The University of Chicago

FEM Software Automation, with a case study on the Stokes Equations [1]

A Masters Paper submitted to
The Faculty of the Computer Science Department
In Candidacy for the degree of
Masters of Science

by

Andy R. Terrel

Advisors:
L. Ridgway Scott and Robert C. Kirby

Additional Committee Members:
Todd Dupont and Matt Knepley

Chicago, Illinois
March 1, 2007

Abstract

Finite Element Methods are a popular method for solving Partial Differential Equations. Unlike Ordinary Differential Equations, there has not been a complete solution to automating any simulation. Here we provide a discussion on the trends in automation of FEM methods, some challenges to automation, and a novel computational study of different methods solving Stokes Equations made possible through automation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

State-of-the-art software projects usually implement new original features needed by the respective field. For example, state-of-the-art games use the fastest machines with better rendering and more life-like play, graphics editors do ray tracing and batch processing or any number of features that were impossible two to five years ago. In the same way, state-of-the-art scientific software must push the envelope in several ways. While not inclusive, a few items on that list would include:

1. Ability to solve large problems
2. Ability to solve interesting problems
3. Incorporate the best theoretical methods

To solve the largest problems, simulation software is tuned to the top computers, which today means the code must be massively parallel. Not only does it need to be parallel but the computations must also be efficient, which typically involves a large amount of hand coding and a dedicated team of workers. Features required for for efficient parallel support might include load balancing or automated mesh refinement, both challenging fields of research.

Solving interesting problems means the researcher must be able to characterize real world scenarios as best as possible. This includes incorporating hard-to-characterize boundary conditions, multiphysics and multiscale interactions, complicated domains, and more depending on the exact applications. This broad stroke of possible complications demands code that is

robust enough to switch working assumptions quickly or else the code is pigeonholed to a few applications. Since each of these features can be in and of themselves large fields of research, most simulation codes are content with fine tuning to one application.

And finally, the software needs to incorporate cutting-edge methods. It could be argued that the methods will be included just because they produce better results, but this is not in general true. In Finite Element Methods (FEM), many elements which would provide much better convergence rates are not used because of the complexity to generate appropriate basis functions or handle the appropriate data. Different methods will often be overlooked because the simulation software just does not support it, maybe because of assumptions made to accommodate the other goals..

One aspect left off this list is the ability to automate this process and reduce the number of coding hours required to produce a simulation. This idea of creating an environment that is competitive with state-of-the-art software and automatic is often criticized as too lofty of a goal, but it is exactly the goal of many research groups throughout the world such as the FEniCS Project [18], Sundance [19], Deal.II [3], and FreeFEM [9].

Automation promises to be an ideal solution to providing each of these goals. Often a software package might achieve one or two, but due to the limited scope of the design is not able to fulfill all. By automating FEM, we are able to make new innovations in the field, and incorporate techniques used in state-of-the-art specialized codes. Adding new features, such as splines for the boundary conditions or adaptive mesh refinement, becomes modular thus allowing for a separation of concerns between the user and developer code. Such a separation allows the scientist or engineer to focus on results and not necessarily the implementations. Finally, automation is the only paradigm to provide quick plug-in-play style for many different methods, such as switching from a Generalized Galerkin method to a Discontinuous Galerkin method or changing element descriptions.

The next question to arise, is which parts should be automated. The scope of this paper is interested in the automation of solving PDEs, while often simulations include many other parts. For many scientists and engineers this scope may seem limited because solving the PDE is only one part of the actual simulation. Additional parts may include hooking up different legacy codes, coupling with non-PDE simulations, and many other possibilities. In particular, there have been attempts to implement these different issues by the Sierra and Cactus frameworks [21, 11]. While important issues, they hard

to abstract and often very particular to the specific application; whereas, the PDE is a mathematically abstract object which can be used in all simulation software.

Throughout this analysis of FEM in an automated software context, we put forward three claims. First, there are interesting or missing the mathematical understandings for the complete automation of FEM. This is most notable in the data structures used to transition from local elements to the global domain. Second, automation of FEM can serve as a tool to aid theory. For example, we found a problem with the formulation of the iterated penalty method for the Stokes equation in a common textbook [6]. Finally by giving a case study of the Stokes equation, we present that automation leverages faster prototyping of scientific models which will immediately affect science researchers, engineers, educators, and many other fields.

Chapter 2

Mathematics and Software of FEM

Finite element methods (FEM) are widely used methods in scientific and engineering simulations for solving PDEs [6]. These methods have been widely studied and appear in both mathematics and engineering textbooks [6, 13]. Like many computational methods, there is a large gap between the mathematical analysis and the software algorithms of FEM. Often for the sake of faster results the software is implemented to work for particular cases, rather than a general abstract application.

The natural question to ask is what is inherent in FEM that is not immediately expressible in an elegant and algorithmic way. Certainly one answer is there are just too many algorithms to accomplish the same basic tasks, and not enough understanding about the effects of these algorithms. While this is certainly the case and even visible in different software package, if this were the only reason we could build fully automated codes that piece together a plethora of different algorithms. Rather, this disconnect between mathematics and software is due to lacking of mathematical insight to both the data structures and interweaving of the different aspects of FEM. To demonstrate this claim, we look at how mathematics and software approach FEM.

2.1 A Mathematical Definition of FEM

Here we present a basic outline of the Galerkin method for an analysis of the algorithms that FEM software must implement. We include more specifics about the finite element in Chapter 4 when we discuss different methods for solving the Stokes equations.

FEM software is designed to numerically solve Partial Differential Equations (PDEs). One simple example of a PDE is the Poisson Equation with Dirichlet boundary conditions: Find a function u such that:

$$\begin{array}{c} -\Delta u = f \\ u = 0 \text{ on } \partial\Omega \end{array} \tag{2.1}$$

Another equivalent formulation of the same problem for sufficiently smooth f is the weak formulation. Find a function $u \in V$, such that:

$$a(u, v) = (f, v) \qquad \forall v \in V \tag{2.2}$$

where,

$$\begin{array}{c} V = \{v \in L^2(0,1) : a(v,v) < \infty \text{ and } v(0) = 0\} \\ a(u,v) = \int u'v'dx \\ (f,v) = \int fvdx \end{array}$$

Since the space of function, V, is quite large, one will restrict our solution to a finite dimensional subspace, $S \subset V$, then we have the Ritz-Galerkin Approximation

$$u_S \in S \text{ such that } a(u_S, v) = (f, v) \qquad \forall v \in S \tag{2.3}$$

To solve this formulation, one must use some finite element space to approximate S. The approximation of V with S provides the mechanism for evaluating the convergence of the method. Approximation of the domain of the PDE is done with the finite element, which one popular definition [8] is:

$$\begin{array}{l} \text{A reference element, } K \\ \text{A space of shape functions, } \mathcal{P} \\ \text{A basis, } \mathcal{N} \end{array} \tag{2.4}$$

The shape functions are interpolated on the basis in the reference element. Then the reference element is transformed to each element in the domain to generate the linear system:

$$AU = F \tag{2.5}$$

By solving this matrix equation, we determine the coefficients for our basis on each element. This leads to our solution of the approximated problem.

This mathematical formulation is very simple and quite elegant, but it does avoid many issues that have been major areas of research for more difficult problems. None the less by understanding each part of this process on a mathematical level, one would hope that its abstractions could be used in building smarter software.

2.2 An Algorithmic Viewpoint of FEM

Our mathematical formulation of FEM is completely lacking in algorithmic details. This becomes clear when one starts to implement the simplest parts of the methods such as how to evaluate f in equation 2.1 or input different bilinear forms in equation 2.2. Moreover, these issues are not orthogonal, evaluating f outside our finite element space requires algorithms that must be reformulated for each form.

In Chapter 3, we will discuss more fully how some software the automates these algorithms that are not specified by the mathematical formulation. To get a sense of the differences between the math and software processes, we ask how does each address different concerns of the simulation, see Table 2.2. In the algorithmic column, we give a list of ways to implement each of different concerns, roughly in order of complexity. The different codes we compare have varying implementations as well, see Table 2.2. We give more details about the comparisons, but largely the linear solver and the mesh is are given to these packages and thus are not compare in the table.

This table questions the full mathematical understanding of automating FEM. One glaring example is the lack of element types that are implemented. A response to this sparsity, was general finite element tabulators which automated the tabulations of the functions for the finite element as described in Equation 2.4, for more details see Section 3.2. Tabulators did not solve this sparsity, automation software still only includes a small number of finite elements. This begs the question is there a problem with this definition? The problem is not in the mathematical formulation but rather the data structure used to communicate the between the local element and the global space. Many elements need to know exactly what other elements are doing. For example, Nédélec elements need to have an idea of element orientation in order to maintain the continuity of the trace between elements. Argyris

Concern	**Mathematical**	**Algorithmic**
The Problem Domain	• Allows for as much generality as needed • Define complex relationships between partitions	• Described by a mesh • Different simulation for different meshes • Implementations: – uniform mesh, – arbitrary geometries, – adaptive mesh, – unstructured mesh
Function Spaces	• The space where solution lies • Global and local implications • No global data description	• Only described locally • Implementations: – linears, – menu of options, – arbitrary order, – tabulator
Equation Statements	• Independent of function spaces • Symbolic equation • Many equivalent forms	• Given in variational form or as a routine • Implementations: – menu – language – derived forms
Assembly of the Matrix Equations	• Process determined by method • Gives $AU = F$ system	• Construction of equations to hand to linear solver • Implementations: – Single methods – Menu of methods – Matrix data structure – Action (matrix-free)

Table 2.1: Mathematical and Algorithmic concerns of FEM

Software	Functions	Equations	Assembly
Deal.II	arbitrary order	numerical functions	Not implemented
FEniCS	tabulator	numerical language	Matrix or Action
Sundance	tabulator	semi-symbolic language	Matrix

elements require knowledge of the gradient between elements and thus must know what links it has with other elements. Because of these challenges, the Lagrange element, which requires no knowledge of the other elements, is very attractive, but at the cost of a lower rate of convergence or lacking to preserve essential qualities in the function space.

2.3 Input and Output

One issue that is less mathematical and formal in nature is how to input or output the problem. A scientist or engineer rarely desires only a solution to a PDE but rather prefers to have some sort of feature about the solution reported. For example in the tests of the Stoke's equations, we report the error of the solution rather that giving the reader the computed solution.

For inputting the problem, often there are parameters that are set in the PDE, such as a material density, or maybe are features of the boundary conditions of the supplied mesh. While these parameters are necessary for solving the PDE, they depend on the particular situation and are not included in our discussion of automating the FEM software. It is often the case that FEM software gives an interface for defining how one inputs such parameters.

The output of the problem can be in many different formats. It could be the result of doing some matrix-vector operations, such as taking a norm, or one might desire a visualization the solution. Much like inputting the problem, it is specific to the needs of the developer, but common operations such as taking the norm or producing a graphics format are typically included in the software. If one wants more for an output, the data structures resulting from the linear solver are available for the user to manipulate.

Just as any mathematical algorithm, there are a host of ways to accomplish the different parts of FEM. Sometimes these different algorithms are using different mathematical assumptions and other times they are optimized for complexity. This makes any analysis of the system as a whole a challenging project, but one certain fact is that the implementations of the separate

parts of FEM are not independent of other parts. Which leads to the next chapter with a discussion of how software implements the different parts of FEM from Table 2.2.

Chapter 3

Automation of the Finite Element Method

Even though the mathematical definition of FEM does not give all the details needed for the simulations, many projects are making major accomplishments. These accomplishments produced the software which allowed our case study of the Stokes equation possible. Many mathematical and algorithmic insights have been made and are implemented in these software packages. For our purposes we studied the FEniCS project and Sundance in detail. We also refer to Deal.II; whereas, it does not automate the entire FEM process, it is a well-designed code that gives the programmer access to libraries that have automated different parts.

These three projects are by no means the only attempts of automating FEM, but do provide a base that covers most of the software and mathematical ideas that are involved in the process. Here we describe some of the abstractions used in the software.

3.1 The Simulation Engine

The simulation engine is the piece of software that holds all the pieces of the process together. Although which pieces to hold together vary widely from one software project to another. The important aspect is that it somehow describes a domain, discretizes that domain, builds a finite element space, constructs the matrix equations, and finally solve the discrete equations with a linear solver.

3.1.1 DEAL.II [3]

This project is a library of functions for a programmer to use and less of a project that looked to automated the entire FEM process. For example, it gives no abstractions for the variational form. While it does not provide much interface for the programming aspects of the automation, it does bundle important data types and operations as simple routines. Probably the most notable feature of Deal.II is its ability to do many more elements and methods, but at the cost of user coding more.

3.1.2 Sundance [19]

This project is largely the result of needing a code that could be rapidly developed and used for optimization problems. Because of this, it separates the user interface and implementation features. Much like Matlab, a user does not need to know how a process is implemented in the developer space to get very accurate answers with an efficiency that rivals hand tweaked special purpose codes.

3.1.3 FFC/Dolfin [18]

These projects are part of the FEniCS project, which was designed to develop simple methods that allow for research with automation. FFC auto-generates optimized code for precomputing much of the discrete equations from the finite element space and Dolfin provides the glue for pulling the rest of the simulation together. This project lies between the previous two projects in that it provides a simpler interface than Deal.II but requires more coding to get the robustness of Sundance. Whereas the previous two projects were built to solve engineering problems, this project focuses much more on researching the automation of FEM.

3.2 Finite Element Tabulators

A finite element tabulator is a middleware product that has been developed to give simulation engines access to basis descriptions of hard to form finite elements. Tabulating the finite element space is not a new idea, since every simulation engine must do it in some way, but by using some more mathematical operations a tabulator can easily make a large number of elements with

arbitrary order. This is a case where there are two codes that accomplish the same results but with a different computing model.

3.2.1 FIAT [18]

This code develops a mathematical paradigm where arbitrary order is based upon recurrence relations between the orders. The evaluation of function in the space is generated numerically by first constructing a matrix of inner products between basis functions and an orthogonal set of polynomials. To extract the basis of the finite element it only needs to find the span of the null space of this matrix. The process is a combination between higher order functions and a higher order programming language to handle the implementation. This code can be used by both Sundance and FFC/Dolfin.

3.2.2 SyFi [18]

This code has recently become part of the FEniCS project and effectively does the same work of FIAT and FFC. It uses the symbolic engine GiNaC [4] to compute the basis functions symbolically as opposed to FIAT, which uses numerical quadrature rules to integrate. SyFi also generates precomputed forms originally introduced by FFC to hand the simulation engine rules for evaluation for the assembly of the global matrix.

3.3 Equation Description

Possibly one of the largest factors in code expressivity and ability is how one can describe the problems to be solved. If it is the case that one can only solve a list of equations, the software immediately becomes useless as soon as a different equation is needed. On the other hand if one uses a symbolic system that will allow for any equation that can be written down, one risks a very slow implementation that will not be able to handle large problems. Additionally, if one has an optimization or control problem, without differentiation of the solutions methods are limited.

Mathematically there are a large number of ways to describe the PDE. We have shown a strong form and a weak form. The weak form is a nice description because it describes the equations in the manner FEM uses. The

problem is able to be split, block solve, reformulate, or any number of techniques to facilitate a solution. Finally one can pose any well formed formula without worries of mixing data types or poorly implemented routines.

Current FEM software completely disagree with one another on how to describe the problem. For example, Sundance allows for virtually any weak form and uses a symbolics engine to evaluate the equations. FFC takes in a weak form, but has limits in the types of operators available. FFC then precompiles parts of the transformation on the reference element and generates efficient routines for Dolfin to use in assembly. Finally Deal.II leaves the user to input the problem as part of the assembly loop, thus it becomes the users responsibility to control how the problem is entered into the simulation.

3.3.1 The Use of Symbolics

Typically in scientific codes, symbolics is seen as requiring huge amounts of resources that can be used to solve larger problems. But when looking at automated simulation software, symbolics are a way to add additional functionality and optimize calculations. Both Sundance and Syfi are built on top of two very different styles of symbolic engines.

The use of the weak form can be motivation for supporting a larger symbolics engine that can then be used for differentiation. Both Sundance and SyFi are built on top of a symbolics engine and are able to easily extend to optimization or error estimation problems. This allows simulations to take advantage of optimization immediately as oppose to using an automatic differentiation tool to put derivatives into the code.

SyFi is a middleware project that does not do the heavy matrix solves or assembly, and is able to use a symbolics that preserves variables like the popular Mathematica code [14]. This type of symbolic engine is slow because it must keep the different symbols around and be able to operate on them appropriately.

Sundance does not let symbolics be the bottleneck as one might see in codes but rather it uses it to optimize the computations. For example, if an integral has a one in it, a different routine is used to make use of that information. Sundance does not keep the symbols around in a fashion as SyFi but does use relations between equations and implement the chain rule. Thus the code has all the ability to do optimization loops but not return a symbolic equation at the end of a computation.

3.4 The Problem Domain

One significant challenge for almost any real world problem is specifying the problem domain. Not only do problem have complicated geometries but also can have highly sensitive areas.

In our simple example, the domain does nothing more than give integration bounds. But we should be able to effortlessly hook up with different domain or refine and only solve explicitly on certain parts of the domain. In simulation this the domain is more or less just a mesh with boundary conditions. Thus one would except a simulation engine to do these kinds of operations on a mesh, thus requiring some sort of mesh generation.

Mesh generation is an aspect of scientific computing that has been widely studied and implemented. Unfortunately, most of the codes are proprietary and highly secretive so many industry standards are just accepted. This standardization has the effect of many finite element codes expecting to be given a mesh rather than generating the mesh, with the exception of highly regular meshes such as partitioning a rectangle. Simulations do give good ways of picking out parts of the domain, but this is necessary to develop iterators for the operators in the assembly process.

Even still there are some impressive projects that are undertaking the challenge of mesh generation incorporated with finite element methods. With this ability, it becomes feasible to automatically do hard to implement methods such as unstructured multigrid or adaptive mesh refinement.

3.5 Solving the Matrix Equation

To solve the matrix equations most engines will assemble the matrix to hand it to a linear solver package. The assembly of the matrices is equivalent to building the global function space and evaluating the weak form at the same time. The mathematical operation is merely a rote application of simple algebra operations. This lack of mathematical insight persists in the lack of automation for many elements. For simulation software it is the heart of implementing the method and often the most computationally demanding with the exception of the linear solve.

While the problem description varied with software packages, the assembly is pretty similar. Sundance and Dolfin use a declarative style where the user declares a linear problem and the assembly is automatic. Deal.II

requires the user to iterate over the mesh to manually assemble but the process is largely the same. A major difference between these routines is the underlying data structures. Accessing the matrix to experiment with is very easy in Deal.II, it is completely assembled by the user, but there are more layers of abstraction in Dolfin and Sundance, which do not prevent access to the matrix but are there to provide services which are implemented. This process does vary the length of code and scripting style that facilitates rapid development.

3.5.1 Linear Solvers

Our mathematical definition only formulates the problem and can be extended to give properties of the solution, thus we typically give the matrix equation to a linear solver. Linear solvers are a very widely studied field, and numerous codes have been developed to handle large matrix equations efficiently. More or less one chooses whether to use a direct or iterative solver and serial or parallel implementation upon the needs of the simulation. For smaller problems, a direct solver often give better accuracy but usually will not scale to large problems and parallel implementations. Thus for larger problems, one uses an iterative solver with parallel support. Then comes the choice of preconditioners and other decisions on how to solve the equations.

Because of this large design space and the fact that there are some very good packages already developed, most simulation engines hand this job off some library to handle. This either means the engine will give some sort of interface to this library or the user is responsible for calling it. Unfortunately, this puts the additional burden on the user to determine how to solve the matrix equations but at the same time allows the fine tuning of their problem. See Table 3.1 for a small table of some solvers that we use frequently, largely which one is used is based on the developers biases or needs. Each library is also used in many different ways such as Trilinos provides for major services such as memory management and UMFPack can be used directly or through other libraries.

3.6 Why are we NOT Automated?

While from a computer science perspective automation seems to be very natural and valuable, one must ask why things have not been automated

	parallel	direct	iterative
uBlas [24]		X	
UMFPack [10]		X	
PETSc [2]	X	X	X
Trilinos [12]	X	X	X

Table 3.1: Some important linear solvers used by FEM software.

already. A cynical answer that one often encounters is the limited scope of the researchers and engineers who use the methods. While there is certainly varying abilities among FEM code writers, there has not emerged a single automated code for some very important reasons. Perhaps the most fundamental reasoning for the lack of automation is the lack of mathematical understanding of the global data structures needed

Additionally there is a healthy sense of skepticism of outside code in the scientific community. Hand-coding FEM provides with assurance that things are done right, and to a certain degree this is quite true, if that is the expertise of the coder. When using automated software, it is hard to know what routine is being used to take a norm or parallelize parts of the code. This is precisely the reason this project was used with open source codes, but as with anything a certain amount of checking to see that things work is necessary. The complexity of FEM and the trade off in coding time far outweigh the challenge of this type of debugging. Imagine a researcher who finds twenty computational models of some natural phenomenon. If this researcher is required to hand code twenty different models, the research becomes more about coding rather than researching the phenomenon. By giving the researcher the tools to develop all these models very quickly, the impetus of the project is no longer on coding but the original intention.

The final reason we give for not yet achieving automation is that there is a large number of ways to solve a problem. New elements are continually being developed, alternate methods are explored, and even how to solve the matrix equations is challenged daily. How is someone suppose to know what element works with which method and then how to precondition the matrix? The difficulty of making all these ideas work together can become daunting, but ultimately by creating such automated software we will be better able to understand the pieces of FEM computationally.

3.6.1 What hasn't been done?

With this long analysis of the steps of automated simulation software, it begs the question what hasn't been done. It seems that in any one area there seems to be some piece of software that has it implemented. But in general a list of pieces that are not implemented include:

Arbitrary Elements Usually only Lagrange elements are implemented and fully supported. Although more support is available in Deal.II than the other codes presented.

Parallel/Partial assembly This is one of the most computationally demanding parts of the process, but yet only a few codes even try to do assemble in parallel or only assemble parts of the domain. Sundance stands out allowing the programmer to do both.

Adaptive/unstructured grids This really stems from the lack of mesh controls in most simulation software. Although there are other codes that specifically address this challenge, but are not automated in other ways.

Error estimators or optimization loops Usually out of the scope of most projects.

Boundary Condition calculus or embedded geometries Also lack of support for hard boundary conditions.

Chapter 4

The Stokes Equation

The Stokes equation is a standard equilibrium equation for incompressible flow. In the strong form it is:

$$\begin{aligned} -\Delta \mathbf{u} + \nabla \mathbf{p} &= \mathbf{f} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \tag{4.1}$$

The problem has been well studied and is documented in several books [6, 7]. It is a challenging problem due to the coupling between the velocity and pressure resulting in a saddle point problem. The matrix equations will be indefinite which proves to be quite taxing on linear solvers.

This case was chosen for our study because there are many stable methods but very few numerical studies including multiple methods. Often this is because it is so difficult to code one method that the cost outweighs the value of coding another method, especially if a large legacy simulation is already using one method. With automated codes, we can make meaningful comparisons quite easily.

Here we provide a brief explanation of the different methods that were included in our study. In the following section, we provide numbers for both evaluation of the different methods and the software we used.

4.1 Mixed method Formulation

Because of the coupling of the velocity and pressure, we are essentially solving two PDE's at once. One natural way to solve the system is to create a mixed system, where the blocks of the matrix are aligned so the solution to one

variable affect the other. The variational form of this mixed system is as such:

> Let $V = H^1(\Omega)^n$ and $\Pi = \{q \in L^2(\Omega) : \int_\Omega q dx = 0\}$. Given $F \in V'$, find functions $\mathbf{u} \in V$ and $p \in \Pi$ such that

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) &= F(\mathbf{v}) \quad \forall c \in V \\ b(\mathbf{u}, q) &= 0 \quad \forall q \in \Pi \end{aligned} \tag{4.2}$$

Where,

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) &:= \int_\Omega \nabla \mathbf{u} \cdot \nabla \mathbf{v} dx, \\ b(\mathbf{v}, q) &:= \int_\Omega (\nabla \cdot \mathbf{v}) q dx \end{aligned}$$

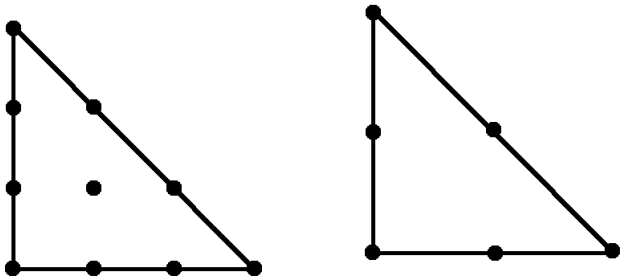
This mixed method formulation gives two discrete spaces V and Π to be solved. Developing different finite element spaces for this system is quite challenging. Using the same continuous elements for both the pressure and velocity often cause an over-determined problem. Also it is very attractive to use a discontinuous pressure space because this will provide a better solution for the divergence of the velocity, but can lead to singular points dependent on mesh. We present elements that have both continuous and discontinuous pressure spaces, and look at the consequences of both.

4.1.1 Taylor - Hood Elements

The Taylor-Hood element [22] is one of the most widely used elements for solving Stokes flow. It consists of a P_k element for the velocity space and P_l where $l < k$ for the pressure space, see Figure 4.1. Because of the simplicity of using Lagrangian elements, it can be extended to higher order easily. This element give a continuous pressure space but with the condition that the order of the pressure finite element is lower than the velocity space. This requirement avoids the problem of an over-determined system.

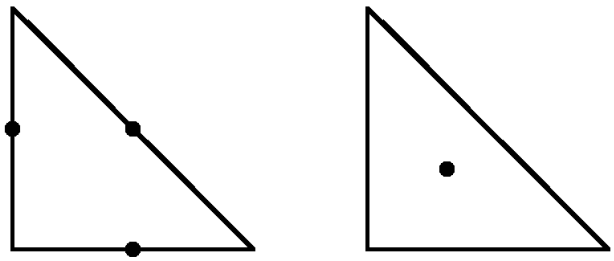
4.1.2 Crouzeix - Raviart Elements

The Crouzeix-Raviart Element is a non-conforming element that is uses the integral of the element edges as a basis in the pressure space and a discontinuous pressure space (P_0), see Figure 4.2. For the low order case the velocity



(a) P_3 for V (b) P_2 for Π

Figure 4.1: Taylor-Hood Elements



(a) Crouzeix-Raviart for V (b) P_0 for Π

Figure 4.2: Crouzeix-Raviart Elements

space element is equivalent to evaluating the basis functions at the center of the edge.

4.1.3 $C^0P_iC^{-1}P_{i-1}$ Elements

Another possibility is just to use the arbitrary Lagrange element for velocity but use a discontinuous element (of a lower order) in the pressure space. It is important to note that stable convergence of this element is highly dependent upon the mesh of the domain. It may not satisfy the inf sup condition due to singular points in the pressure spaces [23]. But if one is able to limit these points the method becomes useful.

4.2 Iterated Penalty Method

In order to avoid the problems with a discontinuous pressure space, the Uzawa iteration method and penalty methods were developed. A compilation of these two elements results in the iterated penalty method.

Let $r \in \mathbb{R}$ and $\rho > 0$ define u^n and $p = w^n$ by

$$\begin{aligned} a(\mathbf{u^n}, \mathbf{v}) + r(\nabla \cdot \mathbf{u^n}, \nabla \cdot \mathbf{v}) &= F(\mathbf{v}) - (\nabla \cdot \mathbf{v}, \nabla \cdot \mathbf{w^n}) \\ \mathbf{w}^{n+1} &= \mathbf{w}^n + \rho \mathbf{u}^n \end{aligned}$$

This method give only one space to be discretized but requires a higher order continuous element. We use Lagrangian elements with degree greater than four. The stopping criteria is the size of the incompressibility term, $||\nabla \cdot \mathbf{u^n}||_{\mathbf{V}} < \epsilon$. The iteration count and accuracy is highly dependent upon the penalties coefficients ρ and r. For our experiments we use $\rho = -r = 1.0e3$.

The Iterated Penalty method give a formulation of the $C^0P_iC^{-1}P_{i-1}$ element using only one space. Thus we have two models that are doing the same mathematical process but the algorithms are very different.

Chapter 5

Tests and Results

To evaluate these methods we want to compare mesh sizes and orders in a series of increasingly difficult problems. The degrees of freedom is a good measure of how much work the method will require, and as of now we are not setting these to be equal. For reference to the variance of the degrees of freedoms, see Figure 5.1

For our tests we use a $n \times n$ uniform mesh for a domain [0,1]×[0,1] and the aforementioned methods, solve the following three problems and compare with analytic solutions.

- Case 0:

$$\begin{aligned} \mathbf{f} &= \begin{bmatrix} -2y+1 \\ 2x+1 \end{bmatrix}, \\ \mathbf{u} &= \begin{bmatrix} x^2 y \\ -xy^2 \end{bmatrix}, \text{ and} \\ p &= x + y - 1.0 \end{aligned}$$

- Case 1:

$$\begin{aligned} \mathbf{f} &= \begin{bmatrix} 8\pi^2 \sin(2\pi x)\cos(2\pi y) \\ -8\pi^2 \cos(2\pi x)\sin(2\pi y) \end{bmatrix}, \\ \mathbf{u} &= \begin{bmatrix} \sin(2\pi x)\cos(2\pi y) \\ -\cos(2\pi x)\sin(2\pi y) \end{bmatrix}, \text{ and} \\ p &= \sin(2\pi x)\sin(2\pi y) \end{aligned}$$

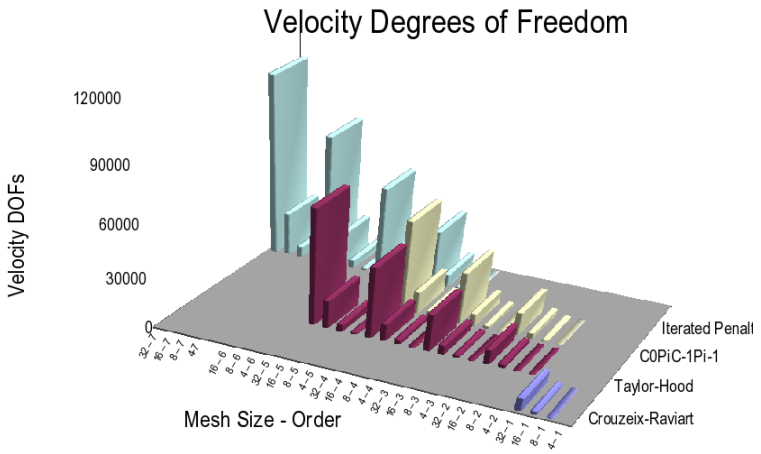


Figure 5.1: Velocity Degrees of Freedom versus Order and Mesh Size

- Case 2:

$$\begin{aligned} \mathbf{f} &= \begin{bmatrix} 18\pi^2 \sin(3\pi x)\cos(3\pi y) \\ -18\pi^2 \cos(3\pi x)\sin(3\pi y) \end{bmatrix}, \\ \mathbf{u} &= \begin{bmatrix} \sin(3\pi x)\cos(3\pi y) \\ -\cos(3\pi x)\sin(3\pi y) \end{bmatrix}, \text{ and} \\ p &= \sin(3\pi x)\sin(3\pi y) \end{aligned}$$

Case 0 gives us a low order polynomial that will be in most of the approximating spaces. Thus many of our methods should give an exact answer to machine precision. The next two are harder functions to approximate and are both used to give better testing on the convergence rates of the methods.

The simulation for each method uses a basis tabulated from FIAT. A handwritten code for generating the $n \times n$ uniform mesh, see Figure 5.2. The code to generate a uniform mesh was quite simple addition and required because in each code the default mesh had boundary elements that caused singular points in the pressure space. For each linear solve, we used the UMFPACK LU direct solver either with the Trilinos Amesos Package in Sundance or directly with FFC/Dolfin of the FEniCS package. At the time of the study, Sundance was not supporting discontinuous elements in its Python environment and thus reduced our methods tested in that system to only Taylor-Hood and Iterated Penalty, see Table 5.1. Other iterative

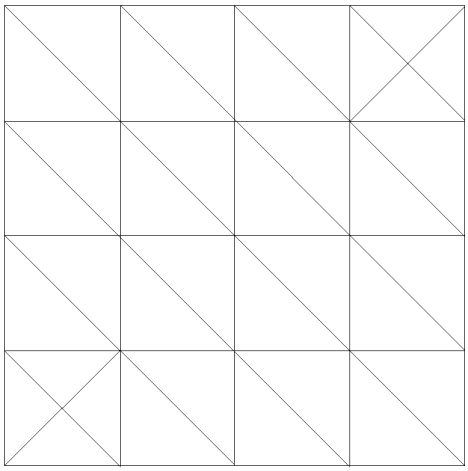


Figure 5.2: The uniform mesh with $n \times n$ rectangles used in the tests.

	Sundance	FEniCS
Taylor-Hood	X	X
Crouzeix-Raviart	-	X
$C^0P^iC^{-1}P^{i-1}$	-	X
Iterated Penalty	X	X

Table 5.1: The different simulation engines used for each method

solvers from the Trilinos or PETSc Toolkits would also work and make the code parallel, but this was not done since it was not the focus of this work.

5.1 Numerical Results

To judge how well the software and methods preform, it is important to compare a few features. The error should correspond to the increased ordered and refined meshes. These trends in the error should be shifted by a constant but have the same slope when changing from case 1 to case 2. Runtimes should be dependent on the number of DOFs although it is also expected to be affect by the ability of the computer to make effective use of a cache. And finally, the features of divergence free elements versus continuous elements should be reflected in the error of the methods.

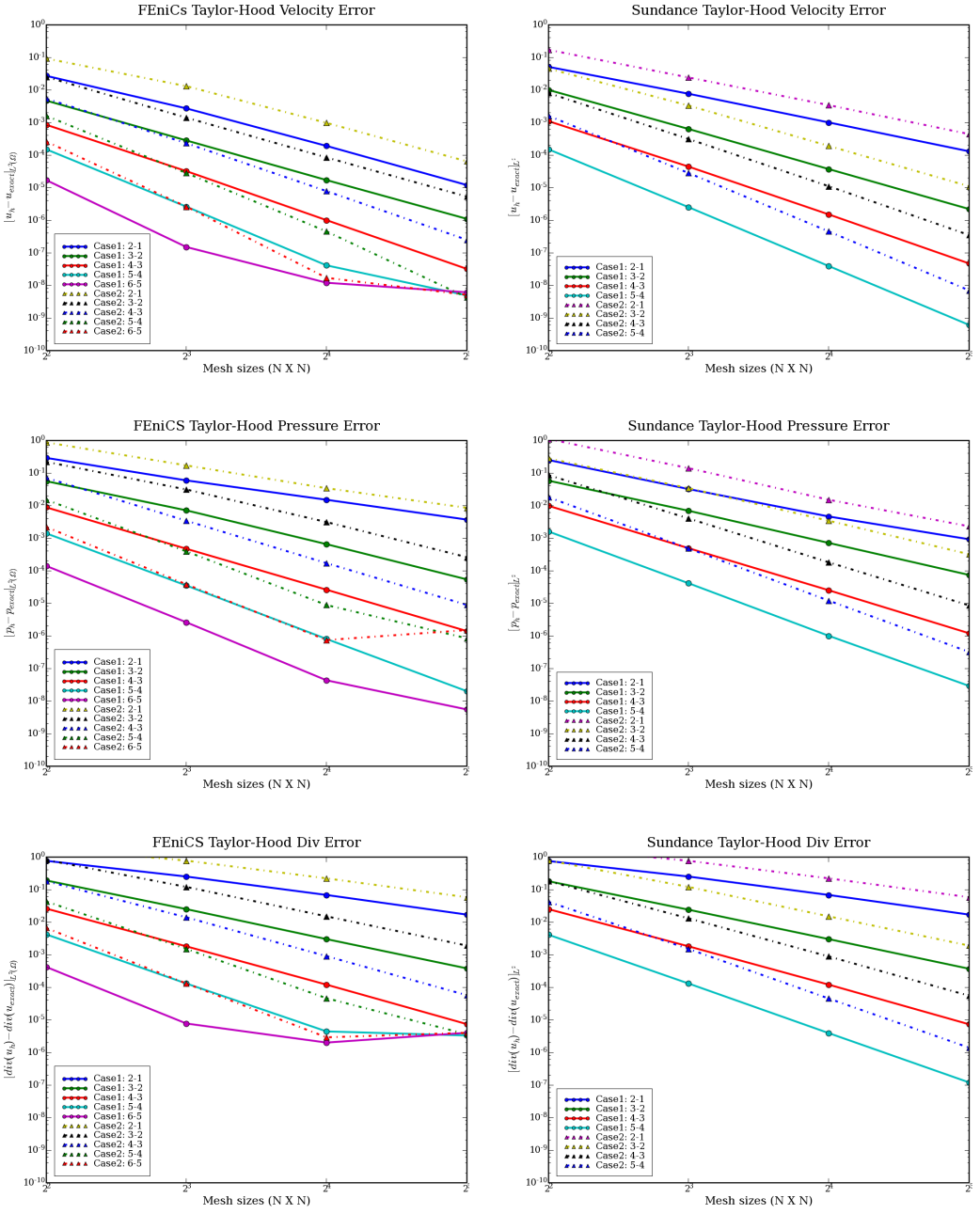


Figure 5.3: Taylor Hood Results

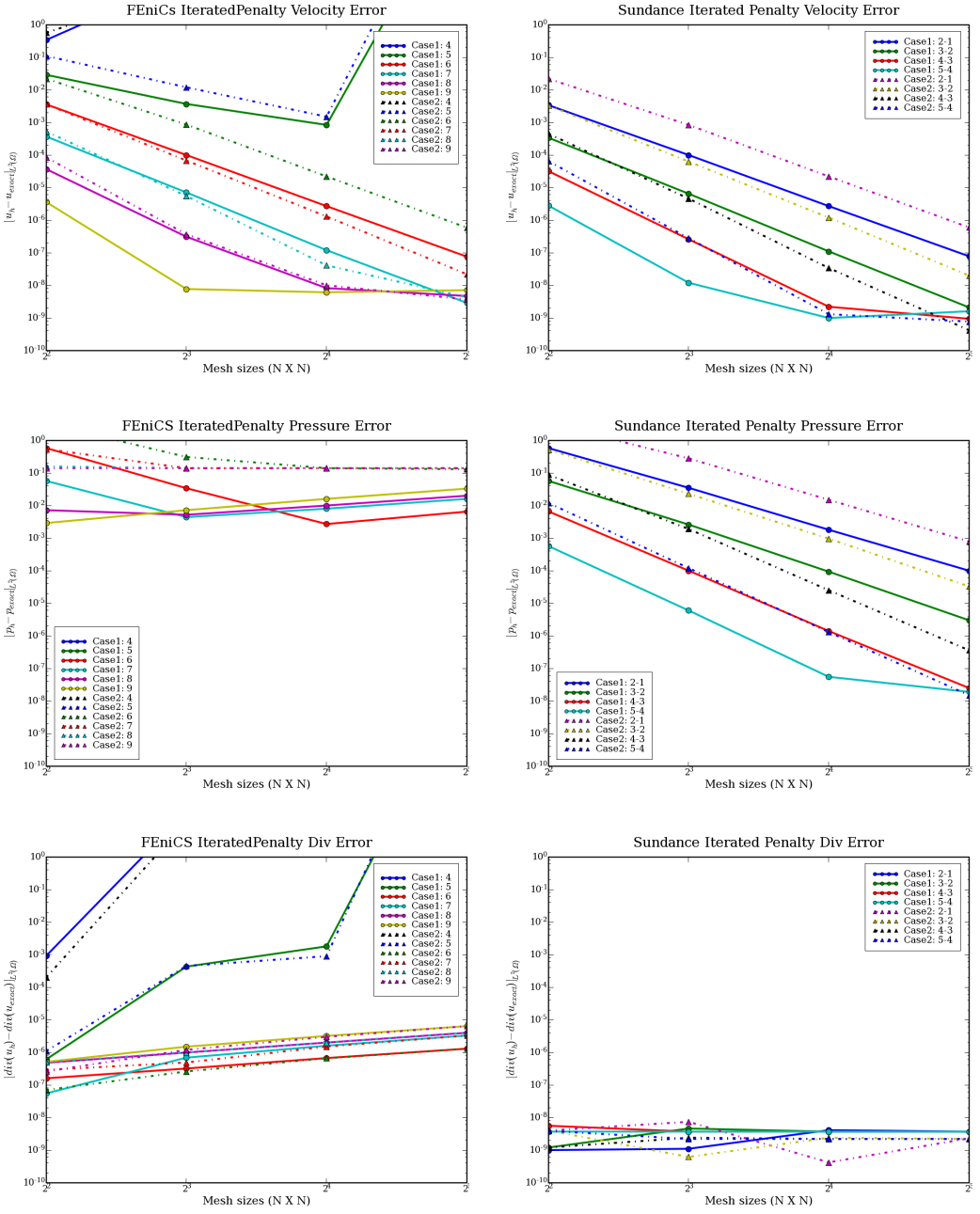


Figure 5.4: Iterated Penalty Results

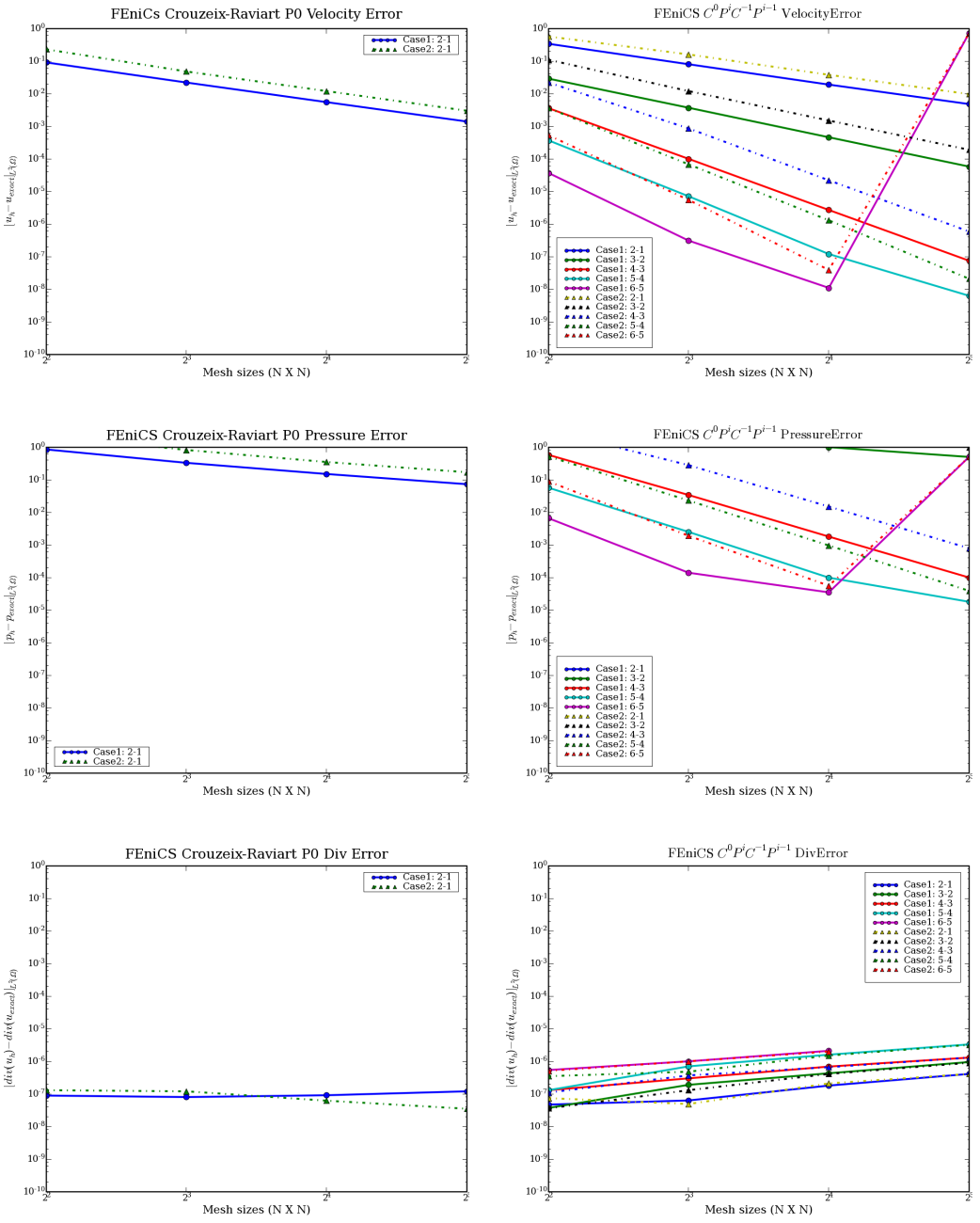


Figure 5.5: Crouzeix-Raviart and $C^0P_iC^{-1}P_{i-1}$ Results

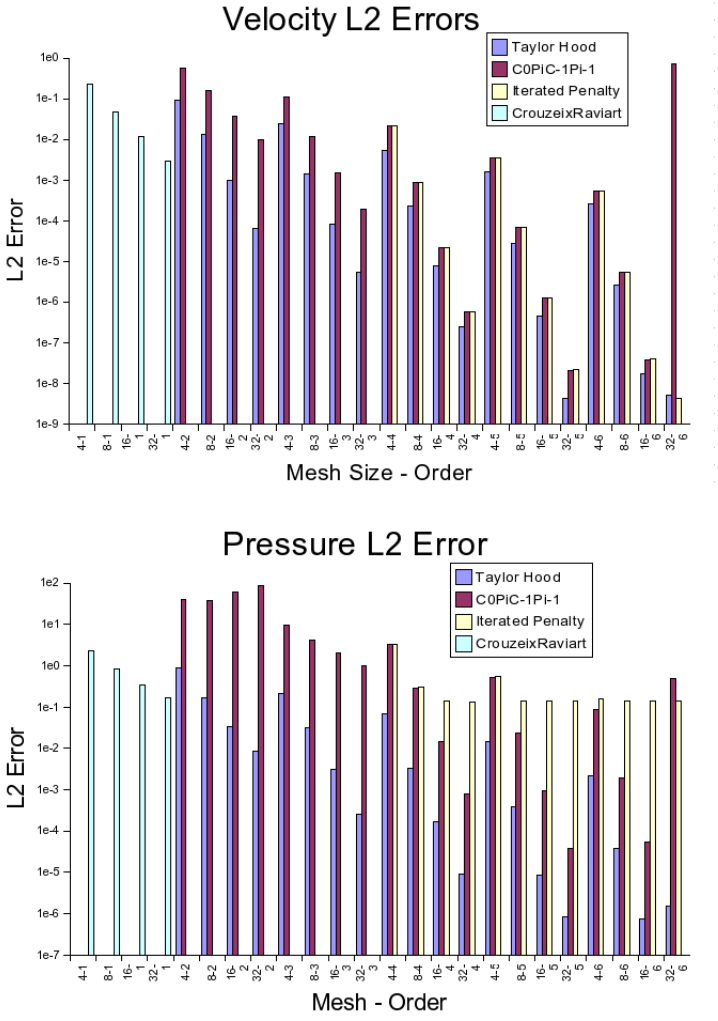


Figure 5.6: Comparison for Case 2, computed with FEniCS code, ordered by mesh size and order

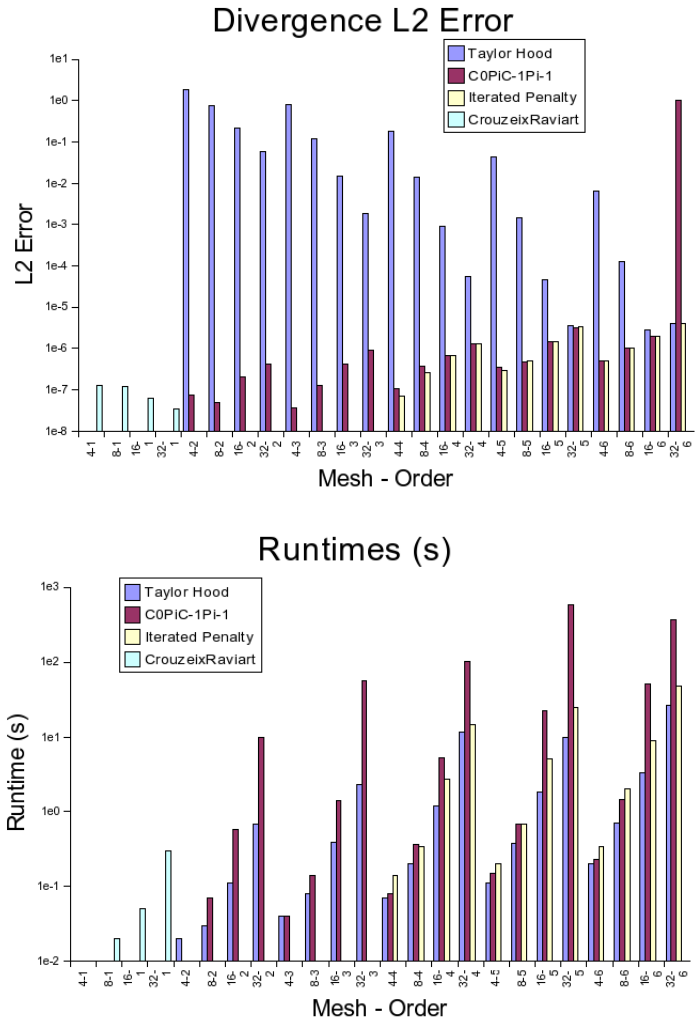


Figure 5.7: Comparison for Case 2, computed with FEniCS code, ordered by mesh size and order

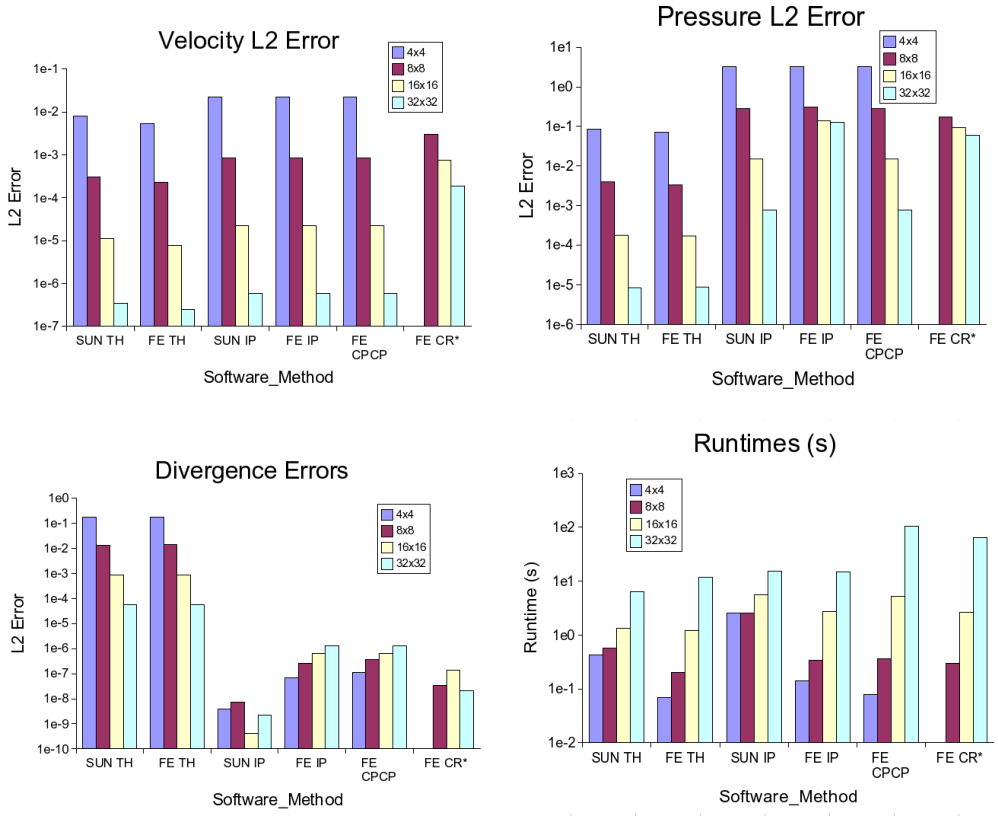


Figure 5.8: Comparison of 4th Order methods on Case 2 (* with Crouzeix-Raviart on a finer mesh to have equivalent number of DOFs)

The different numbers in Figures 5.3, 5.4, and 5.5 show how both Sundance and FEniCS compare, but also the trends of the methods with the Case 1 and Case 2. We do not include Case 0, simply because the numbers show no real variances. These figures show that the trends are as expected up to a certain precision which is dependent on the software package. For example in Figure 5.3, Sundance is able to achieve errors of 10^{-12} while the trends with FEniCS stop at 10^{-8}.

Figure 5.6 and 5.7, show how the methods compare depending on the mesh and order. Here we show the numbers from the FEniCS project because more methods were compared using this software. An interesting feature displayed in these charts is the comparison between Iterated Penalty and the CPCP element. Both of these methods are achieving the same results, which is expected since they are both implementing similar features, but these two methods have quite different algorithms used to implement them.

Figure 5.8, gives a slice of the data on the 4th order methods with the different software packages. Crouziex-Raviart, with an equivalent number of DOFs, is included to show the differences in a low order method for our case problem. Interestingly, Taylor Hood persists as the most accurate in pressure and velocity but as theory suggests is much worse for divergence. Since our case study only includes very smooth functions, this is expected but it should be noted that such results could be different in a non-smooth context. The runtimes are not a rigorous study, because there are many more optimizations one could use which were not the focus of this study. Rather, these runtimes show some data about the amount of work need for each method, thus these time correspond the the DOFs from Figure 5.1

The most notable information presented is that the theory behind using either a continuous element or a discontinuous element can be seen with the difference in pressure and velocity errors. Also shown is some problems with the different software packages such as FEniCS with pressure error of the iterated penalty method and Sundance seems to have a large initialization cost with the implementation of the iterated penalty method.

5.2 User Experience Results

Possibly a more interesting but unquantifiable evaluation is the user experience between implementing the method in the different codes. The first immediate evaluation is that both of these codes treat the assembly and

solution process very much the same. Both take a problem description, assemble a matrix and pass it to one of a number of linear solvers. Then one is able to evaluate the solution by writing it to a VTK format or doing some operations on it such as taking the L2 norm of the error. Furthermore, each has an interface to FIAT to define elements, and include both a C++ or Python (provided through the SWIG [5] package) interface. So at a first glance of the code everything except function names seem almost identical.

It is not until one starts pushing some of the limits of these codes do you really feel the differences. One should also realize that both of these codes are under active development.

The FEniCS code provides a smaller more self contained package, whereas Sundance depends heavily on Trilinos and the services it provides. The interface to FEniCS is less of a scripting style because one must include many precompiled header files that define the element spaces with FFC. This makes the process require both FFC and Dolfin compilations, but can be combined into one file in the Python interface. Sundance on the other hand has a much more seamless scripting style in both C++ and Python. Because of FEniCS smaller code base it has been easier to spot bugs and add new elements as we needed them. Sundance has provided a more robust system that is harder to break, both in the sense of memory management and floating point arithmetic. This lack of memory management can lead to problems with the solver which is the reason for the high error in 32x32 mesh for the 6th order $C^0P^iC^{-1}P^{-1}$ case.

The experience of the different methods varied for each software package. For example, when the study was started FEniCS did not have mixed method formulations implemented and Sundance did not have higher order methods. Both of these features were easy extensions, but required some implementation. As the project continued on, there were other hurdles such as learning the mesh structures and redefining them in a way to minimize singular points. Perhaps the largest bottleneck in the study was the implementation of the Iterated Penalty method. It so happens that the formulation we initially used was wrong and the software was giving was show large errors in the solutions. It was not until the proper penalties ($\rho = -r$) were discovered that the software gave consistent results.

Chapter 6

Conclusion

Automation of FEM is a process that has been motivated both by mathematical insight and utility in engineering and science. Although the process of building the discrete equations based on the function space and mesh is a rote application of calculus and algebra, the complexity of choosing the different implementations and knowing their affects gives rise to the need for this automation. As more research and implementations of automation is completed, the separation from the scientific models and low level coding of algorithms grows allowing for more evaluation and development of different models.

The lack of full automation is a call to further this growing field. One highlight we would especially like to see formalized is the missing link between global and local finite element spaces. While it is easy to do simple elements with nothing more than a consistent ordering of the degrees of freedom, more complicated elements are left out due to their interactions with other elements.

Our study on the Stokes equations shows the value of such automated FEM software, by highlighting the features predicted of different elements. It also shows how a researcher can compare their methods quickly with those that are predicted in literature. While this study is certainly meaningful and interesting to numerical analysts, it is still not at the level of the application scientist. We are currently looking at comparing some grade two fluid models that will both provide more novel studies numerically rather than theoretically and push the simulation engines in more intense computations.

Finally, the software that is currently being produced has already enabled the study of many different systems that were not done before. Possibly the

most interesting differences between the Sundance and FEniCS projects is how the goals of the developers affected the software's capabilities. As the field moves onward, software must look to incorporating symbolic differentiation to aid in new areas of automation include optimization, error correction, and control. These projects currently lack ability to interface with legacy codes, which would be valuable for scientists before they are able to completely switch to an automated system. Nonetheless, automated FEM software is growing in its popularity and is a valuable tool for scientists, engineers, and educators.

Bibliography

[1] The code and current copy of this paper is availble online at http://people.cs.uchicago.edu/~aterrel/Masters.

[2] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Web page*, 2001. Available from: `http://www.mcs.anl.gov/petsc`.

[3] W. Bangerth, R. Hartmann, and G. Kanschat, *Deal.ii*. Available from: `http://www.dealii.org`.

[4] C. Bauer and et al, *Ginac is not a cas*. Available from: `http://www.ginac.de/`.

[5] D. Beazley and et al, *Simplified wrapper and interface generator (swig)*. Available from: `http://www.swig.org`.

[6] S. Brenner and L. R. Scott, *The Mathematical Theory of Finite Element Methods*, Springer-Verlag, New York, 2nd ed., 2002.

[7] F. Brezzi and M. Fortin, *Mixed and Hybrid Finite Element Methods*, vol. 15 of Springer Series in Computational Mathematics, Springer-Verlag, New York, 1991.

[8] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, New York, Oxford, 1978.

[9] I. Danaila, F. Hecht, and O. Pironneau, *Freefem*. Available from: `http://www.freefem.org`.

[10] T. A. Davis, *Algorithm 832: Umfpack, an unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software, 30 (2004), pp. 196 – 199. Available from: `http://www.cise.ufl.edu/research/sparse/umfpack/`.

[11] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf, *The cactus framework and toolkit: Design and applications*, in Vector and Parallel Processing - VECPAR '2002, 5th International Conference, Springer, 2003. Available from: `http://www.cactuscode.org/Articles/Cactus_Goodale03a.pre.pdf`.

[12] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring,

A. Williams, and K. S. Stanley, *An overview of the trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423. Available from: `http://software.sandia.gov/trilinos/TrilinosACMTOMS2004.pdf`.

[13] T. Hughes, *The Finite Element Method, Linear Static and Dynamic Finite Element Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[14] W. R. INC, *Mathematica*. Available from: `http://www.wolfram.com`.

[15] R. C. Kirby, *Algorithm 839: Fiat, a new paradigm for computing finite element basis functions*, ACM Transactins on Mathematical Software, 30 (2004). Available from: `http://www.fenics.org/pub/documents/fiat/papers/fiat-toms-2004.pdf`.

[16] R. C. Kirby and A. Logg, *A compiler for variational forms*, ACM Transactions on Mathematical Software, 32 (2006). Available from: `http://www.fenics.org/pub/documents/ffc/papers/ffc-toms-2005.pdf`.

[17] A. Logg, *Automating the finite element method*. Preprint from The Finite Element Center, January 2006. Available from: `http://www.femcenter.org/pub/preprints/phiprint-2006-01.pdf`.

[18] A. Logg, J. Hoffman, J. Jansson, R. C. Kirby, and G. N. Wells, *The fenics project*. Available from: `http://www.fenics.org`.

[19] K. R. Long, *Sundance*. Available from: `http://software.sandia.gov/sundance/`.

[20] K. Mardal, *SyFi Tutorial*, June 2006. Available from: `http://www.fenics.org/pub/documents/syfi/syfi-user-manual/syfi-user-manual.pdf`.

[21] J. R. Stewart and H. C. Edwards, *A framework approach for developing parallel adaptive multiphysics applications*, Finite Elem. Anal. Des., 40 (2004), pp. 1599–1617.

[22] C. Taylor and P. Hood, *A numerical solution of the navier-stokes equations using the finite element technique*, Computers and Fluids, 1 (1973).

[23] M. Vogelius and L. R. Scott, *Conforming finite element methods for incompressible and nearly incompressible continua*, Large Scale Computations in Fluid Mechanics,, 22 (1985).

[24] J. Walter and M. Koch, *ublas*. Available from: `http://www.boost.org/libs/numeric/ublas/doc/index.htm`.