IMPLEMENTATION OF A VERTICALLY INTEGRATED ICE SHEET MOMENTUM BALANCE MODEL

Joshua Charles Campbell
The University of Montana

IMPLEMENTATION OF A VERTICALLY INTEGRATED ICE SHEET MOMENTUM BALANCE MODEL

By

Joshua Charles Campbell

Bachelor of Arts in Mathematics and Computer Science Mathematics
With Options in Abstract Mathematics and Combinatorics and Optimization,
The University of Montana, Missoula, MT, 2009

Thesis

presented in partial fulfillment of the requirements
for the degree of

Master of Science
in Computer Science

The University of Montana
Missoula, MT

Summer 2012

Campbell, Joshua Charles, M.S., August 2012 Computer Science

Implementation of a Vertically Integrated Ice Sheet Momentum Balance Model

Chairperson: Dr. Jesse Johnson

A new high-fidelity ice sheet momentum balance model meant for inclusion in the Glimmer community ice-sheet model is presented. As a component of the Community Earth Systems Model the newly developed momentum balance will directly benefit from ice/ocean and ice/atmosphere coupling efforts occurring elsewhere. The objectives of this thesis are to develop a model which converges quickly (quadratic convergence rates) for non-Newtonian Stokes flow approximations, and to provide a clear and low-level discussion of its derivation, variation and discretization.

The model utilizes the Finite Element Method to discretize variational forms of the first variation arising from the Galerkin method and for vertically-integrated Stokes flow. The model employs a hybridization of two commonly used approximations to Stokes flow. It couples the Shallow Shelf Approximation (SSA) and Shallow Ice Approximation (SIA). This approximation is then differentiated symbolically. Efficient sparse matrix formats are manipulated directly to avoid invoking costly sorting routines in the underlying linear solvers. The code was not only developed for standards-compliant FORTRAN 90 compilers but also for automatic differentiation tools. The model is verified against published model intercomparison projects.

ACKNOWLEDGEMENTS

I would like to thank Dr. Jesse Johnson for his patience in answering many, many questions and his mentoring and guidance during the thesis project. Additionally, Dr. Glen Granzow provided many hours of help, thoughts and encouragement. I would also like to thank my family for their support and encouragement. This material is based upon work supported by the National Science Foundation under Grant No. 0934662.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Thesis Organization

The rest of this thesis is organized as follows:

- **Chapter 1** provides an overview of this document, an overview of the model presented, motivates its construction discusses the related work.
- **Chapter 2** describes the scientific principles and equations behind the model. These begin with the upper half of Figure 1.1.
- **Chapter 3** covers the implementation of the model. The model implementation computes the momentum balance as in the bottom half of Figure 1.1.
- **Chapter 4** provides model validation results.
- **Chapter 5** reviews the features of this thesis, and covers possible future directions for the implementation presented here.

1.2 Ice Sheet Modelling

It is clearly important to obtain accurate predictions of sea-level rise and the contribution to that rise from ice sheets. Sea-level rise is a major component of climate change, and has huge socio-economic impact. Mitigating the potential damage from sea-level rise of just 50cm is projected to cost hundreds of billions of dollars [Anthoff et al., 2010]. Even just 25cm of sea-level rise would necessitate either abandoning or protecting over 125000km^2 from flooding [Bosello et al., 2007]. At the same time, the Greenland ice sheet alone could raise the sea-level 7m if its entire mass was added to the ocean [Cazenave and Llovel, 2010].

The third largest contribution to future sea-level rise, after thermal expansion, and glaciers and ice caps will be mass from ice sheets. This can happen in two ways: ice

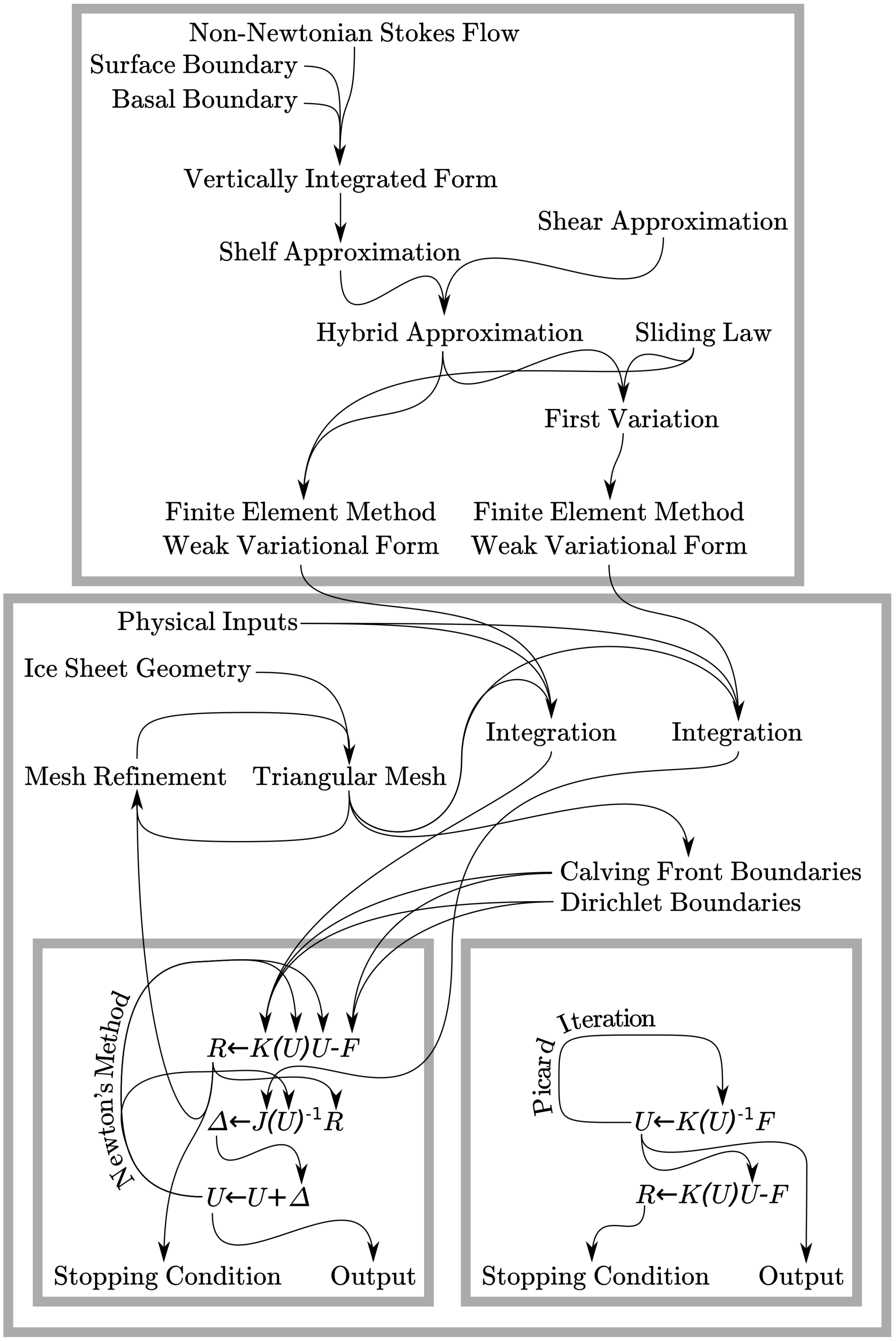


Figure 1.1 Model Overview

either freezing or melting on the ice sheet, and ice flowing off of land and into the ocean [Change, 2007]. In 2007 for the 4th IPCC report on climate change, however, "quantitative projections [could not] be made with confidence" about accelerations of flow in ice sheets due to ice shelf and ice sheet dynamics, including those caused by changes in friction with the underlying bed.

In order to model the movement and changes in ice sheets, ice sheet models such as GLIMMER-CISM are used. One part of such a model is the *momentum balance solver*, responsible for computing the velocity of the ice. Momentum balance models, such as the one presented in this thesis, treat ice as an incompressible fluid: ice behaves like a fluid on scales of kilometers and over periods of years.

1.3 Approximation of the Momentum Balance

Incompressible fluids, including ice, are governed by Stokes' equations. This is a system of partial differential equations (PDEs) solved by three continuous functions defined on a three-dimensional domain. The viscosity of ice, varies based on the strain under which it is flowing, adding a fourth continuous function which is related to the first three by Glen's flow law.

Unfortunately, to solve Stokes' equations, it is required that the viscosity of the fluid is known, and the viscosity must be calculated based on the strain that the ice experiences which is determined by relative velocities at nearby points in the ice. Therefore, the viscosity and the velocity must be found simultaneously.

Solving these equations in three dimensions is often slow or unfeasible on a computer. Furthermore it causes additional numerical difficulties arising from the fact that the ice is typically much thinner than it is wide or long. This extreme aspect ratio complicates the solution of three-dimensional PDEs numerically.

The Shallow Shelf Approximation and Shallow Ice Approximation are two classic partial differential equation approximations to the full three-dimensional Stokes equations described above for ice sheets. They both can be solved in two dimensions to approximate the three dimensional flow of ice sheets. Additionally, they both have strengths and weaknesses.

The Shallow Shelf Approximation (SSA) assumes zero vertical shearing. This is known as plug flow, because the top and bottom of the ice, as well as the entire column between them move at the same velocity. However, the Shallow Shelf Approximation provides a highly accurate approximation of the horizontal stresses and forces in ice with plug flow.

Conversely, the Shallow Ice Approximation (SIA) does not approximate horizontal stresses in the ice, and is solely concerned with vertical shearing. Another advantage of the Shallow Ice Approximation is that exact solution functions can be found without relying on numerical solutions.

Therefore, the model presented in this thesis consists of both SSA and the SIA which are coupled to each other to provide approximations of both horizontal stresses and vertical shearing, in a combined formulation called L1L2. In this formulation they are coupled in two ways: their contributions to ice velocity are summed, and their negative effects on ice viscosity are also summed. This allows both be solved with a more accurate viscosity approximation.

Unfortunately, all four PDE models are made nonlinear by Glen's flow law, which governs the viscosity, and therefore, the velocity of ice. In order to solve nonlinear PDEs, three techniques are commonly employed: fixed-point iteration, Newton's method, and Jacobian-Free Newton-Krylov (JFNK).

This thesis focuses on Newton's method for two reasons: it has good performance (asymptotically faster convergence to the solution in positive-definite regions) and yields the full Jacobian which is useful for inverting the model. Inverting the model is the processes of solving for one of the model inputs given that the velocities are known.

Fixed point iterations are slower and less likely to converge in a reasonable amount of time. However, they consume less memory (by a constant) factor. JFNK methods are a hybrid approach which attempts to yield the fast convergence of Newton's method without increasing memory footprint, however it does not form the full Jacobian and so the full Jacobian cannot be used for any other purposes.

Since the model PDEs are solved by continuous functions, on a computer these functions must be approximated by solving them at a finite number of locations. This is called discretization and the model presented here is discretized by the Finite Element Method.

The Finite Element Method involves splitting the domain into non-overlapping polygons, in this case triangles, and solving the equations at their vertices. The continuous function solutions are approximated as the sum of trial functions which are piecewise-defined polynomials (in this case, linear functions) summing to 1 at a vertex and taking on the value 0 over elements not incident to that vertex.

The Finite Element Method has several benefits. First, it allows the resolution (or element size) at which we approximate the continuous solutions to vary in different regions of the domain so that computational resources may be prioritized to some areas of ice and not others. Second, it involves integrating over the domain which removes all second derivatives from the equations being solved, leaving first derivatives as the highest derivative. Last, it provides an implicit variational principle which can be minimized. This principle is the residual, which represents the distance from the best solution.

1.4 Overview of the Construction of the Model Equations

In this document we present an ice sheet momentum balance model and its implementation, IceCamp. Specifically, the purpose of this model is to relate an estimate for the velocity of the ice in the sheet given to estimates of its physical shape, properties and interaction with the ocean and bed at a point in time. These models are implemented as FORTRAN programs.

We present a fast, yet reasonably accurate, ice sheet momentum balance model. This model strikes a compromise between fully 3-D models such as the one presented in Lemieux et al. [2011], and models employing the Shallow Shelf Approximation (SSA) or Shallow Ice Approximation (SIA).

Our model is based on several mathematical theories and approximations to the Stokes equations for fluid flow. Starting with the Stokes equations, we first integrate them along the vertical dimension to create a system of two equations as in MacAyeal [1997]. These two equations define the momentum balance for a planar "map view" of an ice sheet which will be solved for the velocities of that sheet. We then apply several approximations to form the Shallow Shelf Approximation (SSA). In order to provide an approximation not just for ice shelves but also for ice sheets, we incorporate the extra strain softening to form the L1L2 approximation from Pollard and Deconto [2009]. For both SSA and L1L2 we provide Fortran 90 software to solve for the velocities of the ice sheet given inputs of the ice sheet's physical properties and configuration at a point in time.

The model presented here differs from the model presented in Pollard and Deconto [2009] in several ways. First, the full Jacobian is formed and Newton's method is used. Second, SIA is symbolically integrated and coupled to SSA. Third, the steps required to reach the model from simpler models are described in detail. Fourth, we provide our own, independent, implementation.

After forming the system of equations to be solved, we additionally find their first variations with respect to the velocities of the ice. This is a precise, symbolic, route to solving the non-linear equations using Newton's method.

Two ways of implementing a fast Newton's Method-based solver for the system of

$$\begin{tikzcd}
\text{Original System} \arrow[r] \arrow[d] & \text{Finite Element Method} \arrow[r] & \text{Picard Iteration Solver} \arrow[d] \\
\text{First Variation} \arrow[d] & & \text{Automatic Differentiation} \arrow[d] \\
\text{Tangent Linear System} \arrow[r] & \text{Finite Element Method} \arrow[r] & \text{Newton's Method Solver}
\end{tikzcd}$$

Figure 1.2 Two Paths to a Newton's Method Solver

nonlinear PDEs were originally considered. The first option was to implement a Picard Iteration-based solver, and then use Automatic Differentiation tools to obtain a Jacobian which can then be used with Newton's Method. However, this option was not chosen.

Instead, we did not use Automatic Differentiation, and took an alternate approach, which was to find the first variation manually and then apply the Finite Element Method to find solutions of the tangent linear system for Newton's Method. This method yields a better understanding of the equations and mathematical machinery involved.

After the tangent linear system is found, we use the Finite Element Method as in Strang [2007] to solve the system of equations for a finite set of points in the domain, characterized by a triangular mesh. In fact, the entire implementation is heavily based on the techniques presented in Strang [2007], and the author of this document suggests that text as the primary reference to be used in understanding the techniques he has employed.

Specifically the elements employed are first-order, linear, triangular elements. These elements are characterized by binomial linear piecewise-defined test and trial functions $\phi(\vec{x})$ and $\psi(\vec{x})$. These functions take a value of 1 at a vertex p_j in the mesh, and are zero for all other vertices. Between p and adjacent vertices they are a linear function from 1 to 0.

The solution to our set of partial differential equations

$$f_1(\vec{u}) = 0$$

$$f_2(\vec{u}) = 0$$

are functions $u_i(\vec{x})$ of the plane (x_1, x_2). The Finite Element Method approximates equations as equations of the form

$$f_k \approx \sum_{l=1}^{n} K_{kl}\left(\sum_{j=1}^{n} U_{ij}\psi_{ij}\right)\phi_{kl}$$

where $K_{kl}(\dots)$ is a scalar as is U_{ij}. In order to solve them using well known linear solver routines, however, K_{kl} must be a discrete, affine function of $\vec{u}$. Unfortunately, the system of equations we set out to solve are continuous and non-linear so they cannot be factored as multiples of the form $K_{kl}u_i$. Therefore we employ the Finite Element Method and Newton's method to repeatedly solve a discrete linear approximation of δf_i around a point u. In order to obtain this linear approximation, we take the first variation of f_i at a point u first:

$$\delta f(\vec{u})(\vec{w}) = \lim_{\varepsilon \to 0} \frac{f(\vec{u} + \varepsilon w) - f(u)}{\varepsilon} = \left.\frac{d}{d\varepsilon}\right|_{\varepsilon=0} f(u + \varepsilon w).$$

Another mathematical technique employed is solving the weak form of the equations f rather than solving f directly. This is necessary because f contains second derivatives and the second derivative of $U_{ij}\phi_{ij}$ with respect to x is 0, which would result in solving the system

$$0\vec{u} = 0$$

after applying the Finite Element Method, which is certainly possible, but not particularly useful. In order to avoid this situation we solve instead the weak form of f which is $\int_\Omega f = \int_\Omega 0$. This is advantageous as the first derivative of a linear test (or trial) function is merely a constant. This has the secondary advantage of putting f into a "variational form" where the residual $r = f(u_{guess}) - 0$ has a physical meaning which we are particularly interested in, the excess force unaccounted for on the ice.

For the Picard iteration, we factor K into coefficients of U_{iter+1} as far as possible, and all remaining uses of U are estimated from the previous iterations solution U_{iter}. This process of re-estimating U is repeated until $|U_{iter+1} - U_{iter}|_2$ falls below a threshold. For Newton's method we repeatedly solve

$$0 = J_U(W) \approx \int_\Omega \delta f(\vec{u}_{iter})(\vec{w}) = 0$$

for the Newton update, $\vec{w}$ and then repeat the process with $\vec{U}_{iter+1} \leftarrow \vec{U} + \vec{W}$. This is the approach described by Knoll and Keyes [2004]. We also use the weak form here, $\int_\Omega \delta f$, as δf also contains second derivatives with respect to $\vec{x}$.

For the L1L2 approximation, we add two additional sub-equations to compute the vertical shearing terms $\overline{\frac{\partial u_i}{\partial x_3}}$ which are then used to adjust the viscosity of the ice. We do not use the weak form of these equations since they do not contain second derivatives and are already in the units we are interested in.

1.5 Design Considerations for the Implementation

FORTRAN was chosen because of its common use in the field of physical modelling. Since the model was intended to be able to be integrated with the GLIMMER-CISM ice sheet model eventually, which is written in FORTRAN, FORTRAN was the obvious choice for compatibility reasons.

However, FORTRAN also has other advantages, including its high performance [Bull et al., 2001], static types, array bounds checking, and availability of automatic differentiation tools such as OpenAD/F [Utke et al., 2008], though they are not employed in this implementation. In fact, the Fortran 90 code is specifically written with the possible future application of automatic differentiation tools in mind.

Even though it is written to be Fortran 90 compliant, some Fortran 90 features are intentionally avoided, especially `POINTER` and `ALLOCATABLE`. This is intended to make it easier to use with automatic differentiation tools in the future. Additionally, to allow for easy reuse of the code in other projects, global variables are never used, in order to avoid potential namespace conflicts.

Automatic Differentiation tools (AD tools) such as OpenAD/F [Utke et al., 2008] are tools which accept source code as input, and output source code which is the derivative of that code with respect to some input variable or parameter. Code is most easily handled by AD tools when it does not use advanced language features such as objects, pointers, references, recursion, or dynamic allocation.

In addition, the Community Earth Systems Model (CESM) community, of which GLIMMER-CISM is a part, avoids relying upon external libraries. Since this model is meant to be included, eventually, in GLIMMER-CISM, we only rely on two libraries: a linear solver and a MergeSort routine. GLIMMER-CISM already provides a variety of linear solvers, so this dependency would not add to the dependencies of GLIMMER-CISM. The MergeSort routine is a single file, and easily re-implemented if necessary. Using few dependencies means that many other components needed to be, and were, implemented from scratch.

1.6 Implementation Overview

In order to solve the equations above the Fortran 90 code must do many things. It must first set up the physical dimensions, shape, and properties of an ice sheet. Then it must generate a triangular mesh over the domain of the ice sheet. Next, it must analyze the mesh to determine various geometric properties. Then it must repeatedly form and solve affine systems of the form $0 = KU - F$ and possibly $0 = JW - R$ if Newton's method is selected until the solution is acceptable.

Each equation requires computing the entries of K_{ij} and possibly J_{ij} which depend on many factors. K and J are sparse matrices built in the sparse matrix format which they will be eventually consumed in. Sparse matrices are never converted between formats. These entries are the coefficients from an equation in the system of equations to be solved which multiply a single pair of test and trial functions, themselves representing coefficients of the velocity solution. The factors necessary for K_{ij} include the area of each element, the previous guess as to how fast the ice is flowing at each point and on average for each element, and the physical A or B value from Glen's flow law for the ice at each vertex. The A or B quantity depends on the temperature of the ice.

Additionally, K_{ij} depends on the viscosity at each point and on average over each element, the extra viscosity due to vertical shearing for the L1L2 version of the code, driving forces from the action of gravity on the ice, the friction with the bed, and the resulting forces from that friction. The spatial derivatives of these items and various derived quantities such as strain rates and averages are also required. For the L1L2 approximation, the quantities $\overline{\frac{\partial u_1}{\partial x_3}}$ and $\overline{\frac{\partial u_2}{\partial x_3}}$ must be computed by code that was generated by a computer algebra system.

For each point the code must also determine what boundary conditions apply and apply them. These boundary conditions include borders with the sea and locations of known velocity. In order to apply the boundary conditions at the border between the sheet and the sea the implementation must first locate and compute the normal vectors of these borders. It can apply known velocity boundary conditions using two methods.

Each time there is a quantity relating to ϕ and ψ and not their spatial derivatives, Gaussian quadrature rules must be invoked to find the integral $\int_{e\subset\Omega} c\phi d\psi$ where c and d are scalars. In order to determine the vertices on the borders of the mesh and their normal vectors, which are used where the ice borders the ocean, the mesh must be analyzed. First, the elements on the borders must be found, which are precisely all of the elements with no adjacent elements, then their incident edges on the borders. In order to find the entries of K and J that must necessarily be 0, adjacency lists for every pair of equations and vertices in the mesh must be generated.

Once the affine equations are formed, they are solved by an external library. If Newton's method is being employed, the residual function R must be computed first. The solution must be checked for distance from the previous solution, and the entire process repeated many times. Optionally, new vertices can be generated to divide elements that the implementation is having difficulty solving into additional elements. Finally, the solutions and other diagnostic data must be written out to a file. A short python program is also included to read this output file and display the output graphically.

1.7 Related Work

The works that this thesis depends on begin with Paterson [1983]. This reference contains all the fundamentals of ice sheet physics that our model is based on, including all of the relevant research that came before its publication. This includes our treatment of glacial ice as a non-Newtonian fluid following Glen's flow law.

We then proceed to apply techniques from another comprehensive book on the approximation and implementation of ice sheet models: MacAyeal [1997]. Specifically, we follow the shallow-shelf approximation (SSA) derivation presented by MacAyeal quite closely. MacAyeal also presents other approximations and a large number of implementation code examples.

Strang [2007] contains most of the relevant theory and practice for the finite element method (FEM), including code examples. The implementation presented in this thesis is structured similarly to the FEM example presented in Strang. Strang also covers many other related topics including fluid flow Modelling and finite differencing techniques.

Zienkiewicz and Taylor [2000] is a comprehensive resource on the finite element method. It covers the fundamentals and theory of FEM in detail along with examples. Additionally, it provides enhancements such as adaptive mesh refinement which are employed in this thesis. Zienkiewicz and Taylor also present the theory of variational principles and forms.

In addition to the three books listed, this thesis depends on several other publications. Pollard provides the L1L2 formulation in Pollard and Deconto [2009] which is used in this thesis. L1L2 is an improvement to the SSA approximation first proposed by Hindmarsh [2004]. In this work, Hindmarsh presents formulae for estimating the extra strain softening that a fully three dimensional Stokes flow implementation would model directly. We apply this approximation in our implementation.

Though the implementation presented in this thesis forms the full Jacobian instead of employing a Jacobian-free technique for the solutions of Stokes flow, the implementation is based on the theory described in Knoll and Keyes's discussion of Jacobian-free methods. Knoll and Keyes [2004] present a variety of approaches for efficiently solving non-linear Stokes flow problems.

Habermann et al. [2012] inform the techniques employed in this thesis to form the symbolic tangential linear system to SSA. This paper presents a method for iteratively solving the SSA equations for the basal traction instead of for the velocity. In doing so, Habermann *et al.* describes the transformations necessary for converting the forward SSA momentum balance model, which is typically solved for velocity, into an inverse model which can be solved for basal traction. Goldberg and Sergienko [2011] also provide a method for solving an approximation of Stokes flow inversely for the basal traction.

Bueler and Brown [2009] provides a thorough discussion of the modelling of the basal mechanics of ice sheets. This discussion includes general discussion of the treatment of vertical shearing stresses within ice sheets. Bueler *et al.* also provide a comparison of the shallow shelf approximation (SSA) and shallow ice approximation (SIA) approximation, and their treatment of basal and vertical shear stresses.

The momentum balance model in this thesis is intended to eventually work within the Glimmer community ice sheet model (Glimmer-CISM), presented in Rutt et al. [2009] and Lipscomb et al. [2009]. This model contains more than just momentum balance models: it also models ice sheet thickness evolution over time, internal thermodynamics within the ice sheet, basal hydrology, geothermal heat flux, and lithosphere elasticity. The implementation provided in this thesis takes these other factors as inputs meant to be calculated by Glimmer-CISM in a future implementation.

Glimmer-CISM contains other momentum balance models, including the model presented in Lemieux et al. [2011]. Lemieux et al. employ finite differences and Jacobian-free methods. In comparison, the model presented in this thesis employs finite elements and explicit-Jacobian methods. Jacobian-free methods have the advantage of avoiding the explicit formation of the Jacobian and flatten the solution process by combining the linear and nonlinear iterative solution processes, resulting in dramatic speed increases.

The implementation presented in this thesis is also designed to be easily adapted to processing by automatic differentiation tools such as OpenAD/F. Some of the necessary adaptations are described in Fagan et al. [2006].

Dukowicz et al. [2010] presents an approximation for ice sheet Stokes flow based on variational principles. The variational principles employed are specific to the Stokes flow equations and has a natural physical meaning. This technique minimizes the change in energy. Conversely, this thesis employs a variational formulation arising from the finite element method. This variational formulation is not specific to Stokes flow and minimizes the solution residual.

Goldberg [2011] also presents a vertically integrated approximation to Stokes flow momentum balance model employing the minimization of a variational principle. This model has similar goals to the model presented in this thesis and is validated with the same experiments. However, Goldberg's model is derived using variational principles other than the finite element method and does not provide a Jacobian suitable for solution with Newton's method, instead relying entirely on fixed-point iteration.

CHAPTER 2 Theoretical and Physical Foundation

2.1 Introduction

2.2 Notation

$A(T)$	Temperature-dependent ice rheological coefficient ($s^{-1}Pa^{-n}$ of the order 10^{-15} to 10^{-20}) (Pa = Pascals)
b	Ice sheet base elevation (m)
e	First variation of $\overline{\dot{\varepsilon}^2}$
e_p	First variation of $\dot{\varepsilon}_p^2$
G_i	Substitution variable for quotient rule
g	Acceleration due to gravity (9.8 m/s^2)
H	Substitution variable for quotient rule
h	Ice sheet thickness (m)
n	Ice rheological exponent (varies, about 3)
P	Pressure (Pascals)
s	Ice sheet surface elevation (m)
T	Temperature
T_p	Substitution variable representing terms of τ_p
t_i	First variation of τ_i
u_i	Velocity along coordinate x_i (m/a) (a = years, m/a = meters per year)
$u_{i\text{basal}}$	Velocity at the base along coordinate x_i (m/a)
w_i	Change in velocity along coordinate x_i (m/a)
$w_{i\text{basal}}$	Change in velocity at the base along coordinate x_i (m/a)
x_1, x_2	Orthogonal horizontal coordinates (m)
x_3	Vertical coordinates (m)
β^2	Basal sliding coefficient in ISMIP-HOM sliding law
η	Half viscosity
ρ	Density of ice ($\sim 910\ kg/m^3$)
ρ_w	Density of seawater ($\sim 1028\ kg/m^3$)
ε_v^2	Viscosity normalization constant (a^{-2})
ε	Scalar limit variable $\to 0$
$\dot{\varepsilon}$	Effective strain rate, the second invariant of the strain rate tensor (a^{-1})
$\dot{\varepsilon}_{ij}$	Strain rate components (a^{-1})
$\dot{\varepsilon}_p$	Effective plane strain rate, the second invariant of the strain rate tensor with no vertical terms (a^{-1})

σ	Effective stress, the second invariant of the deviatoric stress tensor (Pa)
σ_{ij}	Deviatoric stress components (Pa)
τ_i	Force due to friction with the bed in the x_i direction
τ_p	Shorthand representing $\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)$
ξ	Vertical integration variable (m), also used to represent $\frac{\zeta}{\dot{e}^2}$
ζ	First variation of η
$\mathcal{A}\ldots\mathcal{Z}$	Substitution variables representing functions of $\vec{u}$

2.3 Ice Sheet Physics

The flow of ice is governed by the nonlinear Stokes equations for incompressible fluid flow. From Paterson [1983]:

$$\begin{aligned}
\frac{\partial\sigma_{11}}{\partial x_1}+\frac{\partial\sigma_{12}}{\partial x_2}+\frac{\partial\sigma_{13}}{\partial x_3}-\frac{\partial P}{\partial x_1} &= 0 \\
\frac{\partial\sigma_{12}}{\partial x_1}+\frac{\partial\sigma_{22}}{\partial x_2}+\frac{\partial\sigma_{23}}{\partial x_3}-\frac{\partial P}{\partial x_2} &= 0 \\
\frac{\partial\sigma_{13}}{\partial x_1}+\frac{\partial\sigma_{23}}{\partial x_2}+\frac{\partial\sigma_{33}}{\partial x_3}-\frac{\partial P}{\partial x_3} &= -\rho g,
\end{aligned} \tag{2.1}$$

where σ_{ij} are the deviatoric stress components, ρ is the density of ice, g is the acceleration due to gravity, and P is the pressure. By Paterson [1983] we have that

$$\sigma_{ij}=\frac{\sigma^{\frac{1-n}{2}}}{2A\left(T\right)}\dot{\varepsilon}_{ij}.$$

This is Glen's flow law for ice. $\dot{\varepsilon}_{ij}$ are strain rate components. $A\left(T\right)$ is the temperature-dependent ice rheological coefficient. Substituting this into equations 2.1 gives us:

$$\begin{aligned}
\frac{\sigma^{\frac{1-n}{2}}}{2A(T)}\left(\frac{\partial\dot{\varepsilon}_{11}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{12}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{13}}{\partial x_3}\right)-\frac{\partial P}{\partial x_1} &= 0 \\
\frac{\sigma^{\frac{1-n}{2}}}{2A(T)}\left(\frac{\partial\dot{\varepsilon}_{12}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{22}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_3}\right)-\frac{\partial P}{\partial x_2} &= 0 \\
\frac{\sigma^{\frac{1-n}{2}}}{2A(T)}\left(\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{33}}{\partial x_3}\right)-\frac{\partial P}{\partial x_3} &= -\rho g
\end{aligned} \tag{2.2}$$

$$\dot{\varepsilon}=A(T)\,\sigma^n$$

We use the fact that $\frac{1}{2}\dot{\varepsilon}^{\frac{1}{2}}=A(T)\,\frac{1}{2}\sigma^{\frac{n}{2}}$ to see that

$$\dot{\varepsilon}^{\frac{1}{2}}=A(T)\,\sigma^{\frac{n}{2}},$$

which gives us

$$\frac{\sigma^{\frac{1-n}{2}}}{2A(T)}=\frac{\sigma^{\frac{1}{2}}}{2A(T)\,\sigma^{\frac{n}{2}}}=\frac{A(T)^{\frac{-1}{n}}\dot{\varepsilon}^{\frac{1}{2n}}}{2\dot{\varepsilon}^{\frac{1}{2}}}=\frac{A(T)^{\frac{-1}{n}}}{2\dot{\varepsilon}^{\frac{n-1}{2n}}}. \tag{2.3}$$

Substituting 2.3 into the system of equations 2.2 yields

$$\begin{aligned}
\frac{A(T)^{\frac{-1}{n}}}{2\dot{\varepsilon}^{\frac{n-1}{2n}}}\left(\frac{\partial\dot{\varepsilon}_{11}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{12}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{13}}{\partial x_3}\right)-\frac{\partial P}{\partial x_1} &= 0 \\
\frac{A(T)^{\frac{-1}{n}}}{2\dot{\varepsilon}^{\frac{n-1}{2n}}}\left(\frac{\partial\dot{\varepsilon}_{12}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{22}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_3}\right)-\frac{\partial P}{\partial x_2} &= 0 \\
\frac{A(T)^{\frac{-1}{n}}}{2\dot{\varepsilon}^{\frac{n-1}{2n}}}\left(\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{33}}{\partial x_3}\right)-\frac{\partial P}{\partial x_3} &= -\rho g.
\end{aligned} \tag{2.4}$$

We must also consider the upper and lower surfaces of the ice as boundary conditions. We take the following equations from MacAyeal, which represent the necessary force balances on the upper and lower surfaces.

Surface boundary condition:

$$\begin{aligned}
(\eta\dot{\varepsilon}_{11}-P)\frac{\partial s}{\partial x_1}+\eta\dot{\varepsilon}_{12}\frac{\partial s}{\partial x_2}-\eta\dot{\varepsilon}_{13} &= 0 \\
\eta\dot{\varepsilon}_{12}\frac{\partial s}{\partial x_1}+(\eta\dot{\varepsilon}_{22}-P)\frac{\partial s}{\partial x_2}-\eta\dot{\varepsilon}_{23} &= 0 \\
\eta\dot{\varepsilon}_{13}\frac{\partial s}{\partial x_1}+\eta\dot{\varepsilon}_{23}\frac{\partial s}{\partial x_2}-(\eta\dot{\varepsilon}_{33}-P) &= 0
\end{aligned} \tag{2.5}$$

Basal boundary condition:

$$\begin{aligned}
(\eta\dot{\varepsilon}_{11}-P)\frac{\partial b}{\partial x_1}+\eta\dot{\varepsilon}_{12}\frac{\partial b}{\partial x_2}-\eta\dot{\varepsilon}_{13} &= -\rho g h\frac{\partial b}{\partial x_1} \\
\eta\dot{\varepsilon}_{12}\frac{\partial b}{\partial x_1}+(\eta\dot{\varepsilon}_{22}-P)\frac{\partial b}{\partial x_2}-\eta\dot{\varepsilon}_{23} &= -\rho g h\frac{\partial b}{\partial x_2} \\
\eta\dot{\varepsilon}_{13}\frac{\partial b}{\partial x_1}+\eta\dot{\varepsilon}_{23}\frac{\partial b}{\partial x_2}-(\eta\dot{\varepsilon}_{33}-P) &= \rho g h
\end{aligned} \tag{2.6}$$

2.4 Vertical Integration of the Governing Equations

For the model we want a vertically-integrated set of equations that can be solved on the plane. To this end, we start by integrating the Stokes equations 2.4 over the vertical.

$$\begin{aligned}
\int_b^s \left[\frac{A(T)^{\frac{-1}{n}}}{2\dot{\varepsilon}^{\frac{n-1}{2n}}}\left(\frac{\partial\dot{\varepsilon}_{11}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{12}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{13}}{\partial x_3}\right)-\frac{\partial P}{\partial x_1}\right]dx_3 &= 0 \\
\int_b^s \left[\frac{A(T)^{\frac{-1}{n}}}{2\dot{\varepsilon}^{\frac{n-1}{2n}}}\left(\frac{\partial\dot{\varepsilon}_{12}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{22}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_3}\right)-\frac{\partial P}{\partial x_2}\right]dx_3 &= 0 \\
\int_b^s \left[\frac{A(T)^{\frac{-1}{n}}}{2\dot{\varepsilon}^{\frac{n-1}{2n}}}\left(\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}+\frac{\partial\dot{\varepsilon}_{33}}{\partial x_3}\right)-\frac{\partial P}{\partial x_3}\right]dx_3 &= -\rho g h
\end{aligned} \tag{2.7}$$

We can factor equations 2.7 to (all variables of integration are x_3, the vertical coordinate, where unspecified)

$$\begin{aligned}
\eta\left[\int_b^s\left(\frac{\partial\dot{\varepsilon}_{11}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{12}}{\partial x_2}\right)dx_3+\int_b^s\frac{\partial\dot{\varepsilon}_{13}}{\partial x_3}dx_3\right]-\int_b^s\frac{\partial P}{\partial x_1}dx_3 &= 0 \\
\eta\left[\int_b^s\left(\frac{\partial\dot{\varepsilon}_{12}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{22}}{\partial x_2}\right)dx_3+\int_b^s\frac{\partial\dot{\varepsilon}_{23}}{\partial x_3}dx_3\right]-\int_b^s\frac{\partial P}{\partial x_2}dx_3 &= 0 \\
\eta\left[\int_b^s\left(\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right)dx_3+\int_b^s\frac{\partial\dot{\varepsilon}_{33}}{\partial x_3}dx_3\right]-\int_b^s\frac{\partial P}{\partial x_3}dx_3 &= -\rho gh.
\end{aligned} \tag{2.8}$$

Then, moving the integrals in 2.8 to the inside, evaluating the vertical integral over pressure, and substituting averages for derivatives with respect to the vertical coordinate yields

$$\begin{aligned}
\frac{\partial}{\partial x_1}\int_b^s\eta\dot{\varepsilon}_{11}+\frac{\partial}{\partial x_2}\int_b^s\eta\dot{\varepsilon}_{12}+\eta\overline{\dot{\varepsilon}_{13}}h-\frac{\partial}{\partial x_1}\int_b^s Pdx_3 &= 0 \\
\frac{\partial}{\partial x_1}\int_b^s\eta\dot{\varepsilon}_{12}+\frac{\partial}{\partial x_2}\int_b^s\eta\dot{\varepsilon}_{22}+\eta\overline{\dot{\varepsilon}_{23}}h-\frac{\partial}{\partial x_2}\int_b^s Pdx_3 &= 0 \\
\frac{\partial}{\partial x_1}\int_b^s\eta\dot{\varepsilon}_{13}+\frac{\partial}{\partial x_2}\int_b^s\eta\dot{\varepsilon}_{23}-\eta\left(\overline{\dot{\varepsilon}_{11}}+\overline{\dot{\varepsilon}_{22}}\right)h-(P_s-P_b) &= -\rho gh..
\end{aligned} \tag{2.9}$$

When we apply the same steps to the basal boundary conditions 2.6, we get the following boundary condition:

$$\begin{aligned}
\frac{\partial}{\partial x_1}\int_b^s\eta\dot{\varepsilon}_{11}+\frac{\partial}{\partial x_2}\int_b^s\eta\dot{\varepsilon}_{12}+\eta\overline{\dot{\varepsilon}_{13}}h-\frac{\partial}{\partial x_1}\int_b^s Pdx_3 &= \rho gh\frac{\partial b}{\partial x_1} \\
\frac{\partial}{\partial x_1}\int_b^s\eta\dot{\varepsilon}_{12}+\frac{\partial}{\partial x_2}\int_b^s\eta\dot{\varepsilon}_{22}+\eta\overline{\dot{\varepsilon}_{23}}h-\frac{\partial}{\partial x_2}\int_b^s Pdx_3 &= \rho gh\frac{\partial b}{\partial x_1} \\
\frac{\partial}{\partial x_1}\int_b^s\eta\dot{\varepsilon}_{13}+\frac{\partial}{\partial x_2}\int_b^s\eta\dot{\varepsilon}_{23}-\eta\left(\overline{\dot{\varepsilon}_{11}}+\overline{\dot{\varepsilon}_{22}}\right)h+\rho gh &= (P_s-P_b).
\end{aligned} \tag{2.10}$$

Note that $s-x_3$ is our effective depth.

We will now apply Leibniz's rule as in pp100 of MacAyeal to the equations 2.9. Overbraced terms are terms substituted from the surface boundary conditions 2.5 and underbraced terms are terms substituted from the basal boundary conditions 2.10.

$$
\begin{aligned}
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{11} \overbrace{-\eta\dot{\varepsilon}_{11}\frac{\partial s}{\partial x_1}}^{\text{Surface BC}} \underbrace{+\eta\dot{\varepsilon}_{11}\frac{\partial b}{\partial x_1}}_{\text{Basal BC}} & \\
+\frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{12} \overbrace{-\eta\dot{\varepsilon}_{12}\frac{\partial s}{\partial x_2}} \underbrace{+\eta\dot{\varepsilon}_{12}\frac{\partial b}{\partial x_2}} & \\
\eta\overline{\dot{\varepsilon}_{13}}h \overbrace{+\dot{\varepsilon}_{13}} \underbrace{-\dot{\varepsilon}_{13}} & \\
-\frac{\partial}{\partial x_1}\int_b^s P dx_3 \overbrace{+P\frac{\partial s}{\partial x_1}} \underbrace{-P\frac{\partial b}{\partial x_1}} &= 0 \\
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{12} - \eta\dot{\varepsilon}_{12}\frac{\partial s}{\partial x_1} + \eta\dot{\varepsilon}_{12}\frac{\partial b}{\partial x_1} & \\
+\frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{22} - \eta\dot{\varepsilon}_{22}\frac{\partial s}{\partial x_2} + \eta\dot{\varepsilon}_{22}\frac{\partial b}{\partial x_2} & \\
+\eta\overline{\dot{\varepsilon}_{23}}h + \dot{\varepsilon}_{23} - \dot{\varepsilon}_{23} & \\
-\frac{\partial}{\partial x_2}\int_b^s P dx_3 + P\frac{\partial s}{\partial x_2} - P\frac{\partial b}{\partial x_2} &= 0 \\
\eta\left[\int\left(\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1} + \frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right)dx_3 + \dot{\varepsilon}_{33}\right] - P &= -\rho g\,(s - x_3)
\end{aligned}
\tag{2.11}
$$

Many terms of 2.11 cancel and we are left with

$$
\begin{aligned}
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{11} + \frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{12} + \eta\overline{\dot{\varepsilon}_{13}}h - \frac{\partial}{\partial x_1}\int_b^s P dx_3 &= \rho g h\frac{\partial b}{\partial x_1} \\
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{12} + \frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{22} + \eta\overline{\dot{\varepsilon}_{23}}h - \frac{\partial}{\partial x_2}\int_b^s P dx_3 &= \rho g h\frac{\partial b}{\partial x_2} \\
\eta\left[\int\left(\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1} + \frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right)dy + \dot{\varepsilon}_{33}\right] + \rho g\,(s - x_3) &= P.
\end{aligned}
\tag{2.12}
$$

We can now reduce the system to two equations 2.12 by substituting P to get

$$\begin{aligned}
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{11}+\frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{12}+\eta\overline{\dot{\varepsilon}_{13}}h & \\
-\frac{\partial}{\partial x_1}\int_b^s\left[\eta\left(\int\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3+\dot{\varepsilon}_{33}\right)+\rho g\left(s-x_3\right)\right]dx_3 &= \rho g h\frac{\partial b}{\partial x_1}\\
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{12}+\frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{22}+\eta\overline{\dot{\varepsilon}_{23}}h & \\
-\frac{\partial}{\partial x_2}\int_b^s\left[\eta\left(\int\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3+\dot{\varepsilon}_{33}\right)+\rho g\left(s-x_3\right)\right]dx_3 &= \rho g h\frac{\partial b}{\partial x_2}.
\end{aligned}$$

Replacing some vertically integrated terms with averages and move remaining integrals of equations 2.13 to the inside, taking note of which terms are missing from MacAyeal's derivation of the Shallow Shelf Approximation (SSA) yields equations 2.13. These terms are 0 because of the assumption of zero sheer in SSA, an assumption which we will apply later.

$$\begin{aligned}
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{11}+\frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{12}-\rho g\frac{\partial}{2\partial x_1}\left(b-s\right)^2 & \\
+\underbrace{\eta\overline{\dot{\varepsilon}_{13}}h-\frac{\partial}{\partial x_1}\eta\int_b^s\int_0^{\xi}\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3 d\xi}-\frac{\partial}{\partial x_1}\eta h\overline{\dot{\varepsilon}_{33}} &= \rho g h\frac{\partial b}{\partial x_1}\\
\frac{\partial}{\partial x_1}\int_b^s \eta\dot{\varepsilon}_{12}+\frac{\partial}{\partial x_2}\int_b^s \eta\dot{\varepsilon}_{22}-\rho g\frac{\partial}{2\partial x_2}\left(b-s\right)^2 & \\
+\underbrace{\eta\overline{\dot{\varepsilon}_{23}}h-\frac{\partial}{\partial x_2}\eta\int_b^s\int_0^{\xi}\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3 d\xi}_{\text{0 by zero-sheer approximation (SSA)}}-\frac{\partial}{\partial x_2}\eta h\overline{\dot{\varepsilon}_{33}} &= \rho g h\frac{\partial b}{\partial x_2}
\end{aligned} \tag{2.13}$$

We now apply the chain rule and flip both instances of $b-s$ in equations 2.13:

$$\begin{aligned}
& \frac{\partial}{\partial x_1}\eta h\overline{\dot{\varepsilon}_{11}} + \frac{\partial}{\partial x_2}h\eta\overline{\dot{\varepsilon}_{12}} - \rho g\left(s-b\right)\frac{\partial}{\partial x_1}\left(s-b\right) \\
& \underbrace{+\eta\overline{\dot{\varepsilon}_{13}}h - \frac{\partial}{\partial x_1}\eta\int_b^s\int_0^\xi\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3 d\xi} - \frac{\partial}{\partial x_1}\eta h\overline{\dot{\varepsilon}_{33}} = \rho g h\frac{\partial b}{\partial x_1} \\
& \frac{\partial}{\partial x_1}\eta h\overline{\dot{\varepsilon}_{12}} + \frac{\partial}{\partial x_2}h\eta\overline{\dot{\varepsilon}_{22}} - \rho g\left(s-b\right)\frac{\partial}{\partial x_2}\left(s-b\right) \\
& \underbrace{+\eta\overline{\dot{\varepsilon}_{23}}h - \frac{\partial}{\partial x_2}\eta\int_b^s\int_0^\xi\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3 d\xi}_{\text{0 by zero-sheer approximation (SSA)}} - \frac{\partial}{\partial x_2}\eta h\overline{\dot{\varepsilon}_{33}} = \rho g h\frac{\partial b}{\partial x_2}
\end{aligned} \tag{2.14}$$

Then we substitute the incomressibility condition into the equations 2.14 and simplify. This condition comes from assumed incomressibility of ice which allows us to make use of the fact that $\overline{\dot{\varepsilon}_{33}} = \overline{\dot{\varepsilon}_{11}} + \overline{\dot{\varepsilon}_{22}}$.

$$\begin{aligned}
& \frac{\partial}{\partial x_1}\eta h\overline{\dot{\varepsilon}_{11}} + \frac{\partial}{\partial x_2}h\eta\overline{\dot{\varepsilon}_{12}} - \rho g h\frac{\partial h}{\partial x_1} \\
& \underbrace{+\eta\overline{\dot{\varepsilon}_{13}}h - \frac{\partial}{\partial x_1}\eta\int_b^s\int_0^\xi\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3 d\xi} + \frac{\partial}{\partial x_1}\eta h\left(\overline{\dot{\varepsilon}_{11}}+\overline{\dot{\varepsilon}_{22}}\right) = \rho g h\frac{\partial b}{\partial x_1} \\
& \frac{\partial}{\partial x_1}\eta h\overline{\dot{\varepsilon}_{12}} + \frac{\partial}{\partial x_2}h\eta\overline{\dot{\varepsilon}_{22}} - \rho g h\frac{\partial h}{\partial x_2} \\
& \underbrace{+\eta\overline{\dot{\varepsilon}_{23}}h - \frac{\partial}{\partial x_2}\eta\int_b^s\int_0^\xi\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3 d\xi}_{\text{0 by zero-sheer approximation (SSA)}} + \frac{\partial}{\partial x_2}\eta h\left(\overline{\dot{\varepsilon}_{11}}+\overline{\dot{\varepsilon}_{22}}\right) = \rho g h\frac{\partial b}{\partial x_2}
\end{aligned} \tag{2.15}$$

Collecting terms of 2.15 and making use of the fact that $b+h=s$ yields:

$$\begin{aligned}
\frac{\partial}{\partial x_1}\eta h\left(2\overline{\dot{\varepsilon}_{11}}+\overline{\dot{\varepsilon}_{22}}\right)+\frac{\partial}{\partial x_2}h\eta\overline{\dot{\varepsilon}_{12}} & \\
\underbrace{+\eta\overline{\dot{\varepsilon}_{13}}h-\frac{\partial}{\partial x_1}\eta\int_b^s\int_0^\xi\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3d\xi} &= \rho gh\frac{\partial s}{\partial x_1} \\
\frac{\partial}{\partial x_1}\eta h\overline{\dot{\varepsilon}_{12}}+\frac{\partial}{\partial x_2}h\eta\left(2\overline{\dot{\varepsilon}_{22}}+\overline{\dot{\varepsilon}_{11}}\right) & \\
\underbrace{+\eta\overline{\dot{\varepsilon}_{23}}h-\frac{\partial}{\partial x_2}\eta\int_b^s\int_0^\xi\left[\frac{\partial\dot{\varepsilon}_{13}}{\partial x_1}+\frac{\partial\dot{\varepsilon}_{23}}{\partial x_2}\right]dx_3d\xi}_{0\text{ by zero-sheer approximation (SSA)}} &= \rho gh\frac{\partial s}{\partial x_2}.
\end{aligned} \tag{2.16}$$

We then substitute the definition of $\dot{\varepsilon}_{ij}$ into equations 2.16, which gives us the equations:

$$\begin{aligned}
\frac{\partial}{\partial x_1}\left(\eta h\left(2\frac{\partial\overline{u_1}}{\partial x_1}+\frac{\partial\overline{u_2}}{\partial x_2}\right)\right)+\frac{\partial}{\partial x_2}\left(\eta h\left(\frac{\partial\overline{u_1}}{\partial x_2}+\frac{1}{2}\frac{\partial\overline{u_2}}{\partial x_1}\right)\right) & \\
+\eta\frac{1}{2}\left(\frac{\partial u_3}{\partial x_1}+\frac{\partial u_1}{\partial x_3}\right) & \\
-\frac{\partial}{\partial x_1}\eta\int_b^s\int_0^\xi\left[\frac{1}{2}\left(\frac{\partial u_3}{\partial x_1}+\frac{\partial u_1}{\partial x_3}\right)+\frac{1}{2}\left(\frac{\partial u_3}{\partial x_2}+\frac{\partial u_2}{\partial x_3}\right)\right]dx_3d\xi &= \rho gh\frac{\partial s}{\partial x_1} \\
\frac{\partial}{\partial x_2}\left(\eta h\left(2\frac{\partial\overline{u_2}}{\partial x_2}+\frac{\partial\overline{u_1}}{\partial x_1}\right)\right)+\frac{\partial}{\partial x_1}\left(\eta h\left(\frac{\partial\overline{u_2}}{\partial x_1}+\frac{1}{2}\frac{\partial\overline{u_1}}{\partial x_2}\right)\right) & \\
+\eta\frac{1}{2}\left(\frac{\partial u_3}{\partial x_2}+\frac{\partial u_2}{\partial x_3}\right) & \\
-\frac{\partial}{\partial x_2}\eta\int_b^s\int_0^\xi\left[\frac{1}{2}\left(\frac{\partial u_3}{\partial x_1}+\frac{\partial u_1}{\partial x_3}\right)+\frac{1}{2}\left(\frac{\partial u_3}{\partial x_2}+\frac{\partial u_2}{\partial x_3}\right)\right]dx_3d\xi &= \rho gh\frac{\partial s}{\partial x_2}.
\end{aligned} \tag{2.17}$$

The last two terms of each equation in system 2.17 are approximated by zero in the the Shallow Shelf Approximation (SSA):

$$\begin{aligned}
\frac{\partial}{\partial x_1}\left(\eta h\left(2\frac{\partial\overline{u_1}}{\partial x_1}+\frac{\partial\overline{u_2}}{\partial x_2}\right)\right)+\frac{\partial}{\partial x_2}\left(\eta h\left(\frac{\partial\overline{u_1}}{\partial x_2}+\frac{1}{2}\frac{\partial\overline{u_2}}{\partial x_1}\right)\right) &= \rho gh\frac{\partial s}{\partial x_1} \\
\frac{\partial}{\partial x_2}\left(\eta h\left(2\frac{\partial\overline{u_2}}{\partial x_2}+\frac{\partial\overline{u_1}}{\partial x_1}\right)\right)+\frac{\partial}{\partial x_1}\left(\eta h\left(\frac{\partial\overline{u_2}}{\partial x_1}+\frac{1}{2}\frac{\partial\overline{u_1}}{\partial x_2}\right)\right) &= \rho gh\frac{\partial s}{\partial x_2}.
\end{aligned} \tag{2.18}$$

2.5 Vertical Shear Approximation

We don't wish to approximate all vertical derivatives as zero in equations 2.17, so from Pollard we apply the approximation

$$\frac{\partial u_1}{\partial x_3} = 2A\left[\sigma_{13}^2 + \sigma_{23}^2 + \sigma_{plane}^2\right]^{\frac{n-1}{2}} \sigma_{13} \tag{2.19}$$

with

$$\sigma_{plane}^2 = \eta^2 \dot{\varepsilon}_{plane}^2$$

and

$$\sigma_{i3} = \tau_i \left(\frac{s - x_3}{s - b}\right) = \tau_i h_{\%}.$$

The basal force due to friction along dimension i is τ_i. This defined by some sliding law such as

$$\tau_i = \beta^2 u_{i\text{basal}}, \tag{2.20}$$

which is a common sliding law. Therefore, the complete form of the approximation from equation 2.19 that we apply is:

$$\begin{aligned} \frac{\partial u_1}{\partial x_3} &= -2A\left[(\tau_1 h_{\%})^2 + (\tau_2 h_{\%})^2 + \eta^2 \dot{\varepsilon}_{plane}^2\right]^{\frac{n-1}{2}} \tau_1 h_{\%} \\ \frac{\partial u_2}{\partial x_3} &= -2A\left[(\tau_1 h_{\%})^2 + (\tau_2 h_{\%})^2 + \eta^2 \dot{\varepsilon}_{plane}^2\right]^{\frac{n-1}{2}} \tau_2 h_{\%}. \end{aligned} \tag{2.21}$$

For brevity we will use

$$h_{\%} = \frac{z - b}{h} \tag{2.22}$$

where z is the current height. $h_{\%}$ is used as a variable of integration.

However we need the vertical average of these terms for our implementation so we integrate equations 2.21 and divide by the height.

$$\begin{aligned} \overline{\frac{\partial u_1}{\partial x_3}} &= \frac{1}{s-b}\int_1^0 \left(2A\left[(\tau_1 h_{\%})^2+(\tau_2 h_{\%})^2+\eta^2\dot{\varepsilon}_{plane}^2\right]^{\frac{n-1}{2}}\tau_1 h_{\%}\right) dh_{\%} \\ \overline{\frac{\partial u_2}{\partial x_3}} &= \frac{1}{s-b}\int_1^0 \left(2A\left[(\tau_1 h_{\%})^2+(\tau_2 h_{\%})^2+\eta^2\dot{\varepsilon}_{plane}^2\right]^{\frac{n-1}{2}}\tau_2 h_{\%}\right) dh_{\%}. \end{aligned} \tag{2.23}$$

We perform the integration of equations 2.23 using the Maple Computer Algebra System, which yields the following equations:

$$\begin{aligned} \overline{\frac{\partial u_1}{\partial x_3}} &= \frac{2h\tau_1 A\left(\eta^{n+1}\dot{\varepsilon}_p^{n+1}-\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)^{\frac{n+1}{2}}\right)}{h\left(n\tau_1^2+n\tau_2^2+\tau_1^2+\tau_2^2\right)} \\ \overline{\frac{\partial u_2}{\partial x_3}} &= \frac{2h\tau_2 A\left(\eta^{n+1}\dot{\varepsilon}_p^{n+1}-\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)^{\frac{n+1}{2}}\right)}{h\left(n\tau_1^2+n\tau_2^2+\tau_1^2+\tau_2^2\right)} \end{aligned} \tag{2.24}$$

It should be noted that when $n = 3$ the equations 2.24 reduce to

$$\begin{aligned} \overline{\frac{\partial u_1}{\partial x_3}} &= \frac{A\tau_1\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right)}{2} \\ \overline{\frac{\partial u_2}{\partial x_3}} &= \frac{A\tau_2\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right)}{2}. \end{aligned} \tag{2.25}$$

Other approximations for terms involving u_3 can be obtained from MacAyeal sections 3.1

and 3.3 and by adding the base height:

$$\begin{aligned}
u_3 &= \frac{\partial x_3}{\partial t} = \nabla \cdot (\vec{u}b) - \nabla \cdot (\vec{u}(x_3 - b)) \\
u_3 &= \frac{\partial b}{\partial x_1} u_1 + \frac{\partial b}{\partial x_2} u_2 - \frac{\partial u_2}{\partial x_2}(x_3 - b) - \frac{\partial u_1}{\partial x_1}(x_3 - b) \\
\frac{\partial u_3}{\partial x_1} &= \frac{\partial}{\partial x_1}\left(\frac{\partial b}{\partial x_1} u_1 + \frac{\partial b}{\partial x_2} u_2 - \frac{\partial u_2}{\partial x_2}(x_3 - b) - \frac{\partial u_1}{\partial x_1}(x_3 - b)\right) \\
\frac{\partial u_3}{\partial x_2} &= \frac{\partial}{\partial x_2}\left(\frac{\partial b}{\partial x_2} u_1 + \frac{\partial b}{\partial x_2} u_2 - \frac{\partial u_2}{\partial x_2}(x_3 - b) - \frac{\partial u_1}{\partial x_1}(x_3 - b)\right).
\end{aligned} \tag{2.26}$$

However, we will not apply these approximations in this discussion, instead we will approximate them as zero as in Pollard. The system of equations, combining equations 2.17, 2.24, and zero-approximations is:

$$\begin{aligned}
&\frac{\partial}{\partial x_1}\left(\eta h \left(2\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2}\right)\right) \\
&+\frac{\partial}{\partial x_2}\left(\eta h \left(\frac{\partial u_1}{\partial x_2} + \frac{1}{2}\frac{\partial u_2}{\partial x_1}\right)\right) = \rho g h \frac{\partial s}{\partial x_1} + \tau_1 \\
&\frac{\partial}{\partial x_2}\left(\eta h \left(2\frac{\partial u_2}{\partial x_2} + \frac{\partial u_1}{\partial x_1}\right)\right) \\
&+\frac{\partial}{\partial x_1}\left(\eta h \left(\frac{\partial u_2}{\partial x_1} + \frac{1}{2}\frac{\partial u_1}{\partial x_2}\right)\right) = \rho g h \frac{\partial s}{\partial x_2} + \tau_2
\end{aligned} \tag{2.27}$$

from the Shallow Shelf Approximation,

$$\begin{aligned}
\frac{2\tau_1 A \left(\eta^{n+1}\dot{\varepsilon}_p^{n+1} - \left(\tau_2^2 + \tau_1^2 + \eta^2\dot{\varepsilon}_p^2\right)^{\frac{n+1}{2}}\right)}{\left(n\tau_1^2 + n\tau_2^2 + \tau_1^2 + \tau_2^2\right)} &= \overline{\frac{\partial u_1}{\partial x_3}} \\
\frac{2\tau_2 A \left(\eta^{n+1}\dot{\varepsilon}_p^{n+1} - \left(\tau_2^2 + \tau_1^2 + \eta^2\dot{\varepsilon}_p^2\right)^{\frac{n+1}{2}}\right)}{\left(n\tau_1^2 + n\tau_2^2 + \tau_1^2 + \tau_2^2\right)} &= \overline{\frac{\partial u_2}{\partial x_3}}
\end{aligned}$$

from Pollard's shearing approximation, and

$$\begin{aligned}
\varepsilon_{\nu}^{2}+\left(\overline{\frac{\partial u_1}{\partial x_1}}\right)^2+\left(\overline{\frac{\partial u_2}{\partial x_2}}\right)^2+\frac{1}{4}\left(\overline{\frac{\partial u_1}{\partial x_2}}+\overline{\frac{\partial u_2}{\partial x_1}}\right)^2+\overline{\frac{\partial u_1}{\partial x_1}\frac{\partial u_2}{\partial x_2}} &= \overline{\dot{\varepsilon}_p^2} \\
\left[\overline{\dot{\varepsilon}_p^2}+\frac{1}{4}\left(\overline{\frac{\partial u_1}{\partial x_3}}\right)^2+\frac{1}{4}\left(\overline{\frac{\partial u_2}{\partial x_3}}\right)^2\right] &= \overline{\dot{\varepsilon}^2} \\
\frac{1}{\overline{A}^{\frac{1}{n}}\left(\overline{\dot{\varepsilon}^2}\right)^{\frac{n-1}{2n}}} &= \eta
\end{aligned}$$

for the non-linear viscosity calculation. In the equations 2.27, the first four approximation equations are the equations to be solved, and the remaining three viscosity equations are substituted into them.

2.6 Variational Principles and the Jacobian

In order to apply Newton's method for a solution, we must find the first variation of the system of equations. The first variation of an equation F is,

$$\delta F(u)(w)=\lim_{\varepsilon\to 0}\frac{F(u+\varepsilon w)-F(u)}{\varepsilon}=\left.\frac{d}{d\varepsilon}\right|_{\varepsilon=0}F(u+\varepsilon w). \tag{2.28}$$

Applying this principal to the equations 2.27 above we get, for the right hand sides,

$$\begin{aligned}
\left.\frac{d}{d\varepsilon}\rho g h\frac{\partial z_s}{\partial x_1}\right|_{\varepsilon=0} &= \ldots \\
\left.\frac{d}{d\varepsilon}\rho g h\frac{\partial z_s}{\partial x_2}\right|_{\varepsilon=0} &= \ldots \\
\left.\frac{d}{d\varepsilon}\overline{\frac{\partial}{\partial x_3}}(u_1+\varepsilon w_1)\right|_{\varepsilon=0} &= \ldots \\
\left.\frac{d}{d\varepsilon}\overline{\frac{\partial}{\partial x_3}}(u_1+\varepsilon w_1)\right|_{\varepsilon=0} &= \ldots
\end{aligned} \tag{2.29}$$

Equations 2.29 evaluate to:

$$\begin{aligned}
0 &= \ldots \\
0 &= \ldots \\
\left.\overline{\frac{\partial}{\partial x_3}}\frac{d}{d\varepsilon}(\varepsilon w_1)\right|_{\varepsilon=0} = \overline{\frac{\partial w_1}{\partial x_3}} &= \ldots \\
\left.\overline{\frac{\partial}{\partial x_3}}\frac{d}{d\varepsilon}(\varepsilon w_2)\right|_{\varepsilon=0} = \overline{\frac{\partial w_1}{\partial x_3}} &= \ldots
\end{aligned} \tag{2.30}$$

Filling in the left hand sides of equations 2.27 and applying 2.28 to them, we get:

$$\begin{aligned}
0 &= \frac{d}{d\varepsilon}\frac{\partial}{\partial x_1}\left(2\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(2\frac{\partial}{\partial x_1}\left(u_1+\varepsilon w_1\right)+\frac{\partial}{\partial x_2}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg| \\
&+\frac{d}{d\varepsilon}\frac{\partial}{\partial x_2}\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(\frac{\partial}{\partial x_2}\left(u_1+\varepsilon w_1\right)+\frac{1}{2}\frac{\partial}{\partial x_1}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0} \\
0 &= \frac{d}{d\varepsilon}\frac{\partial}{\partial x_2}\left(2\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(2\frac{\partial}{\partial x_2}\left(u_2+\varepsilon w_2\right)+\frac{\partial}{\partial x_1}\left(u_1+\varepsilon w_1\right)\right)\right)\Bigg|_{\varepsilon=0} \\
&+\frac{d}{d\varepsilon}\frac{\partial}{\partial x_1}\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(2\frac{\partial}{\partial x_2}\left(u_1+\varepsilon w_1\right)+\frac{1}{2}\frac{\partial}{\partial x_1}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0} \\
\overline{\frac{\partial w_1}{\partial x_3}} &= \frac{d}{d\varepsilon}\frac{2\tau_1 A\left(\left[\eta^{n+1}\left(\vec{u}+\varepsilon\vec{w}\right)\right]\left[\dot{\varepsilon}_p^{n+1}\left(\vec{u}+\varepsilon\vec{w}\right)\right]-\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)^{\frac{n+1}{2}}\right)}{\left(n\tau_1^2+n\tau_2^2+\tau_1^2+\tau_2^2\right)}\Bigg|_{\varepsilon=0} \\
\overline{\frac{\partial w_2}{\partial x_3}} &= \frac{d}{d\varepsilon}\frac{2\tau_1 A\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)^{n+1}\dot{\varepsilon}\left(\vec{u}+\varepsilon\vec{w}\right)_p^{n+1}-\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)^{\frac{n+1}{2}}\right)}{\left(n\tau_1^2+n\tau_2^2+\tau_1^2+\tau_2^2\right)}\Bigg|_{\varepsilon=0}
\end{aligned} \tag{2.31}$$

For the sake of brevity we will define the following functions representing first variations of functions employed by the above equations:

$$\zeta\left(\vec{u},\vec{w}\right)=\frac{d}{d\varepsilon}\eta\left(\vec{u}+\varepsilon\vec{w}\right)\Bigg|_{\varepsilon=0} \tag{2.32}$$

$$e\left(\vec{u},\vec{w}\right)=\frac{d}{d\varepsilon}\overline{\dot{\varepsilon}^2}\left(\vec{u}+\varepsilon\vec{w}\right)\Bigg|_{\varepsilon=0} \tag{2.33}$$

$$e_p\left(\vec{u},\vec{w}\right)=\frac{d}{d\varepsilon}\overline{\dot{\varepsilon}_p^2}\left(\vec{u}+\varepsilon\vec{w}\right)\Bigg|_{\varepsilon=0} \tag{2.34}$$

$$t\left(\vec{u}+\varepsilon\vec{w}\right)=\frac{d}{d\varepsilon}\tau_i\left(\vec{u}+\varepsilon\vec{w}\right)\Bigg|_{\varepsilon=0} \tag{2.35}$$

Note that for the common sliding law, by applying the first variation 2.28 to equation 2.20, we get:

$$t_i=\beta^2 w_{i\text{basal}}. \tag{2.36}$$

We also define the function

$$\tau_p^{\frac{n+1}{2}}\left(\vec{u}+\varepsilon\vec{w}\right)=\left(\tau_2^2\left(\vec{u}+\varepsilon\vec{w}\right)+\tau_1^2\left(\vec{u}+\varepsilon\vec{w}\right)+\eta^2(\vec{u}+\varepsilon\vec{w})\dot{\varepsilon}_p^2(\vec{u}+\varepsilon\vec{w})\right)^{\frac{n+1}{2}} \tag{2.37}$$

and its first variation

$$t_p\left(\vec{u},\vec{w}\right)=\frac{d}{d\varepsilon}\tau_p^{\frac{n+1}{2}}\left(\vec{u}+\varepsilon\vec{w}\right)\Big|_{\varepsilon=0}. \tag{2.38}$$

Proceeding with the differentiation of 2.31, we obtain:

$$
\begin{aligned}
0 &= \frac{\partial}{\partial x_1}\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(2\frac{\partial}{\partial x_1}\frac{d}{d\varepsilon}\left(u_1+\varepsilon w_1\right)+\frac{\partial}{\partial x_2}\frac{d}{d\varepsilon}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0} \qquad (2.39)\\
&+\frac{\partial}{\partial x_1}\left(\zeta\left(\vec{u},\vec{w}\right)h\left(2\frac{\partial}{\partial x_1}\left(u_1+\varepsilon w_1\right)+\frac{\partial}{\partial x_2}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0}\\
&+\frac{\partial}{\partial x_2}\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(\frac{\partial}{\partial x_2}\frac{d}{d\varepsilon}\left(u_1+\varepsilon w_1\right)+\frac{1}{2}\frac{\partial}{\partial x_1}\frac{d}{d\varepsilon}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0}\\
&+\frac{\partial}{\partial x_2}\left(\zeta\left(\vec{u},\vec{w}\right)h\left(\frac{\partial}{\partial x_2}\left(u_1+\varepsilon w_1\right)+\frac{1}{2}\frac{\partial}{\partial x_1}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0}\\
0 &= \frac{\partial}{\partial x_2}\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(2\frac{\partial}{\partial x_2}\frac{d}{d\varepsilon}\left(u_2+\varepsilon w_2\right)+\frac{\partial}{\partial x_1}\frac{d}{d\varepsilon}\left(u_1+\varepsilon w_1\right)\right)\right)\Bigg|_{\varepsilon=0}\\
&+\frac{\partial}{\partial x_2}\left(\zeta\left(\vec{u},\vec{w}\right)h\left(2\frac{\partial}{\partial x_2}\left(u_1+\varepsilon w_1\right)+\frac{\partial}{\partial x_2}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0}\\
&+\frac{\partial}{\partial x_1}\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)h\left(\frac{\partial}{\partial x_2}\frac{d}{d\varepsilon}\left(u_1+\varepsilon w_1\right)+\frac{1}{2}\frac{\partial}{\partial x_1}\frac{d}{d\varepsilon}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0}\\
&+\frac{\partial}{\partial x_1}\left(\zeta\left(\vec{u},\vec{w}\right)h\left(\frac{\partial}{\partial x_2}\left(u_1+\varepsilon w_1\right)+\frac{1}{2}\frac{\partial}{\partial x_1}\left(u_2+\varepsilon w_2\right)\right)\right)\Bigg|_{\varepsilon=0}\\
\overline{\frac{\partial w_1}{\partial x_3}} &= \left(G_1'H-G_1H'\right)/H^2\\
\overline{\frac{\partial w_2}{\partial x_3}} &= \left(G_2'H-G_2H'\right)/H^2\\
G_1 &= 2\tau_1 A\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)^{n+1}\dot{\varepsilon}\left(\vec{u}+\varepsilon\vec{w}\right)_p^{n+1}-\tau_p^{\frac{n+1}{2}}\right)\Bigg|_{\varepsilon=0}\\
G_1' &= 2t_1A\left(\eta^{n+1}\dot{\varepsilon}_p^{n+1}-\tau_p^{\frac{n+1}{2}}\right)\Bigg|_{\varepsilon=0}\\
&+2\tau_1A\left(\left(n+1\right)\left[\zeta^n\dot{\varepsilon}_p^{n+1}+\eta^{n+1}e_p^n\right]-t_p\right)\Big|_{\varepsilon=0}\\
G_2 &= 2\tau_1 A\left(\eta\left(\vec{u}+\varepsilon\vec{w}\right)^{n+1}\dot{\varepsilon}_p^{n+1}\left(\vec{u}+\varepsilon\vec{w}\right)-\tau_p^{\frac{n+1}{2}}\right)\Bigg|_{\varepsilon=0}\\
G_2' &= 2t_2A\left(\eta^{n+1}\dot{\varepsilon}_p^{n+1}-\tau_p^{\frac{n+1}{2}}\right)\Bigg|_{\varepsilon=0}\\
&+2\tau_2A\left(\left(n+1\right)\left[\zeta^n\dot{\varepsilon}_p^{n+1}+\eta^{n+1}e_p^n\right]-t_p\right)\Big|_{\varepsilon=0}\\
H &= \left(n\left[\tau_1^2\left(\vec{u}+\varepsilon\vec{w}\right)\right]+n\tau_2^2+\tau_1^2+\tau_2^2\right)\Big|_{\varepsilon=0}\\
H' &= \left(2n\left[\tau_1\left(\vec{u}+\varepsilon\vec{w}\right)\right]\left[t_1\left(\vec{u},\vec{w}\right)\right]+2n\tau_2t_2+2\tau_1t_1+2\tau_2t_2\right)_{\varepsilon=0}
\end{aligned}
$$

In the next step we apply the condition $\varepsilon = 0$ so where in the previous step, equations 2.39 had η, $\dot{\varepsilon}$, and τ_i computed as functions of $\vec{u} + \varepsilon\vec{w}$, in the next step they return to being functions of $\vec{u}$. Their variations ζ, $\dot{e}$ and t_i, however, remain functions of $(\vec{u}, \vec{w})$:

$$
\begin{aligned}
0 &= \frac{\partial}{\partial x_1}\left(\eta(\vec{u})\,h\left(2\frac{\partial}{\partial x_1}(w_1)+\frac{\partial}{\partial x_2}(w_2)\right)\right) \qquad (2.40)\\
&\quad+\frac{\partial}{\partial x_1}\left(\zeta(\vec{u},\vec{w})\,h\left(2\frac{\partial}{\partial x_1}(u_1)+\frac{\partial}{\partial x_2}(u_2)\right)\right)\\
&\quad+\frac{\partial}{\partial x_2}\left(\nu(\vec{u})\,h\left(\frac{\partial}{\partial x_2}(w_1)+\frac{1}{2}\frac{\partial}{\partial x_1}(w_2)\right)\right)\\
&\quad+\frac{\partial}{\partial x_2}\left(\zeta(\vec{u},\vec{w})\,h\left(\frac{\partial}{\partial x_2}(u_1)+\frac{1}{2}\frac{\partial}{\partial x_1}(u_2)\right)\right)\\
0 &= \frac{\partial}{\partial x_2}\left(\eta(\vec{u})\,h\left(2\frac{\partial}{\partial x_2}(w_2)+\frac{\partial}{\partial x_1}(w_1)\right)\right)\\
&\quad+\frac{\partial}{\partial x_2}\left(\zeta(\vec{u},\vec{w})\,h\left(2\frac{\partial}{\partial x_2}(u_1)+\frac{\partial}{\partial x_2}(u_2)\right)\right)\\
&\quad+\frac{\partial}{\partial x_1}\left(\eta(\vec{u})\,h\left(\frac{\partial}{\partial x_2}(w_1)+\frac{1}{2}\frac{\partial}{\partial x_1}(w_2)\right)\right)\\
&\quad+\frac{\partial}{\partial x_1}\left(\zeta(\vec{u},\vec{w})\,h\left(\frac{\partial}{\partial x_2}(u_1)+\frac{1}{2}\frac{\partial}{\partial x_1}(u_2)\right)\right)\\
\overline{\frac{\partial w_1}{\partial x_3}} &= \left(G_1'H-G_1H'\right)/H^2\\
\overline{\frac{\partial w_2}{\partial x_3}} &= \left(G_2'H-G_2H'\right)/H^2\\
G_i &= 2\tau_i A\left(\eta(\vec{u})^{n+1}\,\dot{\varepsilon}(\vec{u})_p^{n+1}-\tau_p^{\frac{n+1}{2}}\right)\\
G_i' &= 2\left[t_i(\vec{u},\vec{w})\right]A\left(\left[\eta^{n+1}(\vec{u})\right]\left[\dot{\varepsilon}_p^{n+1}(\vec{u})\right]-\tau_p^{\frac{n+1}{2}}(\vec{u})\right)\\
&\quad-2\left[\tau_i(\vec{u})\right]A(n+1)\left(\left[\eta^{n}(\vec{u})\right]\left[\zeta(\vec{u},\vec{w})\right]\left[\dot{\varepsilon}_p^{n+1}(\vec{u})\right]\right)\\
&\quad-2\left[\tau_i(\vec{u})\right]A(n+1)\left(\left[\eta^{n+1}(\vec{u})\right]\left[\dot{\varepsilon}_p^{n}(\vec{u})\right]\left[\dot{e}_p(\vec{u},\vec{w})\right]\right)\\
&\quad-2\left[\tau_i(\vec{u})\right]A\left[t_i\right]\\
H &= \left(n\left[\tau_1^2(\vec{u})\right]+n\tau_2^2+\tau_1^2+\tau_2^2\right)\\
H' &= 2n\left[\tau_1(\vec{u})\right]\left[t_1(\vec{u},\vec{w})\right]+2n\tau_2 t_2+2\tau_1 t_1+2\tau_2 t_2
\end{aligned}
$$

By simplifying equations 2.60 we get

$$\begin{aligned} G_i' &= 2t_iA\left(\left[\eta^{n+1}\right]\left[\dot{\varepsilon}_p^{n+1}\right]-\tau_p^{\frac{n+1}{2}}\right) \\ &\quad -2\tau_iA\,(n+1)\,\eta^n\dot{\varepsilon}_p^n\,[\zeta\dot{\varepsilon}_p] \\ &\quad -2\tau_iA\,(n+1)\,\eta^n\dot{\varepsilon}_p^n\,[\eta\dot{e}_p] \\ &\quad -2\tau_iA\,[t_p]\,. \end{aligned}$$

We must apply the same variational process 2.28 to obtain ζ from equation 2.32:

$$\zeta\left(\vec{u},\vec{w}\right) = \frac{d}{d\varepsilon}\frac{A\left(T\right)^{\frac{-1}{n}}}{2\left(\dot{\varepsilon}^2\right)^{\frac{n-1}{2n}}\left(\vec{u}+\varepsilon\vec{w}\right)}\Bigg|_{\varepsilon=0} \tag{2.41}$$

$$= \left(\frac{1-n}{2n}\right)\frac{B}{2}\left[\left(\dot{\varepsilon}^2\right)\left(\vec{u}\right)\right]^{\frac{1-3n}{2n}}\left[\left(\dot{e}^2\right)\left(\vec{u},\vec{w}\right)\right],$$

where $B = A\left(T\right)^{\frac{-1}{n}}$. We note that this implies that

$$\zeta^n = \left(\frac{1-n}{2n}\right)^n\frac{1}{2A}\left[\left(\dot{\varepsilon}^2\right)\left(\vec{u}\right)\right]^{\frac{1-3n}{2}}\left[\left(\dot{e}^2\right)\left(\vec{u},\vec{w}\right)\right] \tag{2.42}$$

Applying the same variational process 2.28 to obtain $\dot{e}^2$ from equation 2.33:

$$\dot{e}^2\left(\vec{u},\vec{w}\right) = \frac{d}{d\varepsilon}\left[2\overline{\dot{\varepsilon}_p^2}+\frac{1}{4}\left(\overline{\frac{\partial u_1}{\partial x_3}}\right)^2+\frac{1}{4}\left(\overline{\frac{\partial u_2}{\partial x_3}}\right)^2\right]\Bigg|_{\varepsilon=0} \tag{2.43}$$

$$= \dot{e}_p^2+\frac{1}{2}\left(\overline{\frac{\partial u_1}{\partial x_3}}\right)\left(\overline{\frac{\partial w_1}{\partial x_3}}\right)+\frac{1}{2}\left(\overline{\frac{\partial u_2}{\partial x_3}}\right)\left(\overline{\frac{\partial w_2}{\partial x_3}}\right),$$

and we obtain $\dot{e}_p^2$ from 2.34:

$$\dot{e}_p^2 = \frac{d}{d\varepsilon}\left[\varepsilon_\nu^2 + \left(\overline{\frac{\partial u_1}{\partial x_1}}\right)^2 + \left(\overline{\frac{\partial u_2}{\partial x_2}}\right)^2 + \frac{1}{4}\left(\overline{\frac{\partial u_1}{\partial x_2}} + \overline{\frac{\partial u_2}{\partial x_1}}\right)^2 + \overline{\frac{\partial u_1}{\partial x_1}\frac{\partial u_2}{\partial x_2}}\right]\Bigg|_{\varepsilon=0} \tag{2.44}$$

Equations 2.45–2.49 show the steps of differentiating equation 2.44.

$$\begin{aligned} \dot{e}_p^2 \quad = \quad & \frac{d}{d\varepsilon}\left(\frac{\partial}{\partial x_1}(u_1+\varepsilon w_1)\right)^2 + \left(\frac{\partial}{\partial x_2}(u_2+\varepsilon w_2)\right)^2\Bigg|_{\varepsilon=0} \\ & + \frac{d}{d\varepsilon}\frac{1}{4}\left(\frac{\partial}{\partial x_1}(u_2+\varepsilon w_2) + \frac{\partial}{\partial x_2}(u_1+\varepsilon w_1)\right)^2\Bigg|_{\varepsilon=0} \\ & + \frac{d}{d\varepsilon}\left(\frac{\partial}{\partial x_1}(u_1+\varepsilon w_1)\right)\left(\frac{\partial}{\partial x_2}(u_2+\varepsilon w_2)\right)\Bigg|_{\varepsilon=0} \end{aligned} \tag{2.45}$$

$$\begin{aligned} \dot{e}_p^2 \quad = \quad & \frac{d}{d\varepsilon}\left(\left[\frac{\partial u_1}{\partial x_1}\right]^2 + 2\frac{\partial u_1}{\partial x_1}\frac{\partial}{\partial x_1}(\varepsilon w_1) + \left[\frac{\partial}{\partial x_1}(\varepsilon w_1)\right]^2\right)\Bigg|_{\varepsilon=0} \\ & + \frac{d}{d\varepsilon}\left(\left[\frac{\partial u_2}{\partial x_2}\right]^2 + 2\frac{\partial u_2}{\partial x_2}\frac{\partial}{\partial x_2}(\varepsilon w_2) + \left[\frac{\partial}{\partial x_2}(\varepsilon w_2)\right]^2\right)\Bigg|_{\varepsilon=0} \\ & + \frac{d}{d\varepsilon}\frac{1}{4}\left(\left[\frac{\partial u_2}{\partial x_1}\right]^2 + 2\frac{\partial u_2}{\partial x_1}\frac{\partial}{\partial x_1}(\varepsilon w_2) + \left[\frac{\partial}{\partial x_1}(\varepsilon w_2)\right]^2\right)\Bigg|_{\varepsilon=0} \\ & + \frac{d}{d\varepsilon}\frac{1}{4}\left(\left[\frac{\partial u_1}{\partial x_2}\right]^2 + 2\frac{\partial u_1}{\partial x_2}\frac{\partial}{\partial x_2}(\varepsilon w_1) + \left[\frac{\partial}{\partial x_2}(\varepsilon w_1)\right]^2\right)\Bigg|_{\varepsilon=0} \\ & + \frac{d}{d\varepsilon}\frac{1}{4}\left(2\left[\frac{\partial u_2}{\partial x_1}\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1}\frac{\partial}{\partial x_2}\varepsilon w_1 + \frac{\partial}{\partial x_1}\varepsilon w_2\frac{\partial u_1}{\partial x_2} + \varepsilon^2\frac{\partial w_2}{\partial x_1}\frac{\partial w_1}{\partial x_2}\right]\right)\Bigg|_{\varepsilon=0} \\ & + \frac{d}{d\varepsilon}\left(\frac{\partial}{\partial x_1}(u_1+\varepsilon w_1)\right)\left(\frac{\partial}{\partial x_2}(u_2+\varepsilon w_2)\right)\Bigg|_{\varepsilon=0} \end{aligned} \tag{2.46}$$

$$
\begin{aligned}
\dot{e}_p^2 \quad = \quad & \left(2\frac{\partial u_1}{\partial x_1}\frac{\partial}{\partial x_1}(w_1) + 2\varepsilon\left[\frac{\partial}{\partial x_1}(w_1)\right]^2\right)\Bigg|_{\varepsilon=0} \\
& + \left(2\frac{\partial u_2}{\partial x_2}\frac{\partial}{\partial x_2}(w_2) + 2\varepsilon\left[\frac{\partial}{\partial x_2}(w_2)\right]^2\right)\Bigg|_{\varepsilon=0} \\
& + \frac{1}{4}\left(2\frac{\partial u_2}{\partial x_1}\frac{\partial}{\partial x_1}(w_2) + 2\varepsilon\left[\frac{\partial}{\partial x_1}(w_2)\right]^2\right)\Bigg|_{\varepsilon=0} \\
& + \frac{1}{4}\left(2\frac{\partial u_1}{\partial x_2}\frac{\partial}{\partial x_2}(w_1) + 2\varepsilon\left[\frac{\partial}{\partial x_2}(w_1)\right]^2\right)\Bigg|_{\varepsilon=0} \\
& + \frac{1}{4}\left(2\left[\frac{\partial}{\partial x_1}u_2\frac{\partial}{\partial x_2}w_1 + \frac{\partial}{\partial x_1}w_2\frac{\partial}{\partial x_2}u_1 + 2\varepsilon\frac{\partial w_2}{\partial x_1}\frac{\partial w_1}{\partial x_2}\right]\right)\Bigg|_{\varepsilon=0} \\
& + \left(\frac{\partial}{\partial x_1}(w_1)\right)\left(\frac{\partial}{\partial x_2}(u_2+\varepsilon w_2)\right) + \left(\frac{\partial}{\partial x_1}(u_1+\varepsilon w_1)\right)\left(\frac{\partial}{\partial x_2}(w_2)\right)\Bigg|_{\varepsilon=0}
\end{aligned}
\tag{2.47}
$$

$$
\begin{aligned}
\dot{e}_p^2 \quad = \quad & \left(2\frac{\partial u_1}{\partial x_1}\frac{\partial}{\partial x_1}(w_1)\right) \\
& + \left(2\frac{\partial u_2}{\partial x_2}\frac{\partial}{\partial x_2}\left(w_{\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right)2}\right)\right) \\
& + \frac{1}{4}\left(2\frac{\partial u_2}{\partial x_1}\frac{\partial}{\partial x_1}(w_2)\right) \\
& + \frac{1}{4}\left(2\frac{\partial u_1}{\partial x_2}\frac{\partial}{\partial x_2}(w_1)\right) \\
& + \frac{1}{4}\left(2\left[\frac{\partial}{\partial x_1}u_2\frac{\partial}{\partial x_2}w_1 + \frac{\partial}{\partial x_1}w_2\frac{\partial}{\partial x_2}u_1\right]\right) \\
& + \left(\frac{\partial}{\partial x_1}(w_1)\right)\left(\frac{\partial}{\partial x_2}(u_2)\right) + \left(\frac{\partial}{\partial x_1}(u_1)\right)\left(\frac{\partial}{\partial x_2}(w_2)\right)
\end{aligned}
\tag{2.48}
$$

$$
\begin{aligned}
\dot{e}_p^2 &= 2\frac{\partial u_1}{\partial x_1}\frac{\partial w_1}{\partial x_1}+2\frac{\partial u_2}{\partial x_2}\frac{\partial w_2}{\partial x_2} \\
&+\frac{1}{2}\left(\frac{\partial u_1}{\partial x_2}\frac{\partial w_1}{\partial x_2}+\frac{\partial u_1}{\partial x_2}\frac{\partial w_2}{\partial x_1}+\frac{\partial u_2}{\partial x_1}\frac{\partial w_1}{\partial x_2}+\frac{\partial u_2}{\partial x_1}\frac{\partial w_2}{\partial x_1}\right)+\frac{\partial u_1}{\partial x_1}\frac{\partial w_2}{\partial x_2}+\frac{\partial u_2}{\partial x_2}\frac{\partial w_1}{\partial x_1} \\
&= \left(2\frac{\partial u_1}{\partial x_1}+\frac{\partial u_2}{\partial x_2}\right)\frac{\partial w_1}{\partial x_1}+\left(2\frac{\partial u_2}{\partial x_2}+\frac{\partial u_1}{\partial x_1}\right)\frac{\partial w_2}{\partial x_2}+ \\
&\quad \frac{1}{2}\left(\frac{\partial u_1}{\partial x_2}+\frac{\partial u_2}{\partial x_1}\right)\frac{\partial w_2}{\partial x_1}+\frac{1}{2}\left(\frac{\partial u_1}{\partial x_2}+\frac{\partial u_2}{\partial x_1}\right)\frac{\partial w_1}{\partial x_2}
\end{aligned} \tag{2.49}
$$

We note that

$$
\begin{aligned}
\dot{e}_p &= \frac{d}{d\varepsilon}\left[\sqrt{\dot{\varepsilon}_p^2}\right]\Big|_{\varepsilon=0} \\
&= \frac{1}{2\sqrt{\dot{\varepsilon}_p^2}}\dot{e}_p = \frac{\dot{e}_p^2}{2\dot{\varepsilon}_p}
\end{aligned} \tag{2.50}
$$

Similarly,

$$
\dot{e} = \frac{\dot{e}^2}{2\dot{\varepsilon}}. \tag{2.51}
$$

For t_p we apply the same variation 2.28 yet again to equation 2.38:

$$
\begin{aligned}
t_p(\vec{u},\vec{w}) &= \frac{d}{d\varepsilon}\left(\tau_2^2(\vec{u}+\varepsilon\vec{w})+\tau_1^2+\eta^2(\vec{u}+\varepsilon\vec{w})\dot{\varepsilon}_p^2(\vec{u}+\varepsilon\vec{w})\right)^{\frac{n+1}{2}}\Big|_{\varepsilon=0} \\
&= \left(\frac{n+1}{2}\right)\left(\tau_2^2(\vec{u})+\tau_1^2+\eta^2(\vec{u})\dot{\varepsilon}_p^2(\vec{u})\right)^{\frac{n-1}{2}} \\
&\quad \cdot\left(2\left[\tau_1(\vec{u})\right]\left[t_1(\vec{u},\vec{w})\right]+2\left[\tau_2(\vec{u})\right]\left[t_2(\vec{u},\vec{w})\right]+T\right)
\end{aligned} \tag{2.52}
$$

where

$$
T = 2\eta(\vec{u})\dot{\varepsilon}_p(\vec{u})\left(\eta(\vec{u})\dot{e}_p(\vec{u},\vec{w})+\zeta(\vec{u},\vec{w})\dot{\varepsilon}_p(\vec{u})\right). \tag{2.53}
$$

2.7 The Structure of the First Variation Form

Since we intend to solve this system of equations by using the first variation as a linear approximation to the system at a specific point, it is necessary to break the equations into linear combinations of $\frac{\partial w_1}{\partial x_1}$, $\frac{\partial w_2}{\partial x_2}$, $\frac{\partial w_2}{\partial x_1}$, and $\frac{\partial w_1}{\partial x_2}$. In this section we investigate the structure of these equations to that end. In order to keep the equations of manageable complexity we define the following substitution variables:

$$\begin{aligned}
\mathcal{A} &= \left(2\frac{\partial u_1}{\partial x_1}+\frac{\partial u_2}{\partial x_2}\right) \\
\mathcal{B} &= \left(2\frac{\partial u_2}{\partial x_2}+\frac{\partial u_1}{\partial x_1}\right) \\
\mathcal{C} &= \frac{1}{2}\left(\frac{\partial u_1}{\partial x_2}+\frac{\partial u_2}{\partial x_1}\right) \\
\mathcal{E} &= \frac{1}{2}\left(\overline{\frac{\partial u_1}{\partial x_3}}\right) \\
\mathcal{F} &= \frac{1}{2}\left(\overline{\frac{\partial u_2}{\partial x_3}}\right) \\
\mathcal{N} &= \eta\left(\vec{u}\right)h \\
\mathcal{R} &= 2A\tau_i\left(n+1\right)\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)^{\frac{n-1}{2}} \\
\mathcal{S} &= \left[2A\tau_i\left(n+1\right)\right]\left[\eta^n\dot{\varepsilon}_p^n+\eta\dot{\varepsilon}_p\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)^{\frac{n-1}{2}}\right] \\
\mathcal{T} &= 2A\left(\left[\eta^{n+1}\left(\vec{u}\right)\right]\left[\dot{\varepsilon}_p^{n+1}\left(\vec{u}\right)\right]-\tau_p^{\frac{n+1}{2}}\left(\vec{u}\right)\right) \\
\mathcal{X} &= \xi\left(\vec{u}\right)h
\end{aligned} \tag{2.54}$$

This allows us to substitute quantities from 2.54 into equation 2.49.

$$\dot{e}_p^2=\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}. \tag{2.55}$$

and for $\dot{e}^2$ we can substitute into equation 2.33:

$$\dot{e}^2=\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}. \tag{2.56}$$

Then, letting

$$\xi(\vec{u}) = \left(\frac{1-n}{2n}\right)\frac{b}{2}\left[\dot{\varepsilon}^2(\vec{u})\right]^{\frac{1-n}{2n}-1} \tag{2.57}$$

we will use the substitution

$$\zeta(\vec{u},\vec{w}) = \xi(\vec{u})\,\dot{e}^2(\vec{u},\vec{w}). \tag{2.58}$$

And, with the substitutions 2.54–2.58, equation 2.53 becomes becomes

$$\begin{aligned} T &= 2\eta^2(\vec{u})\,\dot{\varepsilon}_p(\vec{u})\left(\mathcal{A}\frac{\partial w_1}{\partial x_1} + \mathcal{B}\frac{\partial w_2}{\partial x_2} + \mathcal{C}\frac{\partial w_2}{\partial x_1} + \mathcal{C}\frac{\partial w_1}{\partial x_2}\right) \\ &\quad + 2\eta(\vec{u})\,\dot{\varepsilon}^2{}_p(\vec{u})\xi(\vec{u})\left(\mathcal{A}\frac{\partial w_1}{\partial x_1} + \mathcal{B}\frac{\partial w_2}{\partial x_2} + \mathcal{C}\frac{\partial w_2}{\partial x_1} + \mathcal{C}\frac{\partial w_1}{\partial x_2} + \mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}} + \mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right). \end{aligned} \tag{2.59}$$

Rewriting the main equations 2.39 we get:

$$\begin{aligned}
0 &= \frac{\partial}{\partial x_1}\left(\mathcal{N}\left(2\frac{\partial}{\partial x_1}(w_1)+\frac{\partial}{\partial x_2}(w_2)\right)\right) \\
&+\frac{\partial}{\partial x_1}\left(\mathcal{X}\mathcal{A}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right) \\
&+\frac{\partial}{\partial x_2}\left(\mathcal{N}\left(\frac{\partial}{\partial x_2}(w_1)+\frac{1}{2}\frac{\partial}{\partial x_1}(w_2)\right)\right) \\
&+\frac{\partial}{\partial x_2}\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right) \\
0 &= \frac{\partial}{\partial x_2}\left(\mathcal{N}\left(2\frac{\partial}{\partial x_2}(w_2)+\frac{\partial}{\partial x_1}(w_1)\right)\right) \\
&+\frac{\partial}{\partial x_2}\left(\mathcal{X}\mathcal{B}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right) \\
&+\frac{\partial}{\partial x_1}\left(\mathcal{N}\left(\frac{\partial}{\partial x_2}(w_1)+\frac{1}{2}\frac{\partial}{\partial x_1}(w_2)\right)\right) \\
&+\frac{\partial}{\partial x_1}\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right) \\
\overline{\frac{\partial w_1}{\partial x_3}} &= \left(G_1'H-G_1H'\right)/H^2 \\
\overline{\frac{\partial w_2}{\partial x_3}} &= \left(G_2'H-G_2H'\right)/H^2 \\
G_i &= 2\tau_i A\left(\eta(\vec{u})^{n+1}\dot{\varepsilon}(\vec{u})_p^{n+1}-\left(\tau_2^2(\vec{u})+\tau_1^2(\vec{u})+\left[\eta^2(\vec{u})\right]\left[\dot{\varepsilon}_p^2(\vec{u})\right]\right)^{\frac{n+1}{2}}\right) \\
G_i' &= \mathcal{T}t_i \\
&-\mathcal{S}\left[\xi\dot{\varepsilon}_p\right]\dot{e}^2 \\
&-\mathcal{S}\left[\frac{\eta}{2\dot{\varepsilon_p}^2}\right]\dot{e}_p^2 \\
&-\mathcal{R}\left[\tau_1 t_1+\tau_2 t_2\right] \\
H &= \left(n\left[\tau_1^2(\vec{u})\right]+n\tau_2^2+\tau_1^2+\tau_2^2\right) \\
H' &= 2n\left[\tau_1(\vec{u})\right]\left[t_1(\vec{u},\vec{w})\right]+2n\tau_2 t_2+2\tau_1 t_1+2\tau_2 t_2.
\end{aligned} \tag{2.60}$$

Furthermore, G_i' of equations 2.60 can be simplified to:

$$\begin{aligned}
G_i' &= \mathcal{T}\left[t_i\left(\vec{u},\vec{w}\right)\right] \\
&\quad -\mathcal{R}\tau_1 t_1 \\
&\quad -\mathcal{R}\tau_2 t_2 \\
&\quad -\mathcal{S}\left[\frac{\eta}{2\dot{\varepsilon_p}^2}+\xi\dot{\varepsilon}_p\right]\left(\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}\right) \\
&\quad -\mathcal{S}\left[\xi\dot{\varepsilon}_p\right]\left(\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right)
\end{aligned} \tag{2.61}$$

It should be noted that when $n=3$, equations 2.60 become much simpler:

$$\begin{aligned}
G_i &= A\tau_1\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right) \\
H &= 2 \\
G_i' &= A\cdot\left(t_i\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right)\right) \\
&\quad +2A\tau_i\left(\left[\tau_1\left(\vec{u}\right)\right]\left[t_1\left(\vec{u},\vec{w}\right)\right]+\left[\tau_2\left(\vec{u}\right)\right]\left[t_2\left(\vec{u},\vec{w}\right)\right]\right) \\
&\quad +2A\tau_i 2\eta\xi\varepsilon_p^2 e^2 \\
&\quad +2A\tau_i\eta^2 e_p^2 \\
G_i' &= A\cdot\left(t_i\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right)\right) \\
&\quad +2A\tau_i\left(\left[\tau_1\left(\vec{u}\right)\right]\left[t_1\left(\vec{u},\vec{w}\right)\right]+\left[\tau_2\left(\vec{u}\right)\right]\left[t_2\left(\vec{u},\vec{w}\right)\right]\right) \\
&\quad +2A\tau_i\eta\left[2\xi\varepsilon_p^2+\eta\right]\left(\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}\right) \\
&\quad 2A\tau_i\eta\left[2\xi\varepsilon_p^2\right]\left(\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right) \\
H' &= 0
\end{aligned} \tag{2.62}$$

So when $n=3$, we can simplify some of the substitutions 2.54:

$$\begin{aligned}
\mathcal{R} &= 8A\tau_i\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right) \\
\mathcal{S} &= [8A\tau_i]\left[\eta^3\dot{\varepsilon}_p^3+\eta\dot{\varepsilon}_p\left(\tau_2^2+\tau_1^2+\eta^2\dot{\varepsilon}_p^2\right)\right] \\
\mathcal{T} &= 2A\left(\left[\eta^4\left(\vec{u}\right)\right]\left[\dot{\varepsilon}_p^4\left(\vec{u}\right)\right]-\tau_p\left(\vec{u}\right)\right)
\end{aligned} \tag{2.63}$$

For clarity, we can apply the common sliding law to equations 2.60:

$$\begin{aligned}
\overline{\frac{\partial w_1}{\partial x_3}} &= \left(G_1'H-G_1H'\right)/H^2 \\
\overline{\frac{\partial w_2}{\partial x_3}} &= \left(G_2'H-G_2H'\right)/H^2 \\
G_i &= 2\beta^2 u_i A\left(\eta^{n+1}\dot{\varepsilon}_p^{n+1}-\left(\beta^4u_1^2+\beta^4u_2^2+\left[\eta^2\right]\left[\dot{\varepsilon}_p^2\right]\right)^{\frac{n+1}{2}}\right) \\
G_i' &= \mathcal{T}\left[\beta^2 w_{i\text{basal}}\right] \\
&\quad -\mathcal{R}\tau_1\left[\beta^2 w_{1\text{basal}}\right] \\
&\quad -\mathcal{R}\tau_2\left[\beta^2 w_{2\text{basal}}\right] \\
&\quad -\mathcal{S}\left[\frac{\eta}{2\dot{\varepsilon_p}^2}+\xi\dot{\varepsilon}_p\right]\left(\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}\right) \\
&\quad -\mathcal{S}\left[\xi\dot{\varepsilon}_p\right]\left(\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right) \\
H &= \beta^2\left(nu_1^2+nu_2^2+u_1^2+u_2^2\right) \\
H' &= 2\left(\left(n+1\right)u_1w_1+\left(n+1\right)u_2w_2\right) \\
\mathcal{S}_i &= 2A\left(n+1\right)u_i \\
\tau_p &= u_1^2+u_2^2+\eta^2(\vec{u})\dot{\varepsilon}_p^2(\vec{u})
\end{aligned} \tag{2.64}$$

These are the equations solved by the implementation in Picard iteration mode.

2.8 Finding the Weak Form for use in the Finite Element Method

In order to prepare system 2.60 for solution using the Finite Element Method we put the equations in variational form using Galerkin's Method, the first step of which is to multiply by test functions ϕ_i and integrate.

$$
\begin{aligned}
0 \;=\;& \iint_{\blacksquare}\left[\phi_1\frac{\partial}{\partial x_1}\left(\mathcal{N}\left(2\frac{\partial w_1}{\partial x_1}+\frac{\partial w_2}{\partial x_2}\right)\right)\right]d\Omega \qquad (2.65)\\
&+\iint_{\Omega}\phi_1\frac{\partial}{\partial x_1}\left(\mathcal{X}\mathcal{A}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
&+\iint_{\Omega}\phi_1\frac{\partial}{\partial x_2}\left(\mathcal{N}\left(\frac{\partial w_1}{\partial x_2}+\frac{1}{2}\frac{\partial w_2}{\partial x_1}\right)\right)d\Omega\\
&+\iint_{\Omega}\phi_1\frac{\partial}{\partial x_2}\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
0 \;=\;& \iint_{\Omega}\phi_2\frac{\partial}{\partial x_2}\left(\mathcal{N}\left(2\frac{\partial}{\partial x_2}(w_2)+\frac{\partial}{\partial x_1}(w_1)\right)\right)d\Omega\\
&+\iint_{\Omega}\phi_2\frac{\partial}{\partial x_2}\left(\mathcal{X}\mathcal{B}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
&+\iint_{\Omega}\phi_2\frac{\partial}{\partial x_1}\left(\mathcal{N}\left(\frac{\partial}{\partial x_2}(w_1)+\frac{1}{2}\frac{\partial}{\partial x_1}(w_2)\right)\right)d\Omega\\
&+\iint_{\Omega}\phi_2\frac{\partial}{\partial x_1}\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
0 \;=\;& \iint_{\Omega}\left[\phi_3\left(\frac{G_1'}{H}\right)-\overline{\frac{\partial w_1}{\partial x_3}}-\frac{G_1H'}{H^2}\right]d\Omega\\
0 \;=\;& \iint_{\Omega}\left[\phi_4\left(\frac{G_2'}{H}\right)-\overline{\frac{\partial w_2}{\partial x_3}}-\frac{G_2H'}{H^2}\right]d\Omega
\end{aligned}
$$

We then apply Green's first identity to equations 2.65 in order to eliminate second derivatives:

$$
\begin{aligned}
0 \;=\;& \iint_\Omega \left[\frac{\partial\phi_1}{\partial x_1}\left(\mathcal{N}\left(2\frac{\partial w_1}{\partial x_1}+\frac{\partial w_2}{\partial x_2}\right)\right)\right]d\Omega \qquad (2.66)\\
&+\iint_\Omega \frac{\partial\phi_1}{\partial x_1}\left(\mathcal{X}\mathcal{A}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
&+\iint_\Omega \frac{\partial\phi_1}{\partial x_2}\left(\mathcal{N}\left(\frac{\partial w_1}{\partial x_2}+\frac{1}{2}\frac{\partial w_2}{\partial x_1}\right)\right)d\Omega\\
&+\iint_\Omega \frac{\partial\phi_1}{\partial x_2}\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
&-\oint_{\partial\Omega}\left[\phi_1\left(\mathcal{N}\left(2\frac{\partial w_1}{\partial x_1}+\frac{\partial w_2}{\partial x_2}\right)\right)\mathbf{n}_{x_1}\right]d\Gamma\\
&-\oint_{\partial\Omega}\phi_1\left(\mathcal{X}\mathcal{A}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)\mathbf{n}_{x_1}d\Gamma\\
&-\oint_{\partial\Omega}\phi_1\left(\mathcal{N}\left(\frac{1}{2}\frac{\partial w_1}{\partial x_2}+\frac{\partial w_2}{\partial x_1}\right)\right)\mathbf{n}_{x_2}d\Gamma\\
&-\oint_{\partial\Omega}\phi_1\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)\mathbf{n}_{x_2}d\Gamma\\
0 \;=\;& \iint_\Omega \frac{\partial\phi_2}{\partial x_2}\left(\mathcal{N}\left(2\frac{\partial w_2}{\partial x_2}+\frac{\partial w_1}{\partial x_1}\right)\right)d\Omega\\
&+\iint_\Omega \frac{\partial\phi_2}{\partial x_2}\left(\mathcal{X}\mathcal{B}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
&+\iint_\Omega \frac{\partial\phi_2}{\partial x_1}\left(\mathcal{N}\left(\frac{\partial w_1}{\partial x_2}+\frac{1}{2}\frac{\partial w_2}{\partial x_1}\right)\right)d\Omega\\
&+\iint_\Omega \frac{\partial\phi_2}{\partial x_1}\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)d\Omega\\
&-\oint_{\partial\Omega}\phi_2\left(\mathcal{N}\left(2\frac{\partial w_2}{\partial x_2}+\frac{\partial w_1}{\partial x_1}\right)\right)\mathbf{n}_{x_1}d\Gamma\\
&-\oint_{\partial\Omega}\phi_2\left(\mathcal{X}\mathcal{B}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)\mathbf{n}_{x_1}d\Gamma\\
&-\oint_{\partial\Omega}\phi_2\left(\mathcal{N}\left(\frac{\partial w_1}{\partial x_2}+\frac{1}{2}\frac{\partial w_2}{\partial x_1}\right)\right)\mathbf{n}_{x_2}d\Gamma\\
&-\oint_{\partial\Omega}\phi_2\left(\mathcal{X}\mathcal{C}\left[\mathcal{A}\frac{\partial w_1}{\partial x_1}+\mathcal{B}\frac{\partial w_2}{\partial x_2}+\mathcal{C}\frac{\partial w_2}{\partial x_1}+\mathcal{C}\frac{\partial w_1}{\partial x_2}+\mathcal{E}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{F}\overline{\frac{\partial w_2}{\partial x_3}}\right]\right)\mathbf{n}_{x_2}d\Gamma
\end{aligned}
$$

It should be noted that G_i, H, H', $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$, η, $\dot{\varepsilon}_p$, ξ and τ_i in system 2.66 are functions only of $\vec{u}$, and therefore are constant for the first variation at a single point. It is important for solution to be clear about the factors of the test and trial (w) functions. One more factorization gives us the weak form of the first variation of the system 2.27. This is an affine system of the form

$$0 = J - \vec{c} \tag{2.67}$$

where

$$\begin{aligned}
J_{1,1} &= [2\mathcal{N} + \mathcal{X}\mathcal{A}\mathcal{A}]\frac{\partial \phi_1}{\partial x_1}\frac{\partial w_1}{\partial x_1} + \mathcal{X}\mathcal{A}\mathcal{C}\frac{\partial \phi_1}{\partial x_1}\frac{\partial w_1}{\partial x_2} \\
&\quad + \mathcal{X}\mathcal{A}\mathcal{C}\frac{\partial \phi_1}{\partial x_2}\frac{\partial w_1}{\partial x_1} + [\mathcal{N} + \mathcal{X}\mathcal{C}\mathcal{C}]\frac{\partial \phi_1}{\partial x_2}\frac{\partial w_1}{\partial x_2} \\
J_{1,2} &= \mathcal{X}\mathcal{A}\mathcal{C}\frac{\partial \phi_1}{\partial x_1}\frac{\partial w_2}{\partial x_1} + [\mathcal{N} + \mathcal{X}\mathcal{A}\mathcal{B}]\frac{\partial \phi_1}{\partial x_1}\frac{\partial w_2}{\partial x_2} \\
&\quad + \left[\frac{1}{2}\mathcal{N} + \mathcal{X}\mathcal{C}\mathcal{C}\right]\frac{\partial \phi_1}{\partial x_2}\frac{\partial w_2}{\partial x_1} + \mathcal{X}\mathcal{A}\mathcal{C}\frac{\partial \phi_1}{\partial x_2}\frac{\partial w_2}{\partial x_2} \\
J_{1,3} &= \mathcal{X}\mathcal{A}\mathcal{E}\frac{\partial \phi_1}{\partial x_1}\overline{\frac{\partial w_1}{\partial x_3}} + \mathcal{X}\mathcal{C}\mathcal{E}\frac{\partial \phi_1}{\partial x_2}\overline{\frac{\partial w_1}{\partial x_3}} \\
J_{1,4} &= \mathcal{X}\mathcal{A}\mathcal{F}\frac{\partial \phi_1}{\partial x_1}\overline{\frac{\partial w_2}{\partial x_3}} + \mathcal{X}\mathcal{C}\mathcal{F}\frac{\partial \phi_1}{\partial x_2}\overline{\frac{\partial w_2}{\partial x_3}}
\end{aligned}$$

$$
\begin{aligned}
J_{2,1} &= \mathcal{X}\mathcal{C}\mathcal{A}\frac{\partial\phi_2}{\partial x_1}\frac{\partial w_1}{\partial x_1} + \left[\frac{1}{2}\mathcal{N} + \mathcal{X}\mathcal{C}\mathcal{C}\right]\frac{\partial\phi_2}{\partial x_1}\frac{\partial w_1}{\partial x_2} \\
&\quad + [\mathcal{N} + \mathcal{X}\mathcal{B}\mathcal{A}]\frac{\partial\phi_2}{\partial x_2}\frac{\partial w_1}{\partial x_1} + \mathcal{X}\mathcal{B}\mathcal{C}\frac{\partial\phi_1}{\partial x_2}\frac{\partial w_1}{\partial x_2} \\
J_{2,2} &= [\mathcal{N} + \mathcal{X}\mathcal{C}\mathcal{C}]\frac{\partial\phi_1}{\partial x_1}\frac{\partial w_2}{\partial x_1} + \mathcal{X}\mathcal{C}\mathcal{B}\frac{\partial\phi_1}{\partial x_1}\frac{\partial w_2}{\partial x_2} \\
&\quad + \mathcal{X}\mathcal{C}\mathcal{B}\frac{\partial\phi_1}{\partial x_2}\frac{\partial w_2}{\partial x_1} + [2\mathcal{N} + \mathcal{X}\mathcal{B}\mathcal{B}]\frac{\partial\phi_1}{\partial x_2}\frac{\partial w_2}{\partial x_2} \\
J_{2,3} &= \mathcal{X}\mathcal{B}\mathcal{E}\frac{\partial\phi_2}{\partial x_1}\overline{\frac{\partial w_1}{\partial x_3}} + \mathcal{X}\mathcal{B}\mathcal{F}\frac{\partial\phi_2}{\partial x_2}\overline{\frac{\partial w_1}{\partial x_3}} \\
J_{2,4} &= \mathcal{X}\mathcal{C}\mathcal{E}\frac{\partial\phi_2}{\partial x_1}\overline{\frac{\partial w_2}{\partial x_3}} + \mathcal{X}\mathcal{C}\mathcal{F}\frac{\partial\phi_2}{\partial x_2}\overline{\frac{\partial w_2}{\partial x_3}} \\
J_{3,1} &= \left[\mathcal{T}\beta^2 - \mathcal{R}\tau_1\beta^2\right]\phi_3 w_{1\text{basal}} - \mathcal{S}\left[\frac{\eta}{2\dot{\varepsilon_p}^2} + \xi\dot{\varepsilon}_p\right]\left(\mathcal{A}\phi_3\frac{\partial w_1}{\partial x_1} + \mathcal{C}\phi_3\frac{\partial w_1}{\partial x_2}\right) \\
J_{3,2} &= \left[\mathcal{R}\tau_2\beta^2\right]\phi_3 w_{2\text{basal}} - \mathcal{S}\left[\frac{\eta}{2\dot{\varepsilon_p}^2} + \xi\dot{\varepsilon}_p\right]\left(\mathcal{B}\phi_3\frac{\partial w_2}{\partial x_2} + \mathcal{C}\phi_3\frac{\partial w_2}{\partial x_1}\right) \\
J_{3,3} &= -\mathcal{S}\left[\xi\dot{\varepsilon}_p\right]\left[\mathcal{E}\phi_3\overline{\frac{\partial w_1}{\partial x_3}}\right] \\
J_{3,4} &= -\mathcal{S}\left[\xi\dot{\varepsilon}_p\right]\left[\mathcal{F}\phi_3\overline{\frac{\partial w_2}{\partial x_3}}\right] \\
J_{4,1} &= \left[\mathcal{R}\tau_1\beta^2\right]\phi_4 w_{1\text{basal}} - \mathcal{S}\left[\frac{\eta}{2\dot{\varepsilon_p}^2} + \xi\dot{\varepsilon}_p\right]\left(\mathcal{B}\phi_4\frac{\partial w_2}{\partial x_2} + \mathcal{C}\phi_4\frac{\partial w_2}{\partial x_1}\right) \\
J_{4,2} &= \left[\mathcal{T}\beta^2 - \mathcal{R}\tau_2\beta^2\right]\phi_4 w_{2\text{basal}} - \mathcal{S}\left[\frac{\eta}{2\dot{\varepsilon_p}^2} + \xi\dot{\varepsilon}_p\right]\left(\mathcal{A}\phi_4\frac{\partial w_1}{\partial x_1} + \mathcal{C}\phi_4\frac{\partial w_1}{\partial x_2}\right) \\
J_{4,3} &= -\mathcal{S}\left[\xi\dot{\varepsilon}_p\right]\left[\mathcal{E}\phi_4\overline{\frac{\partial w_1}{\partial x_3}}\right] \\
J_{4,4} &= -\mathcal{S}\left[\xi\dot{\varepsilon}_p\right]\left[\mathcal{F}\phi_4\overline{\frac{\partial w_2}{\partial x_3}}\right]
\end{aligned}
$$

and $c =$

$$\begin{bmatrix} \begin{pmatrix} \left([2\mathcal{N}+\mathcal{XAA}]\frac{\partial w_1}{\partial x_1}+[\mathcal{N}+\mathcal{XAB}]\frac{\partial w_2}{\partial x_2}+\mathcal{XAC}\left[\frac{\partial w_1}{\partial x_2}+\frac{\partial w_2}{\partial x_1}\right]\right)\phi_1\mathbf{n}_{x_1}\ddots \\ +\left([\mathcal{N}+\mathcal{XCC}]\frac{\partial w_1}{\partial x_2}+\left[\frac{1}{2}\mathcal{N}+\mathcal{XCC}\right]\frac{\partial w_2}{\partial x_1}+\mathcal{XCA}\frac{\partial w_1}{\partial x_1}+\mathcal{XCB}\frac{\partial w_2}{\partial x_2}\right)\phi_1\mathbf{n}_{x_2}\ddots \\ +\left(\mathcal{XAE}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{XCE}\overline{\frac{\partial w_1}{\partial x_3}}\right)\phi_1\mathbf{n}_{x_1}+\left(\mathcal{XAF}\overline{\frac{\partial w_2}{\partial x_3}}+\mathcal{XCF}\overline{\frac{\partial w_2}{\partial x_3}}\right)\phi_1\mathbf{n}_{x_2} \end{pmatrix} \\ \begin{pmatrix} \left([2\mathcal{N}+\mathcal{XBB}]\frac{\partial w_2}{\partial x_2}+[\mathcal{N}+\mathcal{XBA}]\frac{\partial w_1}{\partial x_1}+\mathcal{XBC}\left[\frac{\partial w_1}{\partial x_2}+\frac{\partial w_2}{\partial x_1}\right]\right)\phi_2\mathbf{n}_{x_1}+\cdots \\ +\left([\mathcal{N}+\mathcal{XCC}]\frac{\partial w_2}{\partial x_1}+\left[\frac{1}{2}\mathcal{N}+\mathcal{XCC}\right]\frac{\partial w_1}{\partial x_2}+\mathcal{XCA}\frac{\partial w_1}{\partial x_1}+\mathcal{XCB}\frac{\partial w_2}{\partial x_2}\right)\phi_1\mathbf{n}_{x_2}\ddots \\ +\cdots\left(\mathcal{XAE}\overline{\frac{\partial w_1}{\partial x_3}}+\mathcal{XCE}\overline{\frac{\partial w_1}{\partial x_3}}\right)\phi_2\mathbf{n}_{x_1}+\left(\mathcal{XBF}\overline{\frac{\partial w_2}{\partial x_3}}+\mathcal{XBF}\overline{\frac{\partial w_2}{\partial x_3}}\right)\phi_2\mathbf{n}_{x_2} \end{pmatrix} \\ -\frac{G_1H'}{H^2} \\ -\frac{G_2H'}{H^2} \end{bmatrix}.$$

We may also use a different sliding law such as the one in Pollard and Deconto [2009], however this requires the substitution of the much more complex:

$$\begin{aligned} \phi_3 w_1 \;&\leftarrow\; \left[\phi_3 w_1\left(\left(\frac{1-m}{2m}\right)\left|u_1^2+u_2^2\right|^{\frac{1-3m}{2m}}2u_1^2\left|u_1^2+u_2^2\right|^{\frac{1-m}{2m}}\right)\right] \\ &\quad+\left[\phi_3 w_2\left(\frac{1-m}{2m}\right)\left|u_1^2+u_2^2\right|^{\frac{1-3m}{2m}}2u_2u_1\right] \\ \phi_4 w_2 \;&\leftarrow\; \left[\phi_4 w_1\left(\frac{1-m}{2m}\right)\left|u_1^2+u_2^2\right|^{\frac{1-3m}{2m}}2u_2u_1\right] \\ &\quad+\left[\phi_4 w_2\left(\left(\frac{1-m}{2m}\right)\left|u_1^2+u_2^2\right|^{\frac{1-3m}{2m}}2u_2^2+\left|u_1^2+u_2^2\right|^{\frac{1-m}{2m}}\right)\right] \end{aligned} \tag{2.68}$$

for the weak form. As desired, J is in fact the linear approximation of the Jacobian of the planar form of the Stokes equations with approximations applied at a point.

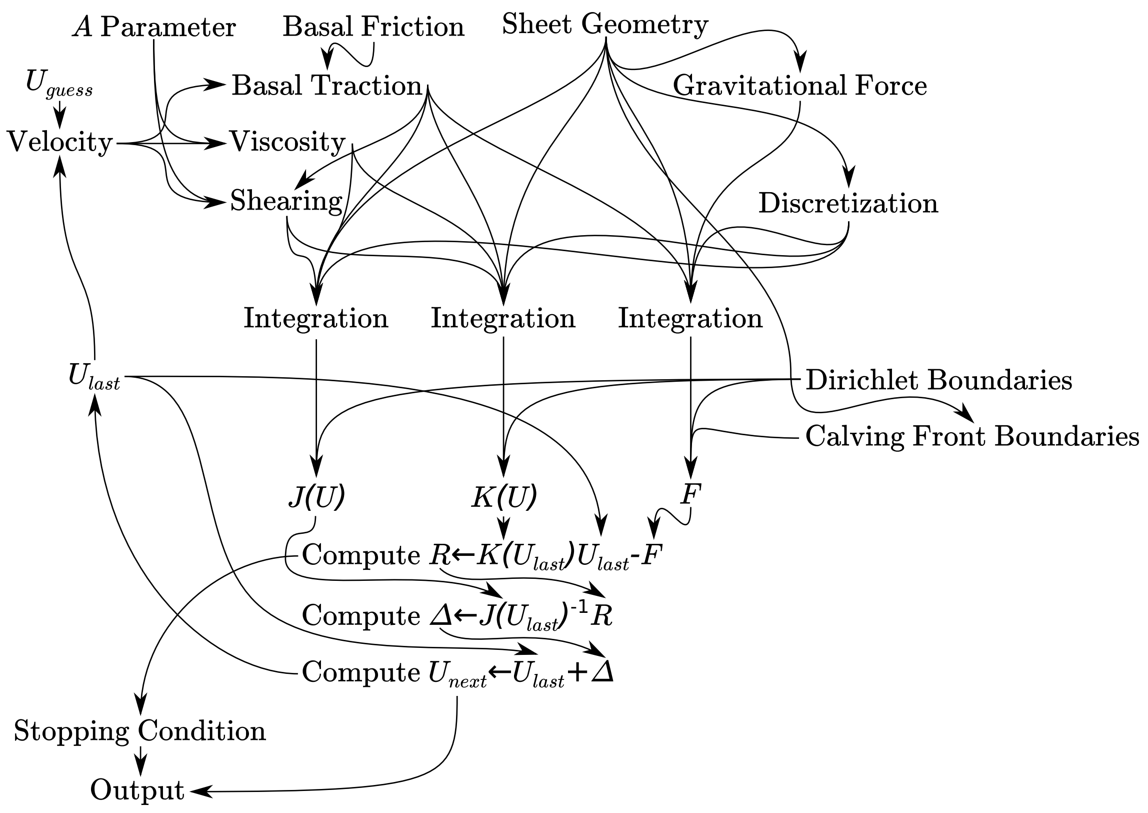


Figure 2.1   General Solution Process

2.9 Approximating and Solving

In this chapter, we presented a derivation of the field equations, which, assuming $n = 3$ and the common linear sliding law, take the form

$$\begin{aligned}
&\frac{\partial}{\partial x_1}\left(\eta h\left(2\frac{\partial u_1}{\partial x_1}+\frac{\partial u_2}{\partial x_2}\right)\right) && \\
&+\frac{\partial}{\partial x_2}\left(\eta h\left(\frac{\partial u_1}{\partial x_2}+\frac{1}{2}\frac{\partial u_2}{\partial x_1}\right)\right) &=& \rho g h\frac{\partial s}{\partial x_1}+\beta^2 u_{1\text{basal}} \\
&\frac{\partial}{\partial x_2}\left(\eta h\left(2\frac{\partial u_2}{\partial x_2}+\frac{\partial u_1}{\partial x_1}\right)\right) && \\
&+\frac{\partial}{\partial x_1}\left(\eta h\left(\frac{\partial u_2}{\partial x_1}+\frac{1}{2}\frac{\partial u_1}{\partial x_2}\right)\right) &=& \rho g h\frac{\partial s}{\partial x_2}+\beta^2 u_{1\text{basal}} \\
&\frac{A\tau_1\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right)}{2} &=& \overline{\frac{\partial u_1}{\partial x_3}} \\
&\frac{A\tau_2\left(\tau_1^2+\tau_2^2+2\eta^2\varepsilon_p^2\right)}{2} &=& \overline{\frac{\partial u_2}{\partial x_3}}.
\end{aligned} \tag{2.69}$$

To these equations we apply the Finite Element Method by multiplying by a test function, replacing the solution variables with a trial function, and integrating over the domain. Optionally, we can follow the procedure in sections 2.7 and 2.8 to compute the first variation for use with Newton's Method.

In order to use the variational form we obtain by applying the Finite Element Method to solve the field equations, we must first divide the planar domain into non-overlapping convex polygonal elements, such as triangles.

Then, we define the test functions θ_{ik} and trial functions ψ_{jl} to be piece-wise defined polynomials, such that they form a basis for the finite solution vector U and finite forcing vector F. These two vectors contain an element for each solution variable at each vertex of each element. The test and trial basis functions must be continuous on the interior of every element. This allows yields an approximation of the continuous functions u_i by a piecewise-continuous function $U\theta$, where θ is the basis matrix.

The remainder of the solution process is performed numerically. An overview of the numerical solution process is shown in Figure 2.1. However, the implementation may be much more complicated than this summary: FEM and boundary condition code requires many supporting numerical computations.

Third, we must compute the matrix K and forcing vector F such that each K_{ijkl} is the definite integral of the field equations over the product $\theta_{ik}\psi_{jl}$ for some specific solution variable pair (i, j) and vertex pair (k,l).

Fourth, we search using some method to find the U which minimizes the residual R:

$$R = K(U)U - F,$$

which can be achieved by several methods including Picard iteration and Newton's Method. Newton's method further requires the Jacobian of $K(U)$ with respect to U. The full integration-ready form of J is defined in equation 2.67. The procedure for solving $K(U)U = F$ using Newton's Method is:

- compute the residual $R \leftarrow K(U_{\text{guess}})U_{\text{guess}} - F$,
- compute the Newton's update $\Delta \leftarrow J(U)^{-1}R$,
- apply the update to the guess $U_{\text{guess}} \leftarrow U_{\text{guess}+\Delta}$,
- repeat steps 1-3 until R is sufficiently small.

Finally, we must modify K and F to apply boundary conditions. Possible boundary conditions include Dirichlet boundary conditions, Calving Front boundary conditions and periodic boundary conditions.

Dirichlet boundary conditions force the solution of a particular point or region of the ice moving at a specific velocity. This is useful, for example, when the velocity of some parts of the ice sheet is known in advance. Dirichlet boundary conditions can be applied in two ways: overwriting the row equation of $KU = F$ with the equation $1 \cdot u = V$ for a numerical value V, or by adding the equation $\lambda 1 \cdot u = \lambda V$. The second method uses the Lagrange multiplier λ, a sufficiently large predefined value.

Calving Front boundary conditions are applied by adding force to F to account for the hydrostatic pressure of the seawater and ice front interface. This force is [MacAyeal, 1997]

$$\vec{n}\frac{\rho g h^2}{2}\left(1 - \frac{\rho}{\rho_w}\right). \tag{2.70}$$

$\vec{n}$ is the normal vector to the ice sheet boundary. This force must be integrated over the adjacent calving front edges of elements on the boundary before being added to F in this discretized finite element formulation.

Periodic boundary conditions are not applied explicitly to K or F. In IceCamp, the implementation presented in this thesis, they are not treated as boundary conditions at all. In a geometry with a periodic domain, there are simply elements which span the periodic "boundary", maintaining a pure torroidal topology. The final form of $K(U)$ contains values which come from integrating over non-zero test and trial functions as they wrap around the periodic "boundary."

CHAPTER 3 Software Implementation

3.1 Overview

This chapter describes the software implementation, IceCamp, of the model presented in the previous chapter. First, an overview of the organization of the code is given. Then each variable used in the implementation is listed, along with a description of what that variable contains. Third, each routine implemented is described in detail including that routines responsibilities and the calculations and algorithms employed by that routine. The routines are listed in depth-first call order.

Figure 3.1 gives a visual depiction of the organization of the subroutines in the software implementation. It also gives an overview of each step those subroutines make and what their responsibilities are.

Processing begins in the software implementation with the test driver. This driver is responsible for setting up a test scenario. Specifically, it must set up the domain size and resolution. Then a triangular mesh is generated by `rect_grid_to_triangles`. `rect_grid_to_triangles` calls the second phase of the test driver which provides per-point data such as thickness, traction, boundary conditions, and Glen's flow law A parameter.

`fem_imr` is then invoked. It begins iteratively refining meshes and calling `fem_l1l2` to solve the momentum balance problem on those meshes. `fem_imr` relies on an external library, `mrgrnk.f90` to perform sort operations.

`fem_l1l2` solves the momentum balance problem for a given input mesh and input data. It does this by first invoking `analyze_mesh` to analyze the mesh geometry and `init_k_sparsity` to build the sparsity pattern of the matrices. `analyze_mesh` depends on a helper function, `maybe_add_neighbor` to maintain sorted, unique, lists.

`fem_l1l2` also depends on Gaussian quadrature integration functions, `duv_dz`, and an external linear solver library. `duv_dz` is a function automatically generated by the Maple computer algebra system to calculate quantities that `fem_1112` depends on.

Finally, a python script, `plot.py` is provided to plot the output of the FORTRAN code.

3.2 Variables

`accel_grav` the acceleration due to gravity, g.

`a_factor` the $\mathcal{A}$ factor used in the expansion of the equations to be solved as in equations 2.54.

`all_p` array of every index into the `p` array.

`approx_ice_visc` initial guess for the viscosity of ice $\approx 2\eta$.

`approx_vel_sqr` initial guess for the velocity squared ($|\vec{u}|^2$).

`Area` area of the current element `e`.

`B` Glen's flow law B parameter used by used by `duv_dz`.

`b` array of points on the boundary in the `eismint_square_bay2` test by index into `p`.

`basal_traction_coeff` basal traction amount for points in `p` with the same index. Units and exact meaning depends on sliding law being applied.

`basal_traction_imr` basal traction amount for points in **`p_imr`** with the same index. Units and exact meaning depends on sliding law being applied.

`basal_sliding_epsilon` ε_b, the extra velocity amount to be added to prevent the friction with the bed from becoming infinite.

`basal_sliding_exp` exponent of Pollard's basal sliding formula.

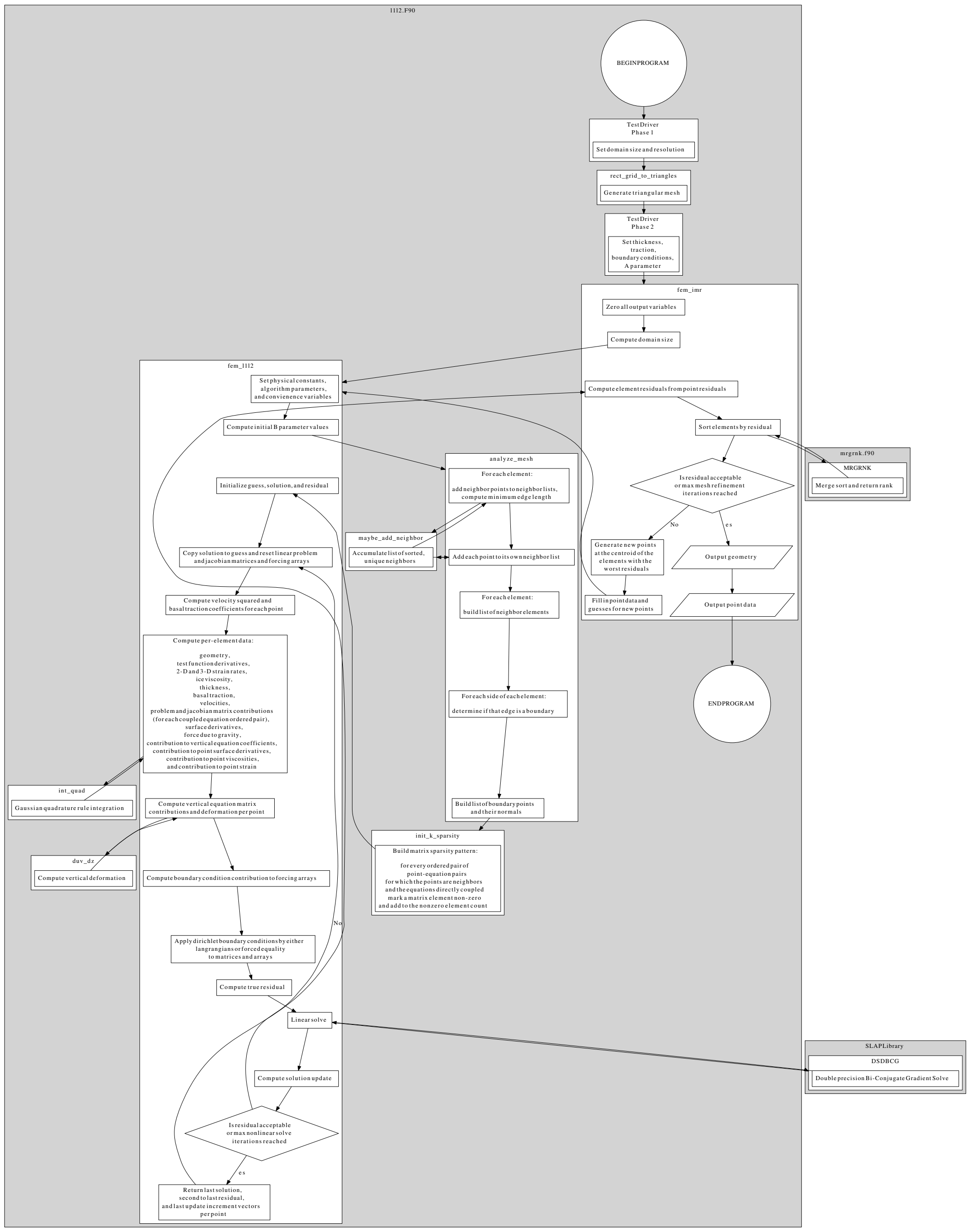

1112.F90
BEGINPROGRAM
TestDriver Phase 1
Set domain size and resolution
rect_grid_to_triangles
Generate triangular mesh
TestDriver Phase 2
Set thickness, traction, boundary conditions, A parameter
fem_imr
Zero all output variables
Compute domain size
Compute element residuals from point residuals
Sort elements by residual
mrgrnk.f90
MRGRNK
Merge sort and return rank
Is residual acceptable or max mesh refinement iterations reached
No
es
Generate new points at the centroid of the elements with the worst residuals
Fill in point data and guesses for new points
Output geometry
Output point data
ENDPROGRAM
fem_1112
Set physical constants, algorithm parameters, and convienence variables
Compute initial B parameter values
Initialize guess, solution, and residual
Copy solution to guess and reset linear problem and jacobian matrices and forcing arrays
Compute velocity squared and basal traction coefficients for each point
Compute per-element data: geometry, test function derivatives, 2-D and 3-D strain rates, ice viscosity, thickness, basal traction, velocities, problem and jacobian matrix contributions (for each coupled equation ordered pair), surface derivatives, force due to gravity, contribution to vertical equation coefficients, contribution to point surface derivatives, contribution to point viscosities, and contribution to point strain
int_quad
Gaussian quadrature rule integration
Compute vertical equation matrix contributions and deformation per point
duv_dz
Compute vertical deformation
Compute boundary condition contribution to forcing arrays
Apply dirichlet boundary conditions by either langrangians or forced equality to matrices and arrays
Compute true residual
Linear solve
Compute solution update
Is residual acceptable or max nonlinear solve iterations reached
No
es
Return last solution, second to last residual, and last update increment vectors per point
analyze_mesh
For each element: add neighbor points to neighbor lists, compute minimum edge length
maybe_add_neighbor
Accumulate list of sorted, unique neighbors
Add each point to its own neighbor list
For each element: build list of neighbor elements
For each side of each element: determine if that edge is a boundary
Build list of boundary points and their normals
init_k_sparsity
Build matrix sparsity pattern: for every ordered pair of point-equation pairs for which the points are neighbors and the equations directly coupled mark a matrix element non-zero and add to the nonzero element count
SLAPLibrary
DSDBCG
Double precision Bi-Conjugate Gradient Solve

Figure 3.1 Call Graph

basal_sliding_exp_param m, parameter of the exponent of Pollard's basal sliding formula.

basal_sliding_exp_diff exponent of the derivative of Pollard's basal sliding formula.

b_e_all array of points on the east boundary in the `eismint_square_bay2` test driver by index into `p`.

bed bed elevation for points in **p** with the same index in meters.

bed_imr bed elevation for points in **p_imr** with the same index in meters.

b_factor the $\mathcal{B}$ factor used in the expansion of the equations to be solved as in equations 2.54.

bi iterator, index into the `b` array.

bj iterator, index into the `b` array.

b_n_all array of points on the north boundary in the `eismint_square_bay2` test by index into `p`.

b_ne array of points on the north-east boundary in the `eismint_square_bay2` test by index into `p`.

b_nw array of points on the north-west boundary in the `eismint_square_bay2` test by index into `p`.

boundary_point_nx array of normal vectors projected onto the x_1 dimension with same index as the `b` array. Vector magnitude is proportional to half the average length of the incident edges.

`boundary_point_ny` array of normal vectors projected onto the x_2 dimension with same index as the `b` array.

`boundary_triangles` array of elements (triangles) on the mesh boundary.

`b_s_all` array of points on the south boundary in the `eismint_square_bay2` test by index into `p`.

`b_se` array of points on the south-east boundary in the `eismint_square_bay2` test by index into `p`.

`b_sw` array of points on the south-west boundary in the `eismint_square_bay2` test by index into `p`.

`b_w_all` array of points on the west boundary in the `eismint_square_bay2` test by index into `p`.

`b_wne` array of points on the north, west, and east boundaries in the `eismint_square_bay2` test by index into `p`.

`bX` basal drag in the x_1 direction value used by `duv_dz`.

`bY` basal drag in the x_2 direction value used by `duv_dz`.

`C` 3×3 inverse matrix of `Pe`. Contains the slopes of the linear test (and trial) functions which are nonzero on the element `e`. `C[i,j]` is the value of

$$\frac{\partial \phi_j}{\partial x_{i-1}} = \frac{\partial \psi_j}{\partial x_{i-1}}.$$

`calving_front` array of points (vertices) by index into **`p`** on the calving front.

c_factor the C factor used in the expansion of the equations to be solved as in equations 2.54.

cgret vector of return values of `duv_dz`. `cgret[i]` is the current estimate of $\overline{\frac{\partial u_i}{\partial x_3}}$.

coeffs slopes of the test and trial functions to be integrated by the various quadrature rules.

converr estimate of the error in the linear solution returned by `DSDBCG`.

domain what domain the test and trial functions should be integrated over by the various quadrature rules.

delta_U solution update vector for Newton's method.

detPe determinant of the `Pe` matrix.

dirichlet list of points (vertices) by index into **p** with dirichlet boundary conditions.

dirichlet_val dirichlet boundary condition values for each variable to apply to the points in `dirichlet` with the same index.

domain_ew width of domain in meters from east to west.

domain_ns height of domain in meters from north to south.

ds_dx $\frac{\partial s}{\partial x_1}$ for the current element `e`. This is the surface slope in the x_1 direction.

ds_dx_fudge adjustment applied to surface derivative calculations in the x_1 direction after calculations based on surface elevation array. This is used to convert a flat sheet into an infinite (periodic) sloped sheet for the ISMIP-HOM tests.

ds_dy $\frac{\partial s}{\partial x_2}$ for the current element `e`. This is the surface slope in the x_2 direction.

`ds_dy_fudge` adjustment applied to surface derivative calculations in the x_2 direction.

`du_dx` $\frac{\partial u_1}{\partial x_1}$ for the current element `e`.

`du_dy` $\frac{\partial u_1}{\partial x_2}$ for the current element `e`.

`du_dz` $\frac{\partial u_1}{\partial x_3}$ for the current element `e`.

`dv_dx` $\frac{\partial u_2}{\partial x_1}$ for the current element `e`.

`dv_dy` $\frac{\partial u_2}{\partial x_2}$ for the current element `e`.

`dv_dz` $\frac{\partial u_2}{\partial x_3}$ for the current element `e`.

`E` 2-D strain rate used by `duv_dz`.

`e2dstrain` 2-D strain rate $\dot{e}_p$ for the current element `e`.

`e3dstrain` 3-D strain rate $\dot{e}$ for the current element `e`.

`edgei` iterator. Current edge of the current element.

`edge_nx` x_1 component of the normal vector of the current edge on the mesh boundary.

`edge_ny` x_2 component of the normal vector of the current edge on the mesh boundary.

`element_basal_coeff` basal traction amount for the vertices in the current element `e`. Exact meaning depends on the basal traction formula selected.

`element_basal_coeff_diff` derivative of the basal traction amount with respect to the magnitude of the velocity for the vertices in the current element `e`. Exact meaning depends on the basal traction formula selected.

`element_basal_final` derivative of the basal traction force with respect to the velocity for for the vertices in the current element `e`.

`element_thickness` thicknesses for the vertices in the current element `e`.

`element_u1vels` velocity estimates for the vertices in the current element `e` in the x_1 direction.

`element_u2vels` velocity estimates for the vertices in the current element `e` in the x_2 direction.

`element_vt` product of the thickness and the viscosity for the vertices in the current element `e`. $\mathcal{N}$ in equations 2.54.

`error_estimate` estimate of the error in the current $\vec{U}$ solution vector.

`error_estimate_imr` estimate of the error in the current $\vec{U}$ solution vector with refined mesh points.

`F` forcing vector.

`Fb` forcing vector temporary copy.

`final_tol` final tolerance for the linear solver's convergence criterion. Used once the nonlinear solution begins to converge.

`flow_law_A` Glen's flow law A parameter to use for the ice sheet.

`flow_law_B` Glen's flow law B parameter, calculated from the A parameter.

`flow_law_epsilon` ε, the extra strain amount to be added to prevent the viscosity from becoming infinite.

`flow_law_exponent` Glen's flow law n parameter controlling the non-linearity.

`FX` elliptical component of the first equation used by `duv_dz`.

FY elliptical component of the second equation used by `duv_dz`.

grounded grounding flag for points (vertices) in **p** with the same index.

- 0 sheet is floating.
- $(0,1)$ sheet is partially grounded.
- 1 sheet is grounded.

grounded_imr grounding flag for points (vertices) in **p_imr** with the same index.

guess guess for the solution of the $\vec{U}$ solution vector.

guess_imr guess for the solution of the $\vec{U}$ solution vector with new points from refined mesh.

ice_density the density of ice, ρ.

ice_visc the viscosity of ice at each point in `p` by the same index $\approx 2\eta$.

ice_visc_zeta the ζ factor of the equations to be solved as in equation 2.58.

ierr integer error code, returned by `DSDBCG`.

iK array of row numbers with the same indexing as `vK`. See the description of the SLAP column-compressed format in section 3.3.7.

initial_tol initial tolerance for the linear solver's convergence criterion until the non-linear solution begins to converge.

invDet inverse of the determinant of the `Pe` matrix.

imri iterator, tracks the current iterative mesh refinement iteration.

iter linear solver iteration count, returned by `DSDBCG`.

is_boundary_point array of flags indicating if each element in p with the same index has an implicit boundary condition.

ismip_size period length of domain, both east to west and north to south. Set to 0 if domain is not periodic.`iwork` scratch space for the `DSDBCG` routine.

Je sub-matrix of the *J* matrix for the current element `e`.

jK array of indexes into `jK`. See the description of the SLAP column-compressed format in section 3.3.7.

Ke sub-matrix of the *K* matrix for the current element `e`.

ki iterator, index into the `iK` and `vK` arrays.

lagrange_multiplier the lagrange multiplier, λ, used for applying boundary conditions.

linear_order number of elements in the `U` array. Order of the *K* matrix.

linear_tol tolerance for the linear solver's convergence criterion. Passed to the `DSDBCG` routine.

list list of neighbors to updated by `maybe_add_neighbor`.

m number of rows of vertices to divide the domain into from north to south.

max_imriter the maximum number iterative mesh refinements to perform.

max_neighbors the maximum order of any vertex in the mesh, including additional vertices created by iterative mesh refinement.

max_nliter the maximum number of nonlinear (Newton's or Picard's) iterations to run in `fem_1112`.

max_nliter_imr the maximum number of nonlinear (Newton's or Picard's) iterations to run in fem_ll12 with an automatically refined mesh.

max_p maximum number of vertices (points) in the new, refined mesh.

max_p_growth parameter controlling maximum growth of the p array during iterative mesh refinement.

max_t maximum number of elements (triangles) in the new, refined mesh.

max_t_growth parameter controlling maximum growth of the t array during iterative mesh refinement.

maxx maximum x_1 coordinate of any vertex in p.

maxy maximum x_2 coordinate of any vertex in p.

mi iterator, current east-west running row.

minx minimum x_1 coordinate of any vertex in p.

miny minimum x_2 coordinate of any vertex in p.

min_edge_length the length of the shortest edge of the current element e.

mode_tol nonlinear convergence threshold used to determine when the linear convergence tolerance should be reduced.

mu viscosity of ice 2η.

n number of columns of vertices to divide the domain into from east to west; also in duv_dz, Glen's flow law parameter n.

`neighbors` the adjacency list for every vertex in the mesh, plus the vertex's self, by the same index as `p`. The length of `neighbors[pi,:]` for any particular `pi` is `nr_neighbors[pi]`.

`ni` iterator, current north-south running column.

`nl_resid` scalar representing the l_2-norm of the residual estimate for the current $\vec{U}$ solution vector.

`nr_b` number of points (vertices) on the boundary.

`nr_bmp_nz` number of possible nonzero coefficients of $\phi_i \psi_j \forall i, j$. This is the number of nonzero coefficients in the block matrix stencil.

`nr_bmp_nz0d` number of possible nonzero coefficients of $\phi_i \psi_j$ where $\phi_i \psi_j$ does not directly depend on adjacent vertices. These occur when the system to be solved has equations not in the weak form.

`nr_bmp_nz2d` number of possible nonzero coefficients of $\phi_i \psi_j$ where $\phi_i \psi_j$ directly depends on adjacent vertices. These occur for all equations in the weak form.

`nr_b_wne` number of points on the north, west, and east boundaries in the `eismint_square_bay2` test.

`nr_boundary_triangles` the number of elements (triangles) on the mesh boundary.

`nr_calving_front` number of points (vertices) on the calving front.

`nr_dirichlet` number of points (vertices) with dirichlet boundary conditions.

`nliter` iterator, current nonlinear (Picard or Newton) iteration.

nr_neighbors the order of every vertex in the mesh, plus one to include the vertex's self, by the same index as `p`.

nr_p number of points (vertices) in the mesh.

nr_p_imr number of points (vertices) in the new, refined mesh.

nr_point_triangles number of elements (triangles) incident to a point (vertex).

nr_t number of triangles (elements) in the mesh.

nr_t_imr number of triangles (elements) in the new, refined mesh.

nr_v number of variables (test functions) in the system to be solved.

nr_v_0d number of variables (test functions) in the system to be solved which do not depend on adjacent vertices.

nr_v_2d number of variables (test functions) in the system to be solved which depend on adjacent vertices.

old_to_new_ratio parameter controlling how many old elements should be considered for the creation of a single new element.

p list of points' (vertices') (x_1, x_2) coordinates.

p_2dstrain 2-D strain rates ($\dot{e}_p$) for each point (vertex).

pa used to keep track of vertices incident to an element, index into the `p` array.

pb used to keep track of vertices incident to an element, index into the `p` array.

pc used to keep track of vertices incident to an element, index into the `p` array.

p_ds surface derivative vectors for each point (vertex) rather than over an element.

p_duv_dz average vertical shear estimates $\approx \overline{\frac{\partial u_i}{\partial x_3}}$ returned by `duv_dz`.

Pe position matrix for a single element. This is always of the form

$$\begin{bmatrix} 1 & p_{1,1} & p_{1,2} \\ 1 & p_{2,1} & p_{2,2} \\ 1 & p_{3,1} & p_{3,2} \end{bmatrix}$$

where $p_{i,j}$ is the x_j coordinate of the ith vertex incident to the element `e`.

p_ellipticals storage for various parts of the elliptical equations (the first two equations) so that these calculations may be reused later by `duv_dz`.

periodic flag indicating if domain is periodic

0 not periodic.

1 periodic domain.

pi iterator, index into the `p` array.

pic value of the mathematical constant π, the ratio of a circle's circumference to its diameter.

p_imr list of new points' (vertices') (x_1, x_2) coordinates for new refined mesh.

pint index into the `p` array of the internal (not on the mesh boundary) vertex incident to the current element `e`.

point_triangles list of elements (triangles) incident to a point (vertex).

pvecs temporary vector storage. Unused.

pvec temporary storage for point information: this is used to avoid recomputation of various quantities which will be needed later by `duv_dz`.

p_visc viscosity values for each point (vertex).

R residual vector.

Rb temporary copy of the residual vector.

RX surface forcing in the x_1 direction value used by `duv_dz`.

RY surface forcing in the x_2 direction value used by `duv_dz`.

resid_estimate estimate of the residual in the current $\vec{U}$ solution vector.

resid_estimate_imr estimate of the residual in the current $\vec{U}$ solution vector including points in the refined mesh.

rghdsdx $\rho g h \frac{\partial s}{\partial x_1}$ for the current element `e`.

rghdsdy $\rho g h \frac{\partial s}{\partial x_2}$ for the current element `e`.

rwork scratch space for the `DSDBCG` routine.

seawater_density the density of seawater, ρ_w.

seed random number generator seed.

sliding_law flag determining which sliding law to apply.

- **1** apply the sliding law from Pollard.
- **2** apply the sliding law from ISMIP-HOM: force is proportional to $\beta^2\vec{u}$.

t list of triangles. 3-tuples of indices into the `p` array.

t_imr list of triangles in the new, refined mesh. 3-tuples of indices into the `p_imr` array.

t_area list of areas for each element (triangle) with the same index as `t` array.

t_err list of error estimates for each element (triangle) with the same index as `t` array.

thickness sheet thickness for points in **p** with the same index in meters.

thickness_imr sheet thickness for points in **p_imr** with the same index in meters.

ti iterator, index into the `t` array.

t_old old `t` list during iterative mesh refinement.

t_rank ranked list of elements (triangles) with same index as `t` array.

triangle index of element sharing a vertex with current element `e`.

triangle_neighbors list of elements adjacent to an element.

use_guess flag indicating whether `guess` has usable values in it.

U solution vector.

Ub solution vector temporary copy.

uv_ok flag indicating whether we should output $\vec{U}$ velocity solutions: this output is suppressed if it is all a single value because that breaks `plot.py`.

vel_sqr the velocity squared ($|\vec{u}|^2$) for each point (vertex) in the mesh by the same index as `p`.

vi iterator, index on variable number (test function number).

vj iterator, index on variable number (trial function number).

vJa array of values of the J matrix. See the description of the SLAP column-compressed format in section 3.3.7.

vK array of values of the K matrix. See the description of the SLAP column-compressed format in section 3.3.7.

vKa temporary copy of `vK`.

wrap_extras tuple of (`x_wrap_extra`, `y_wrap_extra`).

wraps tuple of (`x_wrap`, `y_wrap`).

x_wrap wrapping point for periodic boundary condition on x_1.

x_wrap_extra wrapping margin width for periodic boundary condition on x_1.

y_wrap wrapping point for periodic boundary condition on x_2.

y_wrap_extra wrapping margin width for periodic boundary condition on x_2.

3.3 Routines, Functions and Macros

3.3.1 Test Driver (First Phase)

The first phase of the test driver routine is responsible for setting domain geometry and resolution and calling `rect_grid_to_triangles`. There are currently three test drivers provided for this purpose: `eismint_square_bay` , `ismip_hom_a`, and `ismip_hom_c`, representing the EISMINT and ISMIP-HOM tests they are named after, respectively. The test drivers are also responsible for passing their correct second phase test driver routine as a reference to `rect_grid_to_triangles`. This allows them to share the `rect_grid_to_triangles` routine, which calculates the size of the data structures necessary to hold the triangle grid without needing to allocate memory explicitly.

We employ this technique, which is specific to FORTRAN, throughout the code. First, a subroutine calculates the size of arrays that will be required later and stores those dimensions in integers. The first routine then calls another subroutine, passing the integers into it, and the second subroutine defines local arrays using those integers to specify dimensions. This has the effect of dynamically allocating arrays of unknown size efficiently on the stack. The primary purpose of this technique in IceCamp is to enable easier analysis of routines by automated static analysis tools such as automatic differentiation tools. For this reason, at no point does IceCamp ever employ ALLOCATABLE arrays or pointers.

3.3.2 rect_grid_to_triangles

`rect_grid_to_triangles` performs the second step required to setup the simulation. It calculates the number of vertices and triangles in the initial mesh, and populates the triangle and vertex lists for the initial mesh.

The initial mesh is an even spaced grid of rectangles where each rectangle is divided in half along one diagonal. For each rectangle it chooses either the northwest to southeast diagonal or the northeast to southwest diagonal at random. This is done for debugging purposes: it helps test that the rest of the code can handle arbitrarily oriented and wound elements. Finally, `rect_grid_to_triangles` calls the subroutine that was passed in as an argument, typically this is the second phase of the test driver.

3.3.3 Test Driver (Second Phase)

Now that the domain geometry is set up, the second phase of the test driver is responsible for filling in details of the sheet's geometry, such as bed elevation, sheet thickness, basal traction, grounded areas, A parameter, and boundary conditions. There are currently three test drivers provided for this purpose: `eismint_square_bay2`, `ismip_hom_a2`, and `ismip_hom_c2`. `eismint_square_bay2` describes a flat, floating ice sheet which with zero velocity dirichlet boundary conditions on three sides and a force balance between ice and seawater on the fourth side. This is equivalent a square bay with the ice nailed to the shore. `ismip_hom_c2` and `ismip_hom_a2` describe a square, periodic, grounded sheet on a slope with either basal traction or basal elevation varying with a vertical and horizontal sine function. The second phase of the test driver then calls `fem_imr` which ends the set-up phase of the operation of the model.

3.3.4 fem_imr

The `fem_imr` routine is responsible for performing iterative mesh refinement, calling the core L1L2 FEM solver on a mesh, deciding when the solution is acceptably accurate and outputting the solution. `fem_imr` contains the main outermost loop of the system, which runs the L1L2 FEM solver repeatedly on different meshes until residual vectors of acceptably small magnitude are reached.

`fem_imr` recomputes the domain geometry from the mesh vertex list. This is done so that `fem_imr` does not depend on drivers to do these calculations beforehand. It then calls `fem_l1l2` with the initial mesh. Based on the residual vector returned by `fem_l1l2` it then iteratively refines the mesh and reruns the solver until either the magnitude of the residual vector falls below a threshold or a maximum iteration count is reached.

This process involves first computing an element-wise residual vector, where each element's residual is the l_2-norm of the residual of its vertices, sorting this vector, and then splitting the worst elements. The number of elements split is determined by the ratio `old_to_new_ratio`. Each element is split into three elements with a new vertex at the center of the old element. Once an element is split, values for all of the geometric and boundary properties of the new vertex are estimated, including the basal traction, whether the element is grounded, its thickness, base elevation and a starting velocity estimate.

After the splitting process is complete, `fem_l1l2` is called again on the new mesh. Once iterative mesh refinement exhausts its allowed number of iterations, fem_imr is responsible for outputting the geometry (of the final mesh) and the final velocities in both the x_1 and x_2 directions, the magnitude of the velocity, error estimates and residuals for each point to a text file.

3.3.5 fem_l1l2

fem_l1l2 is the largest routine in IceCamp and is responsible for solving the L1L2 equations using the Finite Element Method on a given triangle mesh. It does this in two stages: first it analyzes the mesh and builds data structures which it will reuse at every iteration of the solution process, then it iteratively solves the nonlinear L1L2 equations using either Newton's Method or Picard iteration.

The first thing calculated by fem_l1l2 is nr_k_nz which determines the maximum number of nonzero elements that can appear in the linear K and J matrices based on the triangular mesh received. This ensures that later processing steps have enough memory to build these sparse matrices. Then fem_l1l2 calls analyze_mesh and init_k_sparsity to extract information from the mesh geometry, such as implied boundaries and build the K and J sparsity pattern.

Depending on compile-time options, fem_l1l2 works by either a Picard iteration or by Newton's Method. As shown in Figure 1.1, in Newton's Method mode it forms both the K matrix, by integrating the weak form of system 2.27 and the J matrix by integrating system 2.67. In Picard iteration mode, it only forms K.

fem_l1l2 then enters its main loop which implements the nonlinear solution process using either Newton's Method or Picard iteration. The first step of this loop is to calculate various quantities for each point which will be used later: the current estimated velocity squared and the current estimate of basal traction magnitude. The next step of the loop is responsible for filling in the K and J matrices. It performs this step on an element-wise basis even though each entry in K and J represent coefficients of a system of equations on each vertex. It does this by accumulating the contribution to each K and J entry from each element.

Various quantities relating to each element are calculated: the element's area, the slopes and volumes of each test (or trial) function that is nonzero at that element, various factors employed by the equations, the approximate viscosity across the entire element, its average thickness, and its average velocity. If the element is on a periodic boundary, the positions of its individual vertices are first recomputed. The elements area is computed by taking the determinant of its position matrix. The slopes of the test and trial functions are computed by inverting the same position matrix.

For the first nonlinear iteration, the ice viscosity, 2-D strain rate and 3-D strain rate are a hard-coded default. For every subsequent iteration they are based on the actual strain rates given the velocity solution of the previous iteration.

Then the precise contributions to K and J are formed in 3×3 submatrices K_e and J_e through a system of C Preprocessor macros. These macros factor out code that would otherwise be extremely repetitive by multiplying the correct factors against the correct test function derivatives or proportions for any particular pair of equations in the system to be solved. Then, K_e and J_e are added to the appropriate subset of K and J.

The C Preprocessor macros employed are `SETUP_ELT_SUBMAT` and `SETUP_ELT_SUBMATIJ`. `SETUP_ELT_SUBMAT` sets all of K_e or J_e at once by running `SETUP_ELT_SUBMATIJ` once for each of the nine entries in K_e or J_e, and storing them. `SETUP_ELT_SUBMAT` takes seven arguments: the matrix to store to, four coefficients which depend on the factors for the specific trial function and test function derivative product being produced, a scalar to scale every factor by and a 3 element vector which will be averaged and used to scale the volume of the trial function and test function product. The four coefficients come from the factorization of the weak form of the system into multiples of $\phi_i \psi_j$ where ϕ_i is a test function and ψ_j is a trial function. For example, when forming K_e for $\phi_1 u_1$, we consider the coefficients of

$$h\nu\left(4\frac{\partial\phi_1}{\partial x_1}\frac{\partial u_1}{\partial x_1}+0\frac{\partial\phi_1}{\partial x_1}\frac{\partial u_1}{\partial x_2}+0\frac{\partial\phi_1}{\partial x_2}\frac{\partial u_1}{\partial x_1}+1\frac{\partial\phi_1}{\partial x_2}\frac{\partial u_1}{\partial x_2}\right)+\beta^2\phi_1 u_1 \\ \underbrace{+\phi_1 f\left(u_{i\neq 1}\right)=\ldots}_{\text{used in other } K_e \text{calculations}} \tag{3.1}$$

we call `SETUP_ELT_SUBMAT` with the arguments $\left(K_e, 4, 0, 0, 1, h\nu, \beta^2\right)$. These calculations are similar to the ones used to compute convenience variables `du_dx`, `dv_dx`, `du_dy`, `dv_dy`, `du_dz` and `dv_dz`, except with additional scaling factors. Unlike the convenience variables, they are also multiplied by the test functions ϕ_i and their derivatives to compute off-diagonal entries.

Next, quantities related to external forcings are computed for the current element. These are the surface slope, and force due to gravity: $\rho g h \frac{ds}{dx_i}$. Unfortunately, these calculations are made quite complex due to the possibility of a periodic boundary condition which induces a torroidal topology on the entire problem. In this case, the effective elevation and location of each vertex must be computed. This depends on the element currently being considered, and whether it spans the periodic boundary. These values are then accumulated into the F forcing vector, and some calculations are saved for the Pollard L1L2 3D strain rate estimation code which is run point-wise, not element-wise.

The fourth set of calculations are done on a point-wise basis. Elements of K and J that couple equations from the system 2.27 are set to 1. Then `duv_dz` is called which estimates the Pollard L1L2 3D strain rate adjustment and saves it to the F forcing vector. Then, points in the implied boundary set are assumed to be boundaries with the ocean and a floating sheet force balance is calculated and saved to the F forcing vector.

The last set of precalculations in the main loop are applied only to points on the implied boundary or points with explicit boundary conditions set. These boundary conditions can be applied in one of two ways: the first being to set the $K_{i,j}$ matrix element to 1 and the F_i vector element to the value to be applied as a boundary condition. The second is to use Lagrangian multipliers to apply the boundary condition by adding λ to the $K_{i,j}$ matrix element and adding $\lambda \cdot v$ to the F_i vector element, where v is the value we wish to apply. For example, for a Dirichlet boundary condition at a particular vertex, the K-matrix elements representing the u_i velocity equations would be adjusted, as would the F vector elements representing their value.

Whether or not Lagrange multiplier boundary conditions are applied instead of forced boundary conditions depends on whether the C Preprocessor flag `LAGRANGEBC` is defined. This can be adjusted at the top of the source file or on the compiler command line.

Now that the matrices and vectors for the two equations $KU = F$ and $JW = R$ are ready, the main loop then enters the phase where it actually solves for U or W. First, $R = KU_{prev} - F$ is computed. The matrix-vector product is performed by a call to `DSMV`. U_{prev} is merely the previous iteration's U solution. Not that this is solution is preserved across iterations, and even across calls to `fem_l1l2`.

Then, if Newton's method is being employed, $JW = R$ is solved for W and applied to U as $U \leftarrow U + W$. Otherwise, $KU = F$ is re-solved with an updated K and F based on the previous iteration's solution for U. This is the Picard iteration. These linear solutions are computed by `DSDBCG`, a double-precision, iterative, preconditioned, bi-conjugate gradient solver. For this purpose, minimum residual solvers are not appropriate: their aggressive solutions cause patterns in the residual which interfere with effectiveness of the Newton iteration.

Finally the l_2-norm of the change in U from the previous iteration to the current is compared against the threshold in `mode_tol` and `final_tol`. If it is lower than those thresholds, or the maximum number of nonlinear iterations has been reached, `fem_l1l2` terminates, returning U, R and $U_{prev} - U$.

It should be noted that the nonlinear iteration passes different convergence thresholds to the linear solver depending on the residual computed by the nonlinear iteration. Once a sufficiently small residual is produced, the linear solver convergence threshold is reduced to produce a more accurate solution.

Whether or not Newton's method is applied instead of a Picard iteration depends on whether the C Preprocessor flag `NEWTONS` is defined. This can be adjusted at the top of the source file or on the compiler command line.

3.3.6 analyze_mesh

The `analyze_mesh` routine is employed by `fem_1112` before looping to efficiently precompute data structures based on the mesh structure. These structures are: an adjacency list describing which vertices are neighbors, which vertices are implied boundary points (points on the edge of the mesh), and normal vectors for those boundary points.

First, an adjacency list is built vertices to each other and an incidence list for vertices to elements is built by iterating over each element and adding its points to the appropriate list. This is necessary since `fem_imr` only requires an incidence list of elements to vertices. This first loop computes the inverse of that list. `fem_imr` also ensures that each vertex is counted as its own neighbor. Each list's sub-list is built by the `maybe_add_neighbor` routine to maintain a specific ordering which will be required by the `init_k_sparsity` routine.

Next, for each edge in each triangular element, `analyze_mesh` finds the neighboring element across that edge from the current element. It does this by considering every other element sharing a vertex in the current edge with the current element, and checking if the other vertex incident with that particular edge is also shared. This is done on an elements basis.

Third, boundary points are found, listed, and their normal vectors are computed. This is done by scanning all boundary edges. Boundary edges are the edges of an element without a neighboring element across that edge. For each of these boundary edges, its incident vertices are added to the list of boundary vertices. The normal vector of the edge times its length is computed and added to the normal vector of both incident vertices. Using this technique, by the time the loop terminates, an each boundary vertex's normal vector has been computed as the sum of the incident boundary edges.

Finally, all normal vectors divided by the number of incident edges. This is necessary to compensate in the case of a single boundary vertex having more than two incident boundary edges. Though geometry with this property should be considered pathological, the code does attempt to handle it as well as it can.

3.3.7 init_k_sparsity

The `init_k_sparsity` subroutine precomputes the K and J matrix sparsity patterns. These matrices are stored in SLAP's compressed column format, and we assemble them directly into this format. Assembling the matrices in this format instead of triad format dramatically improves performance of the solver since the matrices do not need to be sorted and take up a minimal amount of space. Additionally, since the K and J matrices have the same sparsity pattern, they can share data structures.

Briefly, the compressed column format consists of 3 one-dimensional arrays, in the code they are referred to as `iK`, `jK` and `vK`. If K is an $n \times n$ matrix, `jK` is an array of length $n+1$. Each element of `jK` is an index of `iK` representing the start of that column in `iK`. `iK` is an array of length `nr_k_nz`. Each element of `iK` represents a a row number, and the corresponding element of `vK` is the value of that element of K. For example, if we wished to get or set K_{ij} we would get or set $\mathtt{vK}[x]$ where x is such that $\mathtt{iK}[x] = i$ and $\mathtt{jK}[j+1] \geq x \geq \mathtt{jK}[j]$. Furthermore, each section of `iK` representing a column j is sorted with the element on the diagonal, which must exist, first, and then in row order. This enables efficient location of individual elements of K, especially those on the diagonal. Sort order is computed by `maybe_add_neighbor` as called by `analyze_mesh`.

The `init_k_sparsity` subroutine constructs this data structure by initializing `iK`, `jK` and `vK` in index order. First it inserts the diagonal entry for each test function and vertex i combination. Then, for each combination of that test function, any trial function, that vertex i and any vertex j that may be non-zero, it adds an entry to `iK` in sort order and initializes the corresponding element of `vK` to zero. It makes use of the fact that only neighboring vertices as computed by `analyze_mesh` can cause non-zero entries. Additionally, it consults the `bmp` (block matrix pattern) matrix, which defines whether a test function and trial function pair interact in the weak form of the system to be solved. `bmp` can be considered a stencil that is applied once for every edge in the finite element mesh.

3.3.8 duv_dz

This subroutine was produced by the Maple computer algebra system. It requires the current viscosity estimate, surface forcings, 2-D SSA strain rate approximation, the temperature-dependent Glen's flow law B parameter, Glen's flow law exponent n parameter, and the basal drag vector. It uses these to produce and return an estimation of $\overline{\frac{\partial u_1}{\partial x_3}}$ and $\overline{\frac{\partial u_2}{\partial x_3}}$. The Maple input used to produce the statements in `duv_dz` is included in the source repository and as an Appendix.

3.3.9 maybe_add_neighbor

This subroutine maintains a list of points neighboring a single point in sorted array order: the vertex itself first, and then sorted by indices of neighboring vertices. It is currently implemented as an inefficient single iteration insertion sort. Since neighbor lists are of limited length this doesn't post a significant performance concern. It depends on `max_neighbors` being set appropriately so that there is enough space in the array for the maximum degree of all vertices plus one.

3.3.10 k_index

This function locates the index of `vK` representing $K_{i,j}$ or returns an error if $K_{i,j}$ is assumed to be zero.

3.3.11 Integration Routines

IceCamp includes various routines to perform integration. There are three quadrature rules and an exact formula, which was produced by Maple. These routines are designed to integrate the product of two affine functions of a two dimensional space over a given domain. Specifically, a test and a trial function are normally supplied, along with scaling coefficients for each.

3.3.12 External Routines

Three external routines are employed by 1112.F90: `DSDBCG`, `DSMV`, and `MRGRNK`. `DSDBCG` solves an affine system of the form

$$A\vec{X} = \vec{C}$$

by the iterative bi-conjugate gradient method using double-precision numbers. `DSMV` calculates double-precision matrix vector product using matrices in compressed column format. `MRGRNK` sorts a vector using the merge sort algorithm and returns another vector of indices into the original vector in sort order.

3.3.13 Helper Macros

The three C Preprocessor Macros `VPtoI`, `ItoV` and `ItoP` are convenience functions to map variable (trial function) and point (vertex) indices to row or column indices of the K or J matrices back. `VPtoI` produces a row (or column) index from a variable index, point index pair. `ItoV` produces a variable index from a row index. `ItoP` produces a point index from a row index. Additionally, the convenience function `SURFACE` is defined to return the surface elevation at a point.

3.3.14 plot.py

`plot.py` is a short utility written in Python to plot the output of l1l2.F90 to the display.

CHAPTER 4 Results

	Maximum u_1		
	IceCamp	Pattyn *et al.* μ	$n\sigma$
5km	17.23	12.14	0.97
10km	16.66	15.39	0.84
20km	19.53	18.31	1.02
40km	29.83	28.48	0.76
80km	61.63	60.99	0.11
160km	145.63	141.38	0.19

Table 4.1 ISMIP-HOM Test C Numerical Results

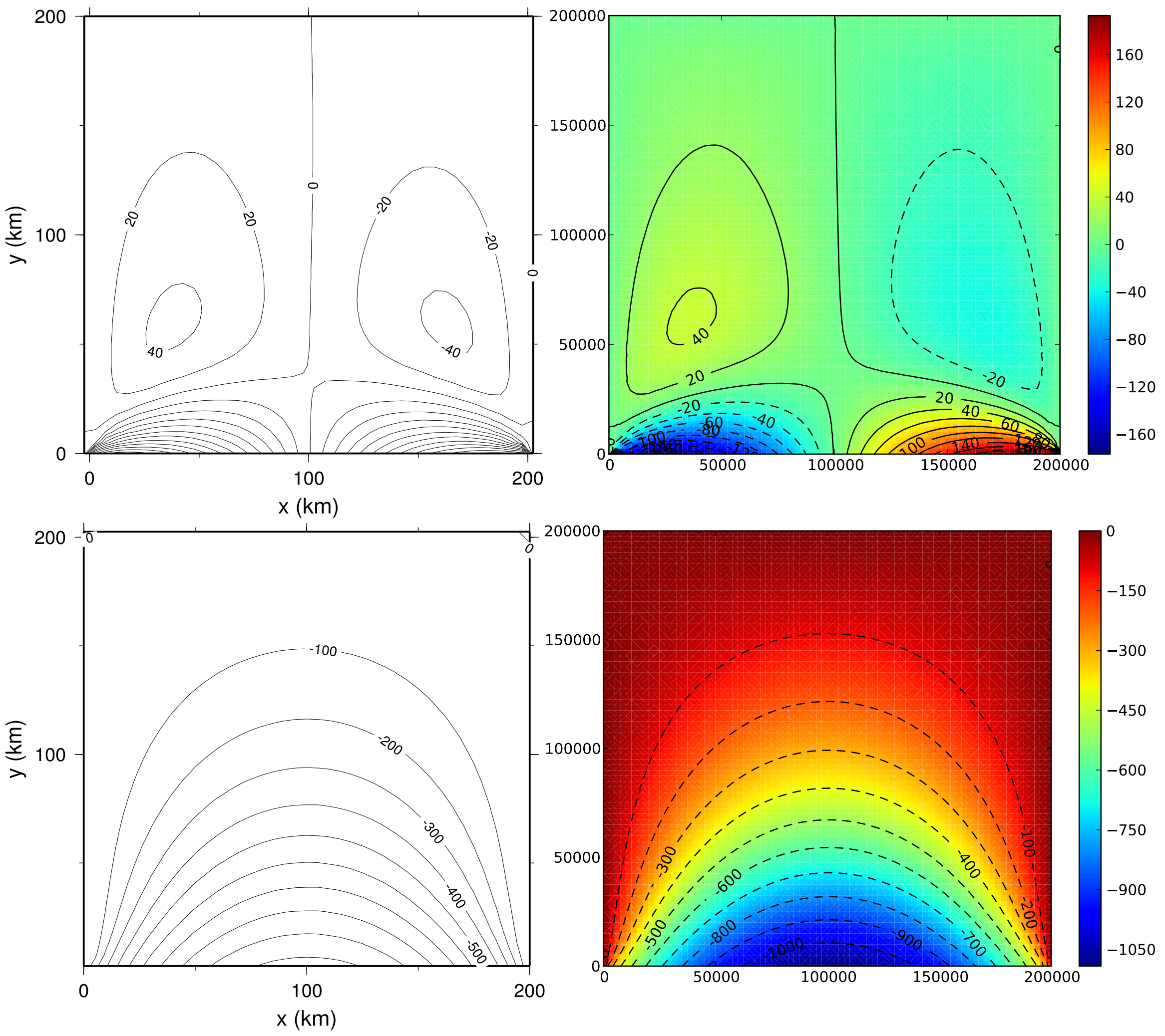
4.1 Results of the EISMINT Square Bay Test

Figure 4.1 shows the results of the software implementation presented here side by side with the expected results from the EISMINT square bay test. The plots for the expected results are taken directly from Rommelaere [1998]. The EISMINT square bay test is a 200km square bay which has 0 velocity on three sides, and a calving front force balance boundary on the south side. Everywhere on the interior is a 500m thick floating ice shelf.

4.2 Results of the ISMIP-HOM Test C

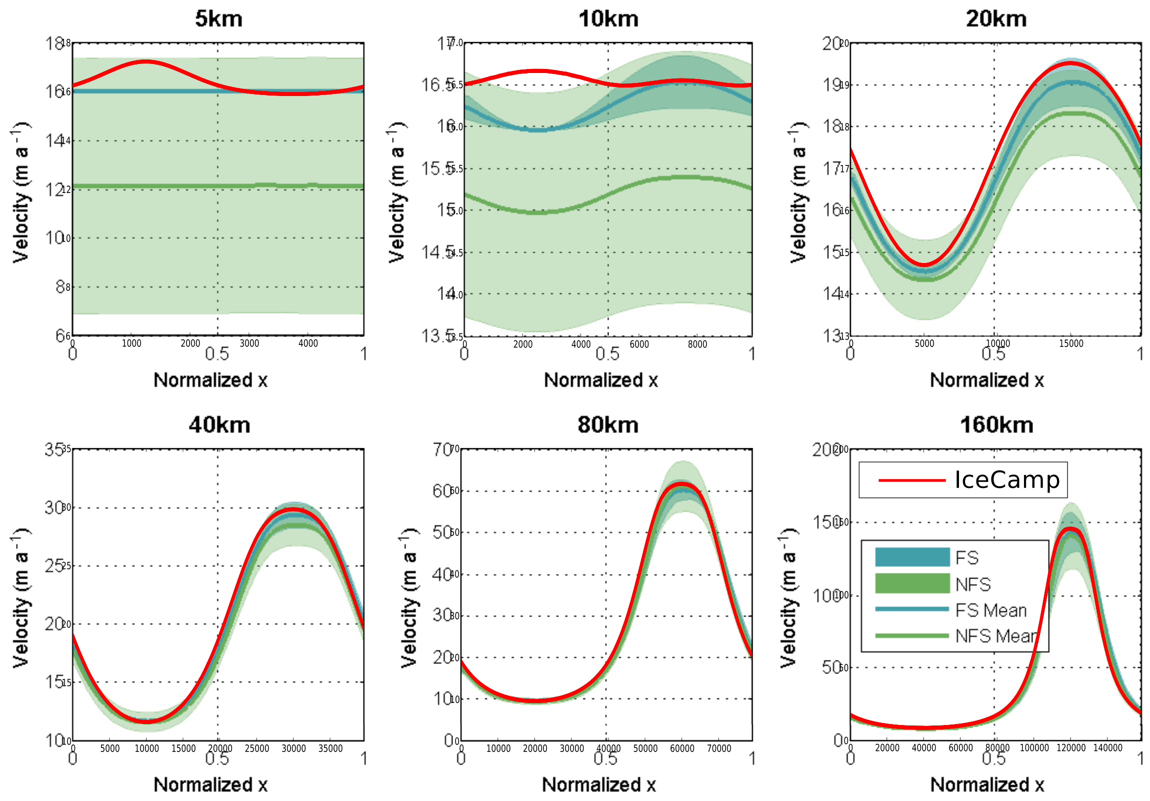
Figure 4.3 shows the results of the software implementation presented in this thesis overlaid on top of the results from other models presented in Pattyn et al. [2008]. One interesting thing to note is that, as discussed in Pattyn et al., the surface velocity field in the 5km test is anti-correlated to the basal friction. However, in the model presented here there is both a correlated and anti-correlated signal present in the 10km test results.

Table 4.1 shows where the maximum horizontal velocity in the direction of ice flow produced by this implementation lies in comparison with other models in Pattyn et al.. It lists, for each domain size, the maximum for this implementation, the mean of the maximums for the models in Pattyn et al. and how many standard deviations outside of that mean this implementation's maximum lies.



Top left: the standard (expected) EISMINT results for the East-West velocity field. Top right: IceCamp (the implementation presented in this thesis) results for the East-West velocity field. Bottom left: the standard (expected) EISMINT results for the North-South velocity field. Bottom right: IceCamp results for the North-South velocity field. Velocities in m/a.

Figure 4.1 EISMINT Square Bay Result Comparison



IceCamp with various models from the ISMIP-Hom comparison project. FS is the mean of various full-stokes models, and NFS is the mean of approximation models.

Figure 4.2 ISMIP-HOM Test C Result Comparison

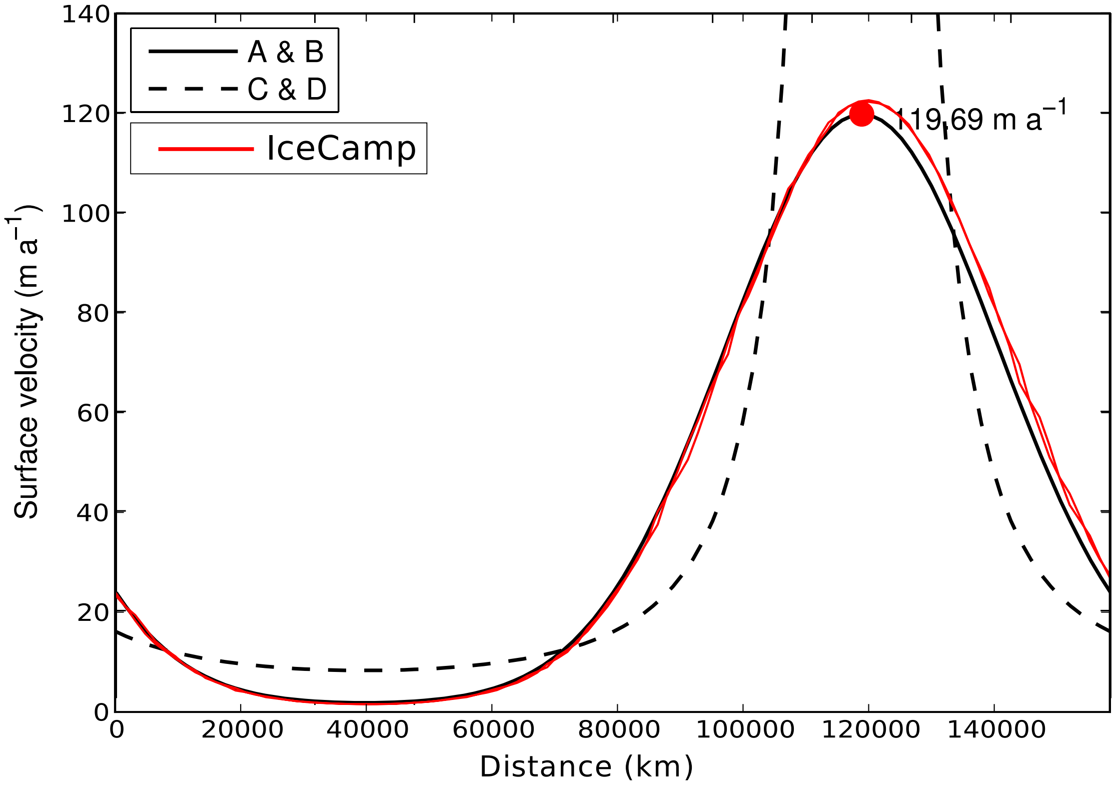


Figure 4.3 ISMIP-HOM A Result Comparison

4.3 Results of the ISMIP-HOM Test A

Figure 4.3 shows the results of the software implementation presented in this thesis overlaid on top of the results from the Shallow Ice Approximation (SIA) results for all domain sizes from Pattyn et al. [2008]. This figure demonstrates a bug in the current implementation that causes the approximation to regress to the SIA when basal sliding is negligible. This figure illustrates the presence of the SIA implicit in the vertical shear terms.

Table 4.2 shows the results of IceCamp on the 5km ISMIP-HOM A test with near the expected results for SIA (199.69m a^{-1}) at high β^2 values. The expected maximum values are 14.65 ± 0.19m a^{-1} for full-Stokes models [Pattyn et al., 2008]. This table also shows the relationship between the two types of flow in the current implementation: sliding goes

β^2	Maximum u_1 Velocities in m a^{-1}		
	Sliding	Shearing	Surface
10^9	0.000118	122.662919	122.663036
10^8	0.001178	122.659613	122.660791
10^7	0.011778	122.558307	122.570085
10^6	0.117398	121.355896	121.473294
10^5	1.133685	109.286043	110.419728
10^4	9.248845	59.349738	68.598583
10^3	76.592625	33.724529	110.317154
10^2	735.333031	29.849443	765.182474
10^1	7319.097179	29.435381	7336.043234
1	73173.247236	29.393673	73156.310658

Table 4.2 ISMIP-HOM Test A 5km Numerical Results

to 0 at high β^2 values, as expected, and shearing is responsible for all flow. As β^2 decreases, we see sliding go to infinity, and shearing converge to exactly twice the expected full-Stokes solution.

CHAPTER 5 Conclusion

5.1 Discussion

5.1.1 Derivation

This thesis includes the full derivation of all discrete systems employed by the implementation. This derivation includes many intermediate algebraic manipulations missing from other treatments of the Shallow-Shelf Approximation and L1L2 approximation. The author felt that this was important for both validation and as a resource for future treatments of the same or similar systems. Other derivations can be very difficult to comprehend as they are very short, and combine many steps into a single step with little or no commentary.

The derivation includes explicit mentions of when, where, and how approximations, discretizations, and boundary conditions are applied to the original system of equations. This is intended to allow future work to start at any of these points by choosing alternate approximations, discretizations and boundary conditions.

5.1.2 Symbolic Differentiation

We provide in this thesis a symbolically differentiated first variation of the Jacobian of the L1L2 model equations. This symbolically differentiated form is suitable for application in any variational form, not only the weak form as applied using the finite element method. Furthermore, it is suitable for any discretization, not only a finite element discretization. Simply applying the discretization of choice to the first variation presented will yield the Jacobian of the same discretization of the original system.

Additionally, by using symbolic differentiation, we avoided the complications of automatic differentiation. By avoiding automatic differentiation, we keep the code simple for use with automatic differentiation tools to be applied later, in order to differentiate with respect to other variables. This can be used, for example, for sensitivity analysis with respect to various parameters, such as the tricky n exponent parameter used in Glen's flow law.

Avoiding Jacobian-free techniques has the disadvantage of requiring the formation of the Jacobian but also has advantages. The biggest advantage is that the Jacobian includes a local linear approximations of the full coupling between all four solution variables. This coupling includes the coupling of the viscosity to all four solution variables implicitly. This allows Newton's method to work on all discretized variables simultaneously, including implicit ones such as the ice viscosity. No fixed point iteration is required for any part of the solution. This choice is in contrast to other implementations such as the one presented in Lemieux et al. [2011], which still requires fixed-point iteration to obtain values for the ice viscosity while using a Jacobian-free Newton-Krylov method to solve the velocity field $\vec{u}$.

5.1.3 Model Accuracy

The model output is accurate. It matches the published results of full-Stokes models for ISMIP-HOM test C at all domain sizes except 10km, where it is still transitioning from the anti-correlated signal to the correlated signal exhibited on larger domains. However, it still achieves an accurate maximum velocity. Additionally, it's output matches the results of the EISMINT square-bay test.

ISMIP-HOM test A results match the analytic solution from the Shallow Ice Approximation (SIA) instead of the full-Stokes solution currently due to a bug in the implementation. However the bug does not affect the validity of the derivation presented in section 2.

5.1.4 Future Directions

The largest deficiency in the current implementation of the model is its lack of parallelization. Hopefully, this can mostly be addressed simply by modifying the implementation to use a parallel linear solver such as one included in a library such as PETSc or Trillinos. We do not expect this to be difficult as long as that solver understands the

SLAP column-compressed format or another format which can be generated from column-compressed format easily. The implementation depends on building sparse matrices directly in an ordered format to maintain efficiency. Using sparse format conversion routines which re-sort the matrix representation would have a large performance impact on the implementation.

BIBLIOGRAPHY

D. Anthoff, R.J. Nicholls, and R.S.J. Tol. The economic impact of substantial sea-level rise. *Mitigation and Adaptation Strategies for Global Change*, 15(4):321–335, 2010.

F. Bosello, R. Roson, and R.S.J. Tol. Economy-wide estimates of the implications of climate change: Sea level rise. *Environmental and Resource Economics*, 37(3):549–571, 2007.

Ed Bueler and Jed Brown. Shallow shelf approximation as a "sliding law" in a thermomechanically coupled ice sheet model. *Journal of Geophysical Research*, 114(F3):F03008, 2009.

J.M. Bull, L.A. Smith, L. Pottage, and R. Freeman. Benchmarking Java against C and Fortran for scientific applications. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, pages 97–105. ACM, 2001.

A. Cazenave and W. Llovel. Contemporary sea level rise. *Annual Review of Marine Science*, 2:145–173, 2010.

I.P.O.C. Change. Climate change 2007: the physical science basis. *Agenda*, 6:07, 2007.

John K Dukowicz, Stephen F Price, and William H Lipscomb. Consistent approximations and boundary conditions for ice-sheet dynamics from a principle of least action. *Journal of Glaciology*, 56(197):480–496, 2010.

Michael Fagan, Laurent Hascoet, and Jean Utke. Data representation alternatives in semantically augmented numerical models. In *Source Code Analysis and Manipulation, 2006. SCAM'06. Sixth IEEE International Workshop on*, pages 85–94. IEEE, 2006.

Daniel N Goldberg. A variationally derived, depth-integrated approximation to a higher-order glaciological flow model. *Journal of Glaciology*, 57(201):157–170, 2011.

Daniel N Goldberg and Olga V Sergienko. Data assimilation using a hybrid ice flow model. *The Cryosphere*, 5:315–327, 2011.

Marijke Habermann, David Maxwell, and Martin Truffer. Reconstruction of basal properties in ice sheets using iterative inverse methods. *Journal of Glaciology*, 58(210): 795–807, 2012.

RCA Hindmarsh. A numerical comparison of approximations to the stokes equations used in ice sheet and glacier modeling. *J. Geophys. Res*, 109(10.1029), 2004.

D.A. Knoll and D.E. Keyes. Jacobian-free newton–krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004.

Jean-François Lemieux, Stephen F. Price, Katherine J. Evans, Dana Knoll, Andrew G. Salinger, David M. Holland, and Antony J. Payne. Implementation of the jacobian-free newton-krylov method for solving the first-order ice sheet momentum balance. *Journal of Computational Physics*, 230(17):6531 – 6545, 2011. ISSN 0021-9991. doi: 10.1016/j.jcp.2011.04.037. URL `http://www.sciencedirect.com/science/article/pii/S0021999111002853`.

William Lipscomb, Robert Bindschadler, Ed Bueler, David Holland, Jesse Johnson, and Stephen Price. A community ice sheet model for sea level prediction: Building a next-generation community ice sheet model; los alamos, new mexico, 18-20 august 2008.

Eos, Transactions American Geophysical Union, 90(3):23–23, 2009. ISSN 2324-9250. doi: 10.1029/2009EO030004. URL `http://dx.doi.org/10.1029/2009EO030004`.

D.R. MacAyeal. Eismint: Lessons in ice-sheet modeling. *University of Chicago, Illinois*, 392, 1997.

W.S.B. Paterson. *The Physics of Glaciers*. Pergamon, 1983. URL `http://books.google.com/books?id=Bl9YMAEACAAJ`.

F Pattyn, L Perichon, A Aschwanden, B Breuer, B De Smedt, Olivier Gagliardini, G Hilmar Gudmundsson, R Hindmarsh, A Hubbard, JV Johnson, et al. Benchmark experiments for higher-order and full stokes ice sheet models (ismip-hom). *The Cryosphere Discussions*, 2(1):111–151, 2008.

D. Pollard and R.M. Deconto. A coupled ice-sheet/ice-shelf/sediment model applied to a marine-margin flowline: forced and unforced variations. *Glacial Sedimentary Processes and Products:(Special Publication 39 of the IAS)*, 23:37, 2009.

SF Price, AJ Payne, and A Shepherd. A three-dimensional, first-order model of ice flow: Numerical implementation, validation, and initial application to iceland and greenland. In *AGU Fall Meeting Abstracts*, volume 1, page 06, 2007.

Vincent Rommelaere. Ice shelf models intercomparison, setup of the expirements. In Philippe Huybrechts, editor, *Report of the Third EISMINT Workshop on Model Intercomparison*, Grindelwald, Switzerland, September 1998.

IC Rutt, M Hagdorn, NRJ Hulton, and AJ Payne. The glimmer community ice sheet model. *J. geophys. Res*, 114:F02004, 2009.

G. Strang. *Computational science and engineering*. Wellesley-Cambridge Press Wellesley, MA, 2007.

J. Utke, U. Naumann, M. Fagan, N. Tallent, M. Strout, P. Heimbach, C. Hill, and C. Wunsch. Openad/f: A modular open-source tool for automatic differentiation of fortran codes. *ACM Transactions on Mathematical Software (TOMS)*, 34(4):18, 2008.

OC Zienkiewicz and R Taylor. The finite element method 5th edition, volume 1: The basis, section 14.4, 2000.