Summer 2013

Pressure Poisson Method for the Incompressible Navier-Stokes Equations Using Galerkin Finite Elements

John Cornthwaite
Georgia Southern University

PRESSURE POISSON METHOD FOR THE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS USING GALERKIN FINITE ELEMENTS

by

JOHN P. CORNTHWAITE

(Under the Direction of Shijun Zheng)

ABSTRACT

In this thesis we examine the Navier-Stokes equations (NSE) with the continuity equation replaced by a pressure Poisson equation (PPE). Appropriate boundary conditions are developed for the PPE, which allow for a fully decoupled numerical scheme to recover the pressure. The variational form of the NSE with PPE is derived and used in the Galerkin Finite Element discretization. The Galerkin finite element method is then used to solve the NSE with PPE. Moderate accuracy is shown.

INDEX WORDS: Thesis, Navier-Stokes, Pressure Poisson Equation, Galerkin Finite Element, Applied Mathematics, Partial Differential Equations

2010 *Mathematics Subject Classification*: 35Q30, 65M60

PRESSURE POISSON METHOD FOR THE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS USING GALERKIN FINITE ELEMENTS

by

JOHN P. CORNTHWAITE

B.A., Rice University, 2003

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment

of the Requirement for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

2013

PRESSURE POISSON METHOD FOR THE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS USING GALERKIN FINITE ELEMENTS

by

JOHN P. CORNTHWAITE

Major Professor: Shijun Zheng

Committee: Scott Kersey
Yan Wu
Cheng Zhang

Electronic Version Approved:

July, 2013

DEDICATION

This thesis is dedicated to my beautiful children Naomi and Quentin who inspire and motivate me.

ACKNOWLEDGMENTS

I would like to thank my advisors, Drs. Shijun Zheng and Cheng Zhang, who poured countless hours into my personal development. I express great appreciation towards my officemates who kept me motivated and on task. Thank you to those who take the time to make their knowledge readily available to the world while asking nothing in return; especially MIT OpenCourseware participants, iTunesU providers, participating universities of Coursera.com and other MOOCs. Their resources allowed me to fill-in background knowledge gaps quickly and competently. I am especially gratefull for the courses in fluid mechanics and computationsl fluid dynamics by Dr. Lorena Barba of Boston University. Finally, thank you to Dr. John Burkardt for making available code and notes from which to learn.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Fluid mechanics is a large and important area of science and engineering due to the many phenomena that fall under its umbrella. Computational fluid dynamics is an influential branch that leads to breakthroughs in designs and general understanding of how the world works. Modeling fluid behavior allows engineers to design the most advanced aircraft and doctors to learn how their drugs move through the human body. At the heart of fluid mechanics and its modeling are the fundamental equations, the Navier-Stokes equations. The equations are known for over 150 years, yet their behavior is still not fully understood. In this paper we examine a particular form of the equations – the incompressible, isothermal Navier-Stokes equations for Newtonian fluids.

Analytic solutions to the Navier-Stokes equations are difficult and few, except for special cases, due to their nonlinear and coupled nature. Mathematicians and physicists continue to search for the existence and uniqueness of analytic solutions (for a brief survey see Appendix B1). Consequently, numerical methods are important to the understanding of the behavior of Navier-Stokes. Though many numerical methods exist, there are often trade-offs between accuracy and efficiency. Different methods are developed for different physical phenomena modelled by the equations; for example, some methods are useful for understanding incompressible fluids, such as water, while others model compressible fluids, such as refrigerants.

We will look specifically at the Navier-Stokes with Pressure Poisson equations (PPE). The PPE is derived from what is known as the primitive variable form, or U-P form, of the equations. By using the PPE and determining the proper boundary conditions we are able overcome the weak coupling between the velocity and the pressure. This form then lends itself to efficient and accurate solvers, one of which we will

demonstrate. There are challenges in using this method, however. Implementing the new boundary conditions proves to be much less straight forward than the primitive variable form.

In the following sections we will introduce the primitive form of the incompressible Navier-Stokes equations and discuss some of the current methods used to numerically solve them. We then direct our attention towards the PPE and the derivation of the Pressure Poisson equation. The question of what are the appropriate boundary conditions for the PPE are asked (and adressed) time and time again over the years (see, for example [8], [11], [22]) as this aspect of the PPE proves to be the most challenging part of the formulation. We consider the formulation by Johnston and Liu [17] in our implementation. A modification introduced by Shirokoff and Rosales [23] is added for stability. Following the proof by Johnston and Liu, the primitive form and the PPE form are shown to be equivalent. We speak briefly about the Sobolev space where the solutions exist before deriving the weak form of the PPE. Using this weak form we are able to discretize the equations using Galerkin Finite Elements and numerically solve a benchmark problem.

1.1 The Navier-Stokes Equations

Numerically solving the incompressible Navier-Stokes equations are challenging for a variety of reasons. First is the nonlinear nature of the partial differential equations. Especially for flows of high velocity or low viscosity, the equations can produce highly unstable flows in the form of eddies. There is also the matter of the constraint imposed by the incompressibility. Any algorithm must ensure a divergence-free flow field at any given time during the calculation. This matter leads to the question of how to recover the pressure from the velocity considering the equations do not provide any boundary conditions for the pressure.

The MAC (Marker and Cell) Scheme is one of the oldest and most common methods. It was introduced in 1965 by Harlow and Welch [10] to solve for time-dependent viscous surface flows, but it continues to be updated. Projection methods were developed in the 1960s and 1970s independently by Temam [26] and Chorin [4]. These method split the computation into multiple temporal steps by first solving for velocity without regard to the incompressibility constraint, then projecting the velocity onto an incompressible velocity field via the PPE. Newer, second-order accurate projection methods were developed and became popular. The projection method suffers from numerical boundary layers, even for relatively stable flows, and is difficult to implement with nonconforming boundaries. The penalty method is successful for complicated domains, but it also introduces numerical boundary layers that reduce accuracy and efficiency. We will utilize a method based on a PPE formulation of the Navier-Stokes equations where the PPE replaces the incompressibility constraint and provides explicit boundary conditions for the pressure. This method was first introduced by Gresho and Sani [8], but this paper will utilize later work by Johnston and Liu [17] and Shirokoff and Rosales [23], whose work allows for the direct and efficient recovery of the pressure for a computed velocity. This method avoids the problems of numerical boundary layers since incompressibility is enforced at all times and pressure has its own explicit boundary conditions

The primitive variable, incompressible Navier-Stokes equations (NSE) on the domain $\Omega \subset \mathbb{R}^2$ (or $\mathbb{R}^3$) are given by

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = \mu \Delta \mathbf{u} - \nabla p + \mathbf{f} \tag{1.1a}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{1.1b}$$

where $\mathbf{u}(\mathbf{x}, t)$ is the velocity, μ is the kinematic viscosity, $p(\mathbf{x}, t)$ is the pressure, $\mathbf{f}(\mathbf{x}, t)$ are the body forces. We denote the gradient by ∇ and the Laplacian by

$\Delta = \nabla^2$. Equation 1.1a is known as the momentum equation and (1.1b) is the continuity equation, which is divergence free due to the incompressibility of the fluid and may be thought more as a constraint rather than a full equation. The boundary conditions for this formulation of the NSE are

$$\mathbf{u} = \mathbf{g}(\mathbf{x}, t) \text{ for } \mathbf{x} \in \Gamma \tag{1.2a}$$

$$\int_\Gamma \mathbf{n} \cdot \mathbf{g} dA = 0 \tag{1.2b}$$

where Γ is the boundary, $\mathbf{n}$ is the unit normal vector on the domain boundary, and dA is the area. Equation (1.2b) constitutes a global conservation of mass since the incompressible fluid must have zero net flux through the boundary. The above formulas are in non-dimensional forms by letting the constant density $\rho = 1$ and $\mu = \frac{1}{Re}$ where Re is the Reynolds number ($Re = \frac{UL}{\nu}$, U is the mean velocity, L is the characteristic length, and ν is the kinematic viscosity).

Remark: Using various identities, the advective term and viscous term may be written in different forms. For example, we may write the viscous term in rotational form using the identity $\Delta\mathbf{u} = \nabla(\nabla \cdot \mathbf{u}) - \nabla \times \nabla \times \mathbf{u} = -\nabla \times \nabla \times \mathbf{u}$ due to the divergence-free constraint. Likewise, the advective term is often written as $(\mathbf{u} \cdot \nabla)\mathbf{u} = \mathbf{u} \cdot \nabla\mathbf{u}$.

1.2 Function Space of the NSE

There is a lot of theory in the functional analysis of the partial derivatives in general and the Navier-Stokes equations in particular. This paper's focus is on the numerical aspects of the equations rather than the analytical aspects, so only the minimum of what is needed is provided below. The interested reader is directed to Temam [27], whose work elucidated the following discussion.

The domain Ω is a compact supported open subset of $\mathbb{R}^d$ where $d = 2$ or 3. Ω is

Lipschitzian. By this we mean that its boundary Γ can be locally represented by a Lipschitz continuous function. Let $\gamma : \mathbb{R}^{d-1} \to \mathbb{R}$ be a function that satisfies the Lipschitz condition $|\gamma(y) - \gamma(x)| \leq M|y - x| \; \forall$ x, y$\in \mathbb{R}^{d-1}$. Then near every boundary point $x \in \Gamma$ there exists a neighborhood $\mathcal{B}$ such that $\Omega \cap \mathcal{B} = \{(x, x_n) \in \mathcal{B} | x_n > \gamma(x)\}$. So Γ is the graph of a Lipschitz function.

L^2 is the space of real valued functions. It possesses the inner product and norm

$$\langle h, g \rangle = \int_\Omega h(x) g(x) \, dx, \qquad |h| = \langle h, h \rangle^{\frac{1}{2}}$$

Here we introduce Sobolev space as the space of solutions to the NSE. Sobolev space, which is a Hilbert space, is used because it allows the weakening of the notion of partial derivatives and thus permits functions that are less smooth. This is achieved by shifting derivatives from differentiable functions to distributions by using smooth test functions. Sobolev space possesses the inner product and norm

$$\langle h, g \rangle_m = \sum_{[\alpha] \leq m} \langle D^a h, D^a g \rangle, \qquad \|h\|_m = \langle h, h \rangle_m^{\frac{1}{2}}$$

$$\text{where} \quad \alpha = [\alpha_1, \ldots, \alpha_n], \quad \alpha_i \in \mathbb{N}, \quad [\alpha] = \sum \alpha_i$$

$$\text{and} \quad D^\alpha = D_1^{\alpha_1} \cdots D_n^{\alpha_n} = \frac{\partial^{[\alpha]}}{\partial x_1^{\alpha_1} \cdots x_n^{\alpha_n}}.$$

The subscript m represents the highest order derivative and α is called a multi-index. Let $\mathcal{D}(\Omega)$ be the set of infinitely differentiable functions with compact support in Ω and $\varphi \in \mathcal{D}(\Omega)$. Then a distribution is $\varphi : \mathcal{D}(\Omega) \to \mathbb{R}$; that is, a continuous linear map.

The particular Sobolev space of interest is $H^1(\Omega)$, the set of L^2 functions satisfying

$$H^1(\Omega) = \left\{ f : \Omega \to R \;\; | \;\; \|f\|_{H_1} < \infty \right\}$$

with the inner product and norm

$$\langle f, g\rangle_{H^1} = \int_\Omega fg + \int_\Omega (\nabla f)\cdot(\nabla g)$$
$$\|f\|_{H_1} = \sqrt{\langle f, f\rangle_{H^1}}.$$

$H^1(\Omega)$ includes functions that are piecewise continuously differentiable.

1.3 The Pressure Poisson Equation

The pressure Poisson equation (PPE) is derived from the momentum equation. Because it is derived by taking the divergence of the momentum equation, it requires that the solution be sufficiently smooth up to the boundary; that is, the solution must be more smooth than would otherwise be required. It will be shown later that the PPE is, for sufficiently smooth solutions, equivalent to the continuum NSE in the non-steady case. For the steady case, [17] notes that equivalence is achieved when the PPE is written without the viscosity term and a divergence free boundary condition is enforced for the velocity. To achieve the pressure Poisson equation formulation, we take the divergence of (1.1a) and use the divergence free condition (1.1b) such that

$$\nabla\cdot(\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\,\mathbf{u}) = \nabla\cdot(\mu\Delta\mathbf{u} - \nabla p + \mathbf{f})$$

Then on the left hand side we have

$$\nabla\cdot(\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\,\mathbf{u}) = \frac{\partial}{\partial t}(\nabla\cdot\mathbf{u}) + \nabla\cdot(\mathbf{u}\cdot\nabla)\,\mathbf{u} = \nabla\cdot(\mathbf{u}\cdot\nabla)\,\mathbf{u}$$

While on the right hand side we have

$$\nabla\cdot(\mu\Delta\mathbf{u} - \nabla p + \mathbf{f}) = \mu\Delta(\nabla\cdot\mathbf{u}) - \Delta p + \nabla\cdot\mathbf{f} = -\Delta p + \nabla\cdot\mathbf{f}$$

Then combining the new left hand side and right hand side we have

$$\nabla\cdot(\mathbf{u}\cdot\nabla)\,\mathbf{u} = -\Delta p + \nabla\cdot\mathbf{f}$$

And rearranging terms

$$\Delta p = \nabla \cdot (\mathbf{f} - (\mathbf{u} \cdot \nabla)\,\mathbf{u}) \tag{1.3}$$

This gives the new system of equations

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = \mu \Delta \mathbf{u} - \nabla p + \mathbf{f} \qquad \text{for } \mathbf{x} \in \Omega \tag{1.4a}$$

$$\Delta p = \nabla \cdot (\mathbf{f} - (\mathbf{u} \cdot \nabla)\,\mathbf{u}) \qquad \text{for } \mathbf{x} \in \Omega \tag{1.4b}$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{for } \mathbf{x} \in \Gamma \tag{1.4c}$$

$$\mathbf{u} = \mathbf{g}\,(\mathbf{x}, t) \qquad \text{for } \mathbf{x} \in \Gamma. \tag{1.4d}$$

1.4 Reformulation of the Pressure Poisson Equation

Can the flow velocity be used to obtain the pressure in the above PPE? This is a question raised and addressed in [23]. The reason for concern is that at this point the boundary conditions exist only for the velocity, so the pressure may not be determined accurately. What are needed are boundary conditions for both the flow velocity and the pressure that allow us to develop the velocity field in time for a given pressure and then solve for the pressure at each fixed time given the velocity. This is achieved by (i) specifying the normal derivative of the normal velocity through the divergence condition and (ii) requiring that the new pressure boundary condition be equivalent to $(\mathbf{n} \cdot (\mathbf{u} - \mathbf{g}))_t = 0$ for $\mathbf{x} \in \Gamma$ [23]. The new system of equations is then

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = \mu \Delta \mathbf{u} - \nabla p + \mathbf{f} \qquad \text{for } \mathbf{x} \in \Omega \tag{1.5a}$$

$$\mathbf{n} \times (\mathbf{u} - \mathbf{g}) = 0 \qquad \text{for } \mathbf{x} \in \Gamma \tag{1.5b}$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{for } \mathbf{x} \in \Gamma \tag{1.5c}$$

and

$$\Delta p = \nabla \cdot (\mathbf{f} - (\mathbf{u} \cdot \nabla)\,\mathbf{u}) \qquad \text{for } \mathbf{x} \in \Omega \tag{1.6a}$$

$$\mathbf{n} \cdot \nabla p = \mathbf{n} \cdot (\mathbf{f} - \mathbf{g}_t + \mu \Delta \mathbf{u} - (\mathbf{u} \cdot \nabla)\,\mathbf{u}) \qquad \text{for } \mathbf{x} \in \Gamma \tag{1.6b}$$

where $\mathbf{n}$ is the unit vector normal to Γ. We require Ω to be a convex polygon or polyhedron or Γ to be smooth.

1.5 Final Stable Reformulation of the Pressure Poisson Equation

The author of [23] demonstrates that the boundary conditions in (1.5) and (1.6) indeed recover the normal velocity boundary condition $\mathbf{n} \cdot \mathbf{u} = \mathbf{n} \cdot \mathbf{g}$ by showing that $(\mathbf{n} \cdot (\mathbf{u} - \mathbf{g}))_t = 0$ for $\mathbf{x} \in \Gamma$. However, the author continues that numericals error from the implementation of (1.5) and (1.6) result in a drift of the normal component velocity, which can have a destabilizing effect on the behavior of a numerical scheme. The author proposes adding a stabilizing term to the boundary condition of the PPE to make the equations suitable for numerical implementation while maintaining equivalence to (1.1):

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = \mu \Delta \mathbf{u} - \nabla p + \mathbf{f} \qquad \text{for } \mathbf{x} \in \Omega \tag{1.7a}$$

$$\mathbf{n} \times (\mathbf{u} - \mathbf{g}) = 0 \qquad \text{for } \mathbf{x} \in \Gamma \tag{1.7b}$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{for } \mathbf{x} \in \Gamma \tag{1.7c}$$

and

$$\Delta p = \nabla \cdot (\mathbf{f} - (\mathbf{u} \cdot \nabla)\,\mathbf{u}) \qquad \text{for } \mathbf{x} \in \Omega \tag{1.8a}$$

$$\mathbf{n} \cdot \nabla p = \mathbf{n} \cdot (\mathbf{f} - \mathbf{g}_t + \mu \Delta \mathbf{u} - (\mathbf{u} \cdot \nabla)\,\mathbf{u}) + \lambda \mathbf{n} \cdot (\mathbf{u} - \mathbf{g}) \qquad \text{for } \mathbf{x} \in \Gamma \tag{1.8b}$$

where $10 \leq \lambda \leq 100$ is a parameter that must be determined by numerical experimentation. Thus the potentially destabilizing error is corrected by using the PPE to enforce the boundary condition $\mathbf{n} \cdot \mathbf{u} = \mathbf{n} \cdot \mathbf{g}$

1.6 Equivalence of the PPE

Below we present a proof from [17] of the equivalence of the PPE formulation (1.4) to the primitive form of the Navier-Stokes equations. Work by Gresho and Sani

in [8] and [22] show various forms of the PPE formulation with different boundary conditions and their equivalence to the u-p formulation.

Theorem 1.6.1. *Assume* $\mathbf{f} \in L^{\infty}\left([0,T], H^{s}\right), s > 1/2.$
For $\mathbf{u} \in L^{\infty}\left([0,T], H^{2+s}\right) \bigcap Lip\left([0,T], H^{s}\right)$ *the formulation of the Navier Stokes equations is equivalent to the PPE formulation* (1.7)-(1.8).

Proof. It has been shown by previous authors that the system expressed in equations (1.4a)-(1.4d) are equivalent to the Navier-Stokes equations (1.1a)-(1.2b). We have already shown that the solution to (1.1a)-(1.2b) satisfies the PPE. As presented in [17], we show the solution $(\mathbf{u}, p)$ to (1.4a)-(1.4d) satisfies the Navier-Stokes equations. We take the divergence of (1.1a) to obtain

$$\partial_t \left(\nabla \cdot \mathbf{u}\right) + \nabla \cdot \left(\mathbf{u} \cdot \nabla \mathbf{u}\right) + \Delta p = \mu \Delta \left(\nabla \cdot \mathbf{u}\right) + \nabla \cdot \mathbf{f}.$$

Using the identity $\nabla \cdot \left(\mathbf{u} \cdot \nabla \mathbf{u}\right) = \nabla \mathbf{u} : \nabla \mathbf{u} - \left(\mathbf{u} \cdot \nabla\right)^2$ in equation (1.4b) and substituting into the above equation we arrive at

$$\partial_t \left(\nabla \cdot \mathbf{u}\right) + \nabla \cdot \left(\mathbf{u} \cdot \nabla \mathbf{u}\right) - \nabla \mathbf{u} : \nabla \mathbf{u} + \left(\mathbf{u} \cdot \nabla\right)^2 + \nabla \cdot \mathbf{f} = \mu \Delta \left(\nabla \cdot \mathbf{u}\right) + \nabla \cdot \mathbf{f}.$$

Cancelling terms we have

$$\partial_t \left(\nabla \cdot \mathbf{u}\right) + \nabla \cdot \left(\mathbf{u} \cdot \nabla \mathbf{u}\right) - \nabla \mathbf{u} : \nabla \mathbf{u} + \left(\mathbf{u} \cdot \nabla\right)^2 = \mu \Delta \left(\nabla \cdot \mathbf{u}\right).$$

We now use the identity $\nabla \cdot \left(\mathbf{u} \cdot \nabla \mathbf{u}\right) = \nabla \mathbf{u} : \nabla \mathbf{u} + \mathbf{u} \cdot \nabla \left(\nabla \cdot \mathbf{u}\right)$ to achieve

$$\partial_t \left(\nabla \cdot \mathbf{u}\right) + \nabla \mathbf{u} : \nabla \mathbf{u} + \mathbf{u} \cdot \nabla \left(\nabla \cdot \mathbf{u}\right) - \nabla \mathbf{u} : \nabla \mathbf{u} + \left(\mathbf{u} \cdot \nabla\right)^2 = \mu \Delta \left(\nabla \cdot \mathbf{u}\right).$$

We cancel terms and denote $\phi = \nabla \cdot \mathbf{u}$. We then have

$$\partial_t \phi + \mathbf{u} \cdot \nabla \phi + \phi^2 = \mu \Delta \phi \tag{1.9}$$

with initial data $\phi|_{t=0} = 0$. Next we use the identity $\Delta \mathbf{u} = \nabla \left(\nabla \cdot \mathbf{u}\right) - \nabla \times \nabla \times \mathbf{u}$ to rewrite (1.4a) in the rotational form of

$$\mathbf{u}_t + \left(\mathbf{u} \cdot \nabla\right) \mathbf{u} + \nabla p = -\mu \nabla \times \nabla \times \mathbf{u} + \mu \nabla \left(\nabla \cdot \mathbf{u}\right) + \mathbf{f}.$$

As in [17], we take the normal component of the trace and use $\mathbf{u}|_\Gamma = 0$ to get

$$\mathbf{n} \cdot p = -\mu \mathbf{n} \cdot \nabla \times \nabla \times \mathbf{u} + \mu \mathbf{n} \cdot \nabla (\nabla \cdot \mathbf{u}) + \mathbf{n} \cdot \mathbf{f}$$

on the boundary Γ. Comparing the above equation with (1.6b), we have

$$\frac{\partial \phi}{\partial \mathbf{n}} = \mathbf{n} \cdot \nabla (\nabla \cdot \mathbf{u}) = 0.$$

Continuing with the proof from [17], we show using energy estimates that ϕ equals zero almost everywhere. First, multiply (1.9) by ϕ and integrate over Ω to obtain

$$\frac{d}{dt} \int_\Omega \phi^2 + \int_\Omega \mathbf{u} \cdot \nabla (\phi)^2 + \int_\Omega \phi^3 = -\mu \int_\Omega |\phi|^2.$$

Integration by parts of the second term gives

$$\int_\Omega \mathbf{u} \cdot \nabla (\phi)^2 = -\int_\Omega (\nabla \cdot \mathbf{u}) \phi^2 = -\int_\Omega \phi^3.$$

So, the second and third terms cancel to yield

$$\frac{d}{dt} \int_\Omega \phi^2 + \mu \int_\Omega |\phi|^2 = 0$$

with the initial conditions

$$\int_\Omega \phi^2 = 0 \text{ for } t = 0.$$

Therefore,

$$\int_\Omega \phi^2 = 0$$

for all $t > 0$. Therefore, $\phi = \mathbf{u} \cdot \nabla = 0$ almost everywhere, which proves that $(\mathbf{u}, p)$ is a solution to the incompressible NSE. □

CHAPTER 2

GALERKIN FINITE ELEMENT METHOD

Several tools are available when choosing how to discretize the Navier-Stokes problem. The three broad choices are the finite element method, finite difference method, and the finite volume method. Each of these has its own strength and weakness. The finite difference method has the benefit of being relatively easy to implement. However, it is not well suited for complex geometries and suffers from poor stability and convergence analysis. The finite volume method is based on physical conservation properties, but it too is difficult to analyze for stability and convergence and it is poorly suited for unstructured meshes. The finite element method, the choice for this study, is able to achieve a high degree of accuracy, is well suited for complex geometries, and facilitates rigorous error analysis [21].

In this chapter we derive the weak formulation of the Navier-Stokes equation with PPE. We then make our choice of element and test function and use the weak form to discretize our equations.

2.1 Weak Formulation

We rewrite equations (1.5a) and (1.6) in their weak form in a manner similar to that found in [17]. First we define an inner product $\langle \cdot, \cdot \rangle$ by $\langle f, g \rangle = \int_{-1}^{1} f(x)\, g(x)\, dx$. For our weak form of the momentum equation we have for all smooth test functions ψ with $\psi|_\Gamma = 0$ find $\mathbf{u}$ such that

$$\langle \mathbf{u}_t, \psi \rangle_\Omega + \langle \mathbf{u} \cdot \nabla \mathbf{u}, \psi \rangle_\Omega + \langle \nabla p, \psi \rangle_\Omega = \mu \langle \Delta \mathbf{u}, \psi \rangle_\Omega + \langle \mathbf{f}, \psi \rangle_\Omega . \tag{2.1}$$

For the diffusion term $\Delta \mathbf{u}$ we lower the order of the derivative through integration by parts and using the fact that $\psi|_\Gamma = 0$. This leads to the final momentum equation

$$\langle \mathbf{u}_t, \psi \rangle_\Omega + \langle \mathbf{u} \cdot \nabla \mathbf{u}, \psi \rangle_\Omega + \langle \nabla p, \psi \rangle_\Omega = -\mu \langle \nabla \mathbf{u}, \nabla \psi \rangle_\Omega + \langle \mathbf{f}, \psi \rangle_\Omega . \tag{2.2}$$

For the PPE use the smooth test function ϕ to obtain the weak form

$$\langle \Delta p, \phi \rangle_\Omega = -\langle \nabla \cdot ((\mathbf{u} \cdot \nabla)\,\mathbf{u}), \phi \rangle_\Omega + \langle \nabla \cdot \mathbf{f}, \phi \rangle_\Omega \tag{2.3a}$$

$$\langle \mathbf{n} \cdot \nabla p, \phi \rangle_\Gamma = \langle \mathbf{n} \cdot (\mathbf{f} - bfg_t + \mu \Delta \mathbf{u} - (\mathbf{u} \cdot \nabla)\,\mathbf{u}), \phi \rangle_\Gamma + \langle \lambda \mathbf{n} \cdot (\mathbf{u} - \mathbf{g}), \phi \rangle_\Gamma \tag{2.3b}$$

By integrating by parts the left side of (2.3a) we have

$$\langle \Delta p, \phi \rangle_\Omega = \langle \mathbf{n} \cdot \nabla p, \phi \rangle_\Gamma - \langle \nabla p, \nabla \phi \rangle_\Omega . \tag{2.4}$$

We make use of the boundary condition given by (2.3b) and substitute into (2.4) and make use of the identity $\nabla \times \nabla \times \mathbf{u} = \nabla (\nabla \cdot \mathbf{u}) - \Delta \mathbf{u} = -\Delta \mathbf{u}$ since $(\nabla \cdot \mathbf{u})\,|_\Gamma = 0$ yielding

$$\langle \mathbf{n} \cdot \nabla p, \phi \rangle_\Gamma - \langle \nabla p, \nabla \phi \rangle_\Omega = \langle \mathbf{n} \cdot \mathbf{f}, \phi \rangle_\Gamma - \langle \mathbf{n} \cdot \mathbf{g}_t, \phi \rangle_\Gamma + \mu \langle \mathbf{n} \cdot (\nabla \times \nabla \times \mathbf{u}), \phi \rangle_\Gamma - \langle \mathbf{n} \cdot (\mathbf{u} \cdot \nabla)\,\mathbf{u}, \phi \rangle_\Gamma + \langle \lambda \mathbf{n} \cdot (\mathbf{u} - \mathbf{g}), \phi \rangle_\Gamma - \langle \nabla p, \nabla \phi \rangle_\Omega . \tag{2.5}$$

We apply the vector identity

$$\langle \mathbf{n} \cdot (\nabla \times \nabla \times \mathbf{u}), \phi \rangle_\Gamma = \langle \nabla \times \nabla \times \mathbf{u}, \nabla \phi \rangle_\Omega = -\langle \nabla \times \mathbf{u}, \mathbf{n} \times \nabla \phi \rangle_\Gamma$$

to give

$$\langle \Delta p, \phi \rangle_\Omega = \langle \mathbf{n} \cdot \mathbf{f}, \phi \rangle_\Gamma - \langle \mathbf{n} \cdot \mathbf{g}_t, \phi \rangle_\Gamma - \mu \langle \nabla \times \mathbf{u}, \mathbf{n} \times \nabla \phi \rangle_\Gamma - \langle \mathbf{n} \cdot (\mathbf{u} \cdot \nabla)\,\mathbf{u}, \phi \rangle_\Gamma + \langle \lambda \mathbf{n} \cdot (\mathbf{u} - \mathbf{g}), \phi \rangle_\Gamma - \langle \nabla p, \nabla \phi \rangle_\Omega . \tag{2.6}$$

Now, for the right side of (2.3a) we use integration by parts to achieve

$$-\langle \nabla \cdot ((\mathbf{u} \cdot \nabla)\,\mathbf{u}), \phi \rangle_\Omega + \langle \nabla \cdot \mathbf{f}, \phi \rangle_\Omega = \langle (\mathbf{u} \cdot \nabla)\,\mathbf{u}, \nabla \phi \rangle_\Omega - \langle \mathbf{f}, \nabla \phi \rangle_\Omega + \langle \mathbf{n} \cdot \mathbf{f}, \phi \rangle_\Gamma . \tag{2.7}$$

Finally, we set (2.6) equal to (2.7), cancel similar terms, realize the incompressibility constraint $\mathbf{u} \cdot \nabla|_\Gamma = 0$ and rearrange to arrive at

$$\langle \nabla p, \nabla \phi \rangle_\Omega = -\langle (\mathbf{u} \cdot \nabla)\,\mathbf{u}, \nabla \phi \rangle_\Omega + \langle \mathbf{f}, \nabla \phi \rangle_\Omega - \langle \mathbf{n} \cdot \mathbf{g}_t, \phi \rangle_\Gamma - \mu \langle \nabla \times \mathbf{u}, \mathbf{n} \times \nabla \phi \rangle_\Gamma + \langle \lambda \mathbf{n} \cdot (\mathbf{u} - \mathbf{g}), \phi \rangle_\Gamma . \tag{2.8}$$

Per [17], we take $\mathrm{X} = \{\psi \in (H_0^1(\Omega))^d, (\nabla \times \mathbf{u})|_\Gamma \in L^2(\Gamma)\}$ and $\mathrm{Y} = \{\phi \in H^1(\Omega)/C, (\mathbf{n} \times \nabla\phi)|_\Gamma \in L^2(\Gamma)\}$ to give the full variational formulation:

If Γ is smooth or if Ω is a convex polygon or polyhedron find $\mathbf{u} \in L^2(0,T;X)$ and p $\in L^2(0,T;Y)$ such that

$$\langle \mathbf{u}_t, \psi \rangle_\Omega + \langle \mathbf{u} \cdot \nabla \mathbf{u}, \psi \rangle_\Omega + \langle \nabla p, \psi \rangle_\Omega = -\mu \langle \nabla \mathbf{u}, \nabla \psi \rangle_\Omega + \langle \mathbf{f}, \psi \rangle_\Omega , \tag{2.9}$$

$$\langle \nabla p, \nabla \phi \rangle_\Omega = \langle \mathbf{f} - (\mathbf{u} \cdot \nabla)\mathbf{u}, \nabla \phi \rangle_\Omega + \langle \lambda \mathbf{n} \cdot (\mathbf{u} - \mathbf{g}) - \mathbf{n} \cdot \mathbf{g}_t, \phi \rangle_\Gamma - \mu \langle \nabla \times \mathbf{u}, \mathbf{n} \times \nabla \phi \rangle_\Gamma , \tag{2.10}$$

is satisfied $\forall \psi \in$ X and $\forall \phi \in$ Y a.e. in (0,T).

2.2 Finite Element Approximation

To develop our finite element discretization, we use the weak form previously developed in the appropriate function space with piecewise continuous polynomials from a trial space. For the two-dimensional case there are two common element types from which to choose: triangular elements and quadrilateral elements. The next task is to describe each matrix in terms of the test functions and summarize the equations in semi-discrete form, postponing briefly the discretization in time.

Let $\{\psi_1, \psi_2, \ldots, \psi_n\}$ be a basis for the velocity shape functions and let $\{\phi_1, \phi_2, \ldots, \phi_m\}$ be the basis for the pressure shape functions. A benefit of using the Galerkin method is that our trial functions and test functions are the same. The velocity and pressure approximations are then uniquely represented by the expansions

$$u_h = \sum_{j=1}^{n} u_j \psi_j, \qquad v_h = \sum_{j=1}^{n} v_j \psi_j, \qquad \text{and} \qquad p_h = \sum_{j=1}^{m} p_j \phi_j.$$

The finite element forms for the momentum equation in the x-direction and y-direction

are

$$\int_\Omega \frac{du_h}{dt}\psi + \int_\Omega u_h \frac{\partial u_h}{\partial x}\psi + v_h \frac{\partial u_h}{\partial y}\psi + \int_\Omega \frac{\partial p_h}{\partial x}\psi = -\mu \int_\Omega \frac{\partial u_h}{\partial x}\frac{\partial \psi}{\partial x} + \frac{\partial u_h}{\partial y}\frac{\partial \psi}{\partial y} + \int_\Omega f_1 \psi \tag{2.11a}$$

$$\int_\Omega \frac{dv_h}{dt}\psi + \int_\Omega u_h \frac{\partial v_h}{\partial x}\psi + v_h \frac{\partial v_h}{\partial y}\psi + \int_\Omega \frac{\partial p_h}{\partial y}\psi = -\mu \int_\Omega \frac{\partial v_h}{\partial x}\frac{\partial \psi}{\partial x} + \frac{\partial v_h}{\partial y}\frac{\partial \psi}{\partial y} + \int_\Omega f_2 \psi. \tag{2.11b}$$

The finite element form for the PPE is

$$\begin{aligned}\int_\Omega \frac{\partial p_h}{\partial x}\frac{\partial \phi}{\partial x} + \frac{\partial p_h}{\partial y}\frac{\partial \phi}{\partial y} = \int_\Omega f_1 \frac{\partial \phi}{\partial x} + f_2 \frac{\partial \phi}{\partial y} - \\ \int_\Omega u_h \frac{\partial u_h}{\partial x}\frac{\partial \phi}{\partial x} + v_h \frac{\partial u_h}{\partial y}\frac{\partial \phi}{\partial x} + u_h \frac{\partial v_h}{\partial x}\frac{\partial \phi}{\partial y} + v_h \frac{\partial v_h}{\partial y}\frac{\partial \phi}{\partial y} + \\ \mu \int_\Gamma n_x \frac{\partial v_h}{\partial x}\frac{\partial \phi}{\partial y} - n_y \frac{\partial v_h}{\partial x}\frac{\partial \phi}{\partial x} + n_y \frac{\partial u_h}{\partial y}\frac{\partial \phi}{\partial x} - n_x \frac{\partial u_h}{\partial y}\frac{\partial \phi}{\partial y} + \\ \int_\Gamma \lambda n \cdot (u_h + v_h - g)\,\phi - n \cdot g_t \phi\end{aligned} \tag{2.12}$$

Equations (2.11) can be written in a matrix form as

$$\begin{bmatrix} M & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} + \begin{bmatrix} N(\mathbf{u}) + \nu D & \\ & N(\mathbf{u}) + \nu D \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} p = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{2.13}$$

where M, N, D, B, and F are defined in Table 2.1. This leads to the compact form of

$$[M]\,\dot{\mathbf{U}} + [K(u)]\,\mathbf{U} + [B]\,P = [F], \tag{2.14}$$

where $K(u)$ is the sum of the convective and diffusive terms. The PPE is a little more complicated since it involves tangential derivatives and terms that appear only on the boundary. In compact form we can write

$$[Mp]\,p = [K_1]\,u + [K_2]\,v + F_3 - \lambda g \tag{2.15}$$

where $K_1 = T_1(\mathbf{u}) + \nu L_1 + \lambda S_1$ and $K_2 = T_2(\mathbf{u}) + \nu L_2 + \lambda S_2$ with each matrix and vector defined in Table 2.2.

Abbreviation	Formula	Representation
M	$\int_\Omega \psi_i \psi_j$	Mass Matrix
D	$\int_\Omega \frac{\partial \psi_i}{\partial x}\frac{\partial \psi_j}{\partial x} + \frac{\partial \psi_i}{\partial y}\frac{\partial \psi_j}{\partial y}$	Diffusion
$N(u)$	$\int_\Omega u_i \psi_i \frac{\partial \psi_i}{\partial x}\psi_j + v_i \psi_i \frac{\partial \psi_i}{\partial y}\psi_j$	Convection
B_1 and B_2	$\int_\Omega \frac{\partial \varphi_i}{\partial x}\psi_j$ and $\int_\Omega \frac{\partial \varphi_i}{\partial y}\psi_j$	Pressure Gradient
F_1 and F_2	$\int_\Omega f_{1i}\psi_j$ and $\int_\Omega f_{2i}\psi_j$	External Force

Table 2.1: Coefficient matrices for the momentum equation.

2.3 The Choice of Element

There exist many choices with regard to the type of element to be used. To begin with, there is the question of equal order elements versus mixed elements, which refers to whether the interpolating functions for pressure and velocity are of equal order or are different. Because the diffusion operator for velocity is of higher order than the gradient operator for the pressure, mixed elements where the velocity is at least one order higher than the pressure are popular choices that typically lead to stable elements provided that the pressure interpolant is at least linear. Other element choices that are not inherently stable can be made so by adding stabilizing terms, but this can lead to inaccurate solutions. When we speak of stable element pairs, we are referring to the Ladyzhenskaya (1969), Babuška (1971), and Brezzi (1974) compatibility condition, also known as the LBB or inf-sup condition, that says that the existence of a stable finite element approximate solution to the steady Stokes problem depends on choosing a pair of spaces V^h and Q^h such that the following condition holds:

$$\inf_{q^h \in Q^h} \sup_{\mathbf{w} \in \mathbb{V}^h} \frac{\left(q^h, \nabla \cdot \mathbf{w}^h\right)}{\|q\|_0 \|\mathbf{w}^h\|_1} \geq \alpha > 0 \tag{2.16}$$

Abbreviation	Formula	Representation
Mp	$\int_\Omega \frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y}\frac{\partial \phi_j}{\partial y}$	Laplacian for pressure
$T_1(u)$	$\int_\Omega u_i\psi_i\frac{\partial \psi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + v_i\psi_i\frac{\partial \psi_i}{\partial y}\frac{\partial \phi_j}{\partial x}$	Convection
$T_2(u)$	$\int_\Omega u_i\psi_i\frac{\partial \psi_i}{\partial x}\frac{\partial \phi_j}{\partial y} + v_i\psi_i\frac{\partial \psi_i}{\partial y}\frac{\partial \phi_j}{\partial y}$	Convection
L_1	$\int_\Gamma \left(-n_x\frac{\partial \phi_j}{\partial y} + n_y\frac{\partial \phi_j}{\partial y}\right)\frac{\partial \psi_i}{\partial y}$	Diffusive term
L_2	$\int_\Gamma \left(n_x\frac{\partial \phi_j}{\partial y} - n_y\frac{\partial \phi_j}{\partial y}\right)\frac{\partial \psi_i}{\partial x}$	Diffusive term
S_1 & S_2	$\int_\Gamma n_x\psi_i\phi_j$ and $\int_\Gamma n_y\psi_i\phi_j$	Stability Term
g	$\int_\Gamma (n_x g_{ui} + n_y g_{vi})\,\phi_j$	Stability Term
F_3	$\int_\Omega f_{1i}\frac{\partial \phi_j}{\partial x} + f_{2i}\frac{\partial \phi_j}{\partial y}$	External Force

Table 2.2: Coefficient matrices for the pressure Poisson equation.

where α is independent of mesh size h. Proving that a mixed element pair satisfies the LBB condition is not simple, and there are many techniques to do so. This thesis will not delve into these details, but the interested reader may consult Girault and Raviart [7].

The next consideration is the shape of element. For the two-dimensional case we have the triangular and quadrilateral elements, which can also be mixed (e.g. quadratic quadrilateral for the velocity and linear triangle for the pressure). Quadrilateral elements are generally easier to implement, but they are not as flexible with respect to complex geometry. Triangular elements, on the other hand, provide better meshes for complex geometries. See [9] for a detailed discussion on element choices.

We will use Taylor-Hood[25] triangle elements with quadratic interpolation for the velocity and linear interpolation for the pressure. These elements are denoted as P_2P_1. Ervin and Jenkins prove the LBB condition for this element pair in [6]. As well as being stable, these elements also converge quadratically. Each component of

velocity will be characterized by 6 nodes while pressure will be characterized by 3 nodes (see figure 2.1). Our approximations for velocity and pressure are then

$$u_h(x,y,t) = \sum_{j=1}^{6} u_j \psi_j, \qquad v_h(x,y,t) = \sum_{j=1}^{6} v_j \psi_j, \qquad \text{and} \qquad p_h(x,y) = \sum_{j=1}^{3} p_j \phi_j. \tag{2.17}$$

(a) (b)

Figure 2.1: Linear and quadratic triangle elements. Figure (a) corresponds to linear pressure elements and figure (b) corresponds to quadratic elements for each component of the velocity. In all, for each triangle element there are 15 degrees of freedom.

For each element there will be 15 unknowns – six each for the two components of velocity and three for pressure. We make use of Lagrange isoparametric interpolating functions. The six shape functions for the velocity over the reference element, denoted by $\hat{\psi}$, and three shape functions for the pressure over the reference element, denoted by $\hat{\phi}$, are given by

$$\begin{bmatrix} \hat{\psi}_1(\xi,\eta) \\ \hat{\psi}_2(\xi,\eta) \\ \hat{\psi}_3(\xi,\eta) \\ \hat{\psi}_4(\xi,\eta) \\ \hat{\psi}_5(\xi,\eta) \\ \hat{\psi}_6(\xi,\eta) \end{bmatrix} = \begin{bmatrix} 1 - 3\xi - 3\eta + 2\xi^2 + 4\xi\eta + 2\eta^2 \\ -\xi + 2\xi^2 \\ -\eta + 2\eta^2 \\ 4\xi - 4\xi\eta - 4\xi^2 \\ 4\eta - 4\xi\eta - 4\eta^2 \\ 4\xi\eta \end{bmatrix} \tag{2.18}$$

and

$$\begin{bmatrix} \hat{\phi}_1(\xi,\eta) \\ \hat{\phi}_2(\xi,\eta) \\ \hat{\phi}_3(\xi,\eta) \end{bmatrix} = \begin{bmatrix} 1-\xi-\eta \\ \xi \\ \eta \end{bmatrix} \tag{2.19}$$

The mapping from an arbitrary triangular element with vertices (x_i, y_i), $1 \leq i \leq 6$, to a reference element with vertices

$$(\xi_1,\eta_1) = (0,0), \quad (\xi_2,\eta_2) = (1,0), \quad (\xi_3,\eta_3) = (0,1),$$
$$(\xi_4,\eta_4) = \left(\tfrac{1}{2},0\right), \quad (\xi_5,\eta_5) = \left(\tfrac{1}{2},\tfrac{1}{2}\right), \quad (\xi_6,\eta_6) = \left(0,\tfrac{1}{2}\right)$$

is given by

$$x(\xi,\eta) = \sum_{j=1}^{6} \psi_j x_j \qquad\qquad y(\xi,\eta) = \sum_{j=1}^{6} \psi_j y_j$$

The shape (and test) functions $\psi(x,y)$ and $\phi(x,y)$ over the physical triangle are defined as

$$\psi(x,y) = \hat{\psi}(\xi(x,y), \eta(x,y)) \qquad \phi(x,y) = \hat{\phi}(\xi(x,y), \eta(x,y)). \tag{2.20}$$

The Jacobian is computed by taking the derivatives of $x(\xi,\eta)$ and $y(\xi,\eta)$ above to get

$$J = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}.$$

The derivative operator is then expressed as

$$\begin{bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{bmatrix} = J^{-1} \begin{bmatrix} \dfrac{\partial}{\partial \xi} \\ \dfrac{\partial}{\partial \eta} \end{bmatrix}.$$

2.4 Temporal Discretization

Typical of time-dependent problems is first the discretization with respect to space. The semi-discrete equations are then integrated forward in time by some method. This strategy is known as the method of lines, and is well suited for linear equations (e.g. Stokes equations). For nonlinear equations, the preference is to discretize in time, then space [5]. In keeping in the spirit of [17] and [23], we already made our choice to use the method of lines as is common practice.

There are many choices for time discretization, so we choose to discuss one class – the so-called θ-methods.

$$[M + \theta\Delta t K(\mathbf{u})^n]\mathbf{u}^n + Bp^n = \left[M - (1-\theta)\Delta t K(\mathbf{u})^{n-1}\right]\mathbf{u}^{n-1} + \theta\Delta t\mathbf{f}^n + (1-\theta)\Delta t\mathbf{f}^{n-1} \quad (2.21)$$

The special cases are first-order explicit forward Euler ($\theta = 0$), first-order implicit backward Euler ($\theta = 1$), and second-order implicit Crank-Nicolson $\left(\theta = \frac{1}{2}\right)$. As is well known, the forward Euler method is the easiest to implement, but is the least stable. Backward Euler is strongly A-stable (smooths oscillations), but is highly dissipative and therefore is not well suited for unsteady problems. The Crank-Nicolson scheme is popular due to its mix of stability and second-order convergence, but it can suffer from unexpected instabilities. Following the discussion presented in [23], we discuss the stability of a second-order Crank-Nicolson scheme where pressure is treated with

a second-order Adams-Bashforth extrapolation:

$$\mathbf{u}^{n+1} - \mathbf{u}^n = \nu\Delta t\left(\Delta\mathbf{u}^{n+1} + \Delta\mathbf{u}^n\right) - \frac{3\Delta t}{2}\left(\mathbf{u}^n \cdot \nabla\mathbf{u} + \nabla p^n\right) + \frac{\Delta t}{2}\left(\mathbf{u}^{n-1} \cdot \nabla\mathbf{u} + \nabla p^{n-1}\right) \qquad x \in \Omega \qquad (2.22a)$$

$$\mathbf{n} \times \mathbf{u}^{n+1} = 0 \qquad x \in \Gamma \qquad (2.22b)$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0 \qquad x \in \Gamma \qquad (2.22c)$$

$$\Delta p = -\nabla \cdot \left(\mathbf{u}^{n+1} \cdot \nabla\mathbf{u}^{n+1}\right) + \nabla \cdot \mathbf{f}^{n+1} \qquad x \in \Omega \qquad (2.22d)$$

$$\frac{\partial p^{n+1}}{\partial n} = -\nu\mathbf{n} \cdot \left(\nabla \times \nabla \times \mathbf{u}^{n+1}\right) + \mathbf{n} \cdot \mathbf{f}^{n+1} \qquad x \in \Gamma \qquad (2.22e)$$

This method is suitable for low Reynolds numbers (viscous flow) where we treat $\nu\Delta\mathbf{u}$ implicitly . Normal mode analysis shows that the eigenvalues of the time stepping operator lie within the unit circle [24], indicating that the normal modes $\tilde{\mathbf{u}}$ (defined as $\mathbf{u}^n = \alpha^n\tilde{\mathbf{u}}$ where $\tilde{\mathbf{u}}$ satisfies (2.22b) and (2.22c) and α is an eigenvalue of the time-stepping operator) remains bounded for the scheme. Therefore the scheme is stable for simple domains.

If the Reynolds number is large, treating the viscous term implicitly does not stabilize the solution. For convective flows an explicit method is used. Johnston and Liu note that a convectively stable scheme, such as Runge-Kutta 4 (RK4), should be used. RK4 is a satisfactory choice since its stable region encompasses a large portion of the imaginary access. However, the price is paid in terms of computational costs. For a thorough discussion on the many time-stepping schemes in computational fluid mechanics, see [28].

In addition to the scheme, another important consideration for any time-dependent algorithm is the time step itself. Diffusive problems, such as the Stokes equation, require smaller time steps than convective flows. Ideally, the most robust algorithm is able to incorporate adaptive time stepping to maximize efficiency in accuracy and computational cost. Johnston and Liu present in [17] Reynolds-dependent stability

constraints that determine what time step should be considered in conjunction with the time discretization:

$$\text{Diffusive stability constraint} \qquad \frac{\nu \Delta t}{\Delta x^2} \leq \frac{1}{2^d} \tag{2.23}$$

$$\text{Convective stability constraint} \qquad \|\mathbf{u}\|_\infty \frac{\Delta t}{\Delta x} = CFL \leq 1 \tag{2.24}$$

where Δx is the smallest grid resolution, d is the dimension, and CFL is the Courant-Friedrichs-Lewy necessary condition for convergence used for explicit schemes.

2.5 Nonlinear Solver

One of the challenges of the Navier-Stokes equations revolves around their nonlinearity. Two common ways to approach the problem are through Newton-Raphson iterations and Picard iterations. The Picard iteration converges linearly, but its radius of convergence is much larger than that of the Newton method and therefore does not require a good initial guess. The well-known Newton iteration converges quadratically, but requires a good starting point to achieve convergence. The method in its pure form also suffers from the requirement to update the Jacobian every iteration. This requirement can be circumvented by, for example, updating the Jacobian perhaps only once or every few terms at the expense of less-than-quadratic convergence. To prepare our formulation for the Newton iteration, we neglect the time derivative term to give

$$[K(\mathbf{u})]\,\mathbf{u} + [G]\,p = \mathbf{f}. \tag{2.25}$$

We search for a fixed point by solving for the residual:

$$R(\mathbf{u}) = [K(\mathbf{u})]\,\mathbf{u} + [G]\,p - \mathbf{f} = 0. \tag{2.26}$$

The truncated Taylor series expansion of $R(\mathbf{u})$ about the known solution $\mathbf{u}^n$ is

$$0 = R(\mathbf{u}^n) + \frac{\partial R(\mathbf{u})}{\partial \mathbf{u}} \Delta \mathbf{u} + O\left(\mathbf{u}^2\right) \tag{2.27}$$

where $\Delta\mathbf{u} = \mathbf{u}^{n+1} - \mathbf{u}^n$. Omitting $O(\mathbf{u}^2)$ and rearranging we have

$$R(\mathbf{u}^n) = -\frac{\partial R(\mathbf{u})}{\partial \mathbf{u}}\Delta\mathbf{u} \approxeq -J(\mathbf{u}^n)\,\Delta\mathbf{u}, \tag{2.28}$$

where $J(\mathbf{u}^n)$ is the Jacobian. Again rearranging we obtain

$$\Delta\mathbf{u} = -J^{-1}(\mathbf{u}^n)\,R(\mathbf{u}^n). \tag{2.29}$$

It is clear that our residual $R(\mathbf{u}^n)$ is simply the momentum equation set to 0 while neglecting the time derivative. The Jacobian is then the derivative of $R(\mathbf{u})$ with respect to $\mathbf{u}$, meaning the pressure disappears. The Jacobian in GFEM notation is

$$J_{11} = \int_\Omega \left(\psi\frac{\partial u}{\partial x} + u\frac{\partial \psi}{\partial x} + v\frac{\partial \psi}{\partial y}\right)\psi + \int_\Omega \left(\left(\frac{\partial \psi}{\partial x}\right)^2 + \left(\frac{\partial \psi}{\partial y}\right)^2\right) \tag{2.30a}$$

$$J_{12} = \int_\Omega \psi\frac{\partial u}{\partial y}\psi \tag{2.30b}$$

$$J_{21} = \int_\Omega \psi\frac{\partial v}{\partial x}\psi \tag{2.30c}$$

$$J_{22} = \int_\Omega \left(u\frac{\partial \psi}{\partial x} + v\frac{\partial \psi}{\partial y} + \psi\frac{\partial v}{\partial y}\right)\psi + \int_\Omega \left(\left(\frac{\partial \psi}{\partial x}\right)^2 + \left(\frac{\partial \psi}{\partial y}\right)^2\right) \tag{2.30d}$$

Substituting (2.29) back into the momentum equation we obtain the expression $M\mathbf{u}_t + J\Delta\mathbf{u} = R$. We can then utilize a time discretization to iterate with Newton-Raphson. For example, if we use the backward Euler method, we would have

$$\Delta\mathbf{u} = -\Delta t\,(M + \Delta t J)^{-1} R$$

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta\mathbf{u}.$$

CHAPTER 3

NUMERICAL EXPERIMENT

In this chapter we test our variational form. We consider the linearized Navier-Stokes since the problems that arise are not connected to the nonlinear advective term. The linearized Navier-Stokes equations, or Stokes equations, have the form

$$\frac{d\mathbf{u}}{dt} = \Delta\mathbf{u} - \nabla p + \mathbf{f} \qquad \text{for} \quad x \in \Omega \tag{3.1}$$

$$\mathbf{u} = 0 \qquad \text{for} \quad x \in \Gamma \tag{3.2}$$

$$\Delta p = \nabla \cdot \mathbf{f} \qquad \text{for} \quad x \in \Omega \tag{3.3}$$

$$\mathbf{n} \cdot \nabla p = \mathbf{n} \cdot -\nu \left(\nabla \times \nabla \times \mathbf{u}\right) - \mathbf{n} \cdot \frac{d\mathbf{g}}{dt} \qquad \text{for} \quad x \in \Gamma. \tag{3.4}$$

Our weak form is then: Find $\mathbf{u} \in L^2\left(0, T; X\right)$ and $\mathrm{p} \in L^2\left(0, T; Y\right)$ such that

$$\left\langle \mathbf{u}_t, \psi \right\rangle_\Omega + \left\langle \nabla p, \psi \right\rangle_\Omega = -\mu \left\langle \nabla \mathbf{u}, \nabla \psi \right\rangle_\Omega + \left\langle \mathbf{f}, \psi \right\rangle_\Omega \tag{3.5}$$

$$\left\langle \nabla p, \nabla \phi \right\rangle_\Omega = \left\langle \mathbf{f} + \left\langle \lambda \mathbf{n} \cdot \left(\mathbf{u} - \mathbf{g}\right) - \mathbf{n} \cdot \mathbf{g}_t, \phi \right\rangle_\Gamma - \mu \left\langle \nabla \times \mathbf{u}, \mathbf{n} \times \nabla \phi \right\rangle_\Gamma . \right. \tag{3.6}$$

Using our shape functions from (2.20) and approximations (2.17), we discretize our domain using finite elements and integrate using fifth-order, 7-point Gauss quadrature.

$$\int_\Omega \frac{du_h}{dt}\psi + \int_\Omega \frac{\partial p_h}{\partial x}\psi = -\mu \int_\Omega \frac{\partial u_h}{\partial x}\frac{\partial \psi}{\partial x} + \frac{\partial u_h}{\partial y}\frac{\partial \psi}{\partial y} + \int_\Omega f_1 \psi \tag{3.7a}$$

$$\int_\Omega \frac{dv_h}{dt}\psi + \int_\Omega \frac{\partial p_h}{\partial y}\psi = -\mu \int_\Omega \frac{\partial v_h}{\partial x}\frac{\partial \psi}{\partial x} + \frac{\partial v_h}{\partial y}\frac{\partial \psi}{\partial y} + \int_\Omega f_2 \psi \tag{3.7b}$$

$$\begin{aligned}\int_\Omega \frac{\partial p_h}{\partial x}\frac{\partial \phi}{\partial x} + \frac{\partial p_h}{\partial y}\frac{\partial \phi}{\partial y} = \int_\Omega f_1 \frac{\partial \phi}{\partial x} + f_2 \frac{\partial \phi}{\partial y} + \mu \int_\Gamma \left(\frac{\partial v_h}{\partial x} - \frac{\partial u_h}{\partial y}\right)\left(n_x \frac{\partial \phi}{\partial y} - n_y \frac{\partial \phi}{\partial x}\right) \\ + \int_\Gamma \lambda \left(n_x u_h + n_y v_h - n \cdot g\right)\phi - n \cdot g_t \phi\end{aligned} \tag{3.7c}$$

3.1 Flow on a square domain

We will solve the linear Navier-Stokes equation on the unit square $0 \le x, y \le 1$ with no slip and no flux boundary conditions ($u = v = 0$) and viscosity $\nu = 1$. We use u

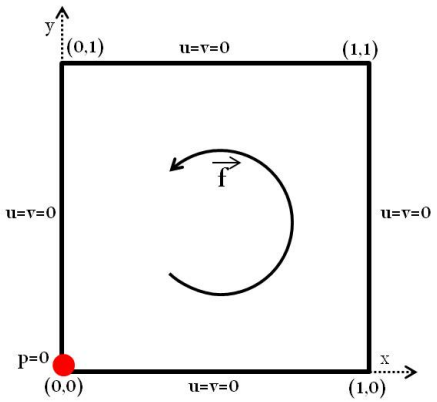


Figure 3.1: Flow on a square domain with no slip and no flux. The external force $\mathbf{f}$ acts on the fluid.

and v of the exact solution given below at $t = 0$ as our initial velocity.

$$u(x, y, t) = \pi \cos(t) \sin(2\pi y) \sin^2(\pi x) \tag{3.8a}$$

$$v(x, y, t) = -\pi \cos(t) \sin(2\pi x) \sin^2(\pi y) \tag{3.8b}$$

$$p(x, y, t) = -\cos(t) \cos(\pi x) \sin(\pi y). \tag{3.8c}$$

The forcing function is $\mathbf{f} = \mathbf{u}_t + \nabla p - \nu \Delta \mathbf{u}$. The finite element x and y components of the forcing function is

$$\begin{aligned} f_x = &- \pi \sin(t) \sin(2\pi y) \sin^2(\pi x) + \pi \cos(t) \sin(\pi x) \sin(\pi y) \\ &- 2\pi^3 \cos(t) \sin(2\pi y) \left(\cos^2(\pi x) - \sin^2(\pi x)\right) + 4\pi^3 \cos(t) \sin(2\pi y) \sin^2(\pi x) \end{aligned} \tag{3.9a}$$

$$\begin{aligned} f_y = &\ \pi \sin(t) \sin(2\pi x) \sin^2(\pi y) - \pi \cos(t) \cos(\pi x) \cos(\pi y) \\ &+ 2\pi^3 \cos(t) \sin(2\pi x) \left(\cos^2(\pi y) - \sin^2(\pi y)\right) - 4\pi^3 \cos(t) \sin(2\pi x) \sin^2(\pi y) \end{aligned} \tag{3.9b}$$

As noted earlier, we choose mixed finite elements using Taylor-Hood triangles with the velocity discretized using quadratic, piecewise continuous Lagrange polynomials and the pressure discretized using linear, piecewise continuous Lagrange poly-

nomials. The approximation spaces $\tilde{X}$ and $\tilde{Y}$ are taken to be

$$\tilde{X} = span\left\{\psi_1(x,y), \psi_2(x,y), \psi_3(x,y), \ldots \psi_n(x,y)\right\}$$
$$\tilde{Y} = span\left\{\phi_1(x,y), \phi_2(x,y), \phi_3(x,y), \ldots \phi_m(x,y)\right\}$$

For $(U^n, V^n, P^n) \in \tilde{X} \times \tilde{X} \times tildeY$ we have the discrete solutions

$$U^n = \sum_{i=1}^{6} \tilde{u}_i^n \psi_i, \qquad V^n = \sum_{i=1}^{6} \tilde{v}_i^n \psi_i, \qquad P^n = \sum_{i=1}^{3} \tilde{p}_i^n \phi_i \tag{3.10}$$

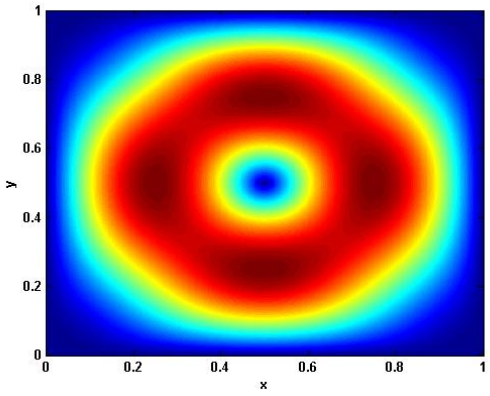
To solve this problem we first set up our momentum equation coefficient matrices M, D, B_1, B_2, F_1, and F_2 as prescribed in table (2.1). After discretizing spatially, we use a semi-implicit temporal discretization. In this case we choose a second-order Crank-Nicholson scheme where the pressure is treated with a second order Adams-Bashforth extrapolation according to [17] and [23]. The procedure is then:

$$\left\langle \frac{U^{n+1} - U^n}{\Delta t}, \psi \right\rangle + \frac{3}{2}\left\langle \frac{\partial P^n}{\partial x}, \psi \right\rangle - \frac{1}{2}\left\langle \frac{\partial P^{n-1}}{\partial x}, \psi \right\rangle$$
$$= -\frac{\nu}{2}\left\langle \nabla\left(U^{n+1} + U^n\right), \nabla\psi \right\rangle + \left\langle F_1^{n+\frac{1}{2}}, \psi \right\rangle \tag{3.11a}$$
$$\left\langle \frac{V^{n+1} - V^n}{\Delta t}, \psi \right\rangle + \frac{3}{2}\left\langle \frac{\partial P^n}{\partial y}, \psi \right\rangle - \frac{1}{2}\left\langle \frac{\partial P^{n-1}}{\partial y}, \psi \right\rangle$$
$$= -\frac{\nu}{2}\left\langle \nabla\left(V^{n+1} + V^n\right), \nabla\psi \right\rangle + \left\langle F_2^{n+\frac{1}{2}}, \psi \right\rangle \tag{3.11b}$$
$$\left\langle \nabla P^{n+1}, \nabla\phi \right\rangle = \left\langle F_1^{n+1}, \frac{\partial\phi}{\partial x} \right\rangle + \left\langle F_2^{n+1}, \frac{\partial\phi}{\partial y} \right\rangle +$$
$$\nu\left\langle \frac{\partial V^{n+1}}{\partial x} - \frac{\partial U^{n+1}}{\partial y}, n_x\frac{\partial\phi}{\partial y} - n_y\frac{\partial\phi}{\partial x} \right\rangle \tag{3.11c}$$

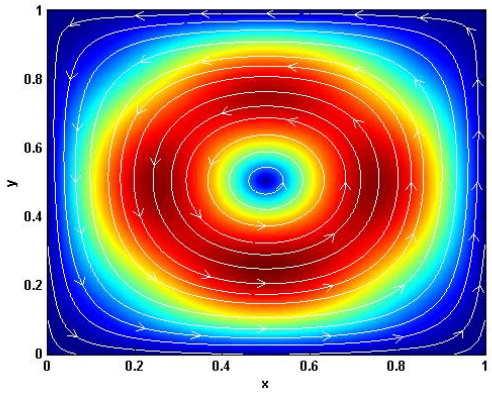
3.2 Results

For our first test we use a 1477-element, unstructured, non-uniform mesh generated by Comsol (version 3.5) and compare the results produced by the PPE formulation and Comsol to the real solutions of (3.8a)-(3.8c). Comsol uses a pressure correction scheme to solve the equations. Comsol is set to use the same P2-P1 elements we use

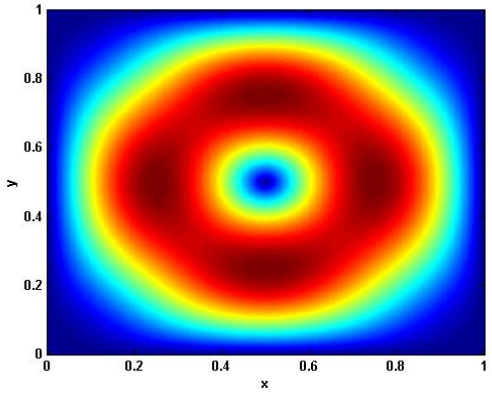
and all of the stabilization options are turned off. The time increment for the PPE is $\Delta t = .8\Delta x =$ while Comsol uses adaptive time-stepping. Both models set the first pressure node to zero and are run until $t = 1$. Figure 3.4 shows moderate velocity accuracy for the PPE, while Comsol's accuracy is an order better and is on par with the results achieved in [23] using an 80 x 80 grid with finite difference and the same time-stepping scheme. However, analysis of the pressure shows superior recovery by the PPE formulation over Comsol, though still not as accurate as seen in [23].



(a) Velocity profile of the true solution at $t = 1$, $\nu = 1$.



(b) Velocity profile computed with PPE.

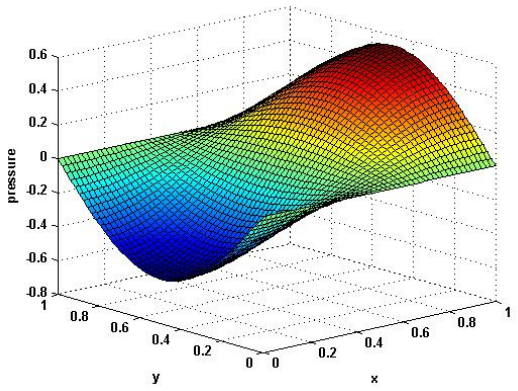


(c) Velocity profile computed with Comsol.

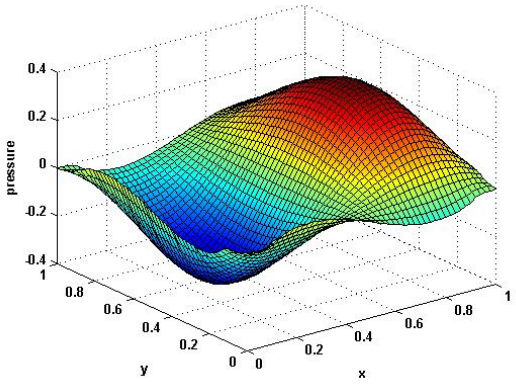
Figure 3.2: The velocity field for the computed solutions (b) and (c) have the same structure as the true solution (a). Comsol's solution, however, is more accurate.

Next, we look at the errors produced for different grid sizes and different Δt to verify the second order convergence of the finite element discretization and Crank-Nicholson/Adams-Bashforth scheme. We first varied the mesh sizes from 32 elements to 1800 elements while evolving the solutions from $t = 0$ to $t = 1$ for $\Delta t = \Delta x^2$. Figures 3.8 and 3.9 show that in meshes of less than 200 elements we see quadratic convergence, but for finer meshes other errors, such as those from the temporal discretization, become comparable. The error in pressure for all mesh sizes are unfavorable, but it is interesting to note that the error for the structured mesh of 1800 is greater than the error of the unstructured mesh of 1477. Figure 3.10 shows a steadily improving level of divergence-free flow for finer meshes, though at least a level of 10^{-4} is sought.

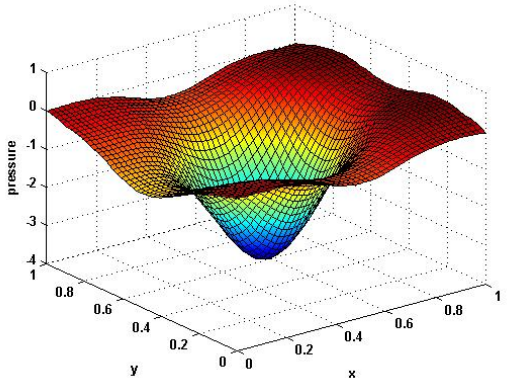
We now illustrate the second order accuracy of the Crank-Nicholson/Adams-Bashforth scheme by varying Δt while fixing the mesh size to 1800 elements and $\nu = 1$. Figure 3.11 shows that the scheme is second order up to $\Delta t = \Delta x = .03$, when the spatial and temporal errors become comparable. Figures 3.12 and Figure 3.13 show that neither the error in pressure nor the error from the divergence-free condition improve appreciably with decreasing Δt.



(a) Pressure of the true solution at $t = 1$, $\nu = 1$.



(b) Pressure computed with PPE.



(c) Pressure computed with Comsol.

Figure 3.3: The pressure field for the computed PPE solution (b) and the true solution have the same structure. Comsol's solution is completely different, although the velocity field it produces is accurate.

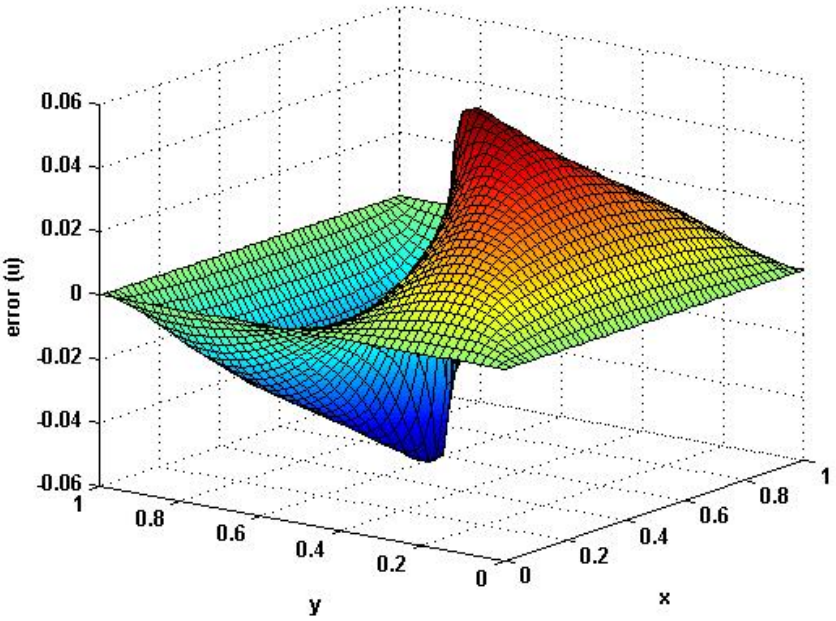


Figure 3.4: Velocity error between the PPE formulation and the real solution at $t = 1$, $\nu = 1$.

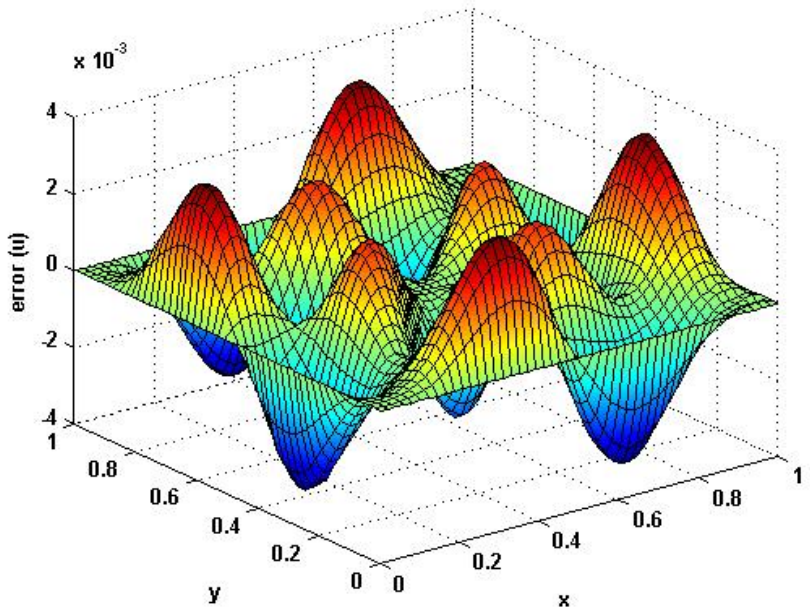


Figure 3.5: Velocity error between Comsol v3.5.3 and the real solution at $t = 1$, $\nu = 1$.

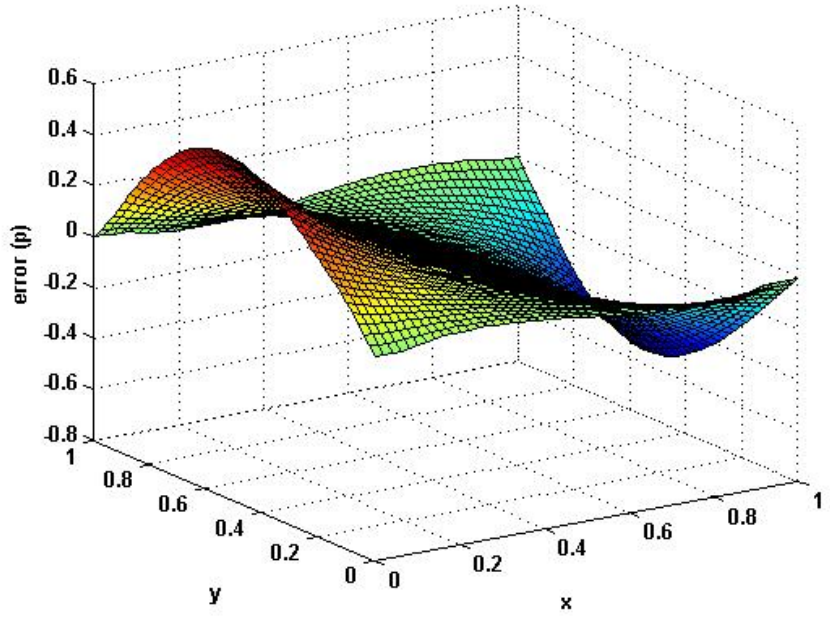


Figure 3.6: Velocity error between the PPE formulation and the real solution at $t = 1$, $\nu = 1$.

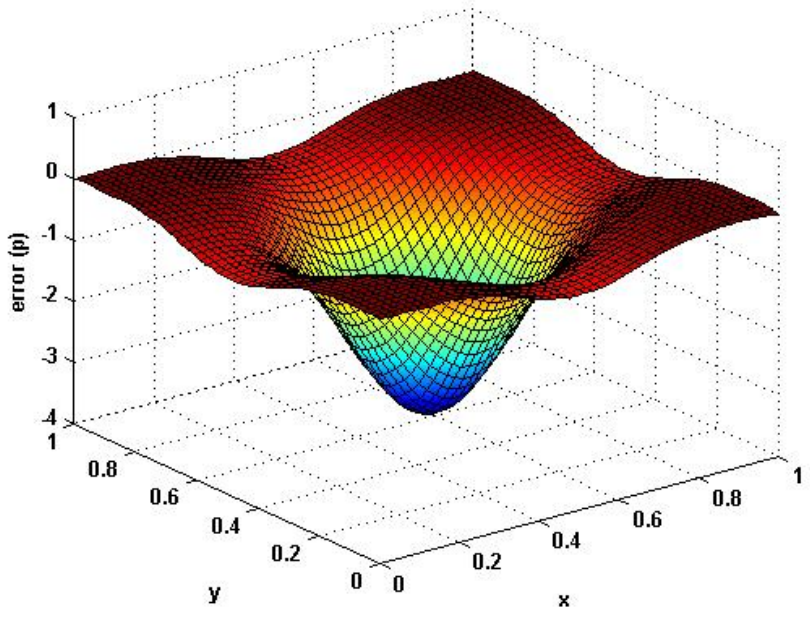


Figure 3.7: Velocity error between Comsol v3.5.3 and the real solution at $t = 1$, $\nu = 1$

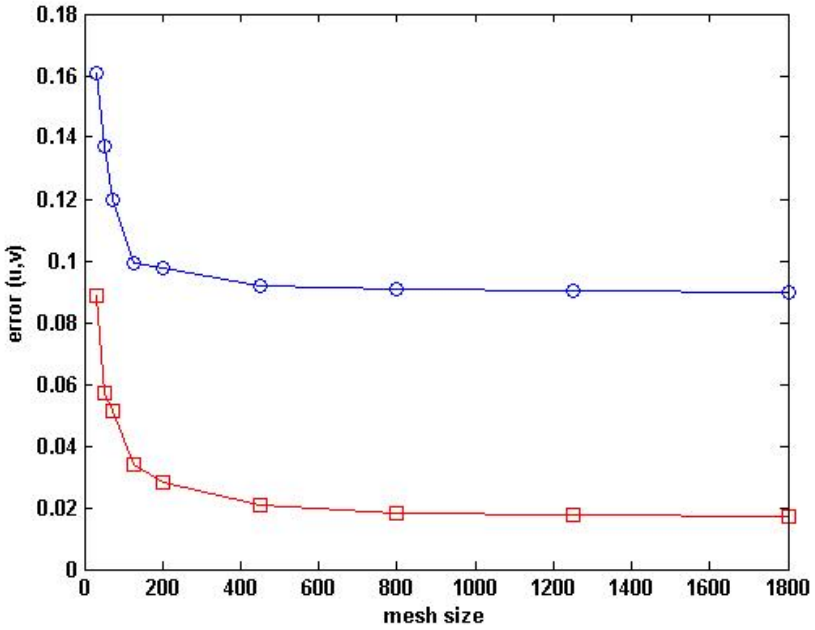


Figure 3.8: The error $\|\tilde{\mathbf{u}} - \mathbf{u}\|_\infty$. The squares ($\square$) corresponds to the horizontal velocity u while the circles ($\circ$) corresponds to the vertical velocity v.

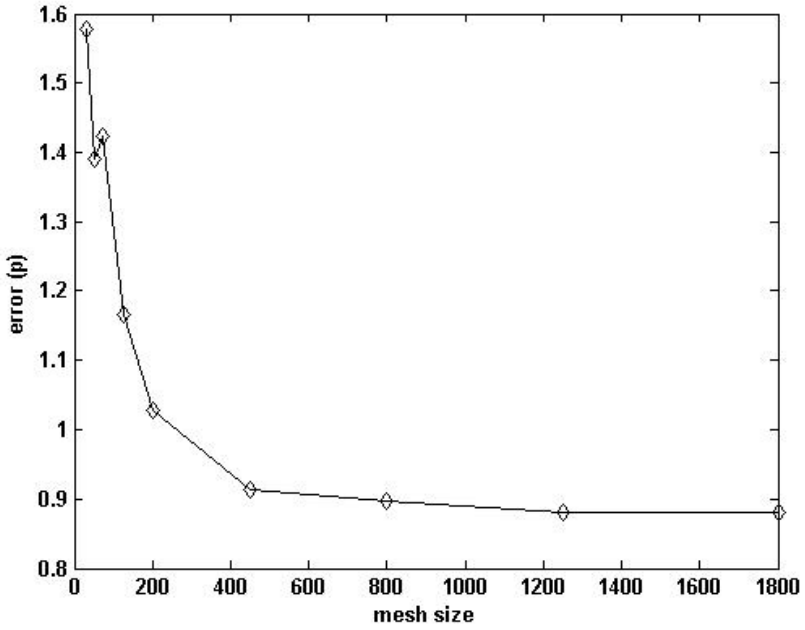


Figure 3.9: The error $\|\tilde{p} - p\|_\infty$.

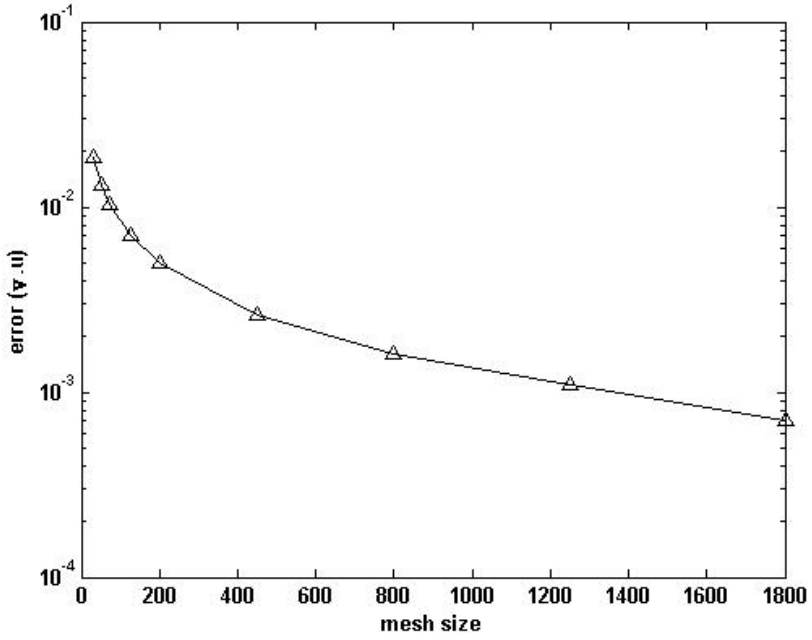


Figure 3.10: The error $\|\nabla \cdot \tilde{\mathbf{u}} - 0\|_\infty$.

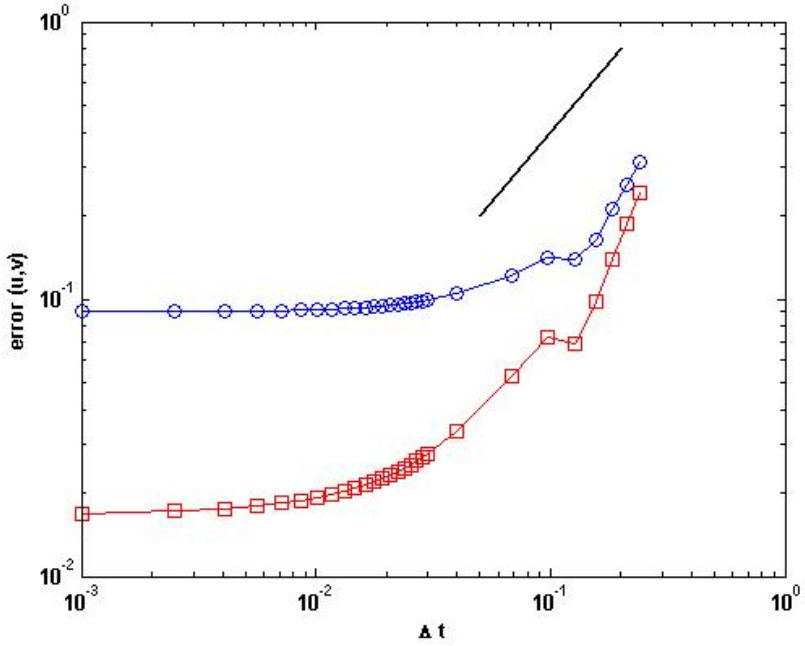


Figure 3.11: The error $\|\tilde{\mathbf{u}} - \mathbf{u}\|_\infty$. The squares ($\square$) corresponds to the horizontal velocity u while the circles ($\circ$) corresponds to the vertical velocity v. The straight line corresponds to second order convergence.

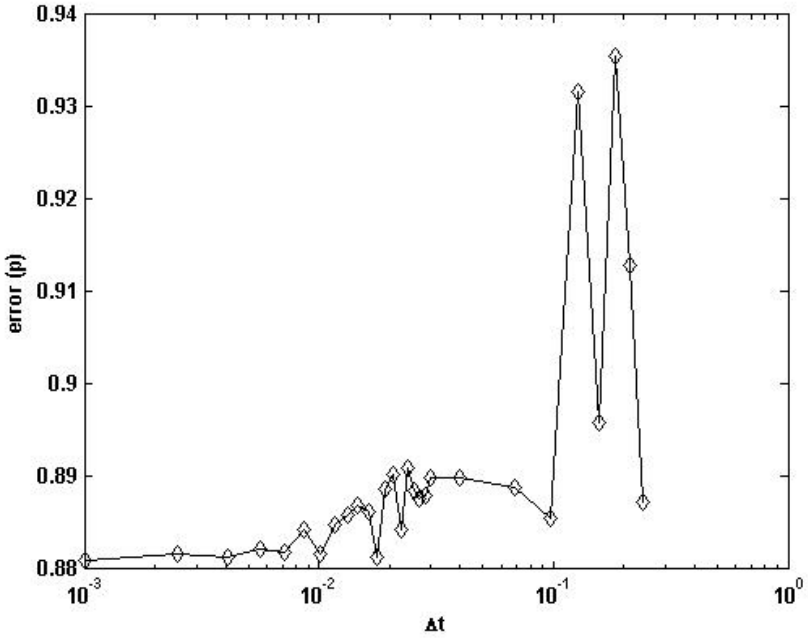


Figure 3.12: The error $\|\tilde{p} - p\|_\infty$.

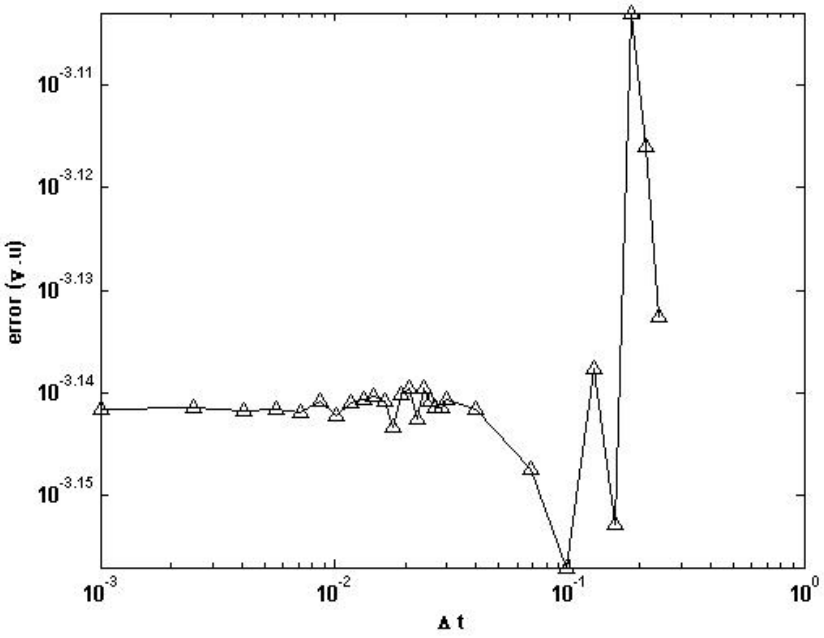


Figure 3.13: The error $\|\nabla \cdot \tilde{\mathbf{u}} - 0\|_\infty$.

CHAPTER 4

CONCLUSIONS

4.1 Overview of Results

As was expected, the implementation of the boundary conditions for the PPE proved to be the most difficult aspect of the formulation. Results showed moderate accuracy for the velocity at reasonably small meshes and times steps. However, compared to the implementation by Shirokoff and the results produced by Comsol, this method's accuracy fell short of its goal of 10^{-4}.

The accuracy for the pressure, again, fell short of the results produced by Shirokoff. However, the results computed far exceeded the results produced by Comsol. Since the same meshes were used and the same pressure node was set to zero, it is likely that the discrepancy is in the pressure correction method used by Comsol.

The finite element method was shown to be second order for the element chosen, which was in line with expectations. The Crank-Nicholson/Adams-Bashforth scheme was also shown to be second order until $\Delta t \sim \Delta x$.

4.2 Conclusion

The method demonstrated in the example clearly has promise, but just as clear is its need for some stabilizing term. Prior to this example, and omitted from this paper, was conducted a simulation of the lid driven cavity. The results in that case showed a similar phenomena with a distorted pressure with moderate accuracy in the velocity. The precise nature of the error is yet to be studied, but would be a worthwhile endeavor.

Personal correspondence with Shirokoff (over discussion of his work [23]) has enlightened us to a way forward; enforcing the no-flux and no-slip boundary con-

ditions indirectly by having the tangential velocity at the boundary approach zero ($\mathbf{n} \times \mathbf{u} = 0$) in the momentum equation. In his method, this (along with $\nabla \cdot \mathbf{u} = 0$ on the boundary in the momentum equation) is precisely his approach. The challenge for the GFEM is how to include these additional boundary conditions. By using rotational form, $\nabla \cdot \mathbf{u} = 0$ is known to be satisfied for all time. If ($\mathbf{n} \times \mathbf{u} = 0$) can be satisfied, it accounts for the no-slip condition, but the no-flux condition remains. The stabilizing term, which is added to the PPE, is meant to address the associated errors from the numerical errors. We have not yet had success employing this approach, but it remains for future work.

BIBLIOGRAPHY

[1] Bourgain, J. ; Pavlovi, N. *Ill-posedness of the NavierStokes equations in a critical space in 3D*, J. Func. Anal. **255** (2008) 2233-2247.

[2] Brezzi, F.; Fortin, F. *Mixed and Hybrid Finite Element Methods*. (1991) Springer-Verlag, New York.

[3] Burkhardt, J. *Steady incompressible Navier Stokes equations in 2D finite element solution banded storage*, http://people.sc.fsu.edu/~jburkardt/m_src/fem2d_navier_stokes/fem2d_navier_stokes.html, accessed on 12/12/2012.

[4] Chorin, A.J. *Numerical solution of the Navier-Stokes equations*, Math. Comp. **22** (1968), 745-762.

[5] Donea, J.; Huerta, A. *Finite element methods for flow problems*. (2003) Wiley, London.

[6] Ervin, V.J.; Jenkins, E.W. *The LBB condition for the Taylor-Hood P2-P1 and Scott-Vogelius P2-discP1 element pairs in 2-D*, Technical Report TR2011_04_EJ, Clemson University,(2011).

[7] Girault, V.; Raviart, P.-A. *Finite element methods for Navier-Stokes equations. Theory and algorithms*. (1987) Springer-Verlag, Berlin.

[8] Gresho, P.M.; Sani, R.L. *On pressure boundary conditions for the incompressible Navier-Stokes equations*, Int. J. Numer. Methods Fluids **7** (1987), 1111-1145.

[9] Gresho, P.M.; Sani, R.L.; Engelman, M.S. *Incompressible flow and the finite element method : advection-diffusion and isothermal laminar flow*. (1999) Wiley.

[10] Harlow, F.H.; Welch, J.E. *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, Phys. of Fluids **8** (1965), 2182-2189.

[11] Hassanzadeh, S.; Sonnad, V.; Foresti, S. *Finite element implementation of boundary conditions for the pressure Poisson equation of incompressible flow*, Int. J. Numer. Methods Fluids **18** (1994), 1009-1019.

[12] Heinrich, J.C. ; Vionnet, C.A. *The penalty method for the Navier-Stokes equations*, Arch. Comp. Meth. Eng. **2** (1995) 51-65.

[13] Hopf, E. *Über die anfang swetaufgabe für die hydrodynamischer grundgleichungan*, Math. Nach. **4** (1951) 213-231.

[14] Jia, H.; Šverák, V. *Local-in-space estimates near initial time for weak solutions of the Navier-Stokes equations and forward self-similar solutions*, arXiv:1204.0529 (2012).

[15] Jia, H.; Šverák, V. *On scale-invariant solutions of the Navier-Stokes equations*, Proc. 6ECM (2012).

[16] Johnston, H.; Liu, J.-G. *Finite difference schemes for incompressible flow based on local pressure boundary conditions*, J. Comput. Phys. **180** (2002), no. 1, 120-154.

[17] Johnston, H.; Liu, J.-G. *Accurate, stable, and efficient Navier-Stokes solvers based on explicit treatment of the pressure term*, J. Comput. Phys. **199** (2004), no. 1, 221-259.

[18] Koch, H. ; Tataru, D. *Well-posedness for the Navier-Stokes equations*, Adv. Math. **157** (2001) Krakow.

[19] Ladyzhenskaya, O. *The mathematical theory of viscous incompressible flow.* (1969) Gordon and Breach, New York.

[20] Leray, J. *Sur le mouvement d'um liquide visqieux emlissant l'space*, Acta Math. **63** (1934) 193-248.

[21] Rannacher, R. *Finite element methods for the incompressible Navier-Stokes equations*, Fund. Dir. Math. Fluid Mech. (2000) 191-293.

[22] Sani, R.L.; Shen, J.; Pironneau, O.; Gresho, P.M. *Pressure boundary condition for the time-dependent incompressible Navier-Stokes equations*, Int. J. Numer. Methods Fluids **50** (2006), 673-682.

[23] Shirkoff, D.; Rosales, R. R. *An efficient method for the incompressible Navier-Stokes equations on irregular domains with no-slip boundary conditions, high order up to the boundary*, J. Comput. Phys. **230** (2011), no. 23, 8619-8646.

[24] Strikwerda, J.C. *Finite difference schemes and partial differential equations.* (1989) Wiley.

[25] Taylor, C.; Hood, P. *A numerical solution of the Navier-Stokes equations using the finite element technique*, Comp. and Fluids **1** (1973), 73-100.

[26] Temam, R. *Sur lapproximation de la solution des equations de Navier-Stokes par la methode des fractionnarires II*, Arch. Rational Mech. Anal. **33** (1969), 377-385.

[27] Temam, R. *Navier-Stokes equations and nonlinear functional analysis.* (1995) SIAM, Philadelphia.

[28] Turek, S. *Efficient solvers for incompressible flow problems: an algorithmic approach.* (1999) Springer-Verlag, Berlin.

Appendix A

MATLAB CODE

The following code is in part adapted from John Burkardt [3] of Florida State University under the GNU license. In particular, full use is made of his basis functions, quadrature, and mapping functions.

```
function [u,v,p] = boxflow(total_time,nu,Lambda,...
    solver_type,save_option)
%************ START MAIN FUNCTION ****************
% INPUT     total_time: amount of time to be evaluated
%           nu: viscosity
%           Lambda: stability parameter (0-100)
%           Solver_type: 1-Backward Euler, 2-Crank-Nicolson
%           save_option: optional; name to save workspace
% OUTPUT    u, v: horizontal and vertical velocity
%           p: pressure

quad_num = 7;% gauss quadrature rule; only 7 is allowed
Re = 1/nu; %Reynolds number

element_node = dlmread('cavity_elements_72.txt');
node_xy = dlmread('cavity_coords_72.txt');

% determines normal vectors, node numbering, boundaries,
% various element info
[pres,pres_num,node_num,element_num,node_boundary,dt,...
    steps,normal_node,variable_num, node_u_variable,...
```

```
    node_v_variable,node_p_variable]...
    = Set_Elements(total_time,element_node,node_xy);

% Compute mass, stiffness, and gradient matrices
[M,D,GP] = Set_Matrix(quad_num,element_node,node_xy,...
    node_num,node_u_variable, node_v_variable,...
    node_p_variable,element_num,pres_num);

im  = M\speye(2*node_num,2*node_num); %inverse M
Mp = GP'*im*GP; % laplace operator for pressure
pbound = logical(node_boundary(pres));
removebnd = Mp(:,pbound);
Mp(pbound,:) = 0;
Mp(:,pbound) = 0;
Mp(pbound,pbound) = diag(ones(nnz(node_boundary(pres)),1));
%
% *************** Initial Conditions *************
%
% initial velocity. initial pressure is solved for
[u v ~] = external_force(node_xy,0,1,Re);
node_c = [u'; v'; zeros(pres_num,1)];

% solve for initial pressure
[Ku Kv] = Set_PPE_RHS(Re,quad_num, element_node,...
    node_xy,node_num, node_p_variable,node_u_variable,...
    element_num, pres_num, normal_node,Lambda); %RHS PPE
```

```
% compute external force funtion for RHS of momentum
[~,~,F3] = rhs(0,dt,node_num,node_u_variable,...
    node_v_variable,node_p_variable, pres_num,element_num,...
    element_node,quad_num,node_xy,Re);
 K1 = Ku*node_c(1:node_num) + Kv*node_c(node_num+1:2*...
     node_num); %PPE RHS
 K2 = removebnd * K1(pbound); %adjust boundary
 K = F3-K2;%adjust boundary
 K(pbound) = K1(pbound);%adjust boundary
 K(1) = 0; % set first pressure node to zero
P=Mp\K;
node_c(2*node_num+1:variable_num,1) = P;
%
% ********** END Initial Conditions **********
%
% ********** Solve System **********%
if(solver_type == 1)
    node_c = Backward_Euler(Re,M,D,GP,Mp,Ku,Kv,node_c,...
    removebnd,pbound,node_boundary,steps,dt,variable_num,...
    element_node,node_xy,quad_num,node_num,element_num,...
    pres_num,node_p_variable,node_u_variable,...
    node_v_variable);
else
    [node_c] = Crank_Nick(Re,M,D,GP,Mp,Ku,Kv,node_c,...
     removebnd,pbound,node_boundary,steps,dt,variable_num,...
```

```
    element_node, node_xy, quad_num, node_num, element_num, ...
    pres_num, node_p_variable, node_u_variable, ...
    node_v_variable);
end

u = node_c(1:node_num);
v = node_c(1+node_num:2*node_num);
p = node_c(1+2*node_num:end);

%post-processing
plotdata(u,v,p,node_xy,node_num,pres,steps,dt,Re)

% Save data if argument exists
if(nargin == 5)
    if(ischar(save_option))
        save(save_option)
    end
end
% ****************** END Main Program ****************
end

function [pres,pres_num,node_num,element_num,...
    node_boundary,dt,steps,normal_node,variable_num,...
    node_u_variable,node_v_variable,node_p_variable]...
    = Set_Elements(total_time,element_node,node_xy)
```

```
pres = unique(element_node(1:3,:)); %pressure nodes
pres_num = length(pres); % number of pressure nodes
node_num = max(max(element_node)); %total number of nodes
element_num = size(element_node,2); %number of elements
node_boundary=triangulation_order6_boundary_node(node_num,...
    node_xy', element_node );

%% *** Determine Time Step ****
del_x = min_dist(node_xy,element_node);
dt = .95*del_x^2;
steps = ceil(total_time/dt)-1;
%% **** END TIME STEP ******
max_x = max(node_xy(1,:));
min_x = min(node_xy(1,:));
max_y = max(node_xy(2,:));
min_y = min(node_xy(2,:));

%******* NORMAL VECTORS *********
normal_node = zeros(2,node_num);
normvec = (node_xy(1,:) == max_x);
normal_node(1,:) = normvec;
normvec = (node_xy(1,:) == min_x);
normal_node(1,:) = normal_node(1,:) - normvec;
normvec = (node_xy(2,:) == max_y);
normal_node(2,:) = normvec;
```

```
normvec = (node_xy(2,:) == min_y);
normal_node(2,:) = normal_node(2,:) - normvec;
normvec = abs(normal_node(1,:)) + abs(normal_node(2,:));
normal_node(:,normvec==2) = 0;
normvec = find(node_xy(1,:) == max_x & node_xy(2,:)...
    == max_y);
normal_node(:,normvec) = [1  1]';
normvec = find(node_xy(1,:) == min_x & node_xy(2,:)...
    == max_y);
normal_node(:,normvec) = [-1  1]';
normvec = find(node_xy(1,:) == min_x & node_xy(2,:)...
    == min_y);
normal_node(:,normvec) = [-1  -1]';
normvec = find(node_xy(1,:) == max_x & node_xy(2,:)...
    == min_y);
normal_node(:,normvec) = [1  -1]';
%******* END NORMAL VECTORS *********

%****** number variables *********
    variable_num = 2*node_num+pres_num;
    node_u_variable = 1:1:node_num;
    node_v_variable = node_num+1:1:2*node_num;
    node_p_variable =zeros(1,node_num);
    node_p_variable(pres) = 1+2*node_num:1:variable_num;
  return
end
```

```
% the external force F
function [u v p] = external_force(xy,t,type,Re)
x = xy(1,:);
y = xy(2,:);

ct = cos(t); st=sin(t);
s2py = sin(2*pi*y); spx = sin(pi*x);
s2px = sin(2*pi*x);spiy = sin(pi*y);
cpx = cos(pi*x); c2px = cos(2*pi*x);
cpy = cos(pi*y); c2py = cos(2*pi*y);

if(type == 1)
    u = pi*ct*s2py.*spx.^2;
    v = -pi*ct*s2px.*spiy.^2;
    p = -ct*cpx.*spiy;

    dudx = 2*pi*ct.*s2py.*spx.*cpx;
    dvdy = -2*pi*ct.*s2px.*spiy.*cpy;
    divu = norm(dudx +dvdy,inf);
else

    ut = -pi*st*s2py.*spx.^2;
    vt = pi*st*s2px.*spiy.^2;
    lapu = 2*pi^3*ct*s2py.*(cpx.^2-spx.^2)...
        -4*pi^3*ct.*s2py.*spx.^2;
```

```
    lapv = 4*pi^3*ct*s2px.*spiy.^2 ...
        - 2*pi^3*ct*s2px.*(cpy.^2-spiy.^2);
    px = pi*ct*spx.*spiy;
    py = -pi*ct*cpx.*cpy;

  u = ut + px - (1/Re)*lapu;
  v = vt + py - (1/Re)*lapv;
  p = 0;
end
return
end
% compute mass, diffusion, and gradient matrices
function [M,D,GP] = Set_Matrix(quad_num,element_node,...
    node_xy, node_num,node_u_variable,...
    node_v_variable,node_p_variable,element_num,pres_num)

vec_len = element_num*36;
mu = zeros(vec_len,3);
mv = mu;
uu = mu;
vv = mu;
row_cnt = 1;

vec_len = element_num*18;
pu = zeros(vec_len,3);
pv = pu;
```

```
row_cnt2 = 1;
velocity_dof = 2*node_num;


[ quad_w, quad_xy ] = quad_rule ( quad_num );


 for element = 1 : element_num
%
%  Make a copy of the triangle.
%
    t3(1:2,1:3) = node_xy(1:2,element_node(1:3,element));
    t6(1:2,1:6) = node_xy(1:2,element_node(1:6,element));
%
%  Map the quadrature points QUAD_XY to points XY in
% the physical triangle.
    xy(1:2,1:quad_num) = reference_to_physical_t6(t6,...
        quad_num, quad_xy );
    area = abs ( triangle_area_2d ( t3 ) );
    w(1:quad_num) = area * quad_w(1:quad_num);
%
%  Evaluate the basis functions at the quadrature points.
%
    [ Psi, dPsidx, dPsidy ] = basis_mn_t6(t6, quad_num,...
        xy );
    [ ~, dPhidx, dPhidy] = basis_mn_t3(t3,quad_num,xy);
```

```
iu(1:6) = node_u_variable(element_node(1:6,element));
iv(1:6) = node_v_variable(element_node(1:6,element));
ip(1:3) = node_p_variable(element_node(1:3,element))...
    -velocity_dof;

for i = 1:6
    for j = 1:6
        % mass matrix
        mu(row_cnt,1:2) = [iu(i),iu(j)];
        mu(row_cnt,3) = sum(w(1:quad_num) .* ...
            Psi(i,1:quad_num) .* Psi(j,1:quad_num));
        mv(row_cnt,1:2) = [iv(i),iv(j)];
        mv(row_cnt,3) = mu(row_cnt,3);

        % stiff matrix
        uu(row_cnt,1:2) = [iu(i),iu(j)];
        uu(row_cnt,3) = sum(w(1:quad_num) .*...
            (dPsidx(i,1:quad_num) .* ...
            dPsidx(j,1:quad_num)+dPsidy(i,1:quad_num)...
            .* dPsidy(j,1:quad_num)));
        vv(row_cnt,1:2) = [iv(i),iv(j)];
        vv(row_cnt,3) = uu(row_cnt,3);

        if i < 4
            % gradient of P
            pu(row_cnt2,1:2) = [iu(j),ip(i)];
```

```
                pu(row_cnt2,3) = sum(w(1:quad_num)...
                    .* dPhidx(i,1:quad_num) .*...
                    Psi(j,1:quad_num));
                pv(row_cnt2,1:2) = [iv(j),ip(i)];
                pv(row_cnt2,3) = sum(w(1:quad_num)...
                    .* dPhidy(i,1:quad_num) .*...
                    Psi(j,1:quad_num));
                row_cnt2 = row_cnt2 + 1;
            end
            row_cnt = row_cnt + 1;
        end
    end

end


mu = mu(any(mu(:,3),2),:);
mv = mv(any(mv(:,3),2),:);
mu = [mu;mv];
M = sparse(mu(:,1),mu(:,2),mu(:,3),2*node_num,2*node_num);

uu = uu(any(uu(:,3),2),:);
vv = vv(any(vv(:,3),2),:);
uu = [uu;vv];
D = sparse(uu(:,1),uu(:,2),uu(:,3),2*node_num,2*node_num);
```

```
pu = pu(any(pu(:,3),2),:);
pv = pv(any(pv(:,3),2),:);
pu = [pu;pv];
GP = sparse(pu(:,1),pu(:,2),pu(:,3),2*node_num,pres_num);


 return;
end
% Backward Euler method
function [node_c] = Backward_Euler(Re,M,D,GP,Mp,Ku,Kv,...
     node_c,removebnd,pbound,node_boundary,steps,dt,...
     variable_num,element_node,node_xy,quad_num,node_num,...
     element_num,pres_num,node_p_variable,...
     node_u_variable,node_v_variable)

boundary_size = nnz([node_boundary,node_boundary]);
lbound = logical([node_boundary,node_boundary]');
Lhs = (M+(dt/Re)*D); % pre-solve LHS momentum eq.
Lhs(lbound,:) = 0; %adjust boundary
Lhs(:,lbound) = 0; %adjust boundary
Lhs(lbound,lbound) = diag(ones(boundary_size,1));%adj b.c.

for time  = 1:steps
     tic
 %compute external force funtion for RHS of momentum
 [F,~,F3] = rhs(time*dt,dt,node_num,node_u_variable,...
      node_v_variable,node_p_variable,pres_num,...
```

```
        element_num,element_node,quad_num,node_xy,Re);

  Rhs = M*node_c(1:2*node_num) -dt*GP...
        *node_c(1+2*node_num:end) + dt*F;
  Rhs(lbound) = 0; %adjust boundary

node_c(1:2*node_num) = Lhs\Rhs;

% Solve PPE
  K1 =Ku*node_c(1:node_num)+Kv*node_c(node_num+1:2*node_num);
  K2 = removebnd * K1(pbound);
  K = F3-K2;
  K(pbound) = K1(pbound);
  K(1) = 0;
P=Mp\K;
node_c(2*node_num+1:variable_num,1) = P;

  tm = toc;
  fprintf('step %i of %i completed in %3.4f s \n', time,...
      steps,toc)
end
return
end
% Crank-Nickolson/Adams-Bashforth method
function [node_c] = Crank_Nick(Re,M,D,GP,Mp,Ku,Kv,node_c,...
      removebnd,pbound,node_boundary,steps,dt,variable_num,...
```

```
    element_node, node_xy, quad_num, node_num, element_num, ...
    pres_num, node_p_variable, node_u_variable, node_v_variable)

solution = zeros(variable_num,2); %current and previous

boundary_size = nnz([node_boundary, node_boundary]);
lbound = logical([node_boundary, node_boundary]');

solution(:,2) = node_c;
        Lhs2 = (M+.5*dt/Re*D);% pre-solve LHS momentum eq.
        Lhs2(lbound,:) = 0;%adjust boundary
        Lhs2(:,lbound) = 0;%adjust boundary
        Lhs2(lbound,lbound) = diag(ones(boundary_size,1));
        Lhs2 = Lhs2\speye(2*node_num,2*node_num);
for time = 1:steps
tic
   if(time == 1)
      [F,~,F3] = rhs(time*dt,dt,node_num,node_u_variable,...
          node_v_variable,node_p_variable,pres_num,...
          element_num,element_node,quad_num,node_xy,Re);

          Rhs = M*node_c(1:2*node_num) -dt*GP...
              *node_c(1+2*node_num:end) + dt*F;
          Rhs(lbound) = 0;%adjust boundary
          Lhs = (M+(dt/Re)*D);% pre-solve LHS momentum eq.
          Lhs(lbound,:) = 0;%adjust boundary
```

```
            Lhs(:,lbound) = 0;%adjust boundary
            Lhs(lbound,lbound) = diag(ones(boundary_size,1));
            node_c(1:2*node_num) = Lhs\Rhs;
    else
        [F,F2,F3] = rhs(time*dt,dt,node_num,...
          node_u_variable,node_v_variable,node_p_variable,...
          pres_num,element_num,element_node,quad_num,node_xy,Re);
        F = .5*(F+F2);
          Rhs = (M-.5*dt/Re*D)*node_c(1:2*node_num,1)... %u_n
              +dt*GP*(-1.5*node_c(1+2*node_num:end)...
              +.5*solution(1+2*node_num:end,1))+dt*F;
         Rhs(lbound) = 0;
          node_c(1:2*node_num,1) = Lhs2*Rhs;
    end

 % Solve PPE
 K1 = Ku*node_c(1:node_num) ...
      + Kv*node_c(node_num+1:2*node_num);
 K2 = removebnd * K1(pbound);
 K = F3-K2;
 K(pbound) = K1(pbound);
 K(1) = 0;
P=Mp\K;
node_c(2*node_num+1:variable_num,1) = P;

 solution = [solution(:,2),node_c];
```

```
 tm = toc;
 fprintf('step %i of %i completed in %3.4f s \n',...
      time,steps,toc)

end
return
end

function [Ku Kv] = Set_PPE_RHS(Re,quad_num,element_node,...
     node_xy,node_num, node_p_variable,node_u_variable,...
     element_num, pres_num, normal_node,Lambda)

vec_len = element_num*18;
uu = zeros(vec_len,3);
vv = uu;
row_cnt = 1;
velocity_dof = 2*node_num;

%
%  Get the quadrature weights and nodes.
%
  [ quad_w, quad_xy ] = quad_rule ( quad_num );
%
%  Consider all quantities associated with a given ELEMENT.
%
  for element = 1 : element_num
```

```
%
%  Make a copy of the triangle.
%

    t3(1:2,1:3) = node_xy(1:2,element_node(1:3,element));
    t6(1:2,1:6) = node_xy(1:2,element_node(1:6,element));
%
%  Map the quad points QUAD_XY to
%  points XY in the physical triangle.
    xy(1:2,1:quad_num) = reference_to_physical_t6(t6,...
        quad_num, quad_xy );
    area = abs ( triangle_area_2d ( t3 ) );
    w(1:quad_num) = area * quad_w(1:quad_num);
%
%  Evaluate the basis functions at the quadrature points.
    [ Psi, dpsidx, dpsidy ] = basis_mn_t6 ( t6, ...
        quad_num, xy );
    [ Phi, dphidx, dphidy ] = basis_mn_t3 ( t3, ...
        quad_num, xy );

%  Extract the node indices  for this element.
    iu(1:6) = node_u_variable(element_node(1:6,element));
    ip(1:3) = node_p_variable(element_node(1:3,element))...
        -velocity_dof;
%  normal vector for each node in the element
     normal_vec = normal_node(:,element_node(1:6,element));
```

```
% compute <curl(u), n * grad(Phi)> + lambda<n * u,Phi>
    for i = 1:3
        for j = 1:6
          uu(row_cnt,1:2) = [ip(i),iu(j)];
          dphidtau = normal_vec(1,i)*dphidy(i,1:quad_num)...
              -normal_vec(2,i)*dphidx(i,1:quad_num);
          uu(row_cnt,3) = sum(w(1:quad_num).* (-dphidtau)...
              .* dpsidy(j,1:quad_num) + Lambda...
              *normal_vec(1,j)*Psi(j,1:quad_num)...
              .*Phi(i,1:quad_num));
          vv(row_cnt,1:2) = [ip(i),iu(j)];
          vv(row_cnt,3) = sum(w(1:quad_num).* dphidtau ...
              .* dpsidx(j,1:quad_num)+ Lambda...
              *normal_vec(2,j)*Psi(j,1:quad_num)...
              .*Phi(i,1:quad_num));
          row_cnt = row_cnt + 1;
        end
    end
  end

uu = uu(any(uu(:,3),2),:);
vv = vv(any(vv(:,3),2),:);
Ku = (1/Re)*sparse(uu(:,1),uu(:,2),uu(:,3),pres_num,node_num);
Kv = (1/Re)*sparse(vv(:,1),vv(:,2),vv(:,3),pres_num,node_num);
```

```
  return
end

function [F1,F2,F3] = rhs (t,dt,node_num,node_u_variable,...
    node_v_variable,node_p_variable,pres_num,element_num,...
    element_node,quad_num,node_xy,Re)

F1 = zeros(2*node_num,1); % F at time = t
F2 = F1;% F at time = t + .5dt
F3 = zeros(pres_num,1);% F at time = t + dt
[ quad_w, quad_xy ] = quad_rule ( quad_num );

%
%  Consider all quantities associated with a given ELEMENT.
%
for element = 1 : element_num
%
%  Make a copy of the triangle.
%
    t3(1:2,1:3) = node_xy(1:2,element_node(1:3,element));
    t6(1:2,1:6) = node_xy(1:2,element_node(1:6,element));
%
%  Map the quadrature points QUAD_XY to
%  points XY in the physical triangle.
%
    xy(1:2,1:quad_num) = reference_to_physical_t6(t6,...
```

```
        quad_num, quad_xy );
    area = abs ( triangle_area_2d ( t3 ) );
    w(1:quad_num) = area * quad_w(1:quad_num);

    iu(1:6) = node_u_variable(element_node(1:6,element));
    iv(1:6) = node_v_variable(element_node(1:6,element));
    ip(1:3) = node_p_variable(element_node(1:3,element))...
        -2*node_num;
%
%  Evaluate the basis functions at the quadrature points.
%
    [ Psi, ~, ~ ] = basis_mn_t6 ( t6, quad_num, xy );
    [ ~, dphidx, dphidy ] = basis_mn_t3 ( t3, quad_num, xy );

    %evaluate forcing function
[u_rhs,v_rhs,~] = external_force(xy,t,2,Re);
[u_rhs2,v_rhs2,~] = external_force(xy,t+.5*dt,2,Re);
[u_rhs3, v_rhs3 ~] = external_force(xy,t+dt,2,Re);

  % weak form
  for i = 1:6
    F1(iu(i)) = F1(iu(i)) + sum(w.*u_rhs(1:quad_num)...
        .* Psi(i,1:quad_num));
    F1(iv(i)) = F1(iv(i)) + sum(w.*v_rhs(1:quad_num)...
        .* Psi(i,1:quad_num));
    F2(iu(i)) = F2(iu(i)) + sum(w.*u_rhs2(1:quad_num)...
```

```
            .* Psi(i,1:quad_num));
        F2(iv(i)) = F2(iv(i)) + sum(w.*v_rhs2(1:quad_num)...
            .* Psi(i,1:quad_num));
        if i<4
            F3(ip(i)) = F3(ip(i))+sum(w.*(u_rhs3(1:quad_num)...
                .* dphidx(i,1:quad_num) + v_rhs3(1:quad_num)...
                .* dphidy(i,1:quad_num)));
        end
    end
end

    return
end

function [ quad_w, quad_xy ] = quad_rule ( quad_num )

%****************************************************
%% QUAD_RULE sets the quadrature rule for assembly.
%     The quadrature rule is given for a reference element,
%     points (X,Y) such that
%       0 <= X,
%       0 <= Y, and
%       X + Y <= 1.
%
%       ^
%     1 | *
```

```
%       |  |\
%    Y  |  |  \
%       |  |   \
%    0  |  *---*
%       +------->
%          0  X  1
%
%  Licensing:
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%    17 July 2005
%
%  Author:
%    John Burkardt
%
%  Parameters:
%    Input, integer QUAD_NUM, the number of quadrature nodes.
%    Legal values are  7
%
%    Output, real QUAD_W(QUAD_NUM), the quadrature weights.
%
%    Output, real QUAD_XY(2,QUAD_NUM), the quadrature nodes.
%
  if ( quad_num == 7 )
```

```
  a = 1.0 / 3.0;
  b = ( 9.0 + 2.0 * sqrt ( 15.0 ) ) / 21.0;
  c = ( 6.0 -       sqrt ( 15.0 ) ) / 21.0;
  d = ( 9.0 - 2.0 * sqrt ( 15.0 ) ) / 21.0;
  e = ( 6.0 +       sqrt ( 15.0 ) ) / 21.0;
  u = 0.225;
  v = ( 155.0 - sqrt ( 15.0 ) ) / 1200.0;
  w = ( 155.0 + sqrt ( 15.0 ) ) / 1200.0;

  quad_xy(1:2,1:quad_num) = [ ...
    a, a; ...
    b, c; ...
    c, b; ...
    c, c; ...
    d, e; ...
    e, d; ...
    e, e ]';

  quad_w(1:quad_num) = [ u, v, v, v, w, w, w ];
else
fprintf(1,'\n' );
fprintf(1,'QUAD_RULE - Fatal error!\n' );
fprintf(1,...
    'No rule is available of order QUAD_NUM = %d\n',...
    quad_num );
  error ( 'QUAD_RULE - Fatal error!\n' );
```

```
  end

  return
end

function phy = reference_to_physical_t6 ( t, n, ref )

%****************************************************
%
%% REFERENCE_TO_PHYSICAL_T6 maps T6 reference points
%  to physical points.
%  Discussion:
%
%    Given the vertices of an order 6 physical triangle
%     and a point (XSI,ETA) in the reference triangle,
%    the routine computes the value of the corresponding
%    image point (X,Y) in physical space.
%    The mapping from (XSI,ETA) to (X,Y) has the form:
%
%    X(ETA,XSI) = A1 * XSI**2 + B1 * XSI*ETA + C1 * ETA**2
%                 + D1 * XSI     + E1 * ETA      + F1
%
%    Y(ETA,XSI) = A2 * XSI**2 + B2 * XSI*ETA + C2 * ETA**2
%                 + D2 * XSI     + E2 * ETA      + F2
%
```

```
%  Reference Element T6:
%
%    |
%    1  3
%    |  |\
%    |  | \
%    S  6  5
%    |  |   \
%    |  |    \
%    0  1--4--2
%    |
%    +--0--R--1-->
%
%  Licensing:
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%    26 June 2005
%
%  Author:
%    John Burkardt
%
%  Parameters:
%  Input, real T(2,6), the coordinates of the vertices.
%  The vertices are assumed to be the images of (0,0), (1,0),
%  (0,1),(1/2,0), (1/2,1/2) and (0,1/2) respectively.
```

```
%
%  Input, integer N, the number of objects to transform.
%
%  Input, real REF(2,N), points in the reference triangle.
%
%    Output, real PHY(2,N), corresponding points in the
%    physical triangle.
%
  for i = 1 : 2

    a(i) =   2.0 * t(i,1) + 2.0 * t(i,2)       ...
           - 4.0 * t(i,4);

    b(i) =   4.0 * t(i,1)                      ...
           - 4.0 * t(i,4) + 4.0 * t(i,5) - 4.0 * t(i,6);

    c(i) =   2.0 * t(i,1)                + 2.0 * t(i,3) ...
                                         - 4.0 * t(i,6);

    d(i) = - 3.0 * t(i,1) -       t(i,2)                ...
           + 4.0 * t(i,4);

    e(i) = - 3.0 * t(i,1)                -       t(i,3) ...
                                         + 4.0 * t(i,6);
    f(i) =         t(i,1);
```

```
  end

  for i = 1 : 2
    phy(i,1:n) = a(i) * ref(1,1:n) .* ref(1,1:n) ...
               + b(i) * ref(1,1:n) .* ref(2,1:n) ...
               + c(i) * ref(2,1:n) .* ref(2,1:n) ...
               + d(i) * ref(1,1:n) ...
               + e(i) * ref(2,1:n) ...
               + f(i);
  end

  return
end

function area = triangle_area_2d ( t )

%****************************************************
%% TRIANGLE_AREA_2D computes the area of a triangle in 2D.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%    28 January 2005
%
```

```
%  Author:
%    John Burkardt
%
%  Parameters:
%    Input, real T(2,3), the triangle vertices.
%    Output, real AREA, the absolute area of the triangle.
%
  area = 0.5 * abs ( ...
      t(1,1) * ( t(2,2) - t(2,3) ) ...
    + t(1,2) * ( t(2,3) - t(2,1) ) ...
    + t(1,3) * ( t(2,1) - t(2,2) ) );

  return
end

function [ phi, dphidx, dphidy, area ]=basis_mn_t6(t,n,p)
%**************************************************
%
%% BASIS_MN_T6: all bases for N points in a T6 element.
%
%  Discussion:
%
%    The routine is given the coordinates of the vertices
%    and midside nodes of a triangle.  It works directly
%    with these coordinates, and does not refer to a
%    reference element.
```

```
%    This routine requires that the midside nodes be
%    "in line" with the vertices, that is, that the sides
%    of the triangle be straight.  However, the midside
%    nodes do not actually have to be halfway along the
%    side of the triangle.
%  Physical element T6:
%    This picture indicates the assumed ordering of the
%     six nodes of the triangle.
%
%              3
%             / \
%            /   \
%           6     5
%          /       \
%         /         \
%        1-----4-----2
%
%  Licensing:
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%    17 February 2006
%
%  Author:
%    John Burkardt
%
```

```
%  Parameters:
%Input, real T(2,6), the nodal oordinates of the element.
%It is common to list these points in counter clockwise order.
%
%    Input, real P(2,N), the evaluation points.
%
%    Output, real PHI(6,N), the basis functions at the
%    evaluation points.
%    Output, real DPHIDX(6,N), DPHIDY(6,N), the basis
%    derivatives at the evaluation points.
%
%  Local Parameters:
%    Local, real AREA, is (twice) the area of the triangle.
%


%
%  Basis func1: PHI(X,Y) = G(3,2) * H(6,4) / normalization.
%

Xi = p(1,1:n);
Eta = p(2,1:n);
x1 = t(1,1); x2 = t(1,2); x3 = t(1,3); x4 = t(1,4); x5...
    = t(1,5); x6 = t(1,6);
y1 = t(2,1); y2 = t(2,2); y3 = t(2,3); y4 = t(2,4); y5...
    = t(2,5); y6 = t(2,6);
```

```
area = t(1,1) * ( t(2,2) - t(2,3) ) ...
     + t(1,2) * ( t(2,3) - t(2,1) ) ...
     + t(1,3) * ( t(2,1) - t(2,2) );
```

```
gx(1:n) = ( Xi - x2 ) * ( y3   - y2 ) ...
        - ( x3   - x2 ) * ( Eta - y2 );
```

```
gn(1:n) = ( x1 - x2 ) * ( y3 - y2 ) ...
        - ( x3 - x2 ) * ( y1 - y2 );
```

```
hx(1:n) = ( Xi - x4 ) * ( y6   - y4 ) ...
        - ( x6   - x4 ) * ( Eta - y4 );
```

```
hn(1:n) = ( x1 - x4 ) * ( y6 - y4 ) ...
        - ( x6 - x4 ) * ( y1 - y4 );
```

```
phi(1,1:n) = ( gx(1:n) .* hx(1:n) ) ./ ( gn(1:n) ...
    .* hn(1:n) );
```

```
dphidx(1,1:n) =  ( ( y3 - y2 ) * hx(1:n) ...
                   + gx(1:n) * ( y6 - y4 ) ) ...
                 ./ ( gn(1:n) .* hn(1:n) );
```

```
   dphidy(1,1:n) = -( ( x3 - x2 ) * hx(1:n) ...
                    + gx * ( x6 - x4 ) ) ...
                   ./ ( gn(1:n) .* hn(1:n) );
%
%  Basis func 2: PHI(X,Y) = G(3,1)* H(4,5)/normalization.
%
   gx(1:n) = ( Xi - x1 ) * ( y3   - y1 ) ...
           - ( x3   - x1 ) * ( Eta - y1 );

   gn(1:n) = ( x2 - x1 ) * ( y3 - y1 ) ...
           - ( x3 - x1 ) * ( y2 - y1 );

   hx(1:n) = ( Xi - x5 ) * ( y4   - y5 ) ...
           - ( x4   - x5 ) * ( Eta - y5 );

   hn(1:n) = ( x2 - x5 ) * ( y4 - y5 ) ...
           - ( x4 - x5 ) * ( y2 - y5 );

   phi(2,1:n) = ( gx(1:n) .* hx(1:n) ) ./ ( gn(1:n) ...
       .* hn(1:n) );

   dphidx(2,1:n) =  ( ( y3 - y1 ) * hx(1:n) ...
                  + gx(1:n) * ( y4 - y5 ) ) ...
                  ./ ( gn(1:n) .* hn(1:n) );

   dphidy(2,1:n) = -( ( x3 - x1 ) * hx(1:n) ...
```

```
                + gx(1:n) * ( x4 - x5 ) ) ...
                ./ ( gn(1:n) .* hn(1:n) );
%
%  Basis func 3: PHI(X,Y) = G(1,2) * H(5,6)/normalization.
%
  gx(1:n) = ( Xi - x2 ) * ( y1   - y2 ) ...
          - ( x1   - x2 ) * ( Eta - y2 );

  gn(1:n) = ( x3 - x2 ) * ( y1 - y2 ) ...
          - ( x1 - x2 ) * ( y3 - y2 );

  hx(1:n) = ( Xi - x6 ) * ( y5   - y6 ) ...
          - ( x5   - x6 ) * ( Eta - y6 );

  hn(1:n) = ( x3 - x6 ) * ( y5 - y6 ) ...
          - ( x5 - x6 ) * ( y3 - y6 );

  phi(3,1:n) = ( gx(1:n) .* hx(1:n) ) ./ ( gn(1:n) ...
      .* hn(1:n) );

  dphidx(3,1:n) =  ( ( y1 - y2 ) * hx(1:n) ...
                + gx(1:n) * ( y5 - y6 ) ) ...
                ./ ( gn(1:n) .* hn(1:n) );

  dphidy(3,1:n) = -( ( x1 - x2 ) * hx(1:n) ...
                + gx(1:n) * ( x5 - x6 ) ) ...
```

```
                ./ ( gn(1:n) .* hn(1:n) );
%
%  Basis func4: PHI(X,Y) = G(1,3) * H(2,3) /normalization.
%
   gx(1:n) = ( Xi - x3 ) * ( y1   - y3 ) ...
           - ( x1   - x3 ) * ( Eta - y3 );

   gn(1:n) = ( x4 - x3 ) * ( y1 - y3 ) ...
           - ( x1 - x3 ) * ( y4 - y3 );

   hx(1:n) = ( Xi - x3 ) * ( y2   - y3 ) ...
           - ( x2   - x3 ) * ( Eta - y3 );

   hn(1:n) = ( x4 - x3 ) * ( y2 - y3 ) ...
           - ( x2 - x3 ) * ( y4 - y3 );

   phi(4,1:n) = ( gx(1:n) .* hx(1:n) ) ./ ( gn(1:n)...
       .* hn(1:n) );

   dphidx(4,1:n) =  ( ( y1 - y3 ) * hx(1:n) ...
                 + gx(1:n) * ( y2 - y3 ) ) ...
                 ./ ( gn(1:n) .* hn(1:n) );

   dphidy(4,1:n) = -( ( x1 - x3 ) * hx(1:n) ...
                 + gx(1:n) * ( x2 - x3 ) ) ...
                 ./ ( gn(1:n) .* hn(1:n) );
```

```
%
%  Basis func 5: PHI(X,Y) = G(2,1) * H(3,1)/normalization.
%
   gx(1:n) = ( Xi - x1 ) * ( y2   - y1 ) ...
           - ( x2   - x1 ) * ( Eta - y1 );

   gn(1:n) = ( x5 - x1 ) * ( y2 - y1 ) ...
           - ( x2 - x1 ) * ( y5 - y1 );

   hx(1:n) = ( Xi - x1 ) * ( y3   - y1 ) ...
           - ( x3   - x1 ) * ( Eta - y1 );

   hn(1:n) = ( x5 - x1 ) * ( y3 - y1 ) ...
           - ( x3 - x1 ) * ( y5 - y1 );

   phi(5,1:n) = ( gx(1:n) .* hx(1:n) ) ./ ( gn(1:n) ...
       .* hn(1:n) );

   dphidx(5,1:n) =  ( ( y2 - y1 ) * hx(1:n) ...
                 + gx(1:n) * ( y3 - y1 ) ) ...
                  ./ ( gn(1:n) .* hn(1:n) );

   dphidy(5,1:n) = -( ( x2 - x1 ) * hx(1:n) ...
                 + gx(1:n) * ( x3 - x1 ) ) ...
                  ./ ( gn(1:n) .* hn(1:n) );
%
```

```
%  Basis func6: PHI(X,Y) = G(1,2) * H(3,2)/normalization.
%
   gx(1:n) = ( Xi - x2 ) * ( y1   - y2 ) ...
           - ( x1   - x2 ) * ( Eta - y2 );

   gn(1:n) = ( x6 - x2 ) * ( y1 - y2 ) ...
           - ( x1 - x2 ) * ( y6 - y2 );

   hx(1:n) = ( Xi - x2 ) * ( y3   - y2 ) ...
           - ( x3   - x2 ) * ( Eta - y2 );

   hn(1:n) = ( x6 - x2 ) * ( y3 - y2 ) ...
           - ( x3 - x2 ) * ( y6 - y2 );

   phi(6,1:n) = ( gx(1:n) .* hx(1:n) ) ./ ( gn(1:n)...
       .* hn(1:n) );

   dphidx(6,1:n) =  ( ( y1 - y2 ) * hx(1:n) ...
                 + gx(1:n) * ( y3 - y2 ) ) ...
                 ./ ( gn(1:n) .* hn(1:n) );

   dphidy(6,1:n) = -( ( x1 - x2 ) * hx(1:n) ...
                 + gx(1:n) * ( x3 - x2 ) ) ...
                 ./ ( gn(1:n) .* hn(1:n) );
```

```
  return
end

function [ phi,dphidx,dphidy,area ] = basis_mn_t3(t,n,p)

%*********************************************************
% BASIS_MN_T3: all bases funcs at N points for a T3 element.
%
%  Discussion:
%    The routine is given the coordinates of the vertices
%    of a triangle. It works directly with these coordinates,
%    and does not refer to a reference element.
%
%    The sides of the triangle DO NOT have to lie along
%    a coordinate axis.
%
%    The routine evaluates the basis functions associated
%     with each vertex, and their derivatives with respect
%      to X and Y.
%  Physical Element T3:
%            3
%           / \
%          /   \
%         /     \
%        /       \
%       /         \
%      1-----------2
```

```
%
%  Licensing:
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%    14 February 2006
%
%  Author:
%    John Burkardt
%
%  Parameters:
%
%    Input, real T(2,3), the vertices of the triangle.  It
%    is common to list these points in counter clockwise
%       order.
%    Input, integer N, the number of evaluation points.
%
%    Input, real P(2,N), the coordinates of the evaluation
%   points.
%    Output, real PHI(3,N), the basis functions at the
%    evaluation points.
%    Output, real DPHIDX(3,N), DPHIDY(3,N), the basis
%    derivatives at the evaluation points.
%
%  Local parameters:
%    Local, real AREA, is (twice) the area of the triangle.
```

```
%
  area = t(1,1) * ( t(2,2) - t(2,3) ) ...
       + t(1,2) * ( t(2,3) - t(2,1) ) ...
       + t(1,3) * ( t(2,1) - t(2,2) );

  phi(1,1:n) =      ( ( t(1,3) - t(1,2) ) * ( p(2,1:n)...
      - t(2,2) )      ...
                    - ( t(2,3) - t(2,2) ) * ( p(1,1:n)...
                    - t(1,2) ) );
  dphidx(1,1:n) =   - ( t(2,3) - t(2,2) );
  dphidy(1,1:n) =     ( t(1,3) - t(1,2) );

  phi(2,1:n) =      ( ( t(1,1) - t(1,3) ) * ( p(2,1:n)...
      - t(2,3) )      ...
                    - ( t(2,1) - t(2,3) ) * ( p(1,1:n)...
                    - t(1,3) ) );
  dphidx(2,1:n) =   - ( t(2,1) - t(2,3) );
  dphidy(2,1:n) =     ( t(1,1) - t(1,3) );

  phi(3,1:n) =      ( ( t(1,2) - t(1,1) ) * ( p(2,1:n)...
      - t(2,1) )      ...
                    - ( t(2,2) - t(2,1) ) * ( p(1,1:n)...
                    - t(1,1) ) );
  dphidx(3,1:n) =   - ( t(2,2) - t(2,1) );
  dphidy(3,1:n) =     ( t(1,2) - t(1,1) );
%
```

```
%  Normalize.
%
   phi(1:3,1:n)     = phi(1:3,1:n) / area;
   dphidx(1:3,1:n) = dphidx(1:3,1:n) / area;
   dphidy(1:3,1:n) = dphidy(1:3,1:n) / area;

   return
end

function plotdata(u,v,p,node_xy,node_num,pres,steps,dt,Re)

% plot velocity magnitude
figure(1);
xlin = linspace(min(node_xy(1,:)),max(node_xy(1,:)),...
    min(500,ceil(.5*node_num)));
ylin = linspace(min(node_xy(2,:)),max(node_xy(2,:)),...
    min(500,ceil(.5*node_num)));
[xx,yy] = meshgrid(xlin,ylin);
vel = sqrt(u.^2 + v.^2);
F = TriScatteredInterp(node_xy(1,:)',node_xy(2,:)',vel);
z = F(xx,yy);
colormap(jet)
imagesc(xlin,ylin,z)
hold on
% plot streamlines
U = TriScatteredInterp(node_xy(1,:)',node_xy(2,:)',u);
```

```
V = TriScatteredInterp(node_xy(1,:)',node_xy(2,:)',v);
uu = U(xx,yy);
vv = V(xx,yy);
U = streamslice(xx,yy,uu,vv); axis('tight');
set(U,'Color','white')
xlabel('x','fontsize',10,'fontweight','b')
ylabel('y','fontsize',10,'fontweight','b')
set(gca,'XTick',[0 0.2 0.4 0.6 0.8 1],'YTick',[0 0.2 0.4...
 0.6 0.8 1],'fontsize',10,'fontweight','b','ydir','normal')
hold off

%plot Pressure
figure(2)
F = TriScatteredInterp(node_xy(1,pres)',node_xy(2,pres)',p);
pp = F(xx,yy);
surf(xx,yy,pp,'FaceColor','interp','FaceLighting','phong')
xlabel('x','fontsize',10,'fontweight','b')
ylabel('y','fontsize',10,'fontweight','b')
zlabel('pressure','fontsize',10,'fontweight','b')
set(gca,'XTick',[0 0.2 0.4 0.6 0.8 1],'YTick',[0 0.2 0.4...
    0.6 0.8 1],'fontsize',10,'fontweight','b')

[ua va pa] = external_force(node_xy,(steps)*dt,1,Re);
norm(ua'-u,inf)
norm(ua'-u,2)
xlin = linspace(min(node_xy(1,:)),max(node_xy(1,:)),...
```

```
    min(50,ceil(.5*node_num)));
ylin = linspace(min(node_xy(2,:)),max(node_xy(2,:)),...
    min(50,ceil(.5*node_num)));
[xx,yy] = meshgrid(xlin,ylin);
%plot velocity error
figure(3);
true_v = sqrt(ua.^2+va.^2)';
F = TriScatteredInterp(node_xy(1,:)',node_xy(2,:)',...
    (vel-true_v));
z = F(xx,yy);
colormap(jet)
surf(xx,yy,z,'FaceColor','interp','EdgeColor','none',...
    'FaceLighting','phong');
title('Flow Velocity error')
colorbar;

%plot Pressure error
figure(4)
F = TriScatteredInterp(node_xy(1,pres)',node_xy(2,pres)',...
    p-pa(pres)');
pp = F(xx,yy);
colormap(jet)
surf(xx,yy,pp,'FaceColor','interp','EdgeColor','none',...
    'FaceLighting','phong')
title('Pressure field error')
colorbar
```

```
return
end
function bnd=triangulation_order6_boundary_node(num_node,xy,t)

%**************************************************************
%
%% TRIANGULATION_ORDER6_BOUNDARY_NODE indicates which nodes
% are on the boundary.
% p contains coordinates
% t contains elements
t = t';
t2 = [t(:,4);t(:,5);t(:,6)];
t = [t(:,1:2);t(:,2:3);[t(:,3),t(:,1)]];
t = sort(t,2);
t = [t,t2];
edges = unique(t,'rows');
tl = length(edges);

% identify boundary nodes
bnd = zeros(1,num_node);
for i = 1:tl
    indx2 = find(t(:,1) == edges(i,1) & ...
        t(:,2) == edges(i,2));
    if(length(indx2) == 1)
        bnd(1,edges(i,1)) = 1;
```

```
        bnd(1,edges(i,2)) = 1;
        bnd(1,edges(i,3)) = 1;
    end
end


% interior boundaries on rectangle domain
max_x = max(xy(:,1))-eps;
min_x = min(xy(:,1))+eps;
max_y = max(xy(:,2))-eps;
min_y = min(xy(:,2))+eps;
for i = 1:num_node
    if(bnd(i) == 1)
        if(xy(i,1) < max_x && xy(i,1) > min_x && ...
                xy(i,2) < max_y && xy(i,2) > min_y)
            bnd(i) = 2;
        end
    end
end

  return
end

function del_x = min_dist(coords,nodes)
[rows cols] = size(nodes);
del_x = 9999;
```

```
for element = 1:1:cols
    side1 = norm(coords(:,nodes(1,element)) -...
        coords(:,nodes(4,element)));
    side3 = norm(coords(:,nodes(1,element)) -...
        coords(:,nodes(6,element)));
    side2 = norm(coords(:,nodes(2,element)) -...
        coords(:,nodes(5,element)));
    del_x = min([del_x,side1,side2,side3]);
end

return
end
```

Appendix B

SURVEY ON ANALYTIC SOLUTIONS

The following are excerpts from various authors on research, both past and present, on the existence and uniqueness of analytic solutions to the Navier-Stokes equations. The papers cited span from 1934 to present. They demonstrate significant progress towards understanding deeply the Navier-Stokes equations, but the also expose how much work is still to be done in this area.

B.1 Excerpts on Existence and Uniqueness

For the NS on $(0,T)\times\mathbb{R}^3$, $\nabla\cdot u=0$, u is called a weak solution if it is a Leray-Hopf solution, namely

$$u\in L^\infty((0,T),L^2(\mathbb{R}^3))\cap L^2((0,T),H^1(\mathbb{R}^3))$$
$$\nabla\cdot u=0 \qquad for\ x\in(0,T)\times\mathbb{R}^3$$
$$\int_0^T\left(-u\cdot\phi_t+\nabla u\cdot\nabla\phi+(u\cdot\nabla u)\cdot\phi\right)dxdt=0$$

for all test functions ϕ in $C_0^\infty\left((0,T)\times\mathbb{R}^3\right)$ with $\nabla\cdot\phi=0$ in $(0,T)\times\mathbb{R}^3$.

Concerning the uniqueness for the Leray-Hopf solutions, since Leray and Hopf proved the global existence of weak solutions [20, 13], it has been an open problem to prove or disprove the uniqueness and regularity (smoothness) of weak solutions to the Navier-Stokes equations. The question of if the solution has continuity or stability that can be reproduced is also a big question in the study of numerical methods.

E. Hopf commented [13], "It is hard to believe that the initial value problem for the viscous fluid in dimension $n=3$ could have more than one solution, and more work should be devoted to the study of the uniqueness question."

From O. A. Ladyzhenskaya [19]: "As regards the class of weak Hopf solutions for the general three-dimensional case, it has always seemed to me that it is too broad, i. e. , that there is missing in it a basic property of of the initial-value problem, viz. its determinacy (a uniqueness theorem)... But I had available only indirect reasons in support of this assertion... which had no formal demonstrative power. At this time I am able to rigorously prove the validity of my opinions."

In [19] she constructed an example of non-uniqueness in a non-standard time-dependent domain which degenerates to one point at time $t = 0$. The boundary conditions are also non-standard and based on the study of certain scale-invariant solutions. The solutions are considered in the class of axi-symmetric velocity fields with no swirl. The time-dependent domain corresponds to the image of a fixed space-time domain in the self-similarity variables. She commented, "We note that a certain exoticness of the domain QT in which our example has been constructed does not imply the loss of the uniqueness theorems just mentioned (they are usually proved for the case of a domain which does not alter with time..."

Jia and Šverák conjectured [15, 14] that for many large scale-invariant initial data the scale invariant solutions are not unique, and suggested possible implications for the non-uniqueness of the Leray-Hopf weak solutions.

The 2D problem in NS is known and settled. In three dimensions, the question of existence in L^2 was proved by Leray. However it is unsettled if the solution is unique or if they contain any singularity (smoothness). The Clay Mathematics Institute has called this one of the seven most important open problems in math and offer US$1,000,000 prize for a solution or a counterexample.

[1]: Kato initiated the study of the Navier-Stokes equations in critical spaces by proving that the problem is locally well-posed in L^3 and globally well-posed if the initial data are small in $L^3(\mathbb{R}^3)$. The study of the Navier-Stokes in critical spaces continues to be studied by many authors.

Koch and Tataru [18] proved the global well-posedness of the Navier-Stokes equations evolving from small initial data. The space has a special role since it is the largest critical space among the spaces listed where such existence results are available.