracy. In order to overcome these problems we presented an approach to weight the data such that more accurate data will be weighted with a larger weight and less accurate data will be weighted with a smaller weight. We have successfully demonstrated the weighting approach through test cases that compare the model prediction with known exact solutions such as the Poiseuille flow. The impact of the assimilated echo-PIV data was also assessed using the WLSFEM functional.

Another issue that was a focus of this thesis is the creation of a data assimilation framework that is computationally cheaper and does not suffer severe mass loss. We have presented a new numerical approach for echo-PIV data assimilating into a numerical solution using a two-step process where the first stage is to solve the numerical model using some discretization technique (e.g., FEM) that does not incorporate any echo-PIV data. The second step, is to assimilate the echo-PIV data using the WLSFEM approach. In this

thesis we presented a framework for the assimilation of velocity data, but this same framework can be extended to almost any other partial differential equation with experimental data available for the unknown (or derivatives of the unknowns). The advantage of the new approach is the computational cost, which is significantly lower compared to using the least-square finite element methods for all aspects of the numerical model.

Future work on this topic could be focused on creating a much more accurate model by seeking a numerical solution with a much finer mesh. The current method presented in this thesis is a second order accurate method both in space and time. Implementing a higher order method in space will help to resolve boundary layers in the numerical solution. Applying this framework as a diagnostic tool to monitor patient's health requires multiple data sets and incorporating statistical analysis of the numerical solution will be vital. Although the current state of blood flow data assimilation requires hours to solve for a reasonable numerical solution, further algorithmic and computer improvements can potentially be made to obtain a numerical solution nearly instantly in the future. In order to improve the current approach for left ventricular blood flow data assimilation we propose several advancements:

1. Develop numerical methods that include mitral valves, without significantly increasing computational cost.
2. Blood flow data currently obtained from the echo-PIV method are at a temporal resolution of $150\,\frac{frames}{s}$. This scan rate is the highest possible

with current technology. A higher temporal resolution than currently available will provide more accurate blood flow data.

3. Currently available methods for generating a 3D model geometry for CFD analysis are limited for the echo-PIV imaging method. Progress in this area will allow better quantification of uncertainty regarding the numerical model.

Application of the new data assimilation framework can be applied to other real world problems described with partial differential equations. The real world problems include meteorology and oceanography, where abundant data are available from satellite imaging. Problems in this field include hurricane predication where the predicted path and intensity of the hurricane are important. A common problem in oceanography is predicting the sea surface temperature, which plays an important role in climate systems. The problem in this area is that the traditional data assimilation methods suffer from strong nonlinearities and turbulence in the ocean model. Implementing the new data assimilation framework to this problem will be challenging, and further research needs to be conducted in the area.

REFERENCES CITED

1. Hall, J.E., *Guyton and Hall textbook of medical physiology*. 2010: Elsevier Health Sciences.

2. Borazjani, I., et al., *Left ventricular flow analysis: recent advances in numerical methods and applications in cardiac ultrasound.* Computational and mathematical methods in medicine, 2013. **2013**.

3. Sengupta, P.P., et al., *Left ventricular form and function revisited: applied translational science to cardiovascular ultrasound imaging.* Journal of the American Society of Echocardiography, 2007. **20**(5): p. 539-551.

4. Sengupta, P.P., et al., *Left ventricular isovolumic flow sequence during sinus and paced rhythms: new insights from use of high-resolution Doppler and ultrasonic digital particle imaging velocimetry.* Journal of the American College of Cardiology, 2007. **49**(8): p. 899-908.

5. Sengupta, P.P., et al., *Left ventricular structure and function: basic science for cardiac imaging.* Journal of the American College of Cardiology, 2006. **48**(10): p. 1988-2001.

6. Wei, F., et al., *Weighted least-squares finite element method for cardiac blood flow simulation with echocardiographic data.* Computational and mathematical methods in medicine, 2012. **2012**.

7. DeMarchi, N. and C. White, *Echo Particle Image Velocimetry.* Journal of visualized experiments: JoVE, 2012(70).

8. Kheradvar, A., et al., *Echocardiographic particle image velocimetry: a novel technique for quantification of left ventricular blood vorticity pattern.* Journal of the American Society of Echocardiography, 2010. **23**(1): p. 86-94.

9. Rajaraman, P.K., et al., *Echocardiographic particle imaging velocimetry data assimilation with least square finite element methods.* Computers & Mathematics with Applications, 2014. **68**(11): p. 1569-1580.

10. Faludi, R., et al., *Left ventricular flow patterns in healthy subjects and patients with prosthetic mitral valves: an in vivo study using echocardiographic particle image velocimetry.* The Journal of Thoracic and Cardiovascular Surgery, 2010. **139**(6): p. 1501-1510.

11. D'Elia, M., M. Perego, and A. Veneziani, *A variational data assimilation procedure for the incompressible Navier-Stokes equations in hemodynamics.* Journal of Scientific Computing, 2012. **52**(2): p. 340-359.

12. Hong, G.-R., et al., *Characterization and quantification of vortex flow in the human left ventricle by contrast echocardiography using vector particle image velocimetry.* JACC: Cardiovascular Imaging, 2008. **1**(6): p. 705-717.

13. D'Elia, M., *Assimilation of velocity data into fluid dynamic simulations, an application to computational hemodynamics*. 2011, Emory University.

14. Gao, H., et al., *How to optimize intracardiac blood flow tracking by echocardiographic particle image velocimetry? Exploring the influence of data acquisition using computer-generated data sets.* European Heart Journal-Cardiovascular Imaging, 2011: p. jer285.

15. Zhang, F., et al., *In vitro and preliminary in vivo validation of echo particle image velocimetry in carotid vascular imaging.* Ultrasound in medicine & biology, 2011. **37**(3): p. 450-464.

16. Jiamsripong, P., et al., *Increase in the late diastolic filling force is associated with impaired transmitral flow efficiency in acute moderate elevation of left ventricular afterload.* Journal of Ultrasound in Medicine, 2009. **28**(2): p. 175-182.

17. Westerdale, J., et al., *Flow velocity vector fields by ultrasound particle imaging velocimetry in vitro comparison with optical flow velocimetry.* Journal of Ultrasound in Medicine, 2011. **30**(2): p. 187-195.

18. Sengupta, P.P., G. Pedrizetti, and J. Narula, *Multiplanar visualization of blood flow using echocardiographic particle imaging velocimetry.* JACC: Cardiovascular Imaging, 2012. **5**(5): p. 566-569.

19. Heys, J.J., et al., *Weighted least-squares finite elements based on particle imaging velocimetry data.* Journal of Computational Physics, 2010. **229**(1): p. 107-118.

20. Thielicke, W. and E.J. Stamhuis, *PIVlab–Towards user-friendly, affordable and accurate digital particle image velocimetry in MATLAB.* Journal of Open Research Software, 2014. **2**(1): p. e30.

21. D'Elia, M. and A. Veneziani. *A data assimilation technique for including noisy measurements of the velocity field into Navier-stokes simulations*. in *Proceedings of V European Conference on Computational Fluid Dynamics, ECCOMAS*. 2010.

22. Heys, J., et al., *First-order system least-squares (FOSLS) for modeling blood flow.* Medical engineering & physics, 2006. **28**(6): p. 495-503.

23. Gunzburger, M.D., *Finite Element Methods for Viscous Incompressible Flows: A guide to theory, practice, and algorithms*. 2012: Elsevier.

24. Gresho, P.M. and R.L. Sani, *Incompressible flow and the finite element method. Volume 1: Advection-diffusion and isothermal laminar flow.* 1998.

25. Rajaraman, P. and J.J. Heys, *Simulation of nanoparticle transport in airways using Petrov–Galerkin finite element methods.* International journal for numerical methods in biomedical engineering, 2014. **30**(1): p. 103-116.

26. Fix, G.J., M.D. Gunzburger, and R. Nicolaides, *On finite element methods of the least squares type.* Computers & Mathematics with Applications, 1979. **5**(2): p. 87-98.

27. Bochev, P.B. and M.D. Gunzburger, *Accuracy of least-squares methods for the Navier-Stokes equations.* Computers & fluids, 1993. **22**(4): p. 549-563.

28. Bochev, P.B. and M.D. Gunzburger, *Analysis of least squares finite element methods for the Stokes equations.* Mathematics of Computation, 1994. **63**(208): p. 479-506.

29. Bochev, P.B. and M.D. Gunzburger, *Finite element methods of least-squares type.* SIAM review, 1998. **40**(4): p. 789-837.

30. Bochev, P.B. and M.D. Gunzburger, *Least-squares finite element methods*. Vol. 166. 2009: Springer Science & Business Media.

31. Heys, J.J., et al., *First-order system least squares for elastohydrodynamics with application to flow in compliant blood vessels.* Biomedical sciences instrumentation, 2001. **38**: p. 277-282.

32. Heys, J.J., et al., *Modeling 3-D compliant blood flow with fosls.* Biomedical sciences instrumentation, 2004. **40**: p. 193-199.

33. Heys, J.J., et al., *First-order system least squares (FOSLS) for coupled fluid-elastic problems.* Journal of Computational Physics, 2004. **195**(2): p. 560-575.

34. Heys, J.J., et al., *On mass-conserving least-squares methods.* SIAM Journal on Scientific Computing, 2006. **28**(5): p. 1675-1693.

35. Heys, J.J., et al., *Enhanced mass conservation in least-squares methods for Navier-Stokes equations.* SIAM Journal on Scientific Computing, 2009. **31**(3): p. 2303-2321.

36. Deang, J.M. and M.D. Gunzburger, *Issues related to least-squares finite element methods for the Stokes equations.* SIAM Journal on Scientific Computing, 1998. **20**(3): p. 878-906.

37. Dwight, R.P., J.A. Witteveen, and H. Bijl, *Adaptive uncertainty quantification for computational fluid dynamics*, in *Uncertainty Quantification in Computational Fluid Dynamics*. 2013, Springer. p. 151-191.

38. Melani, A., *Adjoint-based parameter estimation in human vascular one dimensional models*. 2013, Italy.

39. Bertagna, L., et al., *Data Assimilation in Cardiovascular Fluid–Structure Interaction Problems: An Introduction*, in *Fluid-Structure Interaction and Biomedical Applications*. 2014, Springer. p. 395-481.

40. Dwight, R.P., *Bayesian inference for data assimilation using least-squares finite element methods.* IOP Conference Series: Materials Science and Engineering, 2010. **10**(1): p. 012224.

41. Heys, J.J., et al., *An alternative least-squares formulation of the Navier–Stokes equations with improved mass conservation.* Journal of Computational Physics, 2007. **226**(1): p. 994-1006.

42. Griffith, B.E., *Simulating the blood-muscle-valve mechanics of the heart by an adaptive and parallel version of the immersed boundary method*. 2005, New York University.

43. Griffith, B.E., *Immersed boundary model of aortic heart valve dynamics with physiological driving and loading conditions.* International Journal for Numerical Methods in Biomedical Engineering, 2012. **28**(3): p. 317-345.

44. Griffith, B.E., et al., *An adaptive, formally second order accurate version of the immersed boundary method.* Journal of Computational Physics, 2007. **223**(1): p. 10-49.

45. Griffith, B.E., et al., *Parallel and adaptive simulation of cardiac fluid dynamics*, in *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*. 2010. p. 105.

46. Griffith, B.E., et al., *Simulating the fluid dynamics of natural and prosthetic heart valves using the immersed boundary method.* International Journal of Applied Mechanics, 2009. **1**(01): p. 137-177.

47. Griffith, B.E., D.M. McQueen, and C.S. Peskin, *Simulating cardiovascular fluid dynamics by the immersed boundary method.* 47th AIAA Aerospace Sciences Meeting, 2009.

48. Griffith, B.E. and C.S. Peskin, *On the order of accuracy of the immersed boundary method: Higher order convergence rates for sufficiently smooth problems.* Journal of Computational Physics, 2005. **208**(1): p. 75-105.

49. Gunzburger, M., L. Hou, and T.P. Svobodny, *Analysis and finite element approximation of optimal control problems for the stationary Navier-Stokes equations with Dirichlet controls.* RAIRO-Modélisation mathématique et analyse numérique, 1991. **25**(6): p. 711-748.

50. D'Elia, M., et al., *Applications of variational data assimilation in computational hemodynamics*, in *Modeling of Physiological Flows*. 2012, Springer. p. 363-394.

51. Gunzburger, M., *Adjoint equation-based methods for control problems in incompressible, viscous flows.* Flow, Turbulence and Combustion, 2000. **65**(3-4): p. 249-272.

52. Gunzburger, M.D., *Sensitivities, adjoints and flow optimization.* International Journal for Numerical Methods in Fluids, 1999. **31**(1): p. 53-78.

53. D'Elia, M. and A. Veneziani, *Methods for assimilating blood velocity measures in hemodynamics simulations: Preliminary results.* Procedia Computer Science, 2010. **1**(1): p. 1231-1239.

54. D'Elia, M. and A. Veneziani, *Uncertainty quantification for data assimilation in a steady incompressible Navier-Stokes problem.* ESAIM: Mathematical Modelling and Numerical Analysis, 2013. **47**(04): p. 1037-1057.

55. Gunzburger, M.D., L. Hou, and J. Ming, *Stochastic Steady-State Navier–Stokes Equations with Additive Random Noise.* Journal of Scientific Computing, 2015: p. 1-20.

APPENDICES

APPENDIX A

3D FEM IMPLEMENTATION IN FENICS USING PYTHON

```
from dolfin import *
import numpy, sys, math
parameters["std_out_all_processes"] = False;
parameters["form_compiler"]["optimize"] = True
parameters['form_compiler']['cpp_optimize'] = True
parameters["form_compiler"]["representation"] = 'quadrature'
parameters["form_compiler"]["quadrature_degree"] = 4
parameters['reorder_dofs_serial'] = False
# Create files for storing solution
ufile = File("results_87000/velocity.pvd","compressed")
pfile = File("results_87000/pressure.pvd","compressed")
wfile = File("results_87000/vorticity.pvd","compressed")
movefile = File("results_move_87000/move.pvd","compressed")

maxiter = 1
move_iters = 0
max_move_iters = 135
dt = Constant(0.019)
niter = 135
eps = 1.0
tol = 1.0e-5
nu = Constant(0.001785714)
nu_re = Constant(0.042257713)
nu_move = Constant(150.0)
theta = Constant(0.5)
k = Constant(1.0/dt)
c1 = Constant(4.0/3.0)
c2 = Constant(1.0/3.0)
c3 = Constant(2.0/3.0)
mesh = Mesh("ven_87000.xml")
mesh_facet = MeshFunction('size_t',mesh,"ven_87000_facet_region.xml")
mesh_boundary = BoundaryMesh(mesh,'exterior',True)

#Define Finite Element Space
V = VectorFunctionSpace(mesh, "CG", 2)
V_move = VectorFunctionSpace(mesh_boundary, "CG", 2)
Q = FunctionSpace(mesh, "CG", 1)
W = MixedFunctionSpace([V, Q])

n = FacetNormal(mesh)
ds = Measure("ds")[mesh_facet]
h = CellSize(mesh)
h_F = MinFacetEdgeLength(mesh)
alpha = Constant(150.0)

#Define test functions
(v,q) = TestFunctions(W)

#Define trial Function
w = Function(W)
wold = Function(W)
wolder = Function(W)
```

```
woldest = Function(W)

(u,p) = (as_vector((w[0],w[1],w[2])),w[3])
(uold,pold) = (as_vector((wold[0],wold[1],wold[2])),wold[3])
(uolder,polder) = (as_vector((wolder[0],wolder[1],wolder[2])),wolder[3])
(uoldest,poldest)=(as_vector((woldest[0],woldest[1],woldest[2])),woldes
t[3])

# No-slip boundary condition for velocity
zero = Expression('0.0')
f = Expression(('x[0]/0.019','x[1]/0.019','x[2]/0.019'))
zeroV = Expression(('0.0','0.0','0.0'))
dist = interpolate(zeroV,V)
#1=wall,2=bottomwall,3=in,4=topwall,5=outlet
bc1 = DirichletBC(W.sub(0), G_vel, mesh_facet,1)#wall
bc2 = DirichletBC(W.sub(0), G_vel, mesh_facet,2)#bottomwall
bc5 = DirichletBC(W.sub(0), zeroV, mesh_facet,4)#topwall
bc1v = DirichletBC(W.sub(0), zeroV, mesh_facet,1)#wall
bc2v = DirichletBC(W.sub(0), zeroV, mesh_facet,2)#bottomwall
bc4v = DirichletBC(W.sub(0), zeroV, mesh_facet,3)#in
bc6v = DirichletBC(W.sub(0), zeroV, mesh_facet,5)#out

#NS-Weak form
F=k*inner(u-c1*uold+c2*uolder,v)*dx+c3*(inner(dot(u,nabla_grad(u)),v)*
dx-inner(dot(dist,nabla_grad(u)),v)*dx+nu*inner(grad(u),grad(v))*dx-
p*div(v)*dx)+(q*div(u))*dx-c3*inner(zeroV,v)*dx
F=F+c3*inner(cross(v,n),cross(n,cross(u,n)))*ds(1)+
c3*inner(cross(v,n),cross(n,cross(u,n)))*ds(2)+
c3*inner(cross(v,n),cross(n,cross(u,n)))*ds(3)+
c3*inner(cross(v,n),cross(n,cross(u,n)))*ds(5)
F=F+c3*alpha('+')*h_F**2*inner(jump(grad(u), n), jump(grad(v), n))*dS
F=F+c3*alpha('+')*h_F**3*inner(jump(grad(p),n), jump(grad(q),n))*dS
F=F+c3*alpha('+')*h_F**2*inner(jump(div(u)), jump(div(v)))*dS
# iter
while move_iters < max_move_iters:
    move_iters +=1
    iters = 0
    movefile << mesh
    while iters < maxiter:
        iters += 1
        begin("Solve NS")
        if move_iters >= 1 and move_iters <= 59:
            bcu = [bc1,bc2,bc6v,bc5]
        if move_iters >= 75 and move_iters <= 120:
            bcu = [bc1,bc2,bc4v,bc5]
        if move_iters >= 60 and move_iters <= 74:
            bcu = [bc1v,bc2v,bc4v,bc6v,bc5]
        if move_iters >= 121 and move_iters <= 134:
            bcu = [bc1v,bc2v,bc4v,bc6v,bc5]
        if move_iters >= 60 and move_iters <= 74:
            alpha.assign(nu_move)
        if move_iters >= 121 and move_iters <= 134:
            alpha.assign(nu_move)
        #Derivative of weak form
```

```
dw = TrialFunction(W)
J = derivative(F,w,dw)
problem = NonlinearVariationalProblem(F,w,bcu,J)
solver = NonlinearVariationalSolver(problem)

#Newton Solver parameters
prm = solver.parameters
prm['nonlinear_solver'] = 'snes'
prm['snes_solver']['line_search'] = 'basic'
prm['snes_solver']['linear_solver']= 'mumps'
prm['snes_solver']['absolute_tolerance'] = 1E-4
prm['snes_solver']['relative_tolerance'] = 1E-5
set_log_level(PROGRESS)

solver.solve()
wolder.assign(wold)
wold.assign(w)
print alpha
end()
begin("Mesh motion")
if move_iters <= 59:
    mesh.move(mesh_boundary)
    mesh.smooth()

if move_iters >= 75 and move_iters <= 120:
    mesh.move(mesh_boundary)
    mesh.smooth()
#movefile << mesh
dist = interpolate(f,V)
end()
(u1,p1) = w.split()
ufile << u1
pfile << p1
vort= project(curl(u1),V)
wfile << interpolate(vort,V)
```

The code above is used for creating the Navier-Stokes model in Chapter 5. The implementation is done in an open souce package called Fenics and written in python. The weak for is denied in F, which includes the stablizastion term. The solver used to solve the nonlinear matrix problem uses the PETSC snes solver. Once the velocity is computed the final step is to calcuted the vorticity by taking the projection of the curl of velocity, which defined in the code as $vort$. The vorticity is read in by the code shown in Appendix C, which solves the data assimilation model.

APPENDIX B

2D FEM IMPLEMENTATION IN C++

2D Finite Element Implementation using C++

This appendix is a summary of the C++ code that is used to build all the models in Chapter 4. The important parts of the code will be discussed in detail. The main function is call fem2d, which includes all the prototypes, as shown below. There are 5 prototypes in this model: the first prototypes is build_local, which is the function where the isoparametric mapping is done, the assemble function is where the matrix and vector is assembled, initZero is where the matrix is initialized to zero before assembling the matrix, sol_out is the function called to write the results of the model.

```
int build_local(int, TriMesh *, RefElem *, double **, double *,
Epetra_Vector *, Epetra_Vector *,Epetra_Vector *, double);

void assemble(int, int,int, Epetra_CrsMatrix *, Epetra_Vector *,
double**, double *, TriMesh *, double, Epetra_Vector *);

void initZero(int,int, Epetra_CrsMatrix *, Epetra_Vector *, TriMesh *);

void sol_out(int *, Epetra_Vector *, TriMesh *, int, double);
```

```
TriMesh::TriMesh(){

sprintf(filename, "mesh.g");

// Open the ExodusII file
int ex_id = ex_open(filename, EX_READ, &comp_ws, &io_ws,&exodus_version);

if(ex_id < 0){
std::cerr << "Error:cannot open file " << filename << std::endl;
exit(1);
}

// Read the header/initialization data
ex_err=ex_get_init(ex_id,title,&num_dim,&NumNodes,&NumElem,&num_elem_bl
k,&num_node_sets,&num_side_sets);

// Read in node locations
if(num_dim == 2){
ex_get_coord(ex_id, NodeLoc[0], NodeLoc[1], 0);
} else {
cout << "ERROR: ONLY 2D CURRENTLY SUPPORTED" << endl;
exit(1);
}

// Read in element connectivity information
if(num_elem_blk == 1){
ex_err = ex_get_elem_blk_ids(ex_id, blk_ids);
ex_err=ex_get_elem_block(ex_id,blk_ids[0],elem_type,&num_elem_in_blk,
&NpE,&num_attr);

tot_connect = NumElem * NpE;
tmp_connect = new int [tot_connect];
if((NpE == 6)){
ex_err = ex_get_elem_conn(ex_id, blk_ids[0], tmp_connect);
for(int e=0; e<NumElem; e++){
for(int i=0; i<NpE; i++){
nop[e][i] = (tmp_connect[e*NpE + i]) - 1; // The -1 switches to C numbering
if(nop[e][i] >= NumNodes)
cout << "ERROR in connectivity data" << endl;
}
}
} else {
cout << "ERROR: ONLY 6 Node Triangle Elements currently supported" <<
endl;
}
} else {
cout << "ERROR: ONLY A SINGLE ELEMENT BLOCK IS ALLOWED" << endl;
exit(1);
}


NPpE = 3;
```

```
// Read in Node Set Boundary Data
ns_ids = new int [num_node_sets];
if(num_node_sets > MAX_NS){
cout << "Exceeded MAX_NS: Bad things will happen" << endl;
}
ex_err = ex_get_node_set_ids(ex_id, ns_ids);

for(int ns=0; ns<num_node_sets; ns++){
ex_err    =    ex_get_node_set_param(ex_id,    ns_ids[ns],    &num_nodes,
&df_per_ns);
tmp_nodes = new int [num_nodes];
NumEdge[ns] = num_nodes;
ex_err = ex_get_node_set(ex_id, ns_ids[ns], tmp_nodes);
for(int i=0; i<num_nodes; i++){
BCType[(tmp_nodes[i]) - 1] = ns_ids[ns];
}

delete[] tmp_nodes;
}

for(int i=0; i<NumNodes; i++){
if(BCType[i] == 2){
cout << "X= " << getNodeLocX(i) << " Y= " << getNodeLocY(i) << endl;
}
}
ex_err = ex_close(ex_id);
delete[] tmp_connect;

for (int i=0; i<NumNodes; i++)
pnop[i] = -1;

int counter = 0;
for (int e=0; e<NumElem; e++){
for (int i=0; i<NPpE; i++){
if( pnop[nop[e][i]] == -1){
pnop[nop[e][i]] = counter;
counter++;
}
}
}

NumPNodes = counter;
NumDof = 3 * NumNodes + NumPNodes;
localDof = 3 * NpE + NPpE;
}

double TriMesh::getNodeLocX(int n){
if(n > NumNodes){
cout << "ERROR: trying to get a node larger than NumNodes" << endl;
return(0.0);
} else {
return(NodeLoc[0][n]);
}
}
```

```
double TriMesh::getNodeLocY(int n){
if(n > NumNodes){
cout << "ERROR: trying to get a node larger than NumNodes" << endl;
return(0.0);
} else {
return(NodeLoc[1][n]);
}
}

void TriMesh::moveNodeX(int n, double x){
if(n > NumNodes){
cout << "ERROR: trying to get a node larger than NumNodes" << endl;
} else {
NodeLoc[0][n] = x;
}
}

void TriMesh::moveNodeY(int n, double y){
if(n > NumNodes){
cout << "ERROR: trying to get a node larger than NumNodes" << endl;
} else {
NodeLoc[1][n] = y;
}
}
```

The code above is called the Trimesh object and the RefElem object. The Trimesh object will read in the mesh file and organize the data from the file. In Trimesh, pnop() is the vertex nop() where, given an element number and a node number, we can know whether the node has an unknown on vertex. Similarly, nop() is created for the other degree of freedom such as velocity and concentration, since all these unknowns are on every node in the element, we can lump them into a single nop. The RefElem object is where the basis functions for the triangle are create and the Gauss Quadrature is defined.

```
//AD-equation
for (j=0; j<(mymesh->NpE); j++){
  dof = 2*mymesh->NumNodes + mymesh->nop[e][j];
  switch(mymesh->BCType[mymesh->nop[e][j]]){
    case 1:
    case 2:
    NumEntries = 1;
    vals[0] = 0.0;
    cols[0] = dof;
    A->InsertMyValues(dof, NumEntries, vals, cols);
    break;
    case 3:
    NumEntries = 1;
    vals[0] = 0.0;
    cols[0] = dof;
    A->InsertMyValues(dof, NumEntries, vals, cols);
    break;
    case 4:
    NumEntries = 1;
    vals[0] = 0.0;
    cols[0] = dof;
    A->InsertMyValues(dof, NumEntries, vals, cols);
    break;
    default:
    NumEntries = 3*mymesh->NpE;

    // d(AD)/du
    for (i=0; i < mymesh->NpE; i++){
      cols[i] = mymesh->nop[e][i];
      vals[i] = 0.0;
    }
    // d(AD)/dv
    for (i=0; i < mymesh->NpE; i++){
      cols[mymesh->NpE+i] = mymesh->NumNodes + mymesh->nop[e][i];
      vals[mymesh->NpE+i] = 0.0;
    }
    // d(AD)/dc, no P=0
    for (i=0; i < mymesh->NpE; i++){
      cols[2*mymesh->NpE+i] = 2 * mymesh->NumNodes + mymesh->nop[e][i];
      vals[2*mymesh->NpE+i] = 0.0;
    }
    A->InsertMyValues(dof, NumEntries, vals, cols);
    break;
  }
}
```

The code above initializes the matrix A by filling it in with zeros, and the very first line is the call in the main() function. The next part of the code is the routine used to fill

the matrix in zeros first, and this code is only a part of the function initZero. The code shown here will only fill the first row of the matrix. The case() statements are used to set the boundary conditions.

```
Ifpack Factory;
  Ifpack_Preconditioner *Prec;
  string PrecType = "ILU";
  Prec = Factory.Create(PrecType, &A);
  assert (Prec != 0);
  Teuchos::ParameterList List;

  Epetra_LinearProblem LP(&A,&dsol,&b);
  AztecOO Solver(LP);

  Prec->SetParameters(List);
  Prec->Initialize();

 Solver.SetPrecOperator(Prec);
  Solver.SetAztecOption(AZ_solver,AZ_gmres);
  Solver.SetAztecOption(AZ_output,100);
```

The code above implements the Ifpack solver from the Trilinos package, and the solver uses ILU-type preconditioner with GMRES. Additional information about the solver can be found in Trilinos documentation.

```
 for (int t_step=0; t_step < total_steps; t_step++){
    for (int i_ter=0; i_ter<iter; i_ter++){
      currentTime = (t_step+1.0) * dt;
      A.PutScalar(0.0); b.PutScalar(0.0);
      dsol.PutScalar(0.0); fulldsol.PutScalar(0.0);
      for(int e=0; e<(mymesh.num_local_elem); e++){
build_local(mymesh.MyElem[e], &mymesh, &basis, A_local, b_local, &sol,
&sol_old, &sol_older,dt);
assemble(mymesh.MyElem[e],myid, mymesh.NpE, &A, &b, A_local, b_local,
&mymesh, currentTime, &sol);
       }
      Prec->Compute();
      Solver.Iterate(1000, TOL);
      fulldsol.Import(dsol,FullImporter,Add);
        for (int i=0; i<mymesh.NumDof; i++){
        sol[i] = sol[i]-fulldsol[i];
        }
      //Condition for Newton
```

```
      b.Norm2(&norm);
      if(norm < TOL){
      i_ter = iter;
      }
    } // End of iteration loop
    //Write output
    if(myid==0){
      if((t_step % 10) == 0){
    sol_out(&ex_id, &sol, &mymesh, out_step, currentTime);
    out_step++;
      //Solution update
    for (int i=0; i<mymesh.NumDof; i++){
      sol_old[i] = sol[i];
      sol_older[i] = sol_old[i];
    }
  } // End of time loop
```

The code above implements the time loop and Newton iteration steps, and after the iteration loop and time step loop we insert zeros into the matrix and vectors using Trilinos commands. After this step we loop thru the elements and call the two functions:build_local() and assemble(). The convergence condition for Newton is done by calculating the norm of the residual, which are calculated using the Trilinos package. After ending the iteration loop we write the solution to an Exodus file using the function sol_out. After wring the solution to a file we will update the solution for the old and older time steps since this is need for the temporal discretization before ending the time loop.

In build_local (), the main steps are the isoparametric mapping and the weak form implementation, and these steps are shown below. The weak form shown below is for the advection-diffusion equation where the velocity is obtained by solving the Navier-Stokes equations.

```
for (int gpt=0; gpt<basis->ngp; gpt++){
  for (int i=0; i<(basis->NpE); i++){
   local_x+=mymesh->getNodeLocX(mymesh->nop[e][i])*basis->phi6[i][gpt];
   local_y+=mymesh->getNodeLocY(mymesh->nop[e][i])*basis->phi6[i][gpt];
   uold += sol_old[mymesh->nop[e][i]] * basis->phi6[i][gpt];
   unew += sol[mymesh->nop[e][i]] * basis->phi6[i][gpt];
   vold+=sol_old[mymesh->NumNodes+mymesh->nop[e][i]]*basis-
>phi6[i][gpt];
   vnew+= sol[mymesh->NumNodes+mymesh->nop[e][i]]*basis->phi6[i][gpt];
   cold+=sol_old[2*mymesh->NumNodes+mymesh->nop[e][i]]*basis-
>phi6[i][gpt];
   cnew+=sol[2*mymesh->NumNodes+mymesh->nop[e][i]]* basis->phi6[i][gpt];
   x_xsi+=mymesh->getNodeLocX(mymesh->nop[e][i])*basis-
>dphi6_xsi[i][gpt];
   x_eta+=mymesh->getNodeLocX(mymesh->nop[e][i])*basis-
>dphi6_eta[i][gpt];
   y_xsi+=mymesh->getNodeLocY(mymesh->nop[e][i])*basis-
>dphi6_xsi[i][gpt];
   y_eta+=mymesh->getNodeLocY(mymesh->nop[e][i])*basis-
>dphi6_eta[i][gpt];
   dx_xsi+=mymesh->getNodeLocX(mymesh->nop[e][i])*basis-
>ddphi6_xsi[i][gpt];
   dx_eta+=mymesh->getNodeLocX(mymesh->nop[e][i])*basis-
>ddphi6_eta[i][gpt];
   dy_xsi+=mymesh->getNodeLocY(mymesh->nop[e][i])*basis-
>ddphi6_xsi[i][gpt];
   dy_eta+=mymesh->getNodeLocY(mymesh->nop[e][i])*basis-
>ddphi6_eta[i][gpt];
  }

for (int i=0; i<(basis->NPpE); i++){
    pold+=sol_old[(3*mymesh->NumNodes)+mymesh->pnop[mymesh->
nop[e][i]]]*basis->phi3[i][gpt];
    pnew+=sol[(3*mymesh->NumNodes)+mymesh->pnop[mymesh->nop[e][i]]]*
basis->phi3[i][gpt];
  }

  detJ = (x_xsi*y_eta)-(x_eta*y_xsi);

  for (int i=0; i<(basis->NpE); i++){
    phi_x[i]=((y_eta*basis->dphi6_xsi[i][gpt])-(y_xsi*basis-
>dphi6_eta[i][gpt]))/detJ;
    phi_y[i]=((x_xsi*basis->dphi6_eta[i][gpt])-(x_eta*basis-
>dphi6_xsi[i][gpt]))/detJ;
    dphi_x[i]=((dy_eta*basis->ddphi6_xsi[i][gpt])-(dy_xsi*basis-
>ddphi6_eta[i][gpt]))/detJ;
    dphi_y[i]=((dx_xsi*basis->ddphi6_eta[i][gpt])-(dx_eta*basis-
>ddphi6_xsi[i][gpt]))/detJ;
  }

  for (int i=0; i<(basis->NpE); i++){
    uold_x += sol_old[mymesh->nop[e][i]] * phi_x[i];
```

```
    uold_y += sol_old[mymesh->nop[e][i]] * phi_y[i];
    u_x += sol[mymesh->nop[e][i]] * phi_x[i];//unew_x
    u_y += sol[mymesh->nop[e][i]] * phi_y[i];
    vold_x += sol_old[mymesh->NumNodes+mymesh->nop[e][i]] * phi_x[i];
    vold_y += sol_old[mymesh->NumNodes+mymesh->nop[e][i]] * phi_y[i];
    v_x += sol[mymesh->NumNodes+mymesh->nop[e][i]] * phi_x[i];//vnew_x
    v_y += sol[mymesh->NumNodes+mymesh->nop[e][i]] * phi_y[i];
    cold_x += sol_old[2*mymesh->NumNodes+mymesh->nop[e][i]] * phi_x[i];
    cold_y += sol_old[2*mymesh->NumNodes+mymesh->nop[e][i]] * phi_y[i];
    c_x += sol[2*mymesh->NumNodes+mymesh->nop[e][i]] * phi_x[i];//cnew_x
    c_y += sol[2*mymesh->NumNodes+mymesh->nop[e][i]] * phi_y[i];
    c_xx += sol[2*mymesh->NumNodes+mymesh->nop[e][i]] * dphi_x[i];
    c_yy += sol[2*mymesh->NumNodes+mymesh->nop[e][i]] * dphi_y[i];

        // x-component of N.S.
for (int i=0; i<(basis->NpE); i++){
      for (int j=0; j<(basis->NpE); j++){
            A_local[i][j]+=(basis->phi6[i][gpt]*basis->phi6[j][gpt]+
            dt*(1.0*basis->phi6[j][gpt]*u_x*basis->phi6[i][gpt]
            +1.0*unew*phi_x[j]*basis->phi6[i][gpt]
            +1.0*vnew*phi_y[j]*basis->phi6[i][gpt]+
            (1.0/Re)*(2.0*phi_x[i]*phi_x[j]+phi_y[i]*phi_y[j])))*detJ*
            basis->gw[gpt];
      }
      for (int j=0; j<(basis->NpE); j++){
            A_local[i][basis->NpE+j]+=dt*(1.0*basis-
            >phi6[j][gpt]*u_y*basis->phi6[i][gpt]+(1.0/Re)*phi_x[j]*
            phi_y[i])*detJ*basis->gw[gpt];
      }
      for (int j=0; j<(basis->NpE); j++){
            A_local[i][2*basis->NpE+j]+= 0.0*detJ*basis->gw[gpt];
      }
      for (int j=0; j<(basis->NPpE); j++){
            A_local[i][3*basis->NpE+j]+=-dt*(basis-
            >phi3[j][gpt]*phi_x[i])*detJ*basis->gw[gpt];
      }
      b_local[i]+=(1.0*unew*basis->phi6[i][gpt]-1.0*uold*basis-
      >phi6[i][gpt]+dt*(1.0*unew*u_x*basis->phi6[i][gpt]+
      1.0*vnew*u_y*basis->phi6[i][gpt]+(1.0/Re)*(2.0*u_x*phi_x[i]+
      u_y*phi_y[i]+v_x*phi_y[i])-pnew*phi_x[i]))*detJ*basis->gw[gpt];
}

// y-component of N.S.
for (int i=0; i<(basis->NpE); i++){
      for (int j=0; j<(basis->NpE); j++){
            A_local[basis->NpE+i][j]+=dt*(1.0*basis-
            >phi6[j][gpt]*v_x*basis->phi6[i][gpt]+(1.0/Re)*phi_y[j]*
            phi_x[i])*detJ*basis->gw[gpt];
      }
      for (int j=0; j<(basis->NpE); j++){
            A_local[basis->NpE+i][basis->NpE+j]+=(1.0*basis-
            >phi6[i][gpt]*basis->phi6[j][gpt]+
            dt*(1.0*unew*phi_x[j]*basis->phi6[i][gpt]+1.0*basis-
            >phi6[j][gpt]*v_y*basis->phi6[i][gpt]+
```

```
            1.0*vnew*phi_y[j]*basis->phi6[i][gpt]+
            (1.0/Re)*(phi_x[i]*phi_x[j]+2.0*phi_y[i]*phi_y[j])))*detJ*b
            asis->gw[gpt];
      }
      for (int j=0; j<(basis->NpE); j++){
            A_local[basis->NpE+i][2*basis->NpE+j]+=0.0*detJ*basis->
            gw[gpt];
      }
      for (int j=0; j<(basis->NPpE); j++){
            A_local[basis->NpE+i][3*basis->NpE+j]+=-dt*(basis->
            phi3[j][gpt]*phi_y[i])*detJ*basis->gw[gpt];
      }
      b_local[basis->NpE+i]+=(1.0*vnew*basis->phi6[i][gpt]-1.0*vold*
      basis->phi6[i][gpt]+dt*(1.0*unew*v_x*basis->phi6[i][gpt]+
      1.0*vnew*v_y*basis->phi6[i][gpt]+(1.0/Re)*(u_y*phi_x[i]+v_x*
      phi_x[i]+2.0*v_y*phi_y[i])-pnew*phi_y[i]))*detJ*basis->gw[gpt];
}

//c-component//
for (int i=0; i<(basis->NpE); i++){
      for (int j=0; j<(basis->NpE); j++){
            A_local[2*basis->NpE+i][j]+=dt*(basis->
            phi6[i][gpt]*c_x*basis->phi6[j][gpt]+du_new)*detJ*basis->
            gw[gpt];
      }
      for (int j=0; j<(basis->NpE); j++){
            A_local[2*basis->NpE+i][basis->NpE+j]+=dt*(basis->
            phi6[i][gpt]*c_y*basis->phi6[j][gpt]+dv_new)*detJ*basis->
            gw[gpt];
      }
      for (int j=0; j<(basis->NpE); j++){
            A_local[2*basis->NpE+i][2*basis->NpE+j]+=((basis->
            phi6[i][gpt]*basis->phi6[j][gpt])+dt*(basis->
            phi6[i][gpt]*(unew*phi_x[j]+vnew*phi_y[j])+
            (D*phi_x[i]*phi_x[j]+D*phi_y[i]*phi_y[j])+dc_new))*detJ*
            basis->gw[gpt];
      }
      for (int j=0; j<(basis->NPpE); j++){
            A_local[2*basis->NpE+i][3*basis->NpE+j]+=0.0*detJ*basis->
            gw[gpt];
      }
      b_local[2*basis->NpE+i]+=((basis->phi6[i][gpt]*cnew-basis->
      phi6[i][gpt]*cold)+dt*((unew*c_x*basis->
      phi6[i][gpt]+vnew*c_y*basis->phi6[i][gpt])+(D*phi_x[i]*c_x+
      D*phi_y[i]*c_y)+(i_diff*(phi_x[i]*unew+phi_y[i]*vnew))*(((cnew-
      cold)/dt+(unew*c_x + vnew*c_y)))))*detJ*basis->gw[gpt];
}

// p-component of N.S.
for (int i=0; i<(basis->NPpE); i++){
      for (int j=0; j<(basis->NpE); j++){
            A_local[3*basis->NpE+i][j]+=(phi_x[j]*basis->
            phi3[i][gpt])*detJ*basis->gw[gpt];
      }
```

```
      for (int j=0; j<(basis->NpE); j++){
            A_local[3*basis->NpE+i][basis->NpE+j]+=(phi_y[j]*basis->
            phi3[i][gpt])*detJ*basis->gw[gpt];
      }
      for (int j=0; j<(basis->NpE); j++){
            A_local[3*basis->NpE+i][2*basis->NpE+j]+=0.0*detJ*basis->
            gw[gpt];
      }
      for (int j=0; j<(basis->NPpE); j++){
            A_local[3*basis->NpE+i][3*basis->NpE+j]+=0.0*detJ*basis->
            gw[gpt];
      }
      b_local[3*basis->NpE+i]+=(u_x*basis->phi3[i][gpt]+v_y*basis->
      phi3[i][gpt])*detJ*basis->gw[gpt];
   }
}
```

Assemble is the function used to fill the matrix and residual and where the boundary conditions are set. The case() statement indicates where the boundary conditions are set.

```
cbc=sin(pi*t);

if (cbc<0.0){
  cbc=0.0;
}

for (j=0; j<(mymesh->NpE); j++){
  dof = 2*mymesh->NumNodes + mymesh->nop[e][j];
  x = mymesh->getNodeLocX(dof-2*mymesh->NumNodes);
  y = mymesh->getNodeLocY(dof-2*mymesh->NumNodes);
  switch(mymesh->BCType[mymesh->nop[e][j]]){
    case 1:
      NumEntries = 1;
      vals[0] = 1.0;
      cols[0] = dof;
      A->ReplaceMyValues(dof, NumEntries, vals, cols);
      vals[0] = sol[dof]-0.0;
      b->ReplaceMyValues(NumEntries, vals, &dof);
    break;
    case 3:
    case 2:
      NumEntries = 1;
      vals[0] = 1.0;
      cols[0] = dof;
      A->ReplaceMyValues(dof, NumEntries, vals, cols);
      vals[0] = sol[dof]-0.0;
      b->ReplaceMyValues(NumEntries, vals, &dof);
    break;
```

```
    case 4:
      NumEntries = 1;
      vals[0] = 1.0;
      cols[0] = dof;
      A->ReplaceMyValues(dof, NumEntries, vals, cols);
      vals[0] = sol[dof]-cbc;
      b->ReplaceMyValues(NumEntries, vals, &dof);
      break;
      default:

      NumEntries = 3*mymesh->NpE;
      for (i=0; i < mymesh->NpE; i++){
      cols[i] = mymesh->nop[e][i];
        vals[i] = A_local[2*mymesh->NpE+j][i];
      }
      for (i=0; i < mymesh->NpE; i++){
      cols[mymesh->NpE+i] = mymesh->NumNodes + mymesh->nop[e][i];
      vals[mymesh->NpE+i] = A_local[2*mymesh->NpE+j][mymesh->NpE+i];
      }
      for (i=0; i < mymesh->NpE; i++){
      cols[2*mymesh->NpE+i] = 2*mymesh->NumNodes + mymesh->nop[e][i];
        vals[2*mymesh->NpE+i]    =    A_local[2*mymesh->NpE+j][2*mymesh-
>NpE+i];
      }
      A->SumIntoMyValues(dof, NumEntries, vals, cols);
      NumEntries = 1;
      vals[0] = b_local[2*mymesh->NpE+j];
      b->SumIntoMyValues(NumEntries, vals, &dof);
      break;
  }
}


delete[] cols;
delete[] vals;
return (err);
}
```

APPENDIX C

3D LSFEM IMPLEMENTATION IN C

Data Assimilation Model Implementation in C

This appendix is a summary of the C code that is used for the data assimilation model. The important parts of the code will be discussed in detail. The main function is call parafos, which includes all the prototypes, as shown below. There are 5 prototypes in this model: the first prototypes is read_mesh, which is the function that reads in the input mesh data. The set_bc_verde function is where the boundary condtions are applied to the matrix and vector. While the read_vort function reads in the vorticity solution provided by from the code shown in Appendix A. Solve is the function that assemble the matrix and vector and solves the matrix problem. The final function is called sol_mesh, which is used to write the results of the model.

```
void read_mesh(INFO *,double **,int **,int **,int ***,int **,int **,int
***,double **,double **,double **,double **,double **);
void set_bc_verde(INFO,double *,int **,int *,int **,double **);
void read_vort(INFO,double *,double *,int **,double **,double **,double
*,int *,int *);
void solve(INFO,double *,int **,int **,int *,int *,int *,int **,int **,
int *,int *,double *,double *,double *,double *,double *,double *,double
*,double *,double *,double *,double *,double *,double *,double *,double
*,double *,double *);
void sol_mesh(INFO,double *,double *, int **);
```

```
  /* Opening mesh file */
  if((mesh_file = fopen(info->mesh,"r"))==NULL){
      printf("error opening mesh file on %d\n",info->myid);
      MPI_Abort(MPI_COMM_WORLD,1);
}

 /* Reading header */
count=fscanf(mesh_file,"%s     %s     %d     %d",buff,buff,&(info->type),
&(info->face_type));
count+=fscanf(mesh_file,"%s    %s    %d    %d",buff,buff,&(info->nodes),
&(info->t_nodes));
count+=fscanf(mesh_file,"%s    %s    %d    %d",buff,buff,&(info->elem),
&(info->t_elem));
count+=fscanf(mesh_file,"%s      %s      %d      %d      %d",buff,buff,
&(info->face_elem),&(info->t_face_elem), &(info->t_face));
count += fscanf(mesh_file,"%s %s %d",buff,buff,&(info->pivSteps));

if(count != 20){
     printf("ERROR reading mesh file header\n");
     MPI_Abort(MPI_COMM_WORLD,1);
}

if(!(*oldsol)){
      printf("error allocating oldsol\n");
      MPI_Abort(MPI_COMM_WORLD,1);
}

for(i=0;i<info->t_elem;i++)
     (*map)[i] = (int *)calloc (info->type, sizeof(int));

for(i=0;i<info->t_face;i++)
     (*face_map)[i] = (int *)calloc (info->face_type, sizeof(int));

k = 0;
object = -1;

/* Reading file information */
while(fgets(line_buff,256,mesh_file)) {
   if ((line_buff[0]!='#') && (line_buff[0]!='\n')){
      sscanf(line_buff,"%s %d", buff, &object);

      if((object == 0) && !(strcmp(buff,"object"))){
      /* reading nodes positions */
for(i=0;i<(info->t_nodes);i++){
length=fscanf(mesh_file,"%le%le%le",&((*mesh)[i*3]),&((*mesh)[i*3+1]),
&((*mesh)[i*3+2]));
 }
      object = -1;
      }
      if((object == 1) && !(strcmp(buff,"object"))){
      /* reading node data */
      for(i=0;i<(info->t_nodes);i++){
```

```
    length=fscanf(mesh_file,"%d %d",&((*glob)[i]),&((*proc)[i]));
    if(length != 2)
       printf("error reading node data on proc %d\n",info->myid);
       if((*proc)[i] >= info->numproc){
         printf("error, few processors than specified by mesh!!\n");
         MPI_Abort(MPI_COMM_WORLD,0);
       }
 }
   object = -1;
 }

if((object == 3) && !(strcmp(buff,"object"))){
 /* reading element map */
for(i=0;i<(info->t_elem);i++){
   length=0;
   for(j=0;j<10;j++){
      length += fscanf(mesh_file,"%d",&((*map)[i][j]));
   }
   if(length != 10)
     printf("error reading element data on proc %d\n",info->myid);
     }
     object = -1;
}

if((object == 4) && !(strcmp(buff,"object"))){
    /* reading hoe element map */
   for(i=0;i<(info->t_elem);i++){
      length=0;
      for(j=0;j<19;j++){
         length += fscanf(mesh_file,"%d",&((*map)[i][j+8]));
      }
      if(length != 19)
       printf("error reading element data on proc %d\n",info->myid);
   }
      object = -1;
}



  if((object == 5) && !(strcmp(buff,"object"))){
  /* reading face element map */
     for(i=0;i<(info->t_face);i++){
        count = fscanf(mesh_file,"%d",&((*boundary)[i]));
        count += fscanf(mesh_file,"%d",&((*interior)[i]));
        length=0;
        for(j=0;j<info->face_type;j++){
           length+=fscanf(mesh_file,"%d",&((*face_map)[i][j]));
         }
       }
       object = -1;
     }
  }
 }
```

The code above is called the read_mesh. The function will read in the mesh file and organize the data from the file, the input file must be written in ASCI. The function has the capability of reading in mesh that have been partition using METIS. The mesh file that is being read in contains the information regarding the coordinates, element connectity, face map connectivity and boundary information. This informations which are read in are stored either in a 1 dimensional array or 2 demensional array.

```
if((!(*essential)) || (!(*is_essential))){
    printf("error allocating essential\n");
    MPI_Abort(MPI_COMM_WORLD,1);
}

    for(fe=0;fe<info.t_face;fe++){
      for(i=0;i<info.face_type;i++){
      switch(boundary[fe]){
/* INLET SIDE */
        case 1:
          dof=face_nop(fe,i,0);
          x=mesh[facemap(fe,i)*3];
          y=mesh[facemap(fe,i)*3+1];
          z=mesh[facemap(fe,i)*3+2];

          if(info.Tstep >= 60){
           (*is_essential)[dof]=1;
            (*essential)[dof]=0.0;
            (*is_essential)[dof+1]=1;
            (*essential)[dof+1]=0.0;
            (*is_essential)[dof+2]=1;
            (*essential)[dof+2]=0.0;
          }

        break;

        case 2://outflow
          dof=face_nop(fe,i,0);
          x=mesh[facemap(fe,i)*3];
          y=mesh[facemap(fe,i)*3+1];
          z=mesh[facemap(fe,i)*3+2];

         if(info.Tstep <= 74){
          (*is_essential)[dof]=1;
          (*essential)[dof]=0.0;
          (*is_essential)[dof+1]=1;
          (*essential)[dof+1]=0.0;
          (*is_essential)[dof+2]=1;
          (*essential)[dof+2]=0.0;
         }

        if((info.Tstep >= 121) && (info.Tstep < 134)){
           (*is_essential)[dof]=1;
          (*essential)[dof]=0.0;
          (*is_essential)[dof+1]=1;
          (*essential)[dof+1]=0.0;
          (*is_essential)[dof+2]=1;
          (*essential)[dof+2]=0.0;
        }

        if(info.Tstep == 134){
          (*is_essential)[dof]=1;
          (*essential)[dof]=0.0;
```

```
          (*is_essential)[dof+1]=1;
          (*essential)[dof+1]=0.0;
          (*is_essential)[dof+2]=1;
          (*essential)[dof+2]=0.0;
        }
        break;

/* TOP SIDE */
        case 3:
          dof=face_nop(fe,i,0);
          x=mesh[facemap(fe,i)*3];
          y=mesh[facemap(fe,i)*3+1];
          z=mesh[facemap(fe,i)*3+2];

          (*is_essential)[dof]=1;
          (*essential)[dof]=0.0;
          (*is_essential)[dof+1]=1;
          (*essential)[dof+1]=0.0;
          (*is_essential)[dof+2]=1;
          (*essential)[dof+2]=0.0;

        break;

/* WALL AND BOTTOM SIDE  */
         case 4:
         case 5:
         dof=face_nop(fe,i,0);
         x=mesh[facemap(fe,i)*3];
         y=mesh[facemap(fe,i)*3+1];
         z=mesh[facemap(fe,i)*3+2];

         if((y >= -0.1-EPS)){//here
          (*is_essential)[dof]=1;
          (*essential)[dof]=0.0;
            (*is_essential)[dof+1]=1;
          (*essential)[dof+1]= 0.0;
            (*is_essential)[dof+2]=1;
          (*essential)[dof+2]=0.0;
           }else{
            radi=(x*x + y*y + z*z);
              rad=sqrt(radi);
              phia=sqrt(x*x+z*z);
              phi=atan2(y,phia);
              theta=atan2(x,z);
             if((info.Tstep >= 0) && ( info.Tstep <= 59)){
             if(info.Tstep <= 5){
               drad_in= FRAC/5.0;
             }
             if(info.Tstep >= 6){
               drad_in = FRAC;
             }

                        newr=rad+drad_in;
                        newx=newr*cos(phi)*sin(theta);
```

```
                newy=newr*sin(phi);
                newz=newr*cos(phi)*cos(theta);

           }

              if((info.Tstep >= 75) && ( info.Tstep <=120)){
            if(info.Tstep <= 80){
              drad=FRAC/5.0;
            }

            if(info.Tstep > 80){
              drad=FRAC;
            }

            newr=rad-drad;
            newx=newr*cos(phi)*sin(theta);
            newy=newr*sin(phi);
            newz=newr*cos(phi)*cos(theta);
          }


        if((info.Tstep >= 121) && (info.Tstep < 134)){
          newr=rad;
          newx=newr*cos(phi)*sin(theta);
          newy=newr*sin(phi);
          newz=newr*cos(phi)*cos(theta);
        }

            if((info.Tstep >= 60) && (info.Tstep <= 74)){
          newr=rad;
              newx=newr*cos(phi)*sin(theta);
              newy=newr*sin(phi);
              newz=newr*cos(phi)*cos(theta);
            }

        if((info.Tstep == 134)){
          newr=rad;
              newx=newr*cos(phi)*sin(theta);
              newy=newr*sin(phi);
              newz=newr*cos(phi)*cos(theta);
        }

        (*is_essential)[dof+0]=1;
        (*essential)[dof+0] = (newx-x)/info.dt;
        (*is_essential)[dof+1]=1;
        (*essential)[dof+1] = (newy-y)/info.dt;
        (*is_essential)[dof+2]=1;
        (*essential)[dof+2] = (newz-z)/info.dt;
       }

    break;

}
```

The code above is used to impose the boundary condtion strongly. The boundaries are marked in the boundary array the specific case defined at each boundary. In order to capture the motion of the blood flow the boundary condtions are changed according to the time step. The case() statements are used to set the boundary conditions.

```
dof_elem=info.type*info.unkn;
Q=q*info.type;
R=r*info.type;
S=s*info.type;
x=gp_loc[0];
y=gp_loc[1];
z=gp_loc[2];
detJ = detJ * scaling * scaling;

if((fun_flag == TRUE) || (fun_flag == NO_FUN)){
   for(j=0;j<info.type;j++){
       for(i=0;i<=j;i++){
          A((Q+i),(Q+j)) += (phi_x[i]*phi_x[j])*detJ;
          A((Q+i),(R+j)) += (phi_x[i]*phi_y[j])*detJ;
          A((Q+i),(S+j)) += (phi_x[i]*phi_z[j])*detJ;

          A((R+i),(Q+j)) += (phi_y[i]*phi_x[j])*detJ;
          A((R+i),(R+j)) += (phi_y[i]*phi_y[j])*detJ;
          A((R+i),(S+j)) += (phi_y[i]*phi_z[j])*detJ;

          A((S+i),(Q+j)) += (phi_z[i]*phi_x[j])*detJ;
          A((S+i),(R+j)) += (phi_z[i]*phi_y[j])*detJ;
          A((S+i),(S+j)) += (phi_z[i]*phi_z[j])*detJ;
       }

b(Q+j)+=(unkn_x[q]*phi_x[j]+unkn_y[r]*phi_x[j]+unkn_z[s]*phi_x[j])*detJ
b(R+j)+=(unkn_x[q]*phi_y[j]+unkn_y[r]*phi_y[j]+unkn_z[s]*phi_y[j])*detJ
b(S+j)+=(unkn_x[q]*phi_z[j]+unkn_y[r]*phi_z[j]+unkn_z[s]*phi_z[j])*detJ
}
      }

if((fun_flag == TRUE) || (fun_flag == NO_MAT)){
    fun[0] += (pow((unkn_x[q]+unkn_y[r]+unkn_z[s]),2))*detJ;
}
```

The code above builds the weak form for the divergence of velocity. The unkn() are the variables that need to be solved, while phi () are the test function. The fun[0] calculates the $L_2 - norm$ of the eqaution, which indicates how well the eqaution is stratified.

```
detJ = detJ * scaling * scaling;
if((fun_flag == TRUE) || (fun_flag == NO_FUN)){
   dof_elem=info.type*info.unkn;
   Q=q*info.type;
   R=r*info.type;
   S=s*info.type;

   for(j=0;j<info.type;j++){
      for(i=0;i<=j;i++){
         A((Q+i),(Q+j)) += (phi_y[i]*phi_y[j])*detJ;
         A((Q+i),(R+j)) -= (phi_y[i]*phi_x[j])*detJ;
         A((R+i),(Q+j)) -= (phi_x[i]*phi_y[j])*detJ;
         A((R+i),(R+j)) += (phi_x[i]*phi_x[j])*detJ;
         A((R+i),(R+j)) += (phi_z[i]*phi_z[j])*detJ;
         A((R+i),(S+j)) -= (phi_z[i]*phi_y[j])*detJ;
         A((S+i),(R+j)) -= (phi_y[i]*phi_z[j])*detJ;
         A((S+i),(S+j)) += (phi_y[i]*phi_y[j])*detJ;
         A((Q+i),(Q+j)) += (phi_z[i]*phi_z[j])*detJ;
         A((Q+i),(S+j)) -= (phi_z[i]*phi_x[j])*detJ;
         A((S+i),(Q+j)) -= (phi_x[i]*phi_z[j])*detJ;
         A((S+i),(S+j)) += (phi_x[i]*phi_x[j])*detJ;
      }

      b(Q+j)+=(unkn_y[q]*phi_y[j]-unkn_x[r]*phi_y[j]+piv[s]*phi_y[j])*
      detJ;
      b(Q+j)+=(unkn_z[q]*phi_z[j]-unkn_x[s]*phi_z[j]-piv[r]*phi_z[j])*
      detJ;
      b(R+j)+=(unkn_x[r]*phi_x[j]-unkn_y[q]*phi_x[j]-piv[s]*phi_x[j])*
      detJ;
      b(R+j)+=(unkn_z[r]*phi_z[j]-unkn_y[s]*phi_z[j]+piv[q]*phi_z[j])*
      detJ;
      b(S+j)+=(unkn_x[s]*phi_x[j]-unkn_z[q]*phi_x[j]+piv[r]*phi_x[j])*
      detJ;
      b(S+j)+=(unkn_y[s]*phi_y[j]-unkn_z[r]*phi_y[j]-piv[q]*phi_y[j])*
      detJ;
      }
 }

 if((fun_flag == TRUE) || (fun_flag == NO_MAT)){
    fun[2] += (pow((unkn_x[r]-unkn_y[q]-piv[s]),2))*detJ;
    fun[1] += (pow((unkn_z[q]-unkn_x[s]-piv[r]),2))*detJ;
    fun[0] += (pow((unkn_y[s]-unkn_z[r]-piv[q]),2))*detJ;
}
```

The code above implements the weak for the curl of velocity equation. Where piv[] is the know vorticity vector calculated from the model shown in appendix A.

```
 for(info.Tstep=0; info.Tstep<info.Niter; info.Tstep++){
    setup_comm(&info, map, proc, glob, &sends, &recvs);
    info.time = info.time + info.dt;
    readPIVdata(&info, &veldata);

 if(!strcmp(info.prob,"verde"))
    set_bc_verde(info,mesh,face_map,boundary,&is_essential,&essential);

 info.dof=build_gdof(&info, proc, map, is_essential, &gdof);
 get_cols(info, glob, recvs, sends, gdof);

read_vort(info,mesh,vel_sol,map,&data,&vel_data,dist_sol,para2fenics,
fenics2para);

solve(info,mesh,map,face_map,gdof,glob,proc,recvs,sends,boundary,
interior,gw2,gw3,phi,phi2,phi_1,phi_2,phi_3,phi2_1,phi2_2,essential,
sol,veldata,vel_sol,dist_sol);
info.init = 1;
sol_mesh(info, mesh, sol, map);

   }
```

The code above implements the time loop in the main parafos code and the readPIVdata read in the experimental echo-PIV at each time step. After this step we impose the strong boundary condtion in the function called set_bc_verde. Since the numerical model is solved by requiring the vorticity at each time step the function read_vort provides the known vorticity. The solve function is where the matrix and right hand side are assembled and solved using hypre. After ending the iteration loop we write the solution to an Exodus file using the function sol_mesh.

In solve (), the main steps are assembleing the matrix and right hand side from the weak weak form defined ealier, and these steps are shown below. The function below also shows the implementation of weak boundary condtions (U_tang) and the assimilation of echo-PIV data weakly in Uset_PIV.

```
/* Loop for a set number of iterations.  */
for(niter=0;niter<info.max_iter;niter++){
   initialize_global(info,niter,gdof,map,fun,face_fun,&(matrix),&(rhs),
   &(dsol));

   for(e=0;e<info.t_elem;e++){
      initialize_local(dof_elem, localA, localb);
      for(gpt=0;gpt<info.num_gp;gpt++){
         phi_primes(info,e,gpt,map,mesh,phi,phi_1,phi_2,phi_3,phi_x,
         phi_y,phi_z,&detJ,gp_loc);
         calc_U_at_gp(info,gpt,e,map,phi,phi_x,phi_y,phi_z,unkn,unkn_x,
         unkn_y,unkn_z,sol);
         calc_U_at_gp(info,gpt,e,map,phi,phi_x,phi_y,phi_z,sup,sup_x,
         sup_y,sup_z,vel_sol);
         calc_motion(info, gpt, e, map, phi, motion, sol_move);

         detJ=detJ * gw3[gpt];
         if(e < info.elem)
           fun_flag = TRUE;
         else
           fun_flag = NO_FUN;

    if(!strcmp(info.prob,"verde")){
      curlU_piv(info,0,1,2,gpt,gp_loc,phi_x,phi_y,phi_z,unkn_x,unkn_y,
      unkn_z,sup,sup_x,sup_y,sup_z,localA,localb,detJ,&(fun[0]),info.cu
      rl_sc,fun_flag);
      divU(info,0,1,2,gpt,gp_loc,phi_x,phi_y,phi_z,unkn_x,unkn_y,unkn_z
      ,localA,localb,detJ,&(fun[3]),info.div_sc,fun_flag);
      }
   } /* finished with gp */

build_res(info, e, localb, gdof, map, mesh,essential, sol, &rhs);
build_matrix(info, e, localA, gdof, map, mesh, essential, sol, &matrix);
} /* Finished with all interior elements */

for(e=0;e<info.t_face_elem;e++){
    initialize_local(dof_elem, localA, localb);
     face = boundary[e];
     for(gpt=0;gpt<gp_face;gpt++){
        calc_U_on_face(info, gpt, e, face_map, phi2, set_val, essential);
}
info.h=sqrt(pow(mesh[facemap(e,0)*3]-mesh[facemap(e,1)*3],2)+
pow(mesh[facemap(e,0)*3+1]-mesh[facemap(e,1)*3+1],2)+
pow(mesh[facemap(e,0)*3+2]-mesh[facemap(e,1)*3+2],2));

for(gpt=0;gpt<gp_face;gpt++){
   phi2_primes(info,e,gpt,face_map,mesh,phi2,phi2_1,phi2_2,tang1,tang2,
   norm,&detJ,gp_loc);
   calc_U_on_face(info, gpt, e, face_map, phi2, unkn, sol);
   calc_U_on_face(info, gpt, e, face_map, phi2, set_val, essential);
   detJ=detJ * gw2[gpt];
```

```
if(!strcmp(info.prob,"verde")){
  if((face == 1)){
   if(info.Tstep <= 59){

      U_tang(info,0,1,2,face,gpt,0.0,gp_loc,phi2,unkn,tang1,localA,
      localb,detJ,&(face_fun[0]),fun_flag);
      U_tang(info,0,1,2,face,gpt,0.0,gp_loc,phi2,unkn,tang2,localA,
      localb, detJ, &(face_fun[0]), fun_flag);
   }
}

   if((face == 2)){
    if(info.Tstep >= 75){
      U_tang(info,0,1,2,face,gpt,0.0,gp_loc,phi2,unkn,tang1,localA,
      localb,detJ,&(face_fun[0]),fun_flag);
      U_tang(info,0,1,2,face,gpt,0.0,gp_loc,phi2,unkn,tang2,localA,
      localb,detJ,&(face_fun[0]),fun_flag);
    }
   }

   if((face == 4) || (face == 5)){

      U_tang(info,0,1,2,face,gpt,0.0,gp_loc,phi2,unkn,tang1,localA,
      localb,detJ,&(face_fun[0]),fun_flag);
      U_tang(info,0,1,2,face,gpt,0.0,gp_loc,phi2,unkn,tang2,localA,
      localb,detJ,&(face_fun[0]), fun_flag);
   }

   if(face == 6){

      Uset_PIV(info,2,1,face,gpt,e,face_map,mesh,veldata,sol,gp_loc,
      phi2,unkn,localA,localb,detJ,&(face_fun[1]),fun_flag);
   }
 }
} /* finished with face gp */

build_face_res(info,e,localb,gdof,face_map,mesh,sol,&rhs);
build_face_mat(info,e,localA,gdof,face_map,mesh,sol,essential,&matrix);
} /* Finished with all face elements */
```

The functiocan below show the implemtation of Uset_PIV for data assimilation. The variable tau below is the weight used for a particular time step calculated based on the quality of the data. The function minimizes the distance between the numerical model and the data. The varabiles ux and uy are the 2-demensional vecocity data at the echo-PIV location.

```
if((info.Tstep >= 0) && ( info.Tstep <= 14)){
   tau=0.0673772382;
}
if((info.Tstep >= 15) && ( info.Tstep <= 29)){
   tau=0.0555922669;
}
if((info.Tstep >= 30) && ( info.Tstep <= 44)){
   tau=0.0540069083;
}
if((info.Tstep >= 45) && ( info.Tstep <= 59)){
   tau=0.0524670354;
}
if((info.Tstep >= 60) && ( info.Tstep <= 74)){
   tau=0.0396147743;
}
if((info.Tstep >= 75) && ( info.Tstep <= 89)){
   tau=0.0502846852;
}
if((info.Tstep >= 90) && ( info.Tstep <= 104)){
   tau=0.0472631769;
}
if((info.Tstep >= 105) && ( info.Tstep <= 119)){
   tau=0.0507286866;
}
if((info.Tstep >= 120) && ( info.Tstep <= 134)){
   tau=0.057006382;
}
detJ = (tau*detJ) / info.h;
dof_face=info.face_type*info.unkn;
P=p*info.face_type;
Q=q*info.face_type;
maxX = -100.0; maxY = -100.0; minX = 100.0; minY = 100.0;

for(i=0; i<info.face_type; i++){//face
   facex[i] = mesh[face_map[fe][i] * 3 + p];
   if(facex[i] > maxX)
     maxX = facex[i];
   if(facex[i] < minX)
     minX = facex[i];
   facey[i] = mesh[face_map[fe][i] * 3 + q];
   if(facey[i] > maxY)
     maxY = facey[i];
   if(facey[i] < minY)
     minY = facey[i];
}
if(info.Tstep <= 74){
  del_x=0.3;
  del_y=0.5;
  }else{
   del_x=-0.3;
   del_y=0.0;
}
```

```
for(pt = 0; pt < info.pivPoints; pt++){
   ptx_inv = veldata[pt*5] -600.0;
   ptx = ((ptx_inv + XMID) / XSIZE)+del_x;
   pty = ((veldata[pt*5+1] - YMID) / YSIZE)+del_y;
   pivUx = (veldata[pt*5+2]) / VSCALE;
   pivUy = (veldata[pt*5+3]) / VSCALE;

   if((ptx <= maxX) && (ptx >= minX) && ( pty <= maxY) && (pty >= minY)){
     for(i=0; i<info.face_type; i++){
        solx[i] = sol[face_nop(fe,i,p)];
        soly[i] = sol[face_nop(fe,i,q)];
     }

     inside=mapping(info,facex,facey,solx,soly,deriv,ptx,pty,&ux,&uy);
     if(inside == TRUE)
       pivMag = pow((pivUx*pivUx + pivUy*pivUy),0.5);
       if((fun_flag == TRUE) || (fun_flag == NO_FUN)){
       for(j=0;j<info.face_type;j++){
          for(i=0;i<=j;i++){
             faceA((P+i),(P+j))+=(deriv[i]*phi2(j,gpt))*detJ*pivMag;
             faceA((Q+i),(Q+j))+=(deriv[i]*phi2(j,gpt))*detJ*pivMag;
          }
          faceb(P+j)+=(ux*phi2(j,gpt)-pivUx*phi2(j,gpt))*detJ*pivMag;
          faceb(Q+j)+=(uy*phi2(j,gpt)-pivUy*phi2(j,gpt))*detJ*pivMag;
       }
      }

      if((fun_flag == TRUE) || (fun_flag == NO_MAT)){
        fun[0] += (pow((ux-pivUx),2))*detJ*pivMag;
        fun[1] += (pow((uy-pivUy),2))*detJ*pivMag;
      }
    }
 }
```

The section below implements the output file of the final solution, which is written in Exodus file format. Further information regarding impmentation of Exodus file format can be found in the Exodus documentation [].

```
for(e=0; e<info.t_elem;e++){
   for(i=0; i<info.type;i++){
      connect_tmp[e*info.type+i]= map[e][i]+1;
   }
}

//Open the output file
numc=sprintf(filename,"./output_mesh/Exout_%d_%05d.e",info.myid,
info.Tstep);
ex_id = ex_create(filename, EX_CLOBBER, &comp_ws, &io_ws);
ex_id = ex_open(filename, EX_WRITE, &comp_ws, &io_ws, &exodus_version);

//Write the intialization
numc = sprintf(title, "Output from LS-FEM");
ex_err=ex_put_init(ex_id, title, 3, info.t_nodes, info.t_elem, 1, 0, 0);

//Write Element Block
ex_err=ex_put_elem_block(ex_id, 1, "TETRA10", info.t_elem, info.type, 0);
ex_err=ex_put_elem_conn(ex_id, 1, connect_tmp);


//Write DATA
ex_err = ex_put_var_param(ex_id, "N", num_nodal_fields);
numc = sprintf(var_names[0], "U_x");
numc = sprintf(var_names[1], "U_y");
numc = sprintf(var_names[2], "U_z");

for(i=0; i<info.t_nodes; i++){
   vert_nodeX[i] = mesh[3*i];
   vert_nodeY[i] = mesh[3*i+1];
   vert_nodeZ[i] = mesh[3*i+2];
}

for(i=0; i<info.t_nodes;i++){
   usol[i] = sol[i*info.unkn];
   vsol[i] = sol[i*info.unkn+1];
   wsol[i] = sol[i*info.unkn+2];
}

ex_err = ex_put_coord(ex_id, vert_nodeX, vert_nodeY, vert_nodeZ);
ex_err = ex_put_var_names(ex_id, "N", num_nodal_fields, var_names);
ex_err = ex_put_nodal_var (ex_id, 1, 1 , info.t_nodes, usol);
ex_err = ex_put_nodal_var (ex_id, 1, 2 , info.t_nodes, vsol);
ex_err = ex_put_nodal_var (ex_id, 1, 3 , info.t_nodes, wsol);

ex_close(ex_id);
```

```
//Open the intialization fenics output file
if(info.Tstep == 0){
  sprintf(movefilename, "./move_20000_int/move_0.csv");
  if((movefile = fopen(movefilename,"r"))==NULL){
    printf("error opening move file on %d\n",info.myid);
    MPI_Abort(MPI_COMM_WORLD,1);
  }

  length_move = fscanf(movefile,"%s",(tmp_move));
  if(length_move != 1){
      printf("ERROR reading header of the csv data file\n");
      MPI_Abort(MPI_COMM_WORLD, 1);
  }
    length_move=0;

  for(i=0;i< num_node;i++){

      length_move+=fscanf(movefile,"%200[^,],%200[^,],%s\n",string1,
      string2,string3);

      (*data)[i*num_nodal_fields] = atof(string1);
      (*data)[num_nodal_fields*i+1] = atof(string2);
      (*data)[num_nodal_fields*i+2] = atof(string3);
  }

  if(length_move != num_node*3){
    printf("Check mesh move CSV file format\n");
  }

for(i=0;i<info.t_nodes;i++){
  for(j=0;j<num_node;j++){
    p_x = mesh[i*3];
    p_y = mesh[i*3+1];
    p_z = mesh[i*3+2];
    f_x = (*data)[j*num_nodal_fields];
    f_y = (*data)[j*num_nodal_fields+1];
    f_z = (*data)[j*num_nodal_fields+2];
dist=pow(((p_x-f_x)*(p_x-f_x)+(p_y-f_y)*(p_y-f_y)+(p_z-f_z)*(p_z-
f_z)),0.5);

if(dist < 0.00001){
 fenics2para[j]=i;
 para2fenics[i]=j;
 }
}
}

fclose(movefile);

}
```

```
for(i=0;i<info.t_nodes;i++){
  para_renum[i] = para2fenics[i];
}

sprintf(NSfilename,"./ns_20000/femv_%d.csv",(info.Tstep));

if((NSfile =fopen(NSfilename,"r"))==NULL){
  printf("error opening fenics fem solution on %d\n",info.myid);
  MPI_Abort(MPI_COMM_WORLD,1);
}
length = fscanf(NSfile,"%s",(tmp));
length = 0;
for(i=0;i<num_node;i++){

length+=fscanf(NSfile,"%200[^,],%200[^,],%200[^,],%200[^,],%200[^,],%s\
n",string_u,string_v,string_w,string6,string7,string8);
(*vel_data)[i*6] = atof(string_u);
(*vel_data)[i*6+1] = atof(string_v);
(*vel_data)[i*6+2] = atof(string_w);

 (*vel_data)[i*6+3] = atof(string6);
 (*vel_data)[i*6+4] = atof(string7);
 (*vel_data)[i*6+5] = atof(string8);
}

for(e=0;e<info.t_elem;e++){
  for(i=0;i<4;i++){
    node = map[e][i];
    sol[3*node] = (*vel_data)[6*para_renum[node]];
    sol[3*node+1] = (*vel_data)[6*para_renum[node]+1];
    sol[3*node+2] = (*vel_data)[6*para_renum[node]+2];

    dist_sol[3*node] = (*vel_data)[6*para_renum[node]+3];
    dist_sol[3*node+1] = (*vel_data)[6*para_renum[node]+4];
    dist_sol[3*node+2] = (*vel_data)[6*para_renum[node]+5];
}

sol[3*(map[e][4])] = (sol[3*(map[e][1])] + sol[3*(map[e][0])])/2.0;
sol[3*(map[e][5])] = (sol[3*(map[e][2])] + sol[3*(map[e][1])])/2.0;
sol[3*(map[e][6])] = (sol[3*(map[e][0])] + sol[3*(map[e][2])])/2.0;
sol[3*(map[e][7])] = (sol[3*(map[e][3])] + sol[3*(map[e][0])])/2.0;
sol[3*(map[e][8])] = (sol[3*(map[e][1])] + sol[3*(map[e][3])])/2.0;
sol[3*(map[e][9])] = (sol[3*(map[e][2])] + sol[3*(map[e][3])])/2.0;

sol[3*(map[e][4])+1]= (sol[3*(map[e][1])+1] + sol[3*(map[e][0])+1])/2.0;
sol[3*(map[e][5])+1]=(sol[3*(map[e][2])+1]+sol[3*(map[e][1])+1])/2.0;
sol[3*(map[e][6])+1]=(sol[3*(map[e][0])+1]+sol[3*(map[e][2])+1])/2.0;
sol[3*(map[e][7])+1]=(sol[3*(map[e][3])+1]+sol[3*(map[e][0])+1])/2.0;
sol[3*(map[e][8])+1]=(sol[3*(map[e][1])+1]+sol[3*(map[e][3])+1])/2.0;
sol[3*(map[e][9])+1]=(sol[3*(map[e][2])+1]+sol[3*(map[e][3])+1])/2.0;
sol[3*(map[e][4])+2]=(sol[3*(map[e][1])+2]+sol[3*(map[e][0])+2])/2.0;
sol[3*(map[e][5])+2]=(sol[3*(map[e][2])+2]+sol[3*(map[e][1])+2])/2.0;
sol[3*(map[e][6])+2]=(sol[3*(map[e][0])+2] + sol[3*(map[e][2])+2])/2.0;
sol[3*(map[e][7])+2]=(sol[3*(map[e][3])+2] + sol[3*(map[e][0])+2])/2.0;
```

```
sol[3*(map[e][8])+2]=(sol[3*(map[e][1])+2] + sol[3*(map[e][3])+2])/2.0;
sol[3*(map[e][9])+2]=(sol[3*(map[e][2])+2]+sol[3*(map[e][3])+2])/2.0;

dist_sol[3*(map[e][4])]=
(dist_sol[3*(map[e][1])]+dist_sol[3*(map[e][0])])/2.0;
dist_sol[3*(map[e][5])]=
(dist_sol[3*(map[e][2])]+dist_sol[3*(map[e][1])])/2.0;
dist_sol[3*(map[e][6])]=
(dist_sol[3*(map[e][0])]+dist_sol[3*(map[e][2])])/2.0;
dist_sol[3*(map[e][7])]=
(dist_sol[3*(map[e][3])]+dist_sol[3*(map[e][0])])/2.0;
dist_sol[3*(map[e][8])]=
(dist_sol[3*(map[e][1])]+ dist_sol[3*(map[e][3])])/2.0;
dist_sol[3*(map[e][9])]=
(dist_sol[3*(map[e][2])]+ dist_sol[3*(map[e][3])])/2.0;

dist_sol[3*(map[e][4])+1]=
(dist_sol[3*(map[e][1])+1]+ dist_sol[3*(map[e][0])+1])/2.0;
dist_sol[3*(map[e][5])+1]=
(dist_sol[3*(map[e][2])+1]+ dist_sol[3*(map[e][1])+1])/2.0;
dist_sol[3*(map[e][6])+1]=
(dist_sol[3*(map[e][0])+1]+ dist_sol[3*(map[e][2])+1])/2.0;
dist_sol[3*(map[e][7])+1]=
(dist_sol[3*(map[e][3])+1]+ dist_sol[3*(map[e][0])+1])/2.0;
dist_sol[3*(map[e][8])+1]=
(dist_sol[3*(map[e][1])+1]+ dist_sol[3*(map[e][3])+1])/2.0;
dist_sol[3*(map[e][9])+1]=
(dist_sol[3*(map[e][2])+1]+ dist_sol[3*(map[e][3])+1])/2.0;

dist_sol[3*(map[e][4])+2]=
(dist_sol[3*(map[e][1])+2]+dist_sol[3*(map[e][0])+2])/2.0;
dist_sol[3*(map[e][5])+2]=
(dist_sol[3*(map[e][2])+2]+ dist_sol[3*(map[e][1])+2])/2.0;
dist_sol[3*(map[e][6])+2]=
(dist_sol[3*(map[e][0])+2]+ dist_sol[3*(map[e][2])+2])/2.0;
dist_sol[3*(map[e][7])+2]=
(dist_sol[3*(map[e][3])+2]+ dist_sol[3*(map[e][0])+2])/2.0;
dist_sol[3*(map[e][8])+2]=
(dist_sol[3*(map[e][1])+2]+ dist_sol[3*(map[e][3])+2])/2.0;
dist_sol[3*(map[e][9])+2]=
(dist_sol[3*(map[e][2])+2]+ dist_sol[3*(map[e][3])+2])/2.0;
}

  fclose(NSfile);
```

The final section above reads in the solution provided form Appendix A where the vorticity and mesh motion are read in to two arrays sol [] and dist_sol []. The input file for this function is written in Comma SeparatedValues (CSV) format.