Abstract

DEROCHERS, STEVEN JOSEPH. Numerical Study and Feedback Stabilization of a Linear Hydro-Elasticity Model. (Under the direction of Lorena Bociu.)

Fluid-structure interactions arise in a variety of physical, biological, and engineering problems. Modeling of this phenomenon is crucial in these applications. This work investigates a *total linearization* of a free-boundary nonlinear elasticity - incompressible fluid interaction recently derived in [11, 12]. This linearization uses shape optimization techniques and incorporates the geometry of the problem into the analysis. Mathematically, this model brings up "new" terms–not present in the classical coupling of Stokes flow and linear elasticity–in the matching of the normal stresses and the velocities on the interface. It was demonstrated earlier that this PDE system generates a C_0-semigroup; however, unlike in the standard Stokes-elasticity coupling, the well-posedness result depended on the fluid's viscosity and the new boundary terms which (among other things) involve the curvature of the interface. This work implements a finite element-type numerical scheme for approximating solutions of the fluid-elasticity dynamics and numerically investigate the dependence of the discretized model on the terms not present in the classical Stokes-elasticity coupling.

Furthermore, it is known that the classical Stokes-elasticity coupling can be stabilized by boundary feedback in the form of dissipative matching on the interface [3]. Similarly, the total linearization can be stabilized; however, in addition to an analogous boundary feedback, the total linearization requires dissipation in the elastic interior domain.

Numerical Study and Feedback Stabilization of a Linear Hydro-Elasticity Model

by
Steven Joseph Derochers

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2017

APPROVED BY:

Zhilin Li	Jesus Rodriguez
Xiao-Biao Lin	Lorena Bociu Chair of Advisory Committee

Biography

Steven Derochers was raised in Morristown, Tennessee. He earned his bachelors of science degree in mathematics with an honors concentration from the University of Tennessee in 2011. Afterward, he attended North Carolina State University where he earned his masters degree in 2014.

Acknowledgements

I would like to thank my advisor Dr. Lorena Bociu, my collaborator Dr. Daniel Toundykov, as well as my family and friends for all their guidance and support. Without their tireless effort, this work would not have been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In fluid-elasticity interaction problems, an elastic structure interacts with a gas or fluid flow. These types of problems have many physical applications, including the flutter of airplane wings, vibration of turbine and compressor blades, blood flow in arteries, heart valve dynamics, and interocular dynamics. Fluid-elasticity interactions fall in the category of free and moving boundary problems. In a *free boundary* problem, the geometry of the interface surface is typically unknown but remains stationary (not time dependent). In a *moving boundary* problem, the common interface is dynamic, and a priori information is provided on the evolution of the geometry. A free and moving boundary value problem combines the challenges of both of these and requires the unknown boundary to be determined as a function of time and space. The quintessential example is an elastic body deforming and moving inside a flowing fluid.

Mathematically, this interaction is described by a partial differential equation (PDE) system that couples the parabolic (fluid) and hyperbolic (elasticity) phases, where the key issue is that the traces of the elastic component at the energy level are not defined via the standard trace theory. Due to this mismatch in the coupled system and its highly

nonlinear nature (the equations are nonlinear, and the common boundary is one of the unknowns in the system and has to be determined as part of the solution), fluid-elasticity problems continue to be one of the most challenging topics to date. Obviously, the model has attracted a lot of attention, and its mathematical solvability [6, 13, 18, 19, 22, 21, 25, 33, 34, 35, 36, 39, 41, 53, 58, 55], as well as numerical approximations [23, 28, 40, 57, 56, 48, 50] have been intensively studied in the last three decades.

Various useful simplifications of this models have been considered. A significant amount of fluid-elasticity interactions fall in the category of small but rapid oscillations of the solid body, so that the common interface may be assumed static. Another valid simplification is to consider linearized models, which in some cases provide good approximations of the original nonlinear dynamics. To our knowledge, the first such system appeared in print as early as 1965 in [16] and was prompted by biomedical applications. The dynamics was linearized at rest, which effectively amounted to a fixed/cylindrical geometry on which Stokes flow and linear isotropic elasticity systems were coupled through suitable stress and velocity matching conditions on the common boundary.

In this thesis, we consider a new, more general linear model for hydro-elasticity, which was recently proposed in [11, 12]. This work is motivated by the fact that both stability and control problems related to hydro-elasticity involve the investigation of this new linearized model. The associated well-posedness analysis for the linear system is paramount for both stability investigations and optimal control problems subject to fluid-elasticity interaction, as it is described below:

(i) **Optimal Control:** The work in [11, 12] was performed in the context of an optimal control problem associated with a free and moving boundary interaction (for example, optimizing the pressure in the fluid, or minimizing the flow turbulence). The linearization of the system represents the sensitivity equations needed in the process

of finding necessary optimality conditions, which provide a characterization of optimal control, much needed in numerical computations. In this setting, the nonlinear equations of fluid and solid dynamics (represented by the Navier-Stokes and nonlinear elasticity) were linearized simultaneously with the free and moving interface between them, using shape optimization and tangential calculus techniques. This refinement, which we will address as a 'total linearization', led to a markedly different and more complex model, which agrees with the classical one when the steady regime is the zero equilibrium. In this new linearization, the matching conditions on the common interface account for the geometry of the problem through the presence of boundary curvatures. The well-posedness of the new linearized system is necessary in the investigation of existence and characterization of optimal control.

(ii) **Stability:** An important design aspect in many industrial systems and in understanding many biological phenomena is represented by the stability of equilibrium states in hydro-elasticity [17, 26, 27, 30]. Essentially, one is interested in whether or not resulting perturbations grow or decay when an equilibrium is slightly disturbed. The strategy is to obtain a linearized problem describing the perturbations at first-order, then study the spectral problem associated with this linearization [31]. For fluid-elasticity interactions, complexity arises in particular from large structural displacements and the fact that the position of the common (deformed) interface is not known in advance. This challenge has been solved by transporting the coupled equations to a fixed reference configuration [23, 37, 40, 44, 45] or by using transpiration techniques [38, 46, 47, 51, 24, 26, 27, 28] which are based on modifications of the interface boundary condition. As mentioned above, the authors in [12] investigated a coupled fluid-structure interaction of an incompressible Newtonian fluid and the equations of elastodynamics, subject to small-size perturbations of the distributed forces acting on the system. Unlike earlier approaches,

the authors used shape optimization techniques which incorporate the geometry of the problem in the analysis by linearizing the dynamics equations and the moving boundary concomitantly. This 'total linearization' yields new terms (involving the curvature and the velocity of the interface) in the matching of normal stresses and velocities on the interface that do not appear in the classical linear Stokes flow and linear elasticity coupling.

The well-posedness analysis for the new linearization [12] was addressed in [9]. The "new" first-order terms (2.7) present in the Neumann boundary condition for the elastic solid result in an oblique derivative problem. The hyperbolic component of the dynamics loses regularity under an oblique-derivative condition, even when the coefficients of those first-order terms are relatively very small. This phenomenon has been well-known even in scalar hyperbolic equations [42, 14, 52]. Hence the additional terms appearing in the linearization cannot be trivially handled as mere perturbations of the elastic component even if we impose a size condition on them; with the smallness assumption in place the regularity issues still have to be addressed. In [9] the well-posedness of this model is investigated via semigroup theory in the special case when the linearization is performed near a sufficiently smooth *low-velocity steady-state regime*. In order to accommodate the new terms present on the common interface (in both the velocity and normal stress tensors matchings), the authors (i) derived the basic elliptic theory for the associated elastic boundary value problem, (ii) constructed a suitable equivalent inner product on the fluid-structure state space that accommodated not only the specifics of the hyperbolic component, but also the new perturbed velocity matching condition on the interface, (iii) identified appropriate conditions that preserved the smoothness of the flow under an oblique derivative boundary data in the elasticity system, by taking advantage of the diffusion in the fluid, and (iv) found a suitable maximal dissipative bounded

perturbation of the evolution operator. The maximality result for the evolution generator was demonstrated using a variational approach and the Babuška-Brezzi theorem following the strategy in [2]. This argument also provides a convenient framework for a numerical study of this system via the finite element method. The Babuška-Brezzi approach eliminates the need for divergence-free fluid basis functions in FEM. Moreover, once the algorithm to solve the maximality system is in place, one can use it to solve the dynamical problem with prescribed Cauchy data by using the classical formula for the flow exponential ([49, Thm. 8.3, p. 33]) and corresponding implicit schemes.

One of the traits of the linear semigroup flow approach is that the solution lives in the associated "finite-energy" state space associated in this case to elastodynamics and Stokes flow. This regularity is much weaker than employed in the study of fully nonlinear models. For example, [19] takes advantage of five additional orders of differentiability in the elastic displacement initial data (total of H^6) on top of the finite energy H^1 of solutions to isotropic elasticity. Of course, the gain partly follows from the assertion that linearization is performed around a *smooth solution* to the nonlinear problem in the first place, such as linearization near equilibrium. The analysis in [9], however, shows that if the linearization takes into account geometry of the domain and is performed near a nonzero steady regime then existence of finite-energy solutions likely requires a sufficiently large viscosity—this aspect was irrelevant in the simple classical Stokes-elasticity coupling. The analytic study in [9] does not not provide precise quantitative estimates on the relative parameter sizes, and the current paper aims at investigating this aspect further.

1.1 Goals and Challenges

I. Our first goal is to numerically investigate the linearized model in [12] and study the

effect of the new, when compared to the classical model, trace terms on the discretization. In particular, we numerically examine:

1. how the ellipticity constants and matrix condition numbers in the associated discretized system are affected by the magnitude of the boundary terms present in the matchings of the normal stress tensors and of the velocities on the common boundary (see (2.7)–(2.8) below).

2. whether the size of the boundary terms may affect the convergence rates in the numerical scheme.

The structure of the interface conditions ((2.7) and (2.8) below) is quite involved and the coefficients depend on the exact solution to the associated steady-regime problem around which the linearization is performed. A direct investigation would thus require an exact solution to a nonlinear hydro-elastic system. In addition, elements with curved boundaries would be needed to consider the curvature terms. We can partly circumvent these difficulties by noting that the semigroup-wellposedness analysis [9] critically depends only on the following properties of the linearized system:

(i) The norms (in appropriate spaces) of the fluid velocity, the pressure, and of the deformation in the steady regime around which the linearization is performed, and the size of the fluid viscosity relative to these norms.

(ii) The fact that the additional terms (when compared to classical coupling of Stokes and linear elastodynamics) on the interface can be decomposed into zero-order and first-order *tangential* derivatives only, without normal components.

(iii) The regularity and magnitude of the variable coefficients (accounting for the geometry of the common boundary in the interaction) of the terms on the interface.

Based on these observations, we consider a modified system which admits simplified interface conditions and piecewise-flat geometry. At the same time, this model partly recreates the original dynamics by incorporating zero and first-order perturbations on the interface. Thus a large-curvature scenario is simulated by directly adjusting the size of the coefficients of certain trace terms.

The system is numerically implemented using a finite element scheme with Taylor-Hood elements and illustrates optimal convergence rates [7] for the considered analytic test-solutions. The dependence of the numerical system on the size of the parameters specific to this model is then investigated.

II. Secondly, we investigate how the new linear model in [9] compares to classical linear coupling of fluid and elasticity with respect to stabilization properties. The coupling between linear elasticity and the Stokes system has been investigated in a recent series of papers by Avalos and Triggiani (in [1] the authors studied spectral properties of the semigroup generator; [2], by Avalos and Dvorak, provides an alternative proof of the maximality argument; [4] addresses semigroup well-posedness of a damped model; and [5] treats the backward uniqueness problem). The least amount of damping required for stabilization of the Stokes-elasticity model is the boundary interface feedback. The uniform exponential stability was demonstrated in [3]. Such a feedback represents a refinement accounting for the slip condition on the interface that dissipates energy through friction [54, p. 240]. From an operator theory viewpoint, the corresponding boundary term shifts the generator spectrum to the left, away from the imaginary axis. In comparison, we work with the model obtained in [9] and we demonstrate that a "suitably damped" fluid-structure interaction model, linearized around a steady regime is exponentially stable. Unlike the Stokes-elasticity scenario, the stability in this case is contingent on the presence of viscous dissipation in the elastic solid itself, similar to the analysis provided in

[41]. Moreover, the dissipation through the interface naturally competes with the energy of the steady regime, hence the stability property may be lost if the steady regime's energy is too high. This interplay of forces leads to interesting correlations between the stability of the system and the relative values of the physical parameters as presented in the discussion below.

1.2 Notation

The following notation, following the conventions used in [8], will be used throughout this work.

- $\mathbb{M}^{m\times n}$ denotes the real $m \times n$ matrices, and $\mathbb{M}^m = \mathbb{M}^{m\times m}$.
- A^* denotes the (conjugate) transpose of the matrix A.
- $A..B = \mathrm{Tr}(A^* \cdot B) \in \mathbb{R}$ is the Frobenius product in $\mathbb{M}^2$ or $\mathbb{M}^3$ depending on the size of the matrices A and B.
- $(Df(a))_{ij} = \partial_j f_i(a) \in \mathbb{M}^3$ is the Jacobi matrix at $a \in X$ of any vector field $f = (f_i) : X \subset \mathbb{R}^3 \to \mathbb{R}^3$.
- If T is a 2-tensor, then DT is a 3-tensor whose contraction with a vector gives

  $$[(DT)e]_{ij} = (\partial_k T_{ij})e_k \,.$$

  For a given vector n,

  $$\frac{\partial T}{\partial n} = (DTn) \cdot n$$

- $\mathrm{div}\, f(a) = \partial_i f_i \in \mathbb{R}$ is the divergence of $f : X \subset \mathbb{R}^3 \to \mathbb{R}^3$ at $a \in X$.

- $\operatorname{Div} T(a) = \partial_j T_{ij} e_i \in \mathbb{R}^3$ is the divergence of any 2nd-order tensor field $T = (T_{ij}) : X \subset \mathbb{R}^3 \to \mathbb{M}^3$ at $a \in X$.

- $d_\Omega(x) = \begin{cases} \inf_{y\in\Omega} |y - x| & \Omega \neq \emptyset \\ \infty & \Omega = \emptyset \end{cases}$ is the distance function from a point x to the set $\Omega \subset \mathbb{R}^k$.

- $b_\Omega(x) = d_\Omega(x) - d_{\Omega^c}(x)$, for $x \in \mathbb{R}^k$ is the oriented distance function from x to domain $\Omega \subset \mathbb{R}^k$. Thus $\Delta b_\Omega = \operatorname{Tr}(D^2 b_\Omega)$ is the additive curvature of $\Gamma = \partial\Omega$, and $D^2 b_\Omega$ is the matrix of curvatures of Γ.

Also note the following bilinear forms, norms and duality pairings:

- The bilinear form $(\cdot,\cdot)_X$ indicates the inner product on a given Hilbert space X.

- Product spaces $[X]^k$ are abbreviated with bold $\mathbf{X}$. Likewise the dual $[X']^k \cong ([X]^k)'$ will be denoted $\mathbf{X}'$. The value of k will be clear from the context.

- $\|\cdot\|_X$ will denote the norm on X.

- For a Lipschitz compact boundary-less manifold $\mathcal{M}$, let $\{\cdot,\cdot\}_{\frac{1}{2},\mathcal{M}}$ denote the duality pairing between the Sobolev spaces $\mathbf{H}^{-1/2}(\mathcal{M})$ and $\mathbf{H}^{1/2}(\mathcal{M})$.

1.3 Outline

The remainder of this work proceeds as follows:

- Chapter 2 introduces the total linearization obtained by the authors in [12]. It will also highlight the important aspects of the well-posedness argument for this linearization found in [9].

- Chapter 3 introduces a prototype system as a substitute for the original total linearization. It then constructs and tests a finite element method on this prototype problem. Finally, it investigates the sensitivity of parameters in the system with respect to well-posedness.

- Chapter 4 shows the feedback stability of the total linearization and contrasts with the argument found in [3].

- Chapter 5 provides a summary of the observations found in the preceeding chapters as well as avenues for future work.

- The appendix contains the codes used in the numerical analysis found in Chapter 3.

Chapter 2

Total Linearization

In order to understand the numerical sensitivity analysis in Chapter 3, it is important to understand the contrast between the classical Stokes-linear elasticity system and the total linearization found in [12] as well as the underlying well-posedness argument developed in [9] for this new linearization. The well-posedness argument adapts the argument found in [2] to the new challenges presented by the total linearization.

2.1 PDE Model

2.1.1 Nonlinear PDE System

Let $\mathcal{D} \subset \mathbb{R}^3$ be an open, bounded, and compact domain with fixed boundary $\delta\mathcal{D} = \Gamma_f$. $\mathcal{D}$ contains an elastic solid with domain Ω which is surrounded by a fluid domain $\Omega_f \subset \mathcal{D}$. The union of these non-overlapping subdomains is the full domain $\mathcal{D} = \overline{\Omega} \cup \Omega_f$. Furthermore, the interface between Ω and Ω_f is given by $\Gamma = \delta\Omega$. A sample two-dimensional cross section can be seen in Figure 2.1.

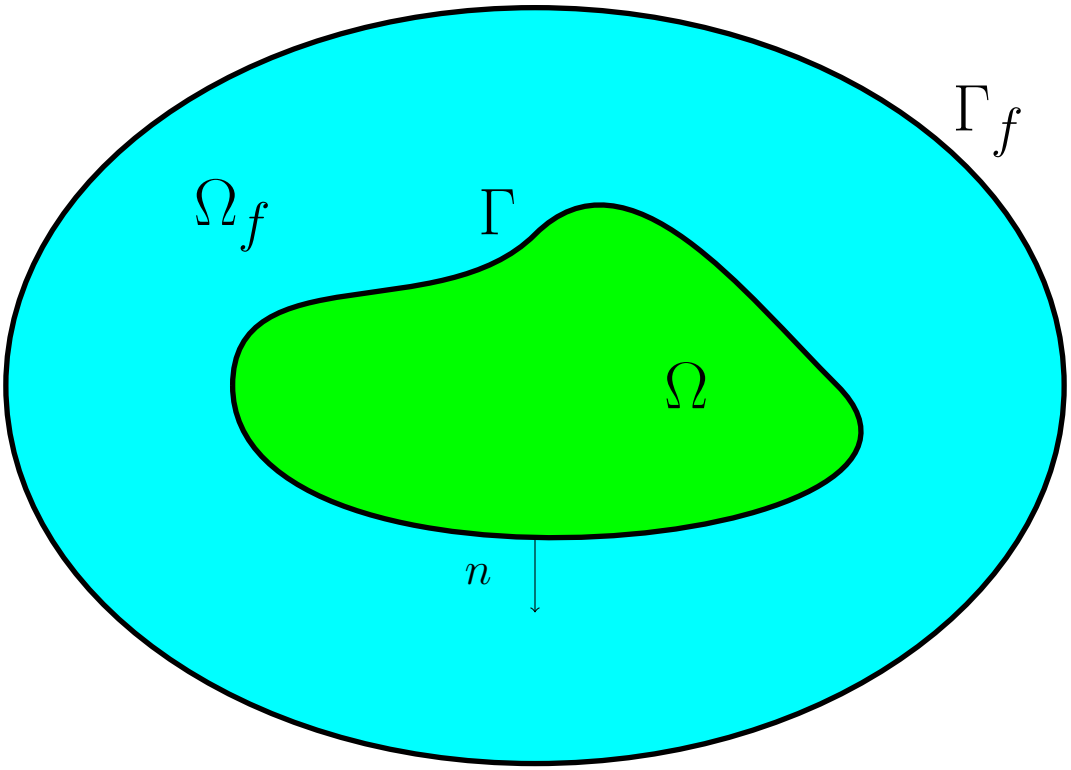

Figure 2.1: A sample two-dimensional geometry for a fluid-structure interaction problem in steady regime

This work assumes that the fluid under consideration is viscous, Newtonian, homogenous, and incompressible. Furthermore, the fluid satisfies the no-slip conditions on the outer boundary Γ_f. The fluid is characterized by its velocity w and pressure p. In the

steady state case, the PDE model is given by

$$\begin{cases} -\nu\,\Delta w \,+\, Dw\cdot w \,+\, \nabla p \,=\, v\Big|_{\Omega_f} & \text{on } \ \Omega_f \\ \text{div } \ w = 0 & \text{on } \ \Omega_f \\ w = 0 & \text{on } \ \Gamma_f \\ -\operatorname{Div}\mathcal{T} = v\Big|_{\Omega} & \text{on } \ \Omega \\ \mathcal{T}n = \sigma(p,w)n & \text{on } \ \Gamma \\ \varphi = I_{\Gamma_f} & \text{on } \ \Gamma_f. \end{cases} \tag{2.1}$$

And the non-steady state system is given by

$$\begin{cases} w_t - \nu\,\Delta w \,+\, Dw\cdot w \,+\, \nabla p \,=\, v\Big|_{\Omega_f(t)} & \text{on } \ \Omega_f(t) \\ \text{div } \ w = 0 & \text{on } \ \Omega_f(t) \\ w = 0 & \text{on } \ \Gamma_f \\ \varphi_{tt}\circ\varphi^{-1} - [J(\varphi)\circ\varphi^{-1}]\operatorname{Div}\mathcal{T} = v\Big|_{\Omega(t)} & \text{on } \ \Omega(t) \\ w\circ\varphi = \varphi_t & \text{on } \ \Gamma(t) \\ \mathcal{T}\cdot n = \sigma(p,w)n & \text{on } \ \Gamma(t) \\ \varphi = I_{\Gamma_f} & \text{on } \ \Gamma_f. \end{cases} \tag{2.2}$$

In systems (2.1) and (2.2), we have the following:

- The fluid dynamics satisfy the Navier-Stokes equations with no-slip conditions on the outer boundary Γ_f. The viscosity of the fluid is $\nu > 0$, and the fluid strain tensor is given by the symmetrized gradient, $\varepsilon(w) = \frac{1}{2}(Dw + (Dw)^*)$. Recall that

Dw is the matrix of partial derivatives for the fluid velocity w.

- The domain of the elastic solid Ω, referred to henceforth as the solid domain, is characterized according to a mapping φ acting in a fixed reference domain (Lagrangian formulation). The evolution of the fluid domain Ω_f is seen via the structural deformation through the boundary interface Γ. Let $\widetilde{\Omega} \subset \mathcal{D}$ be a reference configuration for the solid domain. Further, let $\widetilde{\Gamma} = \partial\widetilde{\Omega}$ be its boundary. Then $\widetilde{\Omega}_f = \mathcal{D} \setminus \overline{\widetilde{\Omega}}$ can be taken as the reference configuration for the fluid domain (again, with $\widetilde{\Gamma}$ as the boundary interface between the two domains). The full domain $\mathcal{D}$ can be characterized by a smooth, injective, orientation-preserving map $\varphi : \overline{\mathcal{D}} \to \overline{\mathcal{D}}$, $x \mapsto \varphi(x)$. If $x \in \widetilde{\Omega}$, $\varphi(x)$ represents the position of the material point x. In the case where $x \in \widetilde{\Omega}_f$, $\varphi(x)$ is defined to be an arbitrary extension of the restriction of $\varphi|_{\widetilde{\Gamma}}$ that preserves the bounary Γ_f. That is to say, $\varphi = \mathrm{Id}$ on Γ_f. Define $J(\varphi) = \det(D\varphi)$ to be the Jacobian of the deformation mapping φ. The orientation-preserving property of φ gives us that $J(\varphi) > 0$.

- The fluid is coupled with the St. Venant-Kirchhoff equations which model large displacement/small deformation elasticity. The equations of elasticity are writen on the deformed configuration Ω using the Cauchy stress tensor $\mathcal{T}$:

$$\mathcal{T} = [J(\varphi)^{-1}\mathcal{P} \cdot (D\varphi)^*] \circ \varphi^{-1} \,.$$

  The Cauchy stress tensor is given in terms of the Green-St. Venant nonlinear strain tensor $\sigma(\varphi)$ and the Piola transform $\mathcal{P}$, given by

$$\mathcal{P} = D\varphi[\lambda_e \operatorname{Tr}(\sigma(\varphi))I + 2\mu_e\sigma(\varphi)] \qquad \text{and} \qquad \sigma(\varphi) = \frac{1}{2}[(D\varphi)^* D\varphi - I]$$

where λ_e and μ_e are the Lamé parameters of the material. Here the "e" subscripts refer to the elasticity and are used to differentiate the Lamé parameters from the resolvant value λ used later in this work. This leads to the matching of the normal stress tensors $\mathcal{T}n$ and $\sigma_f(p, w)n$ on the interface Γ using

$$\sigma_f(p, w) = 2\nu\varepsilon(w) - pI\,,$$

where n is the unit outward normal vector with respect to Ω.

For the steady regime (2.1), well-posedness analysis appears in [33] and [60]. Similarly, well-posedness in the non-steady regime case (2.2) can be found in [19]. Because of this, we assume that the nonlinear systems are well-posed and possess sufficient regularity to take advantage of the existence result from [9], discussed below, for the total linearization.

2.1.2 Perturbed PDE System

The next step in deriving the total linearization is to perturb the nonlinear system (2.2). This is done in the forcing term v on the right hand side of the system. The forcing term v is replaced by the perturbation $v_s(x, t)$ which depends linearly on parameter $s \in [0, s_0]$ for some small s_0 given $v'(x, t)$.

$$v_s(x, t) = v(x) + sv'(x, t)$$

This type of perturbation arises in optimal control problems, and the case when the state of the entire hydro-elastic system is controlled from the fluid domain or its outer layer is of particular interest.

The resulting evolution problem is non-cylindrical in time since the space domains

Ω_f and Ω change as the time parameter t changes. The functions (φ_s, w_s, p_s) solve the perturbed system (2.3).

$$
\begin{cases}
\dfrac{\partial w_s}{\partial t} - \nu\Delta w_s + Dw_s \cdot w_s + \nabla p_s = v_s\big|_{\Omega_f^s(t)} & \Omega_f^s(t) \\[2ex]
\operatorname{div} w_s = 0 & \Omega_f^s(t) \\[2ex]
\left(\dfrac{\partial^2}{\partial t^2}\varphi_s\right) \circ \varphi_s^{-1} - [J_s(t) \circ \varphi_s^{-1}] \operatorname{Div} \mathcal{T}_s = v_s\big|_{\Omega^s(t)} & \Omega^s(t) = \varphi_s(t)(\widetilde{\Omega}) \\[2ex]
w_s = \dfrac{\partial}{\partial t}\varphi_s \circ \varphi_s^{-1} & \Gamma^s(t) = \varphi_s(t)(\widetilde{\Gamma}) \\[2ex]
\mathcal{T}_s \cdot n_s(t) = (2\nu\varepsilon(w_s) - p_s I) n_s(t) & \Gamma^s(t) \\[2ex]
w_s = 0 & \Gamma_f \\[2ex]
\varphi_s(\cdot, 0) = \varphi^0, \dfrac{\partial \varphi_s}{\partial t}(\cdot, 0) = \varphi^1, w_s(\cdot, 0) = w^0, p_s(\cdot, 0) = p^0 &
\end{cases}
\tag{2.3}
$$

where $n_s(t)$ is the unit normal exterior to the solid domain $\Omega^s(t)$. As before, $J_s(t)$ is the Jacobian of the deformation $\varphi_s(t)$. $\mathcal{T}_s$ and $\mathcal{P}_s$ are given by

$$
\begin{aligned}
\mathcal{T}_s &= [(1/J(\varphi_s))\mathcal{P}_s \cdot (D\varphi)^*] \circ \varphi_s^{-1} \\
\mathcal{P}_s &= D\varphi_s[\lambda_e \operatorname{Tr}(\sigma_e(\varphi_s))I + 2\mu_e \sigma_e(\varphi_s)] \,.
\end{aligned}
$$

The strategy here is to differentiate system (2.3) with respect to parameter s at $s = 0$ about a solution (w, p, φ) to the steady regime system (2.1) using shape calculus techniques. The resulting linearized system for the unknown s-derivatives of w_s, p_s, and φ_s–the total linearization–plays a central role in the remainder of this work.

2.1.3 Total Linearization

In order to fully describe the total linearization, a bit more notation is required. Below, the prime " $'$ " notation is used to indicate derivatives in the s parameter evaluated at $s = 0$, i.e.,

$$\varphi' = \frac{\partial}{\partial s}\varphi_s(t)\Big|_{s=0}, \quad w' = \frac{\partial}{\partial s}w_s(t)\Big|_{s=0}, \quad \text{and} \quad p' = \frac{\partial}{\partial s}p_s(t)\Big|_{s=0}.$$

Next, define the elastic displacement vector (for linearized elasticity on the deformed configuration) as

$$U' = \varphi' \circ \varphi^{-1}.$$

Then, let

$$\Theta = D\varphi \circ \varphi^{-1} = [D(\varphi^{-1})]^{-1}. \tag{2.4}$$

Thus, $D\varphi' \circ \varphi^{-1} = (DU')\Theta$. The nonlinear strain tensor associated with φ is given by

$$\sigma(\varphi) \circ \varphi^{-1} = \frac{1}{2}[\Theta^*\Theta - I],$$

and the linearized strain associated with U' can then be written as

$$\begin{aligned} \sigma'(\varphi) \circ \varphi^{-1} &= \frac{1}{2}[\Theta^*(DU')^*\Theta + \Theta^*(DU')\Theta] \\ &= \frac{1}{2}[\Theta^*(DU')\Theta + (\Theta^*(DU')\Theta)^*]. \end{aligned}$$

If we let $\overline{DU'} = \Theta^*(DU')\Theta$, then we can write the strain as the symmetrized gradient conjugated by Θ.

$$\mathcal{E}(U') = \sigma'(\varphi) \circ \varphi^{-1} = \frac{1}{2}[\overline{DU'} + (\overline{DU'})^*] \tag{2.5}$$

Then, since $\mathrm{Tr}(\sigma'(\varphi)\circ\varphi^{-1}) = \mathrm{Tr}(\Theta^*(DU')\Theta)$, we have

$$\begin{aligned}\Sigma(\sigma'(\varphi)\circ\varphi^{-1}) &= \lambda_e \,\mathrm{Tr}(\sigma'(\varphi)\circ\varphi^{-1})I + 2\mu_e(\sigma'(\varphi)\circ\varphi^{-1}) \\ &= \lambda_e \,\mathrm{Tr}(\overline{DU'})I + \mu_e[\overline{DU'} + (\overline{DU'})^*] \\ &= \Sigma(\mathcal{E}(U'))\,.\end{aligned}$$

Note the similarity to the Piola transform $\mathcal{P}$ above.

Furthermore, define

$$\widetilde{\Sigma}(V) = \Theta\Sigma(\mathcal{E}(V))\Theta^*\,.$$

Then the Cauchy stress tensor $\mathcal{T}$ is given by

$$\begin{aligned}\mathcal{T} &= \left(\frac{1}{\det(D\varphi)} D\varphi\cdot\Sigma(\sigma(\varphi))\cdot(D\varphi)^*\right)\circ\varphi^{-1} \\ &= \frac{1}{\det(\Theta)}\Theta\cdot\Sigma\left(\frac{1}{2}[\Theta^*\Theta - I]\right)\Theta^*\,.\end{aligned}$$

This, as calculated in [12], leads to the linearized Cauchy stress tensor $\overline{\mathcal{T}}'$, given by

$$\overline{\mathcal{T}}' = \frac{1}{\det\Theta}\widetilde{\Sigma}(U') + (DU')\mathcal{T}\,. \tag{2.6}$$

With this, we are prepared to state the existence theorem of the total linearization found in [9].

Theorem 2.1.1 (The total linearization around a steady regime [9])**.** *Assume that for all $s\in[0,s_0]$, (2.3) is well-posed on $[0,T]$ such that for $\epsilon>0$ and all $t\in[0,T]$, we have $w_s(t)\in\mathbf{W}^{\frac{11}{4}+\epsilon,4}(\widetilde{\Omega}_f(t))$, $p(t)\in W^{\frac{7}{4}+\epsilon,4}(\widetilde{\Omega}_f(t))$ and $\varphi(t)$ is a $\mathbf{W}^{\frac{15}{4}+\epsilon,4}(\mathcal{D})$ diffeomorphism. Moreover, assume that the system (2.3) is s-differentiable and that the initial condition*

(φ^0, w^0, p^0) *associated with* (2.3) *solves the steady regime problem* (2.1). *Then* (φ', w', p') *satisfy the following* ***cylindrical*** *(i.e. on a time-independent space domain) evolution problem:*

$$\begin{cases} w'_t - \nu\Delta w' + (Dw') \cdot w + Dw \cdot w' + \nabla p' = v'\big|_{\Omega_f} & in \quad \Omega_f \\ \operatorname{div} w' = 0 & in \quad \Omega_f \\ w' + (Dw)U' = U'_t & in \quad \Gamma \\ \dfrac{\partial^2}{\partial t^2} U' - (\det \Theta) \operatorname{Div}\left[\overline{\mathcal{T}}'(U')\right] = v'\big|_{\Omega} & in \quad \Omega \\ \overline{\mathcal{T}}'(U)n = (2\nu\varepsilon(w') - p'I)n + \mathcal{B}(U') & on \quad \Gamma \\ w' = 0 & on \quad \Gamma_f \\ \varphi'(0) = 0, \quad w'(0) = 0, \quad \varphi'_t(0) = 0\,. \end{cases} \tag{2.7}$$

where the boundary operator $\mathcal{B}$ *is given by*

$$\begin{aligned} \mathcal{B}(V) \;=\; & (\mathcal{T} + pI - 2\nu\varepsilon(w)) \cdot [(D_\Gamma V)^* n + (D^2 b_\Omega) V_\Gamma] \\ & -\langle V, n\rangle \left(-\operatorname{Div}_\Gamma(\mathcal{T}) + \left[\tfrac{\partial p}{\partial n} I - 2\nu \tfrac{\partial \varepsilon(w)}{\partial n}\right] \cdot n \right) \\ & +(D\mathcal{T})V \cdot n + \operatorname{div}(V)\mathcal{T} \cdot n - \mathcal{T} \cdot (DV)^* \cdot n\,. \end{aligned} \tag{2.8}$$

Recall from Section 1.2 that the oriented distance function for domain Ω is given by b_Ω. In this case, $D^2 b_\Omega$ in $\mathcal{B}(V)$ yields information on the curvature of the Γ boundary interface, bringing the geometry of the problem more deeply into the analysis.

Furthermore, we have from [9, Prop. 6.2] that the boundary operator $\mathcal{B}(V)$ is comprised of (variable) linear combinations of V and of first-order tangential derivatives of V. This results in an oblique derivative condition on the boundary interface, hence this

boundary term cannot be dismissed as merely a small perturbation of U'. The hyperbolic portion of the dynamics may lose regularity even when the first-order coefficients of the oblique-derivative condition are relatively small. This phenomenon is well-known even in scalar hyperbolic equations [42, 14, 52].

2.2 Well-Posedness of the Total Linearization

The authors in [9] established the well-posedness of (2.7) using semigroup methods. To follow their work, equip the space $L^2(\Omega)$ with the equivalent inner product

$$(V, W)_{\mathbf{L}^2_\Theta(\Omega)} = \int_\Omega \frac{1}{\det \Theta} \langle V, W \rangle \tag{2.9}$$

assuming that $\det(\Theta) > 0$ is bounded below on $\overline{\Omega}$. Similarly, for $H^1(\Omega)$, the space of elastic displacements, define the equivalent inner product

$$(V, W)_{\mathbf{H}^1(\Omega)} = (V, W)_{\mathbf{L}^2_\Theta(\Omega)} + \int_\Omega \frac{1}{\det \Theta} \widetilde{\Sigma}(V) .. DW + \int_\Omega \overline{\mathcal{T}}'(V) .. DW \,. \tag{2.10}$$

The functions U'_t and U' are elements of these spaces respectively.

At this time, we also define the bilinear form $a_S(V, W)$ by

$$a_S(V, W) = (V, W)_{H^1(\Omega)} - \{\mathcal{B}(V), W\}_{\frac{1}{2}, \Gamma} \,. \tag{2.11}$$

Finally, let the fluid velocity space be given by

$$\mathbf{H}^1_{\Gamma_f}(\Omega_f) = \left\{ \varphi \in \mathbf{H}^1(\Omega_f) \middle| \varphi|_{\Gamma_f} = 0 \right\} . \tag{2.12}$$

This leads to the definition of the state space for the full coupled system,

$$\mathscr{H} = \mathbf{H}^1(\Omega) \times \mathbf{L}^2_\Theta(\Omega) \times \mathcal{H}_f$$

where

$$\mathcal{H}_f = \left\{ V \in L^2(\Omega_f) \middle| \operatorname{div}(V) = 0, \langle V, n \rangle|_{\Gamma_f} = 0 \right\}.$$

The authors in [9] equip $\mathscr{H}$ with the following bilinear form:

$$\begin{aligned} \left(\begin{bmatrix} U' \\ V' \\ w' \end{bmatrix}, \begin{bmatrix} \widetilde{U} \\ \widetilde{V} \\ \widetilde{w} \end{bmatrix} \right)_{\mathscr{H}} &= (U', \widetilde{U})_{\mathbf{H}^1(\Omega)} + \left(V' - (D\underline{w})U', \widetilde{V} - (D\underline{w})\widetilde{U} \right)_{\mathbf{L}^2_\Theta(\Omega)} \\ &\quad + (w', \widetilde{w})_{\mathbf{L}^2(\Omega_f)} \end{aligned} \tag{2.13}$$

where $\underline{w}$ is an extension of the steady "linearization velocity" w to the entire control volume domain $\mathcal{D}$. This bilinear form defines an equivalent inner product on $\mathscr{H}$ with assumptions on the smallness of $\underline{w}$ and that $\underline{w}$ is regular enough to admit a small $\|D\underline{w}\|_{L^\infty(\mathcal{D})}$. We will suppress the underline notation on w when no confusion will arise.

Note that the coefficients in the total linearization (2.7) depend on the steady regime solution (w, φ, p) from system (2.1). As such, some constraints on the size and regularity of (w, φ, p) are needed. These assumptions, enforced in [9], are reasonable based on the smoothness of solutions verified in [19].

Assumption 2.2.1. *(Sufficient conditions on the Nonlinear Steady Regime [9, Assumption 3]) Assume that the functions (φ, w, p) satisfying the steady regime problem* (2.1) *obey the following conditions:*

(a) **Regularity Condition:**

(i) w *has an extension to* $\overline{\mathcal{D}}$*:*

$$\underline{w} \in C^1(\overline{\mathcal{D}}) \text{ and } \underline{w} = w \quad \text{in} \quad \overline{\Omega_f}$$

(ii) The coefficients of first-order terms in V *in* $\mathcal{B}(V)$ *defined in* (2.8) *must be multipliers for* $H^{1/2}(\Gamma)$ *topology. Specifically,*

$$\mathcal{T}_{jk}, p, \text{ and }, \delta_{x_i} w_k$$

are multipliers on $H^{1/2}\Gamma$*. The coefficients of zero-order terms in* V *must be in* $L^\infty(\Gamma)$*, i.e.*

$$\delta_{x_i}\mathcal{T}_{jk}, \quad \delta_{x_i} p, \quad \text{and } \delta_{x_i}\delta_{x_j} w_k \quad \text{belong to } L^\infty(\Gamma)$$

Since $H^{1/2}(\Gamma) \overset{dim=2}{\hookrightarrow} L^4(\Gamma)$*, then by [59, Prop. 1.1, p.105 with* $s = \frac{1}{2}$*,* $p = 2$*,* $q_2 = \frac{1}{2}$*,* $q_1 = \infty$*, and* $r_1 = 4$*,* $r_2 = 4$*] to be a multiplier on* $H^{1/2}(\Gamma)$ *it is enough to live in* $W^{\frac{1}{2},4}(\Gamma) \cap L^\infty(\Gamma)$*. So conditions (R-2) are stronger. The space* $L^\infty(\Gamma)$ *contains* $W^{\frac{1}{2}+\varepsilon,4}(\Omega \text{ or } \Omega_f)$*. (Note that by [59, Prop. 1.1], any space* $W^{j+\frac{3}{4}+\varepsilon,4}(\Omega \text{ or } \Omega_f)$*,* $j \in \mathbb{Z}_+$ *forms a Banach algebra). We will therefore assume that*

- $w \in \mathbf{W}^{\frac{11}{4}+\varepsilon,4}(\Omega_f)$
- $p \in \mathbf{W}^{\frac{7}{4}+\varepsilon,4}(\Omega_f)$
- φ *is a* $\mathbf{W}^{\frac{15}{4}+\varepsilon,4}(\Omega)$ *diffeomorphism*

(iii) For the boundary, we assume that $D^2 b_\Omega \in L^\infty(\Gamma)$ *and that* n *is a multiplier with respect to* $H^{1/2}(\Gamma)$ *topology. Since the regularity of* b_Ω *matches that of the*

boundary [20], it suffices to assume that $\Gamma \cup \Gamma_f$ is of class C^2 (and locally on one side of the fluid domain).

(b) **Smallness:** *Let* $\mathbf{R}$ *denote the regularity space* $\mathbf{W}^{\frac{15}{4}+\varepsilon,4}(\Omega)\times\mathbf{W}^{\frac{11}{4}+\varepsilon,4}(\Omega_f)\times\mathbf{W}^{\frac{7}{4}+\varepsilon,4}(\Omega_f)$. *Assume that*

$$r = \|(\varphi - Id, w, p)\|_{\mathbf{R}}$$

is small enough so that there are constants $0 < c = c(r) < 1$ *and* $\omega = \omega(r) \geq 0$, *for which any functions* $\hat{w} \in \mathbf{H}^1(\Omega_f)$, $\hat{U} \in \mathbf{H}^1(\Omega)$, *and* $\hat{V} \in \mathbf{L}^2_\Theta(\Omega)$ *satisfy*

(i) $\{\mathcal{B}(\hat{w}) + (Dw)\hat{U}, \hat{w}\}_{\frac{1}{2},\Gamma} < (c)(2\nu)\int_0^T \int_{\Omega_f} \varepsilon(\hat{w})..\varepsilon(\hat{w}) + C\|\hat{U}\|^2_{\mathbf{H}^1(\Omega)}$

(ii) $2\left|(\hat{V}, (D\underline{w})\hat{U})_{L^2_\Theta(\Omega)}\right| \leq c\left(\|\hat{V}\|^2_{\mathbf{L}^2_\Theta(\Omega)} + \|(D\underline{w})\hat{U}\|^2 + \|\hat{U}\|^2_{\mathbf{H}^1(\Omega)}\right)$

(iii) $\left|\int_\Omega (D\hat{V}\mathcal{T}..D\hat{V}\right| + \left|\{\mathcal{B}(\hat{V}, \hat{V}\}_{\frac{1}{2},\Gamma}\right| \leq c\left(\|\hat{V}\|^2_{\mathbf{L}^2_\Theta(\Omega)} + \int_\Omega \det(\Theta)\widetilde{\Sigma}(\hat{V})..D\hat{V}\right)$

(iv) $(\hat{U}, (D\underline{w})\hat{U})_{\mathbf{H}^1(\Omega)} - ((D\hat{w})w, \hat{w})_{\mathbf{L}^2(\Omega_f)} + \{\mathcal{B}(\hat{U}), \hat{w}\}_{\frac{1}{2},\Gamma}$
$- \left((D\underline{w})\hat{V}, \hat{V} - (D\underline{w})\hat{U}\right)_{\mathbf{L}^2_\Theta(\Omega)} - 2\nu\int_{\Omega_f} \varepsilon(\hat{w})..\varepsilon(\hat{w})$
$\leq \omega\left(\|\hat{U}\|^2_{H^1(\Omega)} + \|\hat{V}\|^2_{\mathbf{L}^2_\Theta(\Omega)}\right)$

The existence of such $c(r)$ and $\omega(r)$ for all small r is justified through Proposition 2.2.2 and Proposition 2.2.1 below.

2.2.1 Resolving Dependence on the Pressure

Typically, the first step in analyzing well-posedness of systems like (2.7) is to resolve the dependence of the dynamics on the pressure p'. The usual Leray projection cannot be applied due to the interface coupling, since the Leray projection requires that the fluid velocity w' be $\mathbf{0}$ on the *entire* fluid boundary. Instead, inspired by [2] and [4], the authors in [9] recast the pressure term as the solution to a second order elliptic boundary value

problem.

$$\begin{cases} -\Delta p' &= (Dw')^*..Dw & \text{in} \quad \Omega_f \\ p' &= \langle 2\nu\varepsilon(w')n, n\rangle + \langle \mathcal{B}(U'), n\rangle - \langle \overline{\mathcal{T}}'(U')n, n\rangle & \text{on} \quad \Gamma \\ \delta_n p' &= \langle \nu\Delta w', n\rangle & \text{on} \quad \Gamma_f \end{cases}$$

The solution p' is found using the following harmonic extension maps.

$$\begin{aligned} p' = \pi'(U', w') &= D_e(\langle 2\nu\varepsilon(w')n, n\rangle + \langle \mathcal{B}(U'), n\rangle - \langle \overline{\mathcal{T}}'(U')n, n\rangle) \\ &\quad + N_f(\langle \nu\Delta w', n\rangle) + A^{-1}((Dw')^*..Dw) \end{aligned}$$

where $A = -\Delta$, $\mathcal{D}(A) = \{f \in H^1(\widetilde{\Omega}_f) \mid Af \in L^2(\widetilde{\Omega}_f) \text{ and } f|_{\Gamma_f} = 0 = \delta_\nu f|_{\Gamma_f}\}$, and the Dirichlet extension $D_e : H^{-1/2}(\widetilde{\Gamma}) \to L^2(\widetilde{\Omega}_f)$ and Neumann extension $N_e : H^{-3/2}(\Gamma_f) \to L^2(\widetilde{\Omega}_f)$ are given by

$$h = D_e(g) \;\Leftrightarrow\; \begin{cases} \Delta h &= 0 & \text{in} \quad \Omega_f \\ h &= g & \text{on} \quad \Gamma \\ \dfrac{\partial h}{\partial n} &= 0 & \text{on} \quad \Gamma_f \end{cases}$$

$$h = N_f(g) \;\Leftrightarrow\; \begin{cases} \Delta h &= 0 & \text{in} \quad \Omega_f \\ h &= 0 & \text{on} \quad \Gamma \\ \dfrac{\partial h}{\partial n} &= g & \text{on} \quad \Gamma_f \end{cases}$$

The solution to the elliptic boundary value problem is denoted by π'.

2.2.2 Define Operator $\mathbb{A}$

After the dependence on the pressure is resolved, it is possible to rewrite system (2.7) as an evolution problem:

$$\frac{d}{dt}y' = \mathbb{A}y' \tag{2.14}$$

on the state space $\mathscr{H}$ where $y' = \begin{bmatrix} U' & V' & w' \end{bmatrix}^*$. The operator $\mathbb{A}$ is given by

$$\mathbb{A} : \begin{bmatrix} U' \\ V' \\ w' \end{bmatrix} \in \mathcal{D}(\mathbb{A}) \subset \mathscr{H} \to \begin{bmatrix} V' \\ -\mathbf{S}(U') \\ \nu\Delta w' - (Dw') \cdot w - \nabla\pi'(w', U') \end{bmatrix} \in \mathscr{H}\,. \tag{2.15}$$

where the elasticity operator S is defined by

$$S(V) = -\det(\Theta)\,\mathrm{Div}[\overline{\mathcal{T}}'(V)] + V \tag{2.16}$$

In [9], the identity was added to S for convenience to ensure the ellipticity of the associated operator; however, semigroup well-posedness is not affected by a bounded perturbation. Here the $(Dw) \cdot w'$ term in the fluid equation was dropped as a small bounded perturbation of w'. Likewise, a U' term was included in the elastic equation for convenience that also has no effect on well-posedness.

If it can be shown that the operator $\mathbb{A}$ generates a C_0-semigroup, then the total linearization is well-posed. With initial conditions $y_0 = \begin{bmatrix} U_0 & V_0 & w_0 \end{bmatrix}^* \in \mathcal{D}(\mathbb{A})$, then the

solution to the total linearization is given by

$$\begin{bmatrix} U'(t) \\ U_t'(t) \\ w'(t) \end{bmatrix} = e^{t\mathbb{A}} \begin{bmatrix} U_0 \\ V_0 \\ w_0 \end{bmatrix}$$

The semigroup exponential notation becomes more clear here when we notice the parallels to an exponential growth or exponential decay differential equation. Section 2.3 outlines the proof justifying the authors' claim in [9] that $\mathbb{A}$ generates a C_0-semigroup to complete the well-posedness argument, and the remainder of this section will define the domain of $\mathbb{A}$ along with additonal observations made in [9] that are needed in the proofs of Section 2.3.

The domain $\mathcal{D}(\mathbb{A})$ of the generator is a subspace of $\mathbf{H}^2(\Omega) \times \mathbf{H}^1(\Omega) \times \mathbf{H}^1(\Omega_f)$ as seen in [9, Sec. 6.3, p.1986-1987]. The exact conditions on $y' = \begin{bmatrix} U' & V' & w' \end{bmatrix}^*$ in order to apply $\mathbb{A}$ are given below.

Definition 2.2.1. *The domain $\mathcal{D}(\mathbb{A})$ of the operator $\mathbb{A}$ given by* (2.15) *is the subset of $\mathbf{H}^2(\Omega) \times \mathbf{H}^1(\Omega) \times \mathbf{H}^1(\Omega_f)$ with the following restrictions. For all $\begin{bmatrix} U' & V' & w' \end{bmatrix}^* \in \mathcal{D}(\mathbb{A})$,*

(i) $(U', V', w') \in \mathscr{H}$

(ii) $V' \in H^1(\Omega)$ *(Consequently, $V'|_\Gamma$ is defined in $L^2(\Gamma)$, in fact $H^{1/2}(\Gamma)$.)*

(iii) $w' \in H^1(\Omega_f)$ *and for some function $\pi_0 \in L^2(\Omega)$*

$$-\Delta w' + \nabla \pi_0 \in L^2(\Omega_f)$$

with $\Delta \pi_0 \in L^2(\Omega_f)$.

(iv) $S(U') \in L^2(\Omega)$ *in the sense that for any* $\Psi \in C_c^\infty(\Omega)$*, there is a* $V \in L^2(\Omega)$ *where*

$$(V, \Psi)_{L^2_\Theta(\Omega)} = (U', \Psi)_{H^1(\Omega)} - \{\mathcal{B}(U'), \Psi\}_{\frac{1}{2},\Gamma}$$

(v) $w' = \mathbf{0}$ *in* $L^2(\Gamma_f)$*.*

(vi) $w' + (D\underline{w})U' = V'$ *in* $H^{1/2}(\Gamma)$*.*

(vii) $\pi_0 = \pi(w', U')$ *and* $\overline{\mathcal{T}}'(U')n - \mathcal{B}(U') = [2\nu\varepsilon(w') - \pi(w', U')I]n$ *in* $H^{-1/2}(\Gamma)$*.*

The following observations from [9] are needed in the proofs given in Section 2.3.

Proposition 2.2.1. *([9, Prop 4.1]) The functional*

$$\hat{w} \mapsto \left(\int_{\Omega_f} \varepsilon(\hat{w}) .. \varepsilon(\hat{w}) \right)^{1/2}$$

defines an equivalent norm on the space $H^1_{\Gamma_f}(\Omega_f)$*.*

Proposition 2.2.2. *(Boundary operator* $\mathcal{B}$ *[9, Prop. 6.2]) The transformation* $V \mapsto \mathcal{B}(V)$ *continuously extends from* $H^2(\Omega) \to L^2(\Gamma)$ *to a bounded linear mapping* $H^1(\Omega) \to H^{-1/2}(\Gamma)$

Remark 2.2.1. *In [9, Prop. 6.2], the proof of Proposition 2.2.2 is based on showing that* $\mathcal{B}(V)$ *is defined in terms of only components of* V *and/or tangential derivatives of components of* V*. This is an important aspect of the numerical analysis in Chapter 3 of this work.*

Proposition 2.2.3. *(Bilinear Form a_S, [9, Prop 6.3]) Consider the linear transformation $H^1(\Omega) \to [H^1(\Omega)]'$:*

$$V \mapsto \big(W \mapsto H^2(\Omega) \text{ for } W \in H^1(\Omega)\big)$$

If V belongs to

$$\mathcal{D}(S) = \left\{ V \in H^2(\Omega) \middle| \overline{\mathcal{T}}'(V)n - \mathcal{B}(V) = 0 \right\}$$

then for any $W \in H^1(\Omega)$

$$a_S(V, W) = (S(V), W)_{L^2_\Theta(\Omega)} \tag{2.17}$$

Moreover, if Assumption 2.2.1 holds, then the bilinear form a_S is continuous elliptic on $H^1(\Omega)$. By m_{a_S} and M_{a_S}, we will denote respectively the ellipticity constant and the continuity modulus for a_S:

$$m_{a_S}\|V\|^2 \leq a_S(V, V) \leq M_{a_S}\|V\|^2$$

for $V \in H^1(\Omega)$.

Proposition 2.2.4. *(Extending $\overline{\mathcal{T}}' \cdot n - \mathcal{B}$ to Functions in $H^1(\Omega)$, [9, Prop. 6.5]) Let $V \in H^1(\Omega)$. Assume $S(V) \in L^2(\Omega)$ in the sense that there is $F \in L^2(\Omega)$ such that for any test function $\Psi \in C_c^\infty(\overline{\Omega})$, we have*

$$a_S(V, \Psi) = (F, \Psi)_{L^2_\Theta(\Omega)} \tag{2.18}$$

Suppose Assumption 2.2.1 holds so that a_S is $H^1(\Omega)$ elliptic. Then $R = \overline{\mathcal{T}}'(V)n - \mathcal{B}(V)$ is uniquely defined as an element of the dual space $H^{-1/2}(\Gamma)$, and we have the identity

$$a_S(V, \cdot) = (S(V), \cdot)_{L^2_\Theta(\Omega)} + \{\overline{\mathcal{T}}'(V)n - \mathcal{B}(V), \cdot\}_{\frac{1}{2},\Gamma} \quad \text{on} \quad H^1(\Omega) \tag{2.19}$$

If $V \in H^2(\Omega)$, then the duality pairing $\{\cdot,\cdot\}_{\frac{1}{2},\Gamma}$ amounts to integration on the boundary Γ.

Proposition 2.2.5. *(Homogeneous S-Dirichlet Problem, [9, Prop 6.7]) Let $G \in H^{-1}(\Omega)$, then for any $\lambda \in \mathbb{R}$, the homogeneous Dirichlet boundary value problem,*

$$\begin{cases} \lambda^2 V_0 + S(V_0) &= G \quad \text{in} \quad \Omega \\ V_0 &= \mathbf{0} \quad \text{in} \quad \Gamma \end{cases} \tag{2.20}$$

has a unique weak solution $V_0 \in H_0^1(\Omega)$ in the sense that

$$\lambda^2 (V_0, W)_{L^2_\Theta(\Omega)} + a_S(V_0, W) = (G, W)_{H^{-1}(\Omega):H_0^1(\Omega)}$$

for all $W \in H_0^1(\Omega)$ with the bilinear form a_S given by (2.11).

Proposition 2.2.6. *(S-Dirichlet Problem, [9, Prop. 6.8]) Let $F \in [H^1(\Omega)]'$ and $R \in H^{1/2}(\Gamma)$. For any $\lambda \in \mathbb{R}$, the boundary value problem*

$$\begin{cases} \lambda^2 V + S(V) &= F \quad \text{in} \quad \Omega \\ V &= R \quad \text{on} \quad \Gamma \end{cases} \tag{2.21}$$

has a unique weak solution $V \in H^1(\Omega)$ in the sense that $V = V_0 + \widetilde{V}$ where $\widetilde{V} \in H^1(\Omega)$ has trace R in $H^{1/2}(\Gamma)$. And $V_0 \in H_0^1(\Omega)$ is the weak solution to the Dirichlet problem (2.20) *with*

$$G(W) = F(W) - a_S(\widetilde{V}, W) - \lambda^2 (\widetilde{V}, W)_{L^2_\Theta(\Omega)}$$

for all $W \in H_0^1(\Omega)$*. Thus by* (2.2.5)*,* V_0 *and* $\widetilde{V}$ *satisfy the variational identity*

$$\begin{aligned} & \lambda^2 (V_0, W)_{L^2_\Theta(\Omega)} + a_S(V_0, W) \\ = \; & (F, W)_{H^{-1}(\Omega):H_0^1(\Omega)} - a_S(\widetilde{V}, W) - \lambda^2 (\widetilde{V}, W)_{L^2_\Theta(\Omega)} \end{aligned} \tag{2.22}$$

for all $W \in H_0^1(\Omega)$*. Equivalently,*

$$\lambda^2 (V, W)_{L^2_\Theta(\Omega)} + a_S(V, W) = (F, W)_{H^{-1}(\Omega):H_0^1(\Omega)} \tag{2.23}$$

for all $W \in H_0^1(\Omega)$*. Moreover, for the ellipticity and continuity constants* m_{a_S} *and* M_{a_S} *of* a_S*, and any* $\delta > 0$*, there are constants* C_p*,* C_e*,* $C_{e,\delta}$ *(detailed in the proof) such that*

$$\begin{aligned} & \lambda^2 C_p \|V_0\|_{L^2_\Theta(\Omega)} + m_{a_S} \|V\|_{H^1(\Omega)} \\ \leq \; & 2\|F\|_{H^{-1}(\Omega)} + \lambda^2 C_p C_{e,\delta} \|R\|_{H^\delta(\Gamma)} + M_{a_S} C_e \|R\|_{H^{1/2}(\Gamma)} \end{aligned} \tag{2.24}$$

Definition 2.2.2. *(S-Dirichlet Extension, [9, Def. 6.9]) Denote by* D_{S,λ^2} *the bilinear solution map* $D_{S,\lambda^2} : (F, R) \mapsto V$ *to the Dirichlet problem given by* (2.21) *in Proposition 2.2.6. According to* (2.24)*, this map is a continuous operator* $[H^1(\Omega)]' \times H^{1/2}(\Gamma) \to H^1(\Omega)$*.*

2.3 Semigroup Generation

The next step in the well-posedness argument is to show that $\mathbb{A}$ generates a C_0-semigroup. Unfortunately, the system

$$\frac{d}{dt}\begin{bmatrix} U' \\ U'_t \\ w' \end{bmatrix} = \mathbb{A}\begin{bmatrix} U' \\ U'_t \\ w' \end{bmatrix} \tag{2.25}$$

does not allow for a direct application of the Lumer-Philips theorem, as was the case in [2].

Theorem 2.3.1. *(Lumer-Phillips [various, e.g. [49, Thm. 4.3, p.14]]) Let $\mathbb{G}$ be a linear operator with dense domain $\mathcal{D}(\mathbb{G})$ in Banach space $\mathbf{X}$. If $\mathbb{G}$ is dissipative and there is a $\lambda_0 > 0$ such that the range, $R(\lambda_0 I - \mathbb{G})$, of $\lambda_0 I - \mathbb{G}$ is $\mathbf{X}$, then $\mathbb{G}$ is the infinitesimal generator of a C_0-semigroup of contractions on $\mathbf{X}$.*

Instead, the authors in [9] identified a bounded perturbation of $\mathbb{A}$ which is maximal dissipative, hence Lumer-Phillips can be applied to the perturbation. This is enough to conclude that $\mathbb{A}$ generates a C_0-semigroup [49, p.12].

2.3.1 ω-Dissipativity of $\mathbb{A}$

In order to apply the Lumer-Phillips theorem 2.3.1 to a bounded perturbation of $\mathbb{A}$, first dissipativity must be shown, i.e. show ω-dissipativity of $\mathbb{A}$ itself. To emphasize the importance of having "large enough" viscosity in the total linearization, we include the following lemma with proof from [9].

Lemma 2.3.1. *(Dissipativity) For viscosity $\nu > 0$, there exists a $r_0 = r_0(\nu) > 0$ and a monotone increasing continuous function $r \mapsto \omega(r)$, $\omega(0) = 0$, such that for any $0 < r \leq$*

r_0 in Assumption 2.2.1 the operator

$$\mathbb{A}_\omega = \mathbb{A} - \omega(r) \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix}$$

is dissipative.

Proof. Recall that

$$\mathbb{A} \begin{bmatrix} U' \\ V' \\ w' \end{bmatrix} = \begin{bmatrix} V' \\ -S(U') \\ \nu\Delta w' - (Dw')w - \nabla\pi(w', U') \end{bmatrix}.$$

Define $y = \begin{bmatrix} U' & V' & w' \end{bmatrix}^* \in \mathcal{D}(\mathbb{A})$. Since the conditions found in Assumption 2.2.1 hold for all small r, it is enough to show the following inequality.

$$(\mathbb{A}y, y)_{\mathscr{H}} \leq \omega(r) \left(\|U'\|^2_{H^1(\Omega)} + \|V'\|^2_{L^2_\Theta(\Omega)} \right)$$

From the inner product on $\mathscr{H}$, we have

$$\begin{aligned} (\mathbb{A}y, y)_{\mathscr{H}} &= (U', V')_{H^1(\Omega)} - (S(U') + (Dw)V', V' - (Dw)U')_{L^2_\Theta(\Omega)} \\ &\quad + (\nu\Delta w' - \nabla\pi', w') - ((Dw')w, w')_{L^2(\Omega_f)}. \end{aligned}$$

Next apply the definition of $a_S(\cdot,\cdot)$ and the identity (2.19).

$$\begin{aligned}(\mathbb{A}y,y)_{\mathscr{H}} &= (U',V')_{H^1(\Omega)} - (U',V')_{H^1(\Omega)} + (U',(Dw)U')_{H^1(\Omega)} \\ &\quad +\{\mathcal{B}(U'),V'-(Dw)U'\}_{\frac{1}{2},\Gamma} + \{\overline{\mathcal{T}}'(U')n - \mathcal{B}(U'),V'-(Dw)U'\}_{\frac{1}{2},\Gamma} \\ &\quad -((Dw)V',V'-(Dw)U')_{L^2_{\Theta}(\Omega)} \\ &\quad +(\nu\Delta w' - \nabla\pi, w')_{L^2(\Omega_f)} - ((Dw')w,w')_{L^2(\Omega_f)}\end{aligned} \tag{2.26}$$

Because $\operatorname{div} w' = 0 \Rightarrow \operatorname{Div}((Dw')^*) = \mathbf{0}$, we have $\nu\Delta w' = 2\nu \operatorname{Div}(\varepsilon(w'))$. Then recalling that n is the unit normal outward with respect to the solid domain

$$\begin{aligned}(\nu\Delta w' - \nabla\pi, w')_{L^2(\Omega_f)} &= (2\nu \operatorname{Div}(\varepsilon(w')) - \nabla\pi, w')_{L^2(\Omega_f)} \\ &= -2\nu\int_{\Omega_f} \varepsilon(w')..\varepsilon(w') - \{(2\nu\varepsilon(w') - \pi I)n, w'\}_{\frac{1}{2},\Gamma}\,.\end{aligned}$$

By the velocity matching condition (vi) and the identification (vii) of 2.2.1, we have

$$\{\overline{\mathcal{T}}'(U')n - \mathcal{B}(U'),V'-(Dw)U'\}_{\frac{1}{2},\Gamma} = \{(2\nu\varepsilon(w') - \pi I)n, w'\}_{\frac{1}{2},\Gamma}\,.$$

Thus we may cancel terms in (2.26). Finally, use the velocity matching condition to expand

$$\{\mathcal{B}(U'),V'-(Dw)U'\}_{\frac{1}{2},\Gamma} = \{\mathcal{B}(U'),w'\}_{\frac{1}{2},\Gamma}\,.$$

This leaves us with

$$\begin{aligned}(\mathbb{A}y,y)_{\mathscr{H}} &= (U',(Dw)U')_{H^1(\Omega)} - ((Dw')w,w')_{L^2(\Omega_f)} \\ &\quad +\{\mathcal{B}(U'),w'\}_{\frac{1}{2},\Gamma} - ((Dw)V',V'-(Dw)U')_{L^2_{\Theta}(\Omega)} \\ &\quad -2\nu\int_{\Omega_f} \varepsilon(w')..\varepsilon(w')\,.\end{aligned}$$

Because $\mathcal{B}$ defines a continuous trace operator on $H^1(\Omega) \to H^{-1/2}(\Gamma)$ using Proposition 2.2.2, we can estimate

$$\begin{aligned}(\mathbb{A}y, y)_{\mathscr{H}} &\leq C(r)\|w'\|^2_{H^1(\Omega_f)} + \omega(r)\left(\|U'\|^2_{H^1(\Omega)} + \|V'\|^2_{L^2_\Theta(\Omega)}\right) \\ &\quad -2\nu \int_{\Omega_f} \varepsilon(w')..\varepsilon(w')\end{aligned}$$

where $C(r)$ and $\omega(r)$ can be chosen continuous monotone increasing in r with $C(0) = 0 = \omega(0)$. We need r to be small enough to ensure that

$$C(r)\|\hat{w}\|_{H^1(\Omega_f)} \leq 2\nu \int_{\Omega_f} \varepsilon(\hat{w})..\varepsilon(\hat{w})$$

for all $\hat{w} \in H^1(\Omega_f)$. Such an r always exists by the property of $C(r)$ and Proposition 2.2.1. This is where the viscosity of the fluid plays its role in counteracting the potentially destabilizing effect of the trace operator $\mathcal{B}$. In contrast with the classical Stokes-elasticity system where any positive viscosity is sufficient, a "large enough" viscosity is needed here. $\square$

2.3.2 Maximality of $\mathbb{A}$

The next step in showing the well-posedness of the total linearization is to show maximality. In the maximality argument, we define a second bounded perturbation of $\mathbb{A}$ for convenience. Given $\omega, \Lambda \geq 0$, suppose

$$\mathbb{A}_{\omega,\Lambda} = \mathbb{A} - \begin{bmatrix} \omega & & \\ & \omega & \\ & & \Lambda \end{bmatrix}$$

Note that our first perturbation, $\mathbb{A}_\omega$, is given by $\mathbb{A}_{\omega,0}$ and that the dissipativity of $\mathbb{A}_{\omega,\Lambda}$ follows from the dissipativity of $\mathbb{A}_\omega$. Following the strategy in [2], the authors in [9] prove the following lemma by reformulating the maximality system into a Babuška-Brezzi system. Again, due to the more complex nature of the coupling in the total linearization, the viscosity of the fluid plays an essential role in verifying the ellipticity of the bilinear form arising in the hypothesis of the Babuška-Brezzi theorem used in the proof.

Lemma 2.3.2. *(Maximality) Suppose Assumption 2.2.1 is in force and the conclusions of lemma 2.3.1 hold for some $\omega_0 \geq 0$, i.e. $\mathbb{A}_\omega$ is dissipative. Then there exists a $\lambda_0 > 0$ such that for every $\lambda \geq \lambda_0$, there is a sufficiently large $\Lambda = \Lambda(\lambda, \omega) \geq \lambda_0$ for which*

$$\lambda I - \mathbb{A}_{\omega,\Lambda}$$

is surjective. Since $\mathbb{A}_{\omega,\Lambda}$ is also dissipative (because $\mathbb{A}_\omega$ is), it is maximal dissipative and generates a C_0-semigroup of contractions on $\mathscr{H}$. Furthermore since $\mathbb{A}$ is a bounded perturbation of $\mathbb{A}_{\omega,\Lambda}$, it can be concluded that $\mathbb{A}$ generates a C_0-semigroup on $\mathscr{H}$ as well.

We include the proof of this lemma from [9] as the remainder of this section.

Surjectivity Problem

Since ω is fixed, for ease of notation, make the following alterations:

- Let $\widetilde{\lambda} = \lambda + \omega$.
- Let $\widetilde{\Lambda} = \Lambda - \omega$.
- Relabel the resulting $\widetilde{\lambda}$ and $\widetilde{\Lambda}$ back to λ and Λ respectively.

With these changes, we must now show that for all large λ and for any $\begin{bmatrix} U_1 & V_1 & w_1 \end{bmatrix}^* \in \mathscr{H}$, there exists a $\begin{bmatrix} U' & V' & w' \end{bmatrix}^* \in \mathcal{D}(\mathbb{A})$ such that for suitably large λ and some Λ depending on λ we have

$$(\lambda I - \mathbb{A}_{0,\Lambda}) \begin{bmatrix} U' \\ V' \\ w' \end{bmatrix} = \begin{bmatrix} U_1 \\ V_1 \\ w_1 \end{bmatrix} . \tag{2.27}$$

We will refer to (2.27) as the maximality system.

Reformulation of the Surjectivity Problem into a Babuška-Brezzi System

Since (2.27) must hold on the Hilbert space $\mathscr{H}$, from the definition of $\mathbb{A}$ and the equivalent inner product 2.13 on $\mathscr{H}$, we obtain:

$$\lambda U' - V' = U_1 \text{ in } H^1(\Omega) \tag{2.28}$$

$$\lambda V' + S(U') - (Dw)(\lambda U' - V') = V_1 - (Dw)U_1 \text{ in } L^2_\Theta(\Omega) \tag{2.29}$$

$$(\lambda + \Lambda)w' - \nu\Delta w' + (Dw')w + \nabla\pi = w_1 \text{ in } \mathcal{H}_f . \tag{2.30}$$

Step 1: [Elliptic Inhomogeneous Dirichlet Problem for U'] Use (2.28) to rewrite V' in (2.29).

$$(\lambda^2 I + S)(U') = V_1 + \lambda U_1 \text{ in } L^2_\Theta(\Omega)$$

Furthermore, V', U', and w' satisfy the velocity matching condition (vi) of 2.2.1, i.e. $w' + (D\underline{w})U' = V'$ on Γ. Rewriting $V' = \lambda U' - U_1$ yields

$$\lambda U' - U_1 = w' + (D\underline{w})U'$$

on Γ. Here we see that λ must be large enough to dominate the (extended into Ω) differential of the steady flow ($D\underline{w}$). Thus we have an elliptic boundary value problem.

$$\begin{cases} (\lambda^2 I + S)(U') &= V_1 + \lambda U_1 & \text{in} \quad \Omega \\ U' &= (\lambda - D\underline{w})^{-1}(w' + U_1) & \text{on} \quad \Gamma \end{cases} \tag{2.31}$$

Invoking [9, Prop. 6.8], this problem has a unique solution $U' \in H^1(\Omega)$ given by the extension mapping D_{S,λ^2}.

$$\begin{aligned} U' &= D_{S,\lambda^2}\left[V_1 + \lambda U_1, (\lambda I - D\underline{w})^{-1}(w' + U_1)\right] \\ &= D_{S,\lambda^2}\left[V_1 + \lambda U_1, (\lambda I - D\underline{w})^{-1}U_1\right] + D_{S,\lambda^2}\left[0, (\lambda I - D\underline{w})^{-1}w'\right] \end{aligned} \tag{2.32}$$

The second equality is based on the bilinearity of D_{S,λ^2}.

This U' satisfies the variational identity (2.23):

$$\lambda^2(U', W)_{L^2_\Theta(\Omega)} + a_S(U', W) = (V_1 + \lambda U_1, W)_{L^2_\Theta(\Omega)}$$

for all $W \in H^1_0(\Omega)$. Then for $F = V_1 + \lambda U_1 - \lambda^2 U' \in L^2_\Theta(\Omega)$

$$S(U') = F \tag{2.33}$$

Hence, by Proposition 2.2.4, U' has a well-defined trace $\overline{\mathcal{T}}'(U')n - \mathcal{B}(U') \in H^{-\frac{1}{2}}(\Gamma)$ uniquely determined by

$$\{\overline{\mathcal{T}}'(U')n - \mathcal{B}(U'), W|_\Gamma\}_{\frac{1}{2},\Gamma} = a_S(U', W) - (S(U'), W)_{L^2_\Theta(\Omega)} \tag{2.34}$$

for all $W \in H^1(\Omega)$. In particular, if we choose W such that in the interior it is given by

the S-harmonic extension of $\varphi \in H^{\frac{1}{2}}(\Gamma)$ via the map in (2.2.2),

$$W = D_{S,\lambda^2}[0, \varphi]\,.$$

Then expanding $S(U')$ via (2.33) and U' through (2.32) yields from (2.34):

$$\begin{aligned}
\{\overline{\mathcal{T}}'(U')n - \mathcal{B}(U'), \varphi\}_{\frac{1}{2},\Gamma} \;=\; & a_S\left(D_{S,\lambda^2}[V_1 + \lambda U_1, (\lambda I - D\underline{w})^{-1}U_1], D_{S,\lambda^2}[0,\varphi]\right) \\
& + a_S\left(D_{S,\lambda^2}[0, (\lambda I - D\underline{w})^{-1}w'], D_{S,\lambda^2}[0,\varphi]\right) \\
& + \lambda^2\left(D_{S,\lambda^2}[V_1 + \lambda U_1, (\lambda I - D\underline{w})^{-1}U_1], D_{S,\lambda^2}[0,\varphi]\right)_{L^2_\Theta(\Omega)} \\
& + \lambda^2\left(D_{S,\lambda^2}[0, (\lambda I - Dw)^{-1}w'], D_{S,\lambda^2}[0,\varphi]\right)_{L^2_\Theta(\Omega)} \\
& - \left(V_1 + \lambda U_1, D_{S,\lambda^2}[0,\varphi]\right)_{L^2_\Theta(\Omega)}
\end{aligned} \tag{2.35}$$

for all $\varphi \in H^{\frac{1}{2}}(\Gamma)$.

Step 2: [Equation Satisfied by the Fluid Component] A priori w' satisfies (2.30). In addition, we have the following regularity for $\pi = \pi_0$:

- $w' \in H^1(\Omega_f)$
- $\pi \in L^2(\Omega_f)$
- $\pi \in H^{-1/2}(\Gamma \cup \Gamma_f)$
- $\frac{\partial w}{\partial n}, \varepsilon(w') \in H^{-1/2}(\Gamma \cup \Gamma_f)$

From this regularity, we may integrate (2.30) by parts against an appropriate test function. Recall that $H^1_{\Gamma_f}(\Omega_f)$ denotes $H^1(\Omega_f)$ functions with zero trace on Γ_f. Let $\varphi \in$

$H^1_{\Gamma_f}(\widetilde{\Omega}_f)$. Then (2.30) gives

$$\begin{aligned} & (\lambda+\Lambda)(w',\varphi)_{L^2(\Omega_f)} + 2\nu \int_{\Omega_f} \varepsilon(w')..\varepsilon(\varphi) + \int_{\Omega_f} (Dw')\underline{w}\cdot\varphi \\ & +2\nu\{\varepsilon(w')n,\varphi\}_{\frac{1}{2},\Gamma} - \int_{\Omega_f} \pi \operatorname{div}(\varphi) - \{\pi n,\varphi\}_{\frac{1}{2},\Gamma} \\ = \; & (w_1,\varphi)_{L^2(\Omega_f)} \end{aligned} \tag{2.36}$$

where n is the unit normal exterior to the *solid domain* Ω.

From the domain condition (vi),

$$\overline{\mathcal{T}}'(U')n - \mathcal{B}(U') = (2\nu\varepsilon(w') - \pi I)n \text{ in } H^{-1/2}(\Gamma)$$

hence,

$$\{\pi n,\varphi\}_{\frac{1}{2},\Gamma} - 2\nu\{\varepsilon(w')n,\varphi\}_{\frac{1}{2},\Gamma} = -\{\overline{\mathcal{T}}'(U')n - \mathcal{B}(U')\}_{\frac{1}{2},\Gamma}.$$

Substituting yields

$$\begin{aligned} & (\lambda+\Lambda)(w',\varphi)_{L^2(\Omega_f)} + 2\nu \int_{\widetilde{\Omega}_f} \varepsilon(w')..\varepsilon(\varphi) \\ & + \int_{\Omega_f} (Dw')w\cdot\varphi + \{\overline{\mathcal{T}}'(U')n - \mathcal{B}(U'),\varphi\}_{\frac{1}{2},\Gamma} - \int_{\Omega_f} \pi \operatorname{div}(\varphi) \\ = \; & (w_1,\varphi)_{L^2(\Omega_f)}. \end{aligned} \tag{2.37}$$

Step 3: [Combining the Equations] Substitute the trace expression (2.35) into (2.37). The condition $\operatorname{div}(w') = 0$ in $L^2(\Omega_f)$ shows that w' and π satisfy the following system for all $\varphi \in H^1_{\Gamma_f}(\Omega_f)$ and $\xi \in L^2(\Omega_f)$:

$$\left\{ \begin{aligned} a_\lambda(w',\varphi) + b(\varphi,\pi) &= F(\varphi) \\ b(w',\xi) &= 0 \end{aligned} \right. \tag{2.38}$$

where

$$
\begin{aligned}
a_\lambda(w',\varphi) &= (\Lambda+\lambda)(w',\varphi)_{L^2(\Omega_f)} + \int_{\Omega_f}(Dw')\cdot w\varphi \\
&\quad + \mathbf{a_S}(D_{\mathbf{S},\lambda^2}[0,(\lambda I - Dw)^{-1}w'], D_{\mathbf{S},\lambda^2}[0,\varphi]) \\
&\quad + \lambda^2(D_{\mathbf{S},\lambda^2}[0,(\lambda I - Dw)^{-1}w'], D_{\mathbf{S},\lambda^2}[0,\varphi])_{L^2_\Theta(\Omega)} + 2\nu\int_{\Omega_f}\varepsilon(w')..\varepsilon(\varphi)
\end{aligned}
\tag{2.39}
$$

$$
b(\varphi,\xi) = -(\xi, \operatorname{div}(\varphi))_{L^2(\Omega_f)}, \tag{2.40}
$$

and

$$
\begin{aligned}
F(\varphi) &= (w_1,\varphi)_{L^2(\Omega_f)} - \mathbf{a_S}(D_{\mathbf{S},\lambda^2}[V_1+\lambda U_1,(\lambda I - Dw)^{-1}U_1], D_{\mathbf{S},\lambda^2}[0,\varphi]) \\
&\quad - \lambda^2(D_{\mathbf{S},\lambda^2}[V_1+\lambda U_1,(\lambda I - Dw)^{-1}U_1], D_{\mathbf{S},\lambda^2}[0,\varphi])_{L^2_\Theta(\Omega)} \\
&\quad + (V_1+\lambda U_1, D_{\mathbf{S},\lambda^2}[0,\varphi])_{L^2_\Theta(\Omega)}\,.
\end{aligned}
\tag{2.41}
$$

Solving the Constrained Variational Problem

Like [2], the authors in [9] invoked the Babuška-Brezzi theorem to guarantee a solution to the constrained variational problem. We include the statement of the Babuška-Brezzi theorem here as found, for example, in [43, p.116].

Theorem 2.3.2. *(Babuška-Brezzi Theorem) Let Σ and Ψ be Hilbert spaces and $a:\Sigma\times\Sigma\to\mathbb{R}$, $b:\Sigma\times\Psi\to\mathbb{R}$ be continuous bilinear forms. Let*

$$
Z = \{\eta\in\Sigma \mid b(\eta,\rho)=0,\ \textit{for every } \rho\in\Psi\}\,.
$$

Assume that $a(\cdot,\cdot)$ is Z-elliptic; i.e., there exists a constant $\alpha>0$ such that

$$
a(\eta,\eta)\geq\alpha\|\eta\|^2_\Sigma
$$

for every $\eta \in \Sigma$. Assume further that there exists a constant $\beta > 0$ such that

$$\sup_{\tau \in \Sigma} \frac{b(\tau, \rho)}{\|\tau\|_\Sigma} \geq \beta \|\rho\|_\Psi$$

for every $\rho \in \Psi$. Then if $\kappa \in \Sigma$ and $\ell \in \Psi$, there exists a unique pair $(\hat{\eta}, \hat{\rho}) \in \Sigma \times \Psi$ such that

$$\begin{cases} a(\hat{\eta}, \tau) + b(\tau, \hat{\rho}) & = (\kappa, \tau)_\Sigma \text{ for every } \tau \in \Sigma \\ b(\hat{\eta}, \rho) & = (\ell, \rho)_\Psi \text{ for every } \rho \in \Psi\,. \end{cases}$$

In order to apply this theorem to (2.38), the required hypotheses must be shown.

Step 1: [Continuity of a_λ] Recall that the operator

$$\psi \mapsto D_{S,\lambda^2}[0, \psi]$$

is continuous from $H^{\frac{1}{2}}(\Gamma)$ to $H^1(\Omega)$. Therefore the mapping

$$\psi \in H^1_{\Gamma_f}(\Omega_f) \mapsto \psi|_\Gamma \in H^{1/2}(\Gamma) \mapsto D_{S,\lambda^2}[0, \psi] \in H^1(\Omega)$$

is continuous. Furthermore by Assumption 2.2.1, $Dw \in C(\overline{\Omega})$, so for $\lambda > \|Dw\|$, the matrix operator $(\lambda I - Dw)^{-1}$ has $C^1(\overline{\Omega})$ coefficients and likewise defines a multiplier on $H^{1/2}(\Gamma)$ (hence by duality, a multiplier on $H^{-1/2}(\Gamma)$). Therefore,

$$\psi \mapsto D_{S,\lambda^2}[0, (\lambda I - Dw)^{-1}\psi]$$

is also continuous. Because a_S is a continuous bilinear form on $H^1(\Omega)$, it follows that a_λ defined in (2.39) is a continuous bilinear form.

Step 2: [Ellipticity of a_λ] We will show that a_λ is elliptic on all of $H^1_{\Gamma_f}(\Omega_f)$, a stronger requirement than is needed for the Babuška-Brezzi theorem. The argument for this is a perturbation result similar to [10].

From Proposition 2.2.1, there always exists a λ large enough so that

$$a_\lambda^{(1)}(w', \varphi) = \lambda(w', \varphi)_{L^2(\Omega_f)} + 2\nu \int_{\Omega_f} \varepsilon(w')..\varepsilon(\varphi) + \int_{\Omega_f} (Dw')w \cdot \varphi \tag{2.42}$$

defines a strongly elliptic bilinear form $H^1(\Omega_f)$ since we may estimate

$$a_\lambda^{(1)}(\varphi, \varphi) \geq (\lambda - C_\delta)\|\varphi\|^2_{L^2(\Omega_f)} + 2\nu \int_{\Omega_f} \varepsilon(\varphi)..\varepsilon(\varphi) - \delta\|\varphi\|_{H^1_{\Gamma_f}(\Omega_f)}$$

which controls the $H^1(\Omega_f)$ norm of φ in the case that δ (which depends on ν) is chosen small enough and $\lambda > C_\delta$. As in the dissipativity argument above, this is another instance of the "large enough" viscosity playing an important role.

Using the ellipticity of $a_\lambda^{(1)}$, we will show that the original form a_λ is elliptic provided that Λ is sufficiently large. If we rewrite

$$a_\lambda = a_\lambda^{(1)} + \Lambda(\cdot, \cdot)_{L^2(\Omega_f)} + a_\lambda^{(2)} \tag{2.43}$$

then it remains to show that the quadratic functional induced by $a_\lambda^{(2)}$, i.e.

$$\begin{aligned} a_\lambda^{(2)}(\psi, \psi) &= a_S\left(D_{S,\lambda^2}[0, (\lambda I - Dw)^{-1}\psi], D_{S,\lambda^2}[0, \psi]\right) \\ &\quad + \lambda^2 \left(D_{S,\lambda^2}[0, (\lambda I - Dw)^{-1}\psi], D_{S,\lambda^2}[0, \psi]\right)_{L^2_\Theta(\Omega_f)} \end{aligned} \tag{2.44}$$

defines a perturbation that does not disturb the ellipticity of (2.43) up to a Λ-multiple of the $L^2(\Omega_f)$ norm.

To simplify the notation in the remainder of the proof of the ellipticity of a_λ, let $\mathfrak{D} = D_{S,\lambda^2}[0,\psi]$, remembering that it still depends implicitly on λ^2 and ψ. Furthermore, let $\mathbf{X}$ represent the space $\mathbb{M}^3(C^1(\overline{\Omega_f}))$. Then whenever $\lambda > \|Dw\|_{\mathbf{X}}$, we have

$$\|(\lambda I - Dw)^{-1}\|_{\mathbf{X}} \leq \frac{1}{\lambda}\sum_{n=0}^{\infty}\frac{1}{\lambda^n}\|Dw\|_{\mathbf{X}}^n\,. \tag{2.45}$$

Thus the norm $\|(\lambda I - Dw)^{-1}\|_{\mathbf{X}}$ is of order λ^{-1}. Next, if $Dw = \mathbf{0}$ in Ω_f, then

$$a^{(2)}_{\lambda,(Dw=\mathbf{0})}(\psi) = \frac{1}{\lambda}a_S(\mathfrak{D},\mathfrak{D}) + \lambda\|\mathfrak{D}\|^2_{L^2_\Theta(\Omega)}$$

defines a non-negative functional according to the ellipticity of a_S. To compare the cases $Dw \neq \mathbf{0}$ and $Dw = \mathbf{0}$, introduce

$$Y = (\lambda I - Dw)^{-1} - \frac{1}{\lambda}I = (\lambda I - Dw)^{-1}\lambda^{-1}Dw\,.$$

Then,

$$\begin{aligned} a^{(2)}_\lambda(\psi,\psi) &= a^{(2)}_{\lambda,(Dw=\mathbf{0})}(\psi,\psi) + a_S(D_{S,\lambda^2}[0,Y\cdot\psi],\mathfrak{D}) \\ &\quad +\lambda^2(D_{S,\lambda^2}[0,Y\cdot\psi],\mathfrak{D})_{L^2_\Theta(\Omega)} \\ a^{(2)}_\lambda(\psi,\psi) &\geq \frac{m_{a_S}}{\lambda}\|\mathfrak{D}\|^2_{H^1(\Omega)} + \lambda\|\mathfrak{D}\|^2_{L^2_\Theta(\Omega)} \\ &\quad - |a_S(D_{S,\lambda^2}[0,Y\cdot\psi],\mathfrak{D})| - \lambda^2\left|(D_{S,\lambda^2}[0,Y\cdot\psi],\mathfrak{D})_{L^2_\Theta(\Omega)}\right|\,. \end{aligned} \tag{2.46}$$

Note that the $\mathbf{X}$-topology defines multipliers on $H^{1/2}(\Gamma)$ [59, Prop. 1.1, p.105], so for some $K > 0$

$$\|Y\cdot V\|_{H^{1/2}(\Omega)} \leq K\|Y\|_{\mathbf{X}}\|V\|_{H^{1/2}(\Gamma)}\,.$$

From (2.45) and if, say, $\lambda > 2\|Dw\|_{\mathbf{X}}$, then

$$\|Y \cdot V\|_{H^{1/2}(\Gamma)} \leq K \frac{1}{\lambda^2} \|Dw\|_{\mathbf{X}} \|V\|_{H^{1/2}(\Gamma)} \,. \tag{2.47}$$

Recall that M_{a_S} denotes the continuity modulus for a_S. Then estimate the negative terms in (2.47).

$$\begin{aligned} & |a_S(D_{S,\lambda^2}[0, Y \cdot \psi], \mathfrak{D})| + \lambda^2 \left| (D_{S,\lambda^2}[0, Y \cdot \psi], \mathfrak{D})_{L^2_\Theta(\Omega)} \right| & (2.48) \\ \leq \; & M_{a_S} \|D_{S,\lambda^2}[0, Y \cdot \psi]\|_{H^1(\Omega)} \|\mathfrak{D}\|_{H^1(\Omega)} + \lambda^2 \|D_{S,\lambda^2}[0, Y \cdot \psi]\|_{L^2_\Theta(\Omega)} \|\mathfrak{D}\|_{L^2_\Theta(\Omega)} & (2.49) \end{aligned}$$

Fix some $\delta \in (0, \frac{1}{2})$. From the property (2.24) of the extension map D_{S,λ^2}, we have for some $K_1, K_2 > 0$

$$\begin{aligned} & \lambda^2 \|D_{S,\lambda^2}[0, Y \cdot \psi]\|_{L^2_\Theta(\Omega)} + \|D_{S,\lambda^2}[0, Y \cdot \psi]\|_{H^1(\Omega)} \\ \leq \; & K_1 \left(\|Y \cdot \psi\|_{H^{1/2}(\Gamma)} + \lambda^2 \|Y \cdot \psi\|_{H^\delta(\Gamma)} \right) \\ \leq \; & K_2 \left(\frac{1}{\lambda^2} \|Dw\|_{\mathbf{X}} \|\psi\|_{H^{1/2}(\Gamma)} + \|Dw\|_{\mathbf{X}} \|\psi\|_{H^\delta(\Gamma)} \right) . \end{aligned}$$

With this inequality, the lower bound (2.47) on $a^{(2)}_\lambda(\psi, \psi)$ and the estimate (2.49) give

us for some constant C such that

$$\begin{aligned}
a_\lambda^{(2)}(\psi,\psi) &\geq \frac{m_{a_S}}{\lambda}\|\mathfrak{D}\|^2_{H^1(\Omega)} - \frac{C}{\lambda^2}\|Dw\|_{\mathbf{X}}\|\psi\|_{H^{1/2}(\Gamma)}\|\mathfrak{D}\|_{H^1(\Omega)} \\
&\quad -C\|Dw\|_{\mathbf{X}}\|\psi\|_{H^\delta(\Gamma)}\|\mathfrak{D}\|_{H^1(\Omega)} \\
&\geq \frac{m_{a_S}}{\lambda}\|\mathfrak{D}\|^2_{H^1(\Omega)} - \frac{C}{2\lambda}\|\psi\|_{H^{1/2}(\Gamma)}\|\mathfrak{D}\|_{H^1(\Omega)} - C\|Dw\|_{\mathbf{X}}\|\psi\|_{H^\delta(\Gamma)}\|\mathfrak{D}\|_{H^1(\Omega)} \\
&\geq \frac{m_{a_S}}{\lambda}\|\mathfrak{D}\|^2_{H^1(\Omega)} - \frac{C^2}{4m_{a_S}\lambda}\|\psi\|^2_{H^{1/2}(\Gamma)} - \frac{m_{a_S}}{2\lambda}\|\mathfrak{D}\|^2_{H^1(\Omega)} \\
&\quad -\frac{C^2\lambda}{2m_{a_S}}\|Dw\|_{\mathbf{X}}\|\psi\|^2_{H^\delta(\Gamma)} - \frac{m_{a_S}}{2\lambda}\|\mathfrak{D}\|^2_{H^1(\Omega)}\,.
\end{aligned}$$

Cancel the terms with $\|\mathfrak{D}\|^2_{H^1(\Omega)}$. Since $\delta < \frac{1}{2}$, the trace embeddings and interpolation give us for any parameter $\eta > 0$:

$$\|\psi\|^2_{H^\delta(\Gamma)} \leq C_\delta \left(\frac{\lambda}{\eta}\right)^{\frac{1+2\delta}{1-2\delta}} \|\psi\|^2_{L^2(\Omega_f)} + \eta\frac{1}{\lambda}\|\psi\|^2_{H^1(\Omega_f)}\,.$$

For instance, with $\delta = \frac{1}{4}$ and for $\widetilde{\eta} > 0$ such that $\eta\|Dw\|^2 \leq \widetilde{\eta}$, we get for some $C' > 0$:

$$a_\lambda^{(2)}(\psi,\psi) \geq -\frac{C'}{\lambda}\|\psi\|^2_{H^1(\Omega_f)} - C'\frac{\lambda^4}{\widetilde{\eta}^3}\|Dw\|^8_{\mathbf{X}}\|\psi\|^2_{L^2(\Omega_f)} - \widetilde{\eta}\|\psi\|^2_{H^1(\Omega_f)}\,.$$

Then for any λ sufficiently large and $\widetilde{\eta}$ small, we have

$$\frac{C'}{\lambda}\|\psi\|^2_{H^1(\Omega_f)} + \widetilde{\eta}\|\psi\|^2_{H^1(\Omega_f)} \leq (1-\epsilon)a_\lambda^{(1)}(\psi,\psi)$$

via the ellipticity of $a_\lambda^{(1)}$. So in order for a_λ to be elliptic, it suffices to choose Λ dependent on λ and the linearization state w such that the coefficient of the $L^2(\Omega_f)$-level term satisfies

$$C'\frac{\lambda^4}{\widetilde{\eta}^3}\|Dw\|^8_{\mathbf{X}} \leq \Lambda\,.$$

Then it follows that

$$a_\lambda(\psi,\psi) \geq \epsilon a_\lambda^{(1)}(\psi,\psi)$$

which is elliptic.

Step 3: [Bilinear Form b] Since they share definitions for b, the proof of the hypothesis on b found below and in [9] is drawn from the one given in [2].

From definition (2.40), it is clear that b is a continuous bilinear form $H^1_{\Gamma_f}(\Omega_f) \times L^2(\Omega_f) \to \mathbb{R}$. For the purposes of the Babuška-Brezzi theorem, we must show that there exists a $\beta > 0$ such that for all $\xi \in L^2(\Omega_f)$,

$$\sup_{\varphi \in H^1_{\Gamma_f}(\Omega_f)} \frac{b(\varphi,\xi)}{\|\varphi\|_{H^1_{\Gamma_f}(\Omega_f)}} \geq \beta \|\xi\|_{L^2(\Omega_f)} \,. \tag{2.50}$$

Let $\eta \in L^2(\Omega_f)$, and consider the following boundary value problem:

$$\begin{cases} \operatorname{div}(\omega) = -\eta & \text{in } \Omega_f \\ \omega|_{\Gamma_f} = 0 & \text{on } \Gamma_f \\ \omega|_{\Gamma} = -\frac{\left(\int_{\Omega_f} \eta\right)}{|\Gamma|} n & \text{on } \Gamma \,. \end{cases} \tag{2.51}$$

From [29, (III.3.31), p.176], we know that (2.51) has a solution $\omega \in H^1_{\Gamma_f}(\Omega_f)$, and there is a $C > 0$ such that

$$\|\nabla \omega\|_{\Omega_f} \leq C \|\eta\|_{L^2(\Omega_f)} \,.$$

For the purposes of (2.50), we may give $H^1_{\Gamma_f}$ an equivalent norm $\|\nabla(\cdot)\|$. For example see

[15, 6.6.-6, p.336]. Thus, for any $\eta \in L^2(\Omega_f)$, we have that

$$\begin{aligned}
\sup_{\varphi \in H^1_{\Gamma_f}(\Omega_f)} \frac{b(\varphi, \eta)}{\|\varphi\|_{H^1_{\Gamma_f}(\Omega_f)}} &= \sup_{\varphi \in H^1_{\Gamma_f}(\Omega_f)} \frac{-(\eta, \operatorname{div}(\varphi))_{L^2(\Omega_f)}}{\|\nabla \varphi\|_{L^2(\Omega_f)}} \\
&\geq \frac{-(\eta, \operatorname{div}(\omega))_{L^2(\Omega_f)}}{\|\nabla \omega\|_{L^2(\Omega_f)}} \\
&= \frac{\|\eta\|^2_{L^2(\Omega_f)}}{\|\nabla \omega\|_{L^2(\Omega_f)}} \\
&\geq \frac{\|\eta\|_{L^2(\Omega_f)}}{C}.
\end{aligned}$$

Therefore the required inf-sup condition is satisfied.

Since the required hypotheses have been shown, the Babuška-Brezzi theorem may be applied to (2.38) to obtain the unique solution $(w', \pi) \in H^1_{\Gamma_f}(\Omega_f) \times L^2(\Omega_f)$.

Chapter 3

Numerical Results

In [9], the authors use semigroup theory to establish the well-posedness of the total linearization in the case where the linearization is performed about a smooth low-velocity steady-state regime. The maximality argument for the semigroup generator was shown using a variational approach and the Babuška-Brezzi theorem following the strategy in [2]. However unlike [2], the well-posedness argument in [9] relied on having a "sufficiently large" viscosity along with "sufficiently small" interface curvature and steady-state regime about which to linearize. Beyond the assertion that such parameter ranges exist, little is given in [9] about quantitative relations among the parameters of the system.

Motivated by and building upon the theory developed in [9], a finite element scheme for approximating solutions of the linearized fluid-elasticity dynamics is implemented. This allows for a numerical investigation of the dependence of the discretized model on the "new" terms present in the total linearization, in contrast with the classical Stokes-linear elasticity system. In particular, the variational framework present in the well-posedness argument of the total linearization is exploited using a finite element method

to numerically investigate how several parameters (quantifying the curvature and velocity of the interface) affect the well-posedness of the representative system. The results are published in [8].

3.1 Prototype Problem

The well-posedness argument of (2.7) relies on assumptions on the size of the viscosity ν and on the magnitude of the "new" terms that arise in the total linearization but not in the classical Stokes-elasticity coupling. Attempting to solve (2.7) directly would involve a domain with non-constant boundary curvature, hence elements with curved edges in a finite element algorithm. Furthermore a direct solution would require an exact solution $[\varphi, w, p]^*$ to the steady regime problem (2.1). This is a costly implementation; however, we may still look into some features of the problem without resorting to those measures. The argument in [9] showed that the exact structure of the coupling in (2.7) is not essential to semigroup well-posedness. The following aspects of the model are important:

- The fact that the components of boundary operator $\mathcal{B}(V)$ defined in (2.8) are (variable) linear combinations of V and of the first-order tangential derivatives of V. This is not obvious from the definition; see Remark 2.2.1.

- The magnitude of the coefficients that perturb the elastic stress tensor in the interior, in particular $\mathcal{T}$ in the second summand of (2.6).

- The size of the coefficients of the zero and first-order terms of V in (2.8). In addition the magnitude of these coefficients relative to the viscosity ν plays a role.

- The norm of the differential Dw, the differential of the velocity in the steady regime, which appears in the velocity matching condition in (2.7).

The semigroup well-posedness relies exclusively on the magnitudes and smoothness of the above terms. For example, the property that w is a steady regime solution did not play a role in the well-posedness proof. In light of this, the following prototype problem (3.1) is designed to capture the above essential features of the model.

Consider the following simplified two-dimensional model on a rectangular domain $\Omega \cup \Omega_f = [0,1] \times [0,1]$.

$$\begin{cases} w'_t - \nu \Delta w' + (Dw') \cdot (\varepsilon_3 w_p) + \nabla p' = v'\big|_{\Omega_f} & \text{in} \quad \Omega_f \\ \operatorname{div} w' = 0 & \text{in} \quad \Omega_f \\ w' + (\varepsilon_3 D w_p) U' = U'_t & \text{in} \quad \Gamma \\ \dfrac{\partial^2}{\partial t^2} U' - \operatorname{Div}\left[\Sigma(\mathcal{E}(U'))\right] = v'\big|_{\Omega} & \text{in} \quad \Omega \\ \Sigma(\mathcal{E}(U')) \cdot n = (-p' I + 2\nu \varepsilon(w')) \cdot n + \varepsilon_1 U' + \varepsilon_2 D_\Gamma U' & \text{on} \quad \Gamma \\ w' = 0 & \text{on} \quad \Gamma_f \\ \varphi'(0) = 0, \quad w'(0) = 0, \quad \varphi'_t(0) = 0\,. \end{cases} \tag{3.1}$$

where $D_\Gamma V(x) = (DV)\tau(x)$ with an oriented tangent field τ on Γ. The following adjustments have been made:

- Since the focus is on the normal stress and velocity matching conditions on the interface, the stress in the interior of the elastic component is modeled by the standard linear isotropic tensor. Namely, $\overline{\mathcal{T}}'(U')$ is replaced with $\Sigma(\mathcal{E}(U'))$.

- Define $w = \varepsilon_3 w_p$ to replace the fluid velocity of the steady state solution. Its magnitude is now controlled by the parameter ε_3. Since the semigroup well-posedness does not depend on the precise structure of this term, we do not require w_p to be

a solution to the steady regime. However, w_p will be chosen to be divergence free. This avoids the need to solve the nonlinear steady regime problem for a suitable fluid velocity.

- The effect of the curvature and of the first-order tangential derivatives in (2.8) is simulated by redefining the boundary operator $\mathcal{B}(V)$ to incorporate such terms explicitly.

$$\mathcal{B}(V) = \varepsilon_1 V + \varepsilon_2 c(x) D_\Gamma V \tag{3.2}$$

  The (positive) sign of the zero-order term, ε_1, is chosen so that it does not contribute to the dissipativity of the uncoupled elastic problem. We will consider choices of $c(x) = 1$ and $c(x) = |x|^2$ in our numerical investigations.

After making these alterations, the maximality system (2.27) becomes

$$\begin{bmatrix} \lambda U' \\ \lambda V' \\ \lambda w' \end{bmatrix} - \begin{bmatrix} V' \\ \mathrm{Div}(\Sigma(\mathcal{E}(U'))) \\ \nu \Delta w' - (Dw') \cdot \varepsilon_3 w_p - \nabla \pi \end{bmatrix} = \begin{bmatrix} U_1 \\ V_1 \\ w_1 \end{bmatrix}. \tag{3.3}$$

3.2 FEM Framework

3.2.1 Theoretical Framework

One of the major benefits of the framework used to prove maximality of the C_0-semigroup generator $\mathbb{A}$ is the allowance for the use of a finite element method (FEM) to numerically approximate the solution of the given fluid-structure system.

The same argument for the maximality of the $\mathbb{A}$ generator also proves that a solution to the finite dimensional maximality problem exists. Consider the finite dimensional

approximating subspaces $V_h = \text{span}\{\varphi_i\}_{i=1}^{n_1}$ of $H^1_{\Gamma_f}(\Omega_f)$ and $\Pi_h = \text{span}\{\eta_i\}_{i=1}^{n_2}$ of $L^2(\Omega_f)$. Then for

$$w_h' = \sum_{i=1}^{n_1} \alpha_i \varphi_i \qquad \text{and} \qquad \pi_h' = \sum_{i=1}^{n_2} \beta_i \eta_i \,,$$

by the Babuška-Brezzi theorem 2.3.2, we are guaranteed a solution for the finite dimensional system

$$\begin{cases} a_\lambda(w_h', \varphi_i) + b(\varphi_i, \pi_h') & = F(\varphi_i) \text{ for all } i = 1, 2, ..., n_1 \\ b(w_h', \eta_i) & = 0 \qquad \text{for all } i = 1, 2, ..., n_2 \end{cases} . \tag{3.4}$$

Translating this system into matrix form, we have

$$\begin{bmatrix} A & B \\ B^* & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} G \\ 0 \end{bmatrix} \tag{3.5}$$

for

$$\begin{aligned} A &= \begin{bmatrix} a_\lambda(\varphi_1, \varphi_1) & \cdots & a_\lambda(\varphi_1, \varphi_{n_1}) \\ \vdots & \ddots & \vdots \\ a_\lambda(\varphi_{n_1}, \varphi_1) & \cdots & a_\lambda(\varphi_{n_1}, \varphi_{n_1}) \end{bmatrix}, \\ B &= \begin{bmatrix} b(\varphi_1, \eta_1) & \cdots & b(\varphi_1, \eta_{n_2}) \\ \vdots & \ddots & \vdots \\ b(\varphi_{n_1}, \eta_1) & \cdots & b(\varphi_{n_1}, \eta_{n_2}) \end{bmatrix} \end{aligned}$$

and

$$G = \begin{bmatrix} F(\varphi_1) \\ \vdots \\ F(\varphi_{n_1}) \end{bmatrix}.$$

In this form, $\alpha = \begin{bmatrix} \alpha_1, \alpha_2, ..., \alpha_{n_1} \end{bmatrix}^*$ and $\beta = \begin{bmatrix} \beta_1, \beta_2, ..., \beta_{n_2} \end{bmatrix}^*$ are coefficient vectors for w'_h and π'_h respectively. Furthermore $a_\lambda(\cdot,\cdot)$, $b(\cdot,\cdot)$, and $F(\cdot)$ are defined in (2.39), (2.40), and (2.41). In particular, we will use a mixed $\{\mathcal{P}_2, \mathcal{P}_1\}$ Taylor-Hood basis element scheme over a triangular mesh with piecewise quadratic basis functions φ_i to approximate the fluid velocity w' and piecewise linear basis functions η_j to approximate the pressure π'.

Note that while the overarching Babuška-Brezzi block matrix is square, the submatrix B is not square since $n_2 < n_1$. This is because of the quadratic basis functions using not only element vertices as nodes but also using the midpoints of the triangular edges as nodes.

Next, for our prototype problem, we will begin with the "first-level" mesh given in Figure 3.1. This mesh, along with its refinements, was created using the open source gmsh software [32]. It partitions the rectangle $\Omega \cup \Omega_f = [0,1] \times [0,1]$. The blue triangles represent elements on the fluid domain Ω_f, and the green triangles are elements on the solid domain Ω. Finer, "higher-level" meshes are developed recursively by connecting the midpoints of each triangular element edge with three line segments to form four new triangular elements. Later in section 3.2.3, we will use the higher-level meshes to examine the convergence rates of our algorithm.

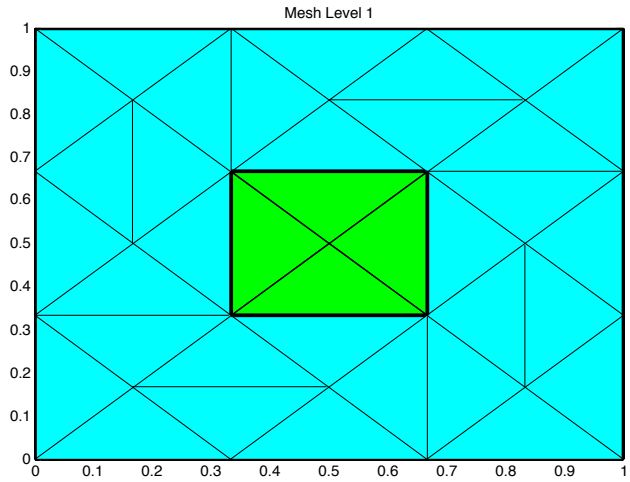

Figure 3.1: "First-level" mesh. Higher level meshes are obtained recursively by connecting the midpoints of the edges for each triangular element with line segments.

To calculate the required integrals in the FEM implementation, we will use a reference triangle $\mathcal{R}$ with ordered vertices $(0,0)$, $(1,0)$, and $(0,1)$. Figure 3.2 shows the ordering used for local nodes on the reference triangle. The standard quadratic basis functions on $\mathcal{R}$ are given in Table 3.1, and the standard linear basis functions are given in Table 3.2.

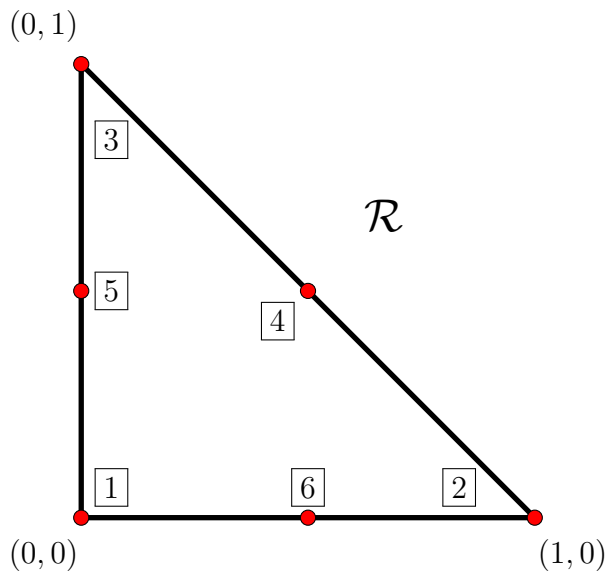

Figure 3.2: Reference Triangle $\mathcal{R}$ with labeled local nodes $\boxed{1}$ through $\boxed{6}$

Table 3.1: Quadratic Local Basis Functions over $\mathcal{R}$

$\varphi_1(x,y)$	$=$	$(1-x-y)(2(1-x-y)-1)$
$\varphi_2(x,y)$	$=$	$x(2x-1)$
$\varphi_3(x,y)$	$=$	$y(2y-1)$
$\varphi_4(x,y)$	$=$	$4xy$
$\varphi_5(x,y)$	$=$	$4(1-x-y)y$
$\varphi_6(x,y)$	$=$	$4(1-x-y)x$

Table 3.2: Linear Local Basis Functions over $\mathcal{R}$

$\eta_1(x,y)$	$=$	xy
$\eta_2(x,y)$	$=$	$1-x$
$\eta_3(x,y)$	$=$	$1-y$

Furthermore, since we are using a reference triangle, we will also use the bijective affine mappings $\mathcal{L}_k : \mathcal{R} \to \mathcal{M}_k$ given by

$$\mathcal{L}_k \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{3.6}$$

that preserve nodes and orientation of elements.

Since the fluid velocity is two-dimensional, the reference basis element will have a total of $6 \times 2 = 12$ local basis functions. These basis functions have the form $\begin{bmatrix} \varphi_k & 0 \end{bmatrix}^*$ or $\begin{bmatrix} 0 & \varphi_k \end{bmatrix}^*$ for $k = 1, 2, ..., 6$ and for φ_k given in Table 3.1. Then, using Gaussian quadrature over the reference triangle $\mathcal{R}$, we may calculate the following exactly:

$$\int_{\mathcal{M}_k} (\varphi_i^{\mathcal{M}_k})^* \varphi_j^{\mathcal{M}_k} = \int_{\mathcal{R}} J \, (\varphi_i^{\mathcal{R}})^* \varphi_j^{\mathcal{R}}$$

where $J = \det(D\mathcal{L}_k)$ and $\varphi_i^{M_k}$ and $\varphi_k^{\mathcal{R}}$ refer to the two dimensional local basis functions on the mesh elements and reference triangle respectively. Similarly we can exactly calculate

integrals of the form

$$\int_{\mathcal{M}_k} (D\varphi_i^{\mathcal{M}_k})..(D\varphi_j^{\mathcal{M}_k}) = \int_{\mathcal{R}} J \ \mathrm{Tr}\left(\left[(D\varphi_i^{\mathcal{R}})(D\mathcal{L}_k)^{-1}\right]^* (D\varphi_j^{\mathcal{R}})(D\mathcal{L}_k)^{-1}\right) .$$

The integrals involving linear basis functions for pressure are calculated similarly. Boundary integrals computed by evaluating similar forms through parameterizations of the element edges in the boundary of the domain mesh.

3.2.2 Implementation of FEM

Extension Problems: There are two main stages of the FEM implementation on the prototype problem (3.1). The first stage is to solve the extension problems $D_{S,\lambda^2}[F,R]$ found in the $a_S(\cdot,\cdot)$ product within the $a_\lambda(\cdot,\cdot)$ product as well as inside the $F(\cdot)$ operator.

Recall that

$$g = D_{S,\lambda^2}[F,R] \qquad \Leftrightarrow \qquad g \text{ solves } \begin{cases} \lambda^2 g + \mathbf{S}(g) = F & \text{on } \Omega \\ g = R & \text{on } \Gamma. \end{cases} \tag{3.7}$$

These extension problems can be solved with a standard finite element method with non-homogeneous Dirichlet boundary conditions. The stiffness matrix for this problem is given by

$$S_{i,j} = \lambda^2 (\varphi_i, \varphi_j)_{L^2_\Theta(\Omega)} + a_S(\varphi_i, \varphi_j) .$$

Note that the size of the extension stiffness matrix S is not equal to the size of the matrices in the Babuška-Brezzi system. This is because the extension problems only involve the elastic domain Ω. On a practical note, this means that a separate "global" node numbering system must be used on the extension problems when compared to the

global node numbering system used in the mesh over the full domain. In the second stage of the implementation, this solid global node numbering system must be translated into the global node numbering system on the full domain.

Assembly of Babuška-Brezzi System: Once a library of coefficient vectors for the extension problems is created, the second stage of the FEM implementation is to build the overarching Babuška-Brezzi system (3.5). The construction of the $B_{i,j} = b(\varphi_i, \eta_j)$ matrix can be done with standard finite element methods; however the $A_{i,j} = a_\lambda(\varphi_i, \varphi_j)$ matrix and $F_i = F(\varphi_i)$ vector both involve the extension problems outlined above. Specifically consider the final terms in the $a_\lambda(\cdot, \cdot)$ product.

$$\begin{aligned} & a_S(D_{S,\lambda^2}[0, (\lambda I - Dw_p)^{-1}\varphi_i, D_{S,\lambda^2}[0, \varphi_j]) \\ + \quad & \lambda^2(D_{S,\lambda^2}[0, (\lambda I - Dw_p)^{-1}\varphi_i, D_{S,\lambda^2}[0, \varphi_j])_{L^2_\Theta(\Omega)} \end{aligned} \tag{3.8}$$

While these terms can be calculated directly by reconstructing the finite element solutions for $D_{S,\lambda^2}[F, R]$, this is not the most efficient method. Instead, suppose u is the coefficient vector of $D_{S,\lambda^2}[0, (\lambda I - Dw_p)^{-1}\varphi_i]$, and suppose v is the coefficient vector of $D_{S,\lambda^2}[0, \varphi_j]$. Then, the terms (3.8) can be calculated by the matrix product $u^* S v$. This property follows from how the stiffness matrix S is defined in the extension problems. The analogous terms in the vector F,

$$\begin{aligned} & a_S(D_{S,\lambda^2}[V_1 + \lambda U_1, (\lambda I - Dw_p)^{-1}U_1], D_{S,\lambda^2}[0, \varphi_i]) \\ + \quad & \lambda^2(D_{S,\lambda^2}[V_1 + \lambda U_1, (\lambda I - Dw_p)^{-1}U_1], D_{S,\lambda^2}[0, \varphi_i])_{L^2_\Theta(\Omega)} \end{aligned}$$

can be calculated similarly.

Once the Babuška-Brezzi system is constructed, we may solve for the coefficient vectors α and β for w_h' and π_h' respectively. However before solving the matrix system, we

must enforce the no-slip conditions on the outer (non-interface) boundary of the domain. This can be done by replacing the rows in the system corresponding to these nodes by the identity in the Babuška-Brezzi matrix and by 0 on the right-hand vector. This will force $w'_h = 0$ on Γ_f as required by the condition on w' in the prototype problem.

3.2.3 Testing the Implementation

Consistency Check: The FEM algorithm described above can be initially tested with an eigenvalue problem on the maximality system. For convenience we set the Lamé parameters $\lambda_e = \mu_e = 1$. Let

$$w = \epsilon_3 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow Dw = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

We want to consider an eigenvalue problem of the form: $\widetilde{\lambda}(\lambda I - \mathbb{A})y' = y'$.

$$(\lambda I - \mathbb{A}) \begin{bmatrix} U' \\ V' \\ w' \end{bmatrix} = \begin{bmatrix} \lambda U' \\ \lambda V' \\ (\lambda + \Lambda)w' \end{bmatrix} - \begin{bmatrix} V' \\ \operatorname{Div} \sigma_f(w', p')(\mathcal{E}(U')) \\ \nu \Delta w' - (Dw') \cdot w - \nabla \pi'(w', U') \end{bmatrix} = \begin{bmatrix} U_1 \\ V_1 \\ w_1 \end{bmatrix} \tag{3.9}$$

Following the example of [2], for $V' = \mathbf{0}$, $w' = \mathbf{0}$, $\pi' = 1$, and U' as the solution to (3.10), we have a solution to (3.9) for eigenvalue $\widetilde{\lambda} = 1$.

$$\begin{cases} S(U') = 0 & \text{on } \Omega \\ \overline{\mathcal{T}}'(U')n - \mathcal{B}(U') = -\pi' n & \text{on } \Gamma \end{cases} \tag{3.10}$$

Again, n is the unit normal vector *outward* to the solid domain. Notice that U' can be

approximated in (3.10) using the same process used when solving the extension problems in the first stage of building the Babuška-Brezzi matrix system. This allows us to feed the approximation for U' into our FEM algorithm for U_1 when $\lambda = 1$ along with $V_1 = \mathbf{0}$. While using this approximation prevents this example from being a true analytic test, we do expect to recover the intended true solution $w' = \mathbf{0}$ and $\pi' = 1$.

Using viscosity $\nu = 1$, the algorithm yields "machine epsilon" in the error as seen in Table 3.3.

Table 3.3: Eigenvalue Consistency Check Errors

Mesh Lvl	No. Fluid Elem.	$\|w'_h - w'\|_{\mathbf{L}^2(\Omega_f)}$	$\|w'_h - w'\|_{\mathbf{H}^1(\Omega_f)}$	$\|\pi'_h - \pi'\|_{L^2(\Omega_f)}$
1	32	$3.4642 * 10^{-17}$	$3.5400 * 10^{-16}$	$8.8056 * 10^{-16}$

Analytic Solution I: The next step is to test the FEM algorithm on closed-form solutions. Unfortunately the form of the elastic stress tensor $S(\cdot)$ and the presence of tangential terms on the boundary interface make it difficult to find an exact analytic solution compatible with our simplified geometry. One way to overcome this obstacle is to consider a case when there is no interaction on the interface.

Again let

$$w = \varepsilon_3 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow Dw = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Now consider

$$U_1 = 10^6 \begin{bmatrix} (x-1/3)^2(x-2/3)^2(y-1/3)^2(y-2/3)^2 \\ 0 \end{bmatrix}.$$

Here the 10^6 factor is included to counter how the eighth degree polynomial is pointwise very small on our domain. More importantly, both U_1 and DU_1 are zero on the interface boundary. We can then use $w_1 = \mathbf{0}$ and compute $V_1 = S(\frac{1}{\lambda}U_1)$. This problem has analytic solution $U' = \frac{1}{\lambda}U_1$, $V' = \mathbf{0}$, $w' = \mathbf{0}$, and $\pi' = 0$.

Testing the algorithm for $\lambda = \Lambda = 2$, viscosity $\nu = 1$, and parameters $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 0.1$, we obtain the error estimates in Table 3.4. The decay exponents for the errors in the table are found in Figure 3.3.

Table 3.4: Analytic Solution I Errors

Mesh Lvl	No. Fluid Elem	$\|w'_h - w'\|_{\mathbf{L}^2(\Omega_f)}$	$\|w'_h - w'\|_{\mathbf{H}^1(\Omega_f)}$	$\|\pi'_h - \pi'\|_{L^2(\Omega_f)}$
1	32	$5.8767 * 10^{-3}$	$1.0924 * 10^{-1}$	$6.4597 * 10^{-2}$
2	128	$4.5561 * 10^{-4}$	$1.6494 * 10^{-2}$	$8.7087 * 10^{-3}$
3	512	$6.4210 * 10^{-5}$	$4.6350 * 10^{-3}$	$1.2419 * 10^{-3}$
4	2048	$6.4039 * 10^{-6}$	$1.0002 * 10^{-3}$	$1.2831 * 10^{-4}$
5	8192	$5.7990 * 10^{-7}$	$1.8994 * 10^{-4}$	$1.2075 * 10^{-5}$

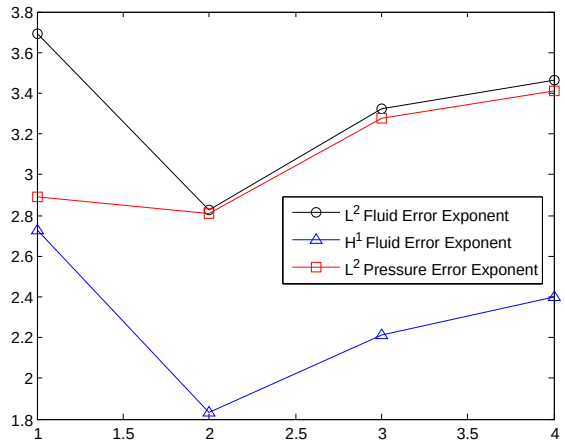

Figure 3.3: Error exponent in the mesh parameter for Analytic Solution I

The convergence rate of approximately 2.4 for the $H^1(\Omega_f)$ norm is slightly better than the predicted rate of 2 for Taylor-Hood elements [7, Thm. 1, p.219]. This scheme also shows slightly improved convergence rates in the $L^2(\Omega_f)$ norm of the solution despite the fact that the fluid domain is polygonal and non-convex [7, Prop. 3, p.222].

Analytic Solution II: We should also test the FEM algorithm with an analytic solution that does not entirely vanish on the interface. For the same $w = \varepsilon_3[1,1]^* \Rightarrow Dw = \mathbf{0}$ along with right hand side data

$$U_1 = \begin{bmatrix} y \\ -x \end{bmatrix} \qquad V_1 = S\left(\frac{1}{\lambda}U_1\right) \qquad w_1 = \mathbf{0}$$

we have analytic solution $U' = \lambda U_1$, $V' = \mathbf{0}$, $w' = \mathbf{0}$ and $\pi' = \varepsilon_2$ from the $\mathcal{B}(\cdot)$ parameters given by (3.11). Notice that to satisfy the boundary interface conditions, $\varepsilon_1 = 0$ must be

enforced in this analytic solution.

For $\lambda = \Lambda = 2$, viscosity $\nu = 1$, and parameters $\varepsilon_1 = 0$ and $\varepsilon_2 = \varepsilon_3 = 0.1$, we recover "machine epsilon" at the first-level mesh as seen in Table 3.5. This immediate convergence is not surprising since the solution consists of low degree polynomials.

Table 3.5: Analytic Solution II Errors

Mesh Lvl	No. Fluid Elem	$\|w_h' - w'\|_{\mathbf{L}^2(\Omega_f)}$	$\|w_h' - w'\|_{\mathbf{H}^1(\Omega_f)}$	$\|\pi_h' - \pi'\|_{L^2(\Omega_f)}$
1	32	$9.3261 * 10^{-17}$	$8.1033 * 10^{-16}$	$1.4496 * 10^{-15}$

3.3 Numerical Sensitivity Analysis

At this point we may numerically investigate the sensitivity of the epsilon parameters introduced in the prototype problem (3.1) with respect to the well-posedness of the system. Recall that these parameters appear in the pseudo-steady regime fluid velocity, $w = \varepsilon_3 w_p$, as well as in the boundary operator $\mathcal{B}(V)$. For the sensitivity analysis, we use the following divergence-free function:

$$w = \varepsilon_3 \begin{bmatrix} (1-y)y \\ (1-x)x \end{bmatrix}.$$

The value of ε_3 simulates the size of the fluid velocity about which the total linearization was centered. The epsilon parameters appear in the boundary operator

$$\mathcal{B}(V) = \varepsilon_1 V + \varepsilon_2 D_\Gamma V \tag{3.11}$$

for $c(x) = 1$ from (3.2). Numerical results below also prompted considering a variable coefficient for the first-order terms. This was done using

$$\mathcal{B}(V) = \varepsilon_1 V + \varepsilon_2 |x|^2 D_\Gamma V \tag{3.12}$$

for $c(x) = |x|^2$.

Recall that originally, components of $\mathcal{B}(V)$ are (variable) linear combinations of V and of the first-order tangential derivatives of V (Remark 2.2.1). In constructing (3.1), $\mathcal{B}(V)$ was defined so that this property was explicit. Thus ε_1 governs the influence of the zero-order terms in $\mathcal{B}(V)$, including the influence of the curvature of the interface. Similarly, ε_2 controls the first-order tangential derivatives of V. Note that in the case where $\varepsilon_i = 0$ for all i, the classical Stokes-elasticity coupling is recovered.

3.3.1 Epsilon Dependence (Constant Coefficient)

The following three sets of plots each explore the sensitivity of the epsilon parameters with regard to the solvability of the Babuška-Brezzi system given in (3.4), hence the well-posedness of the total linearization. First, the ellipticity of the $a_\lambda(\cdot,\cdot)$ operator is observed from the minimum eigenvalue of the matrix $A + A^*$. The ellipticity of $a_\lambda(\cdot,\cdot)$ is necessary to invoke the Babuška-Brezzi theorem justifying well-posedness of the problem. In matrix form, this ellipticity condition requires that for any nonzero $v \in \mathbb{R}^n$, $v^* A v > 0$ which is

equivalent to $v^*Av + v^*A^*v = v^*(A + A^*)v > 0$. The $a_\lambda(\cdot, \cdot)$ operator is not symmetric in general, hence A is not symmetric. This leads to considering the symmetrization $A + A^*$. When $A + A^*$ has a non-positive eigenvalue (i.e. when positive definiteness is lost), $a_\lambda(\cdot, \cdot)$ has lost ellipticity.

In addition to the minimum eigenvalue, the left column of Figures 3.4, 3.5, and 3.6 show log plots of the condition numbers of the A matrix as well as the "full" Babuška-Brezzi matrix given in (3.5). The "spikes" in these plots are indicative of the numerical scheme observing the matrices becoming singular as the ε_i values are varied. Finally, the right column shows the relative change in condition number compared to the $\varepsilon_i = 0$ condition number. In all three figures, a "second-level" mesh with 128 fluid elements and 16 solid elements was used. Furthermore, the resolvent value $\lambda = 1$ was used in all three plots.

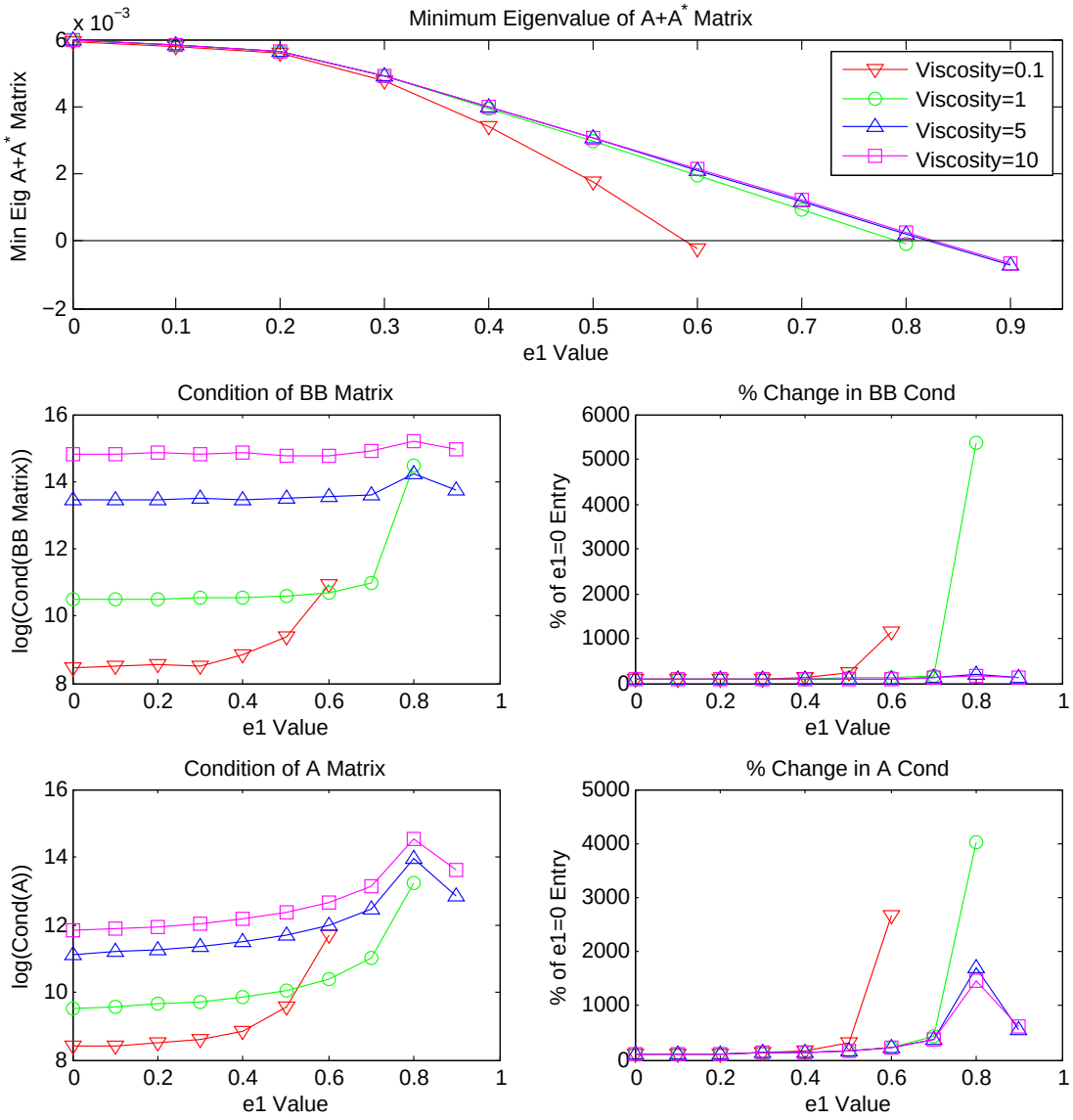

Figure 3.4: Sensitivity w.r.t. parameter ε_1 with $\lambda = 1$, Mesh Level 2.

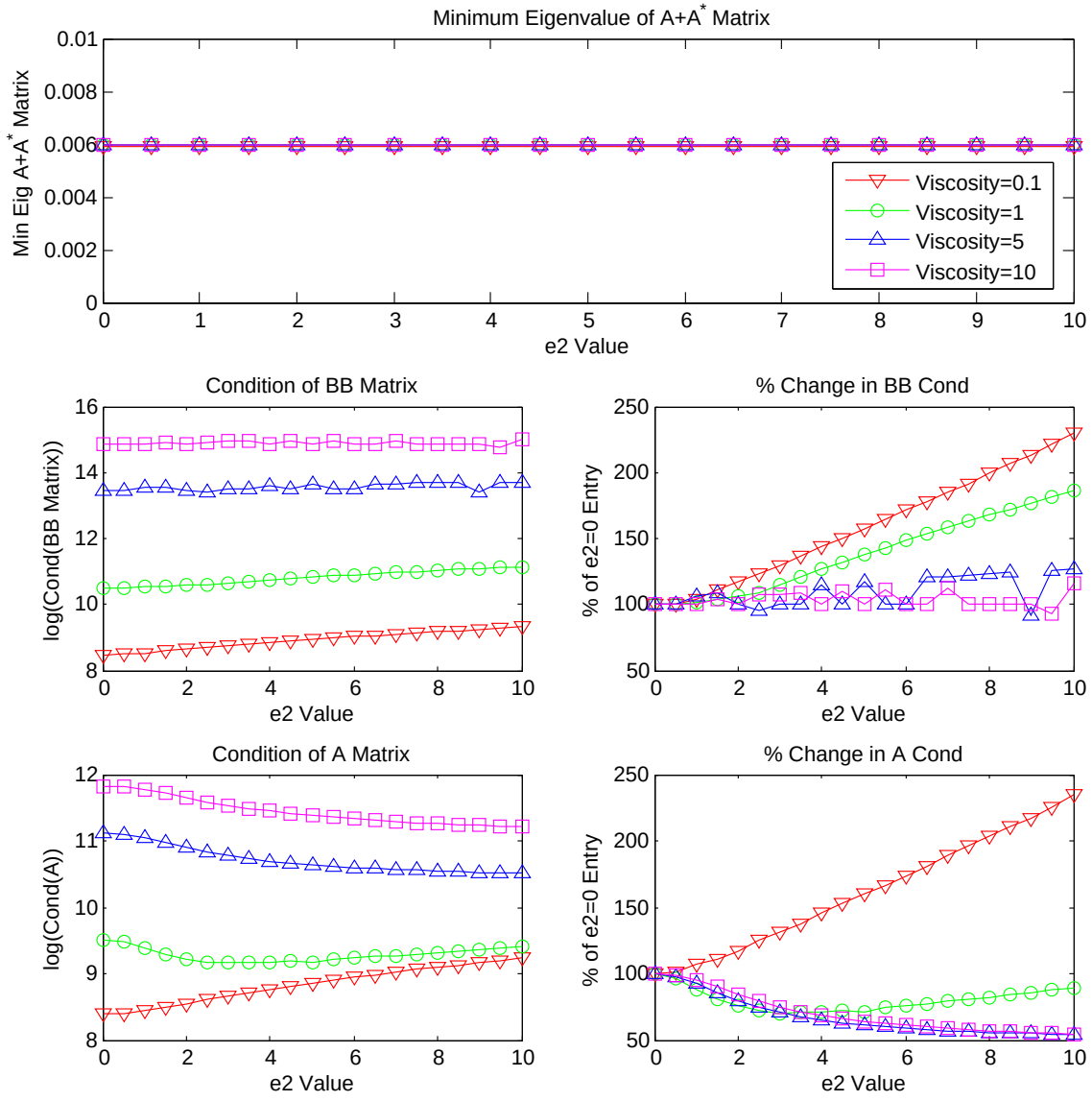

Figure 3.5: Sensitivity w.r.t. parameter ε_2 using (3.11) for $\mathcal{B}(V)$, with $\lambda = 1$, Mesh Level 2.

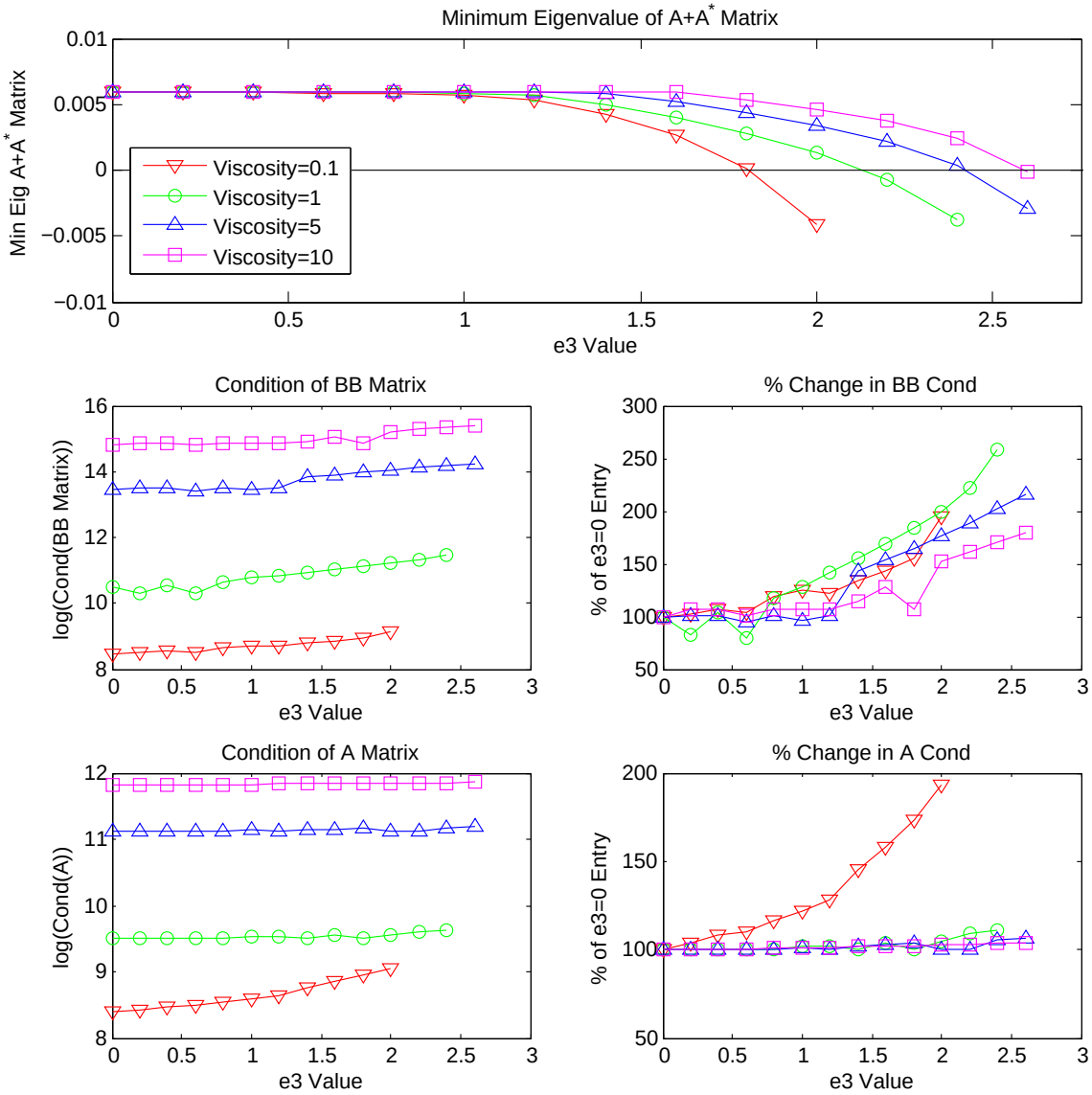

Figure 3.6: Sensitivity w.r.t. parameter ε_3, with $\lambda = 1$, Mesh Level 2.

The following observations can be made from Figures 3.4, 3.5, and 3.6:

- Large values of both ε_1 and ε_3 both lead to a loss of ellipticity in the discretization

of the $a_\lambda(\cdot,\cdot)$ bilinear form of the Babuška-Brezzi system. This can be seen in the eigenvalue plots crossing the horizontal axis in the ε_1 and ε_3 sensitivity tests.

- Tangential derivatives with constant coefficient ε_2 have no observed effect on the solvability of the system, at least in the flat geometry of our test problem. Note that this initially prompted a variable coefficient case to be considered below.

- Larger values of viscosity ν lead to an increase in condition number of both the A matrix and the full Babuška-Brezzi matrix. However, the system becomes more robust with respect to the interface dynamics, i.e. the relative change of the condition numbers with respect to ε_i are negligible at higher viscosity values.

- Higher viscosity extends the threshold of ellipticity for ε_1 and ε_3 where the minimum eigenvalue of $A + A^*$ remains positive. However, the viscosity can only extend this threshold to a certain extent. For example in Figure 3.4, changing the viscosity value from $\nu = 0.1$ to $\nu = 1$ results in a much larger gain in admissible ε_1 values compared to changing the viscosity from $\nu = 1$ to $\nu = 10$ where there is relatively no gain.

3.3.2 Epsilon Dependence (Variable Coefficient)

Figure 3.5 is interesting in the fact that the value of ε_2 seems to be independent of the ellipticity of the discretized bilinear form $a_\lambda(\cdot,\cdot)$. Recall that ε_2 corresponds to the coefficients of the first-order tangential derivatives of the $\mathcal{B}(V)$ boundary operator (3.2). This unexpected behavior may be caused by only considering constant coefficient tangential derivatives. Variable coefficients for the tangential derivatives would be more faithful to the original (2.8). In this run, we repeat the data set for parameter ε_2 using (3.12) as the

definition for $\mathcal{B}(V)$. The results can be seen in Figure 3.7.

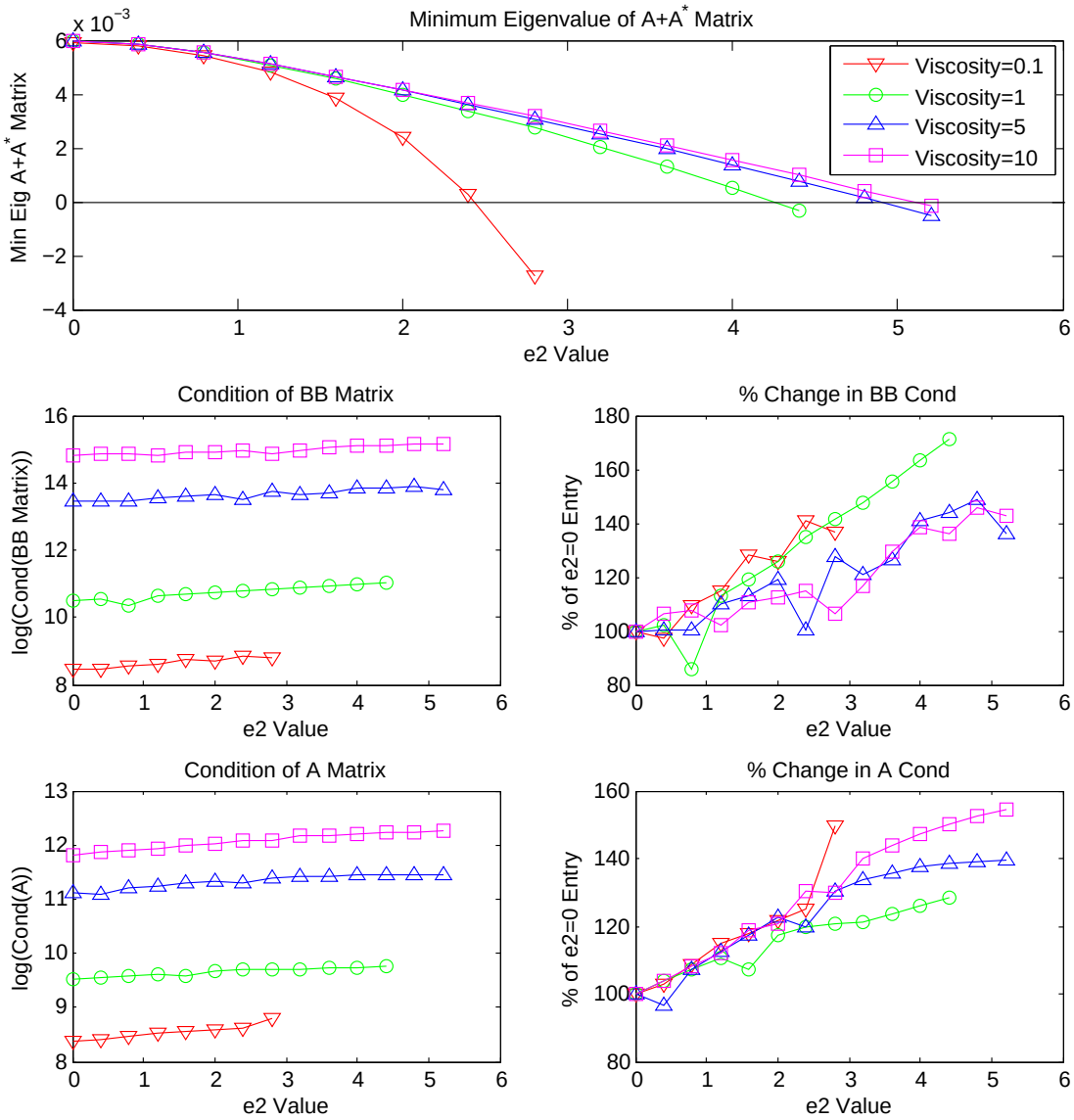

Figure 3.7: Sensitivity w.r.t. parameter ε_2 using (3.12) for $\mathcal{B}(V)$, with $\lambda = 1$, Mesh Level 2.

Using the variable coefficient, we see that ellipticity is lost as ε_2 increases. Like the constant coefficient case for the ε_1 and ε_3 parameters, an increase in viscosity led to an increase in the value that ε_2 can reach before losing ellipticity, and the increase that viscosity can provide appears to be limited. Furthermore, the increase in condition number as viscosity increases is also seen again. However, in the variable coefficient case for ε_2, the increase in robustness as viscosity increases is not observed.

3.3.3 Viscosity and Resolvent Dependence

The information contained in Figures 3.4, 3.6, and 3.7 suggest that given a large ε_i value with a fixed resolvent value λ, there may not be a viscosity value ν that guarantees ellipticity of the $a_\lambda(\cdot,\cdot)$ bilinear form. This can be more easily seen in Figure 3.8. In this figure, whose points were obtained with a level 1 mesh, the minimum eigenvalues of matrix $A + A^*$ are plotted with respect to both parameter ε_1 and viscosity ν. As with earlier cases, the resolvent value $\lambda = 1$ was used for each point. Here the intersection curve of the surface with the horizontal 0-plane appears to have asymptotic behavior at approximately $\varepsilon_1 \approx 0.83$. This matches the results in Figure 3.4 where higher viscosity did not seem to extend the ellipticity threshold significantly.

To better understand how the resolvent value λ affects the ellipticity of the $a_\lambda(\cdot,\cdot)$ bilinear form, we may consider a fixed viscosity, say $\nu = 1$, and look at the minimum eigenvalue of $A + A^*$ with respect to ε_1 and λ. This can be seen (using mesh level 2) in Figure 3.9. Note that in this plot, once the minimum eigenvalue becomes negative for a given ε_1 value, the rest of the points for the particular λ value are artificially set to 0 to save on computing time since no useful information can be gathered after ellipticity is lost.

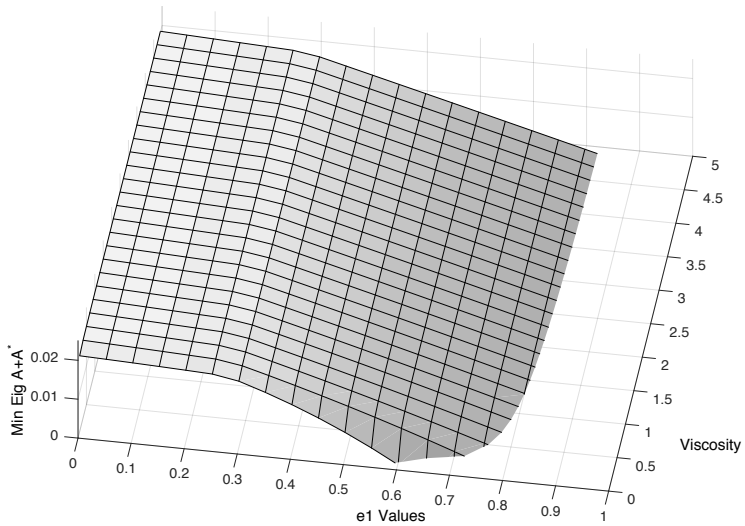

Figure 3.8: Minimum eigenvalue of $A + A^*$ w.r.t. ε_1 and viscosity, with $\lambda = 1$, mesh level 1

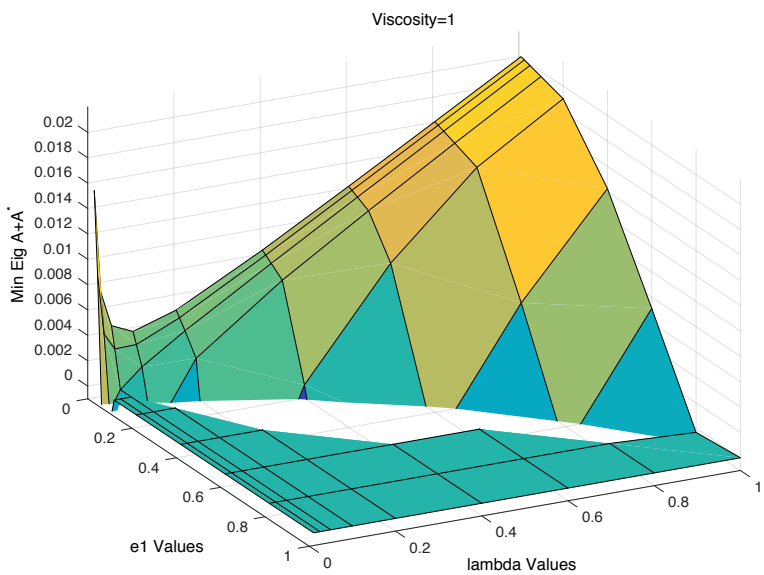

Figure 3.9: Minimum Eigenvalue of $A + A^*$ w.r.t. ε_1 and λ, with viscosity= 1, mesh level 2

In Figure 3.9, we observe that larger resolvent values λ seem to extend the admissible range of ε_1 values. That is to say, for a fixed viscosity ν, larger resolvent values λ lead to larger ε_1 value before the minimum eigenvalue of $A + A^*$ becomes negative. Furthermore in this figure, we see a "spike" near the origin. Recall that for $\varepsilon_i = 0$ for all i, we have the classical Stokes-elasticity coupling which is well-posed for any positive viscosity and positive resolvent value.

It was not possible to recreate Figure 3.9 with arbitrarily small viscosity values. For example, $\nu < 0.05$ with $\lambda = 0.01$ is not admissible because the matrix A cannot be formed. In this case, the extension problems into the solid domain become ill-posed.

In a separate test with $\nu = 1$, $\varepsilon_1 = 0.01$, and $\lambda = 0.001$, the numerical analysis once again points to an ill-posed elliptic problem for the elastic component. This suggests a discontinuity in Figure 3.9 near the origin that is not necessarily clear from the discrete set of data points shown in the figure. This discontinuity would support the authors' suspicion of strict ω-dissipativity ($\omega > 0$) of the semigroup generator $\mathbb{A}$ in [9].

3.4 Summary of Observations from the Numerical Analysis

The numerical experiments above underline the following traits with respect to the well-posedness of the original linear system (2.7).

- The system (as asserted theoretically) degenerates with respect to the zero-order terms of the elasticity component in (2.8), including those dependent on the curvature of the interface. The system also degenerates with respect to the norm of the state around which the linearization is performed.

- The dependence on the oblique derivative condition on the boundary interface does not always manifest itself in polygonal geometry unless coefficients are state-dependent. Thus constant coefficient approximations of (2.8) may turn out to be highly inaccurate.

- A multiple of w with a very large constant coefficient Λ was added to the fluid equation in [9]. Such a term represents a bounded (in fact, compact) perturbation of the evolution generator and thus does not affect the existence of a C_0 semigroup. However, this term was an important artifact of the proof and was used to verify the ellipticity of the bilinear form in the context of the Babuška-Brezzi theorem approach. A numerical version of this argument has now been successfully carried without adding such a term, suggesting that one might be able to further refine the proof in [9] to work without this bounded perturbation.

- The classical Stokes-elasticity coupling admits any positive viscosity. However, both theoretical and numerical evidence indicates that the total linearization requires a large enough viscosity ν to ensure solvability. In addition, for any particular resolvent value $\lambda > 0$, the admissible energy of the linearization state is limited even independently of the viscosity. In other words, the energy of the linearization state cannot be increased beyond a certain threshold for even highly viscous flows without sacrificing the stability properties or well-posedness of the model.

- The system appears strictly ω-dissipative. That is, even for weak steady regimes and relatively large viscosity, the spectrum of the generator appears to overlap the right half-plane. This aspect suggests that merely adding a dissipative term on the boundary, for example, as in stabilization of the Stokes-elasticity model in [3], *is not sufficient to make the dynamics even conservative.* We therefore expect a feedback

stabilization argument for this model to have a size requirement on the coefficient of the dissipative term. The magnitude of the coefficient may depend on the energy of the regime at which the system is linearized. This observation fully supports the analysis of stability discussed in Chapter 4.

Chapter 4

Feedback Stabilization

It is known that classical, linear Stokes-Lamé systems can be stabilized by boundary feedback in the form of dissipative velocity matching on the common interface [3]. In this chapter, we are interested in feedback stabilization for the total linearization. However, as seen in Lemma 2.3.1, the system is not dissipative to begin with, which represents a key departure from the feedback control analysis in [3] on the classical Stokes-elasticity coupling. We prove that a damped version of the total linearization, with viscous dissipation within the elastic solid itself and with boundary dissipation of the form used in [3], is exponentially stable.

4.1 Dissipative Model

As seen in [3], the classical Stokes-elasticity coupling can be stabilized solely by a boundary feedback control where the size of the positive dissipation coefficient was unrestricted. However, in the total linearization (2.7), the same stabilization procedure cannot be repeated due to the extra terms in both the stress matching and velocity matching con-

ditions on the boundary interface. To overcome this (asymptotic) instability, our system requires both viscous damping in the form of a U'_t term and "static" damping in the form of a U' term in the interior of the solid domain. In fact, one would not expect exponential stability without such terms in the general case as already noted in [41] for a nonlinear model. Furthermore, the size of the boundary damping coefficient should depend on the energy of the linearization state (w, φ, p). Therefore, we consider the following damped model with the additional damping terms in red.

$$\begin{cases} w'_t - \nu\Delta w' + (Dw')w + (Dw)w' + \nabla p' &= \mathbf{0} & \text{in} \quad (0,T)\times\Omega_f \\ \operatorname{div}(w') &= 0 & \text{in} \quad (0,T)\times\Omega_f \\ U'_{tt} - \det(\Theta)\operatorname{Div}[\overline{\mathcal{T}}'(U')] + \alpha U'_t + \gamma U' &= \mathbf{0} & \text{in} \quad (0,T)\times\Omega \\ \\ w' + (Dw)U' &= U'_t + K\overline{\mathcal{T}}'(U')n & \text{on} \quad (0,T)\times\Gamma \\ \overline{\mathcal{T}}'(U')n &= \sigma_f(w',p')n + \mathcal{B}(U') & \text{on} \quad (0,T)\times\Gamma \\ w' &= \mathbf{0} & \text{on} \quad (0,T)\times\Gamma_f \\ U'(0) = u^0; \quad U'_t(0) = u^1; \quad w'(0) = w^0 \end{cases} \tag{4.1}$$

for $\sigma_f(w', p') = 2\nu\varepsilon(w') - p'I$.

Here, the terms α and γ correspond to the distributed dissipation in the solid. The term $K\overline{\mathcal{T}}'(U')n$ is the natural generalization of the Stokes-elasticity feedback [3] and describes analogously a slip condition on the interface to [54, p. 240]. The friction forces provide a dissipative effect, which, from the functional-analytic perspective, is reflected in the shift of the generator spectrum to the left complex half-plane.

Next, we define the energy of the system proportional to the square of an equivalent

norm in $\mathscr{H}$:

$$E(w', U', U_t') = \frac{1}{2}\|w'\|^2_{L^2(\Omega_f)} + \frac{1}{2}\|U_t'\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2}|U'|^2_{1,\Omega} + \frac{1}{2}\gamma\|U'\|_{L^2_\Theta(\Omega)} \tag{4.2}$$

where

$$(V, W)_{1,\Omega} = \int_\Omega \overline{\mathcal{T}}'(V)..DW \quad \text{and} \quad |U'|^2_{1,\Omega} = \int_\Omega \overline{\mathcal{T}}'(U')..DU'\,.$$

We will use the shorthand notation $E(t) = E(w'(t), U'(t), U_t'(t))$. Furthermore, $E_{U'}(t)$ will denote the elastic components of the energy.

4.2 Exponential Stability

Now we present the main result of this chapter.

Theorem 4.2.1. *(Stability) For $\alpha, \gamma > 0$, assume that in addition to the condition $\nu > \nu_0$ furnished by the well-posedness result of the total linearization, the viscosity value also admits some $\eta \in (0, 1)$ such that*

$$\left| \int_0^T \int_{\Omega_f} \langle w', (Dw')w + (Dw)w' \rangle \right| \leq 2\eta\nu \int_0^T \|w'\|^2_{H^1(\Omega_f)}\,.$$

Then there exists $K_1 > 0$ dependent on α, γ, ν (and the norm of (w, φ, p)) such that for $K > K_1$ the dynamical system generated by (4.1) is exponentially stable. In particular, there exist $C > 0$ and $\omega > 0$ such that the energy $E(t)$ defined by (4.2) satisfies

$$E(t) \leq C\,E(0)e^{-\omega t}, \quad \textit{for} \quad t \geq 0.$$

The remainder of this chapter provides the proof for the stability theorem. The anal-

ysis strategy is based on multiplier estimates. In this model, the natural energy identity does not imply any dissipation due to the presense of the tangential derivatives in the stress matching condition. Absorption of these terms in stability estimates necessitates use of the interior viscous damping in the elastic body.

4.2.1 Energy Identities

As usual, all energy estimates are performed for strong semigroup solutions and extended to weak ones by density of the generator domain and the fact that the concluding inequalities are continuous with respect to the finite energy space $\mathscr{H}$. Recall that on the common interface n and $n_f = -n$ denote the outward normal fields with respect to the solid and fluid domains respectively. We will also use the shorthand notation $\langle V, W\rangle = V^* W$.

Fluid with w' Multiplier: We begin with the fluid dynamics in (4.1) along with a w' multiplier.

$$\begin{aligned}
\mathbf{0} &= w_t' - \nu\Delta w' + (Dw')w + (Dw)w' + \nabla p' \\
0 &= \int_{\Omega_f} \langle w', w_t'\rangle - \nu\langle w', \Delta w'\rangle + \langle w', (Dw')w + (Dw)w'\rangle + \langle w', \nabla p\rangle \\
0 &= \int_{\Omega_f} \langle w', w_t'\rangle - \nu\langle w', 2\,\mathrm{Div}(\varepsilon(w'))\rangle \\
&\quad \int_{\Omega_f} \langle w', (Dw')w + (Dw)w'\rangle + \langle w', \nabla p\rangle
\end{aligned}$$

We have used the basic identity $\Delta w' = 2\,\mathrm{Div}(\varepsilon(w'))$ since we have $\mathrm{div}(w') = 0$.

Now we consider terms individually, applying Green's formula where needed. In order

to simplify terms, recall that $w' = \mathbf{0}$ on Γ_f and $\operatorname{div}(w') = 0$ in Ω_f.

$$\begin{aligned}
\int_{\Omega_f} \langle w', w'_t \rangle &= \frac{1}{2} \int_{\Omega_f} \frac{d}{dt}(|w'|^2) \\
\int_{\Omega_f} \langle w', \nabla p' \rangle &= -\int_{\Omega_f} \operatorname{div}(w') p' + \int_{\Gamma} \langle w', p' I \, n_f \rangle \\
&= \int_{\Gamma} \langle w', p' I \, n_f \rangle \\
&= -\int_{\Gamma} \langle w', p' I \, n \rangle \\
\int_{\Omega_f} -\langle w', 2\nu \operatorname{Div}(\varepsilon(w')) \rangle &= \nu \int_{\Omega_f} 2\varepsilon(w') .. (Dw') - \nu \int_{\Gamma} \langle 2\varepsilon(w') n_f, w' \rangle \\
&= 2\nu \int_{\Omega_f} \varepsilon(w') .. \varepsilon(w') - \nu \int_{\Gamma} \langle w', 2\varepsilon(w') n_f \rangle \\
&= 2\nu \int_{\Omega_f} \varepsilon(w') .. \varepsilon(w') + \int_{\Gamma} \langle w', 2\nu \varepsilon(w') n \rangle \\
&= 2\nu \int_{\Omega_f} \|w'\|^2_{H^1(\Omega_f)} + \int_{\Gamma} \langle w', 2\nu \varepsilon(w') n \rangle
\end{aligned}$$

With the understanding that $\int_{\Omega_f} \varepsilon(w') .. \varepsilon(w')$ is an equivalent inner product on $H^1_{\Gamma_f}(\Omega_f)$ from Proposition 2.2.1, we can write it as $\|w'\|^2_{H^1(\Omega)}$.

Now, using the fundamental theorem of calculus, we have the following identity:

$$\begin{aligned}
&\frac{1}{2} \|w'\|^2_{L^2(\Omega_f)} \Big|_0^T + 2\nu \int_0^T \|w'\|^2_{\mathbf{H}^1(\Omega_f)} \\
&+ \int_0^T \int_{\Gamma} \langle w', 2\nu \varepsilon(w') n \rangle + \int_0^T \int_{\Omega_f} \langle w', (Dw') w + (Dw) w' \rangle = \int_0^T \int_{\Gamma} \langle w', pn \rangle \, .
\end{aligned} \tag{4.3}$$

Solid with U_t' Multiplier: Similarly, we can use a U_t' multiplier on the solid dynamics in (4.1).

$$\begin{aligned}
\mathbf{0} &= \frac{1}{\det(\Theta)}U_{tt}' - \mathrm{Div}[\overline{\mathcal{T}}'(U')] + \frac{\alpha}{\det(\theta)}U_t' + \frac{\gamma}{\det(\theta)}U' \\
0 &= \int_\Omega \frac{1}{\det(\theta)}\langle U_t', U'\rangle - \int_\Omega \langle U_t', \mathrm{Div}[\overline{\mathcal{T}}'(U')]\rangle \\
&\quad + \int_\Omega \frac{\alpha}{\det(\Theta)}\langle U_t', U'\rangle + \int_\Omega \frac{\gamma}{\det(\Theta)}\langle U_t', U'\rangle
\end{aligned}$$

Considering terms individually as before, we have:

$$\begin{aligned}
\int_\Omega \frac{1}{\det(\theta)}\langle U_t', U'\rangle &= \int_\Omega \frac{d}{dt}\left(\frac{1}{2}|U_t'|^2\right) \\
\int_\Omega \frac{\alpha}{\det(\theta)}\langle U_t', U_t'\rangle &= \int_\Omega \frac{\alpha}{\det(\Theta)}|U_t'|^2 \\
\int_\Omega \frac{\gamma}{\det(\Theta)}\langle U_t', U'\rangle &= \int_\Omega \frac{\gamma}{\det(\Theta)}\frac{d}{dt}\left(\frac{1}{2}|U'|^2\right) \\
-\int_\Omega \langle U_t', \mathrm{Div}(\overline{\mathcal{T}}'(U')\rangle &= -\int_\Gamma \langle U_t', \overline{\mathcal{T}}'(U')n\rangle + \int_\Omega \overline{\mathcal{T}}'(U')..DU_t\,.
\end{aligned}$$

Note here that $\overline{\mathcal{T}}'(U')..DU_t$ can be rewritten as $\frac{1}{2}\frac{d}{dt}\left(\overline{\mathcal{T}}'(U')..DU\right)$. Once we integrate in time, this leaves us with the elastic energy identity:

$$E_{U'}(T) + \alpha\int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)} = E_{U'}(0) + \int_0^T\int_\Gamma \langle U_t', \overline{\mathcal{T}}'(U')n\rangle \tag{4.4}$$

Solid with U' Multiplier: In (4.1), the boundary conditions on the interface incorporate tangential differential operator $\mathcal{B}$ applied to the elastic displacement. This term ultimately contributes a non-dissipative finite-energy perturbation. For this reason, we consider another "equipartition" multiplier U' for the solid equation.

$$
\begin{aligned}
\mathbf{0} &= \frac{1}{\det(\theta)} U'_{tt} - \mathrm{Div}[\overline{\mathcal{T}}'(U')] + \frac{\alpha}{\det(\theta)} U'_t + \frac{\gamma}{\det(\theta)} U' \\
0 &= \int_{t=a}^{b} \int_{\Omega} \frac{1}{\det(\Theta)} \langle U', U'_{tt} \rangle - \int_{t=a}^{b} \int_{\Omega} \langle U', \mathrm{Div}[\overline{\mathcal{T}}'(U')] \rangle \\
&\quad + \int_{t=a}^{b} \int_{\Omega} \frac{\alpha}{\det(\Theta)} \langle U', U'_t \rangle + \int_{t=a}^{b} \int_{\Omega} \frac{\gamma}{\det(\Theta)} \langle U', U' \rangle
\end{aligned}
$$

Then, consider individual terms:

$$
\begin{aligned}
\int_{t=a}^{b} \int_{\Omega} \frac{1}{\det(\Theta)} \langle U', U'_{tt} \rangle &= \int_{\Omega} \frac{1}{\det(\Theta)} \left((U')^* U'_t \big]_{t=a}^{t=b} - \int_{t=a}^{b} |U'_t|^2 \right) \\
\int_{t=a}^{b} \int_{\Omega} \frac{\alpha}{\det(\Theta)} \langle U', U'_t \rangle &= \int_{t=a}^{b} \int_{\Omega} \frac{\alpha}{\det(\Theta)} \frac{d}{dt} \left(\frac{1}{2} |U'|^2 \right) \\
\int_{t=a}^{b} \int_{\Omega} \frac{\gamma}{\det(\Theta)} \langle U', U' \rangle &= \int_{t=a}^{b} \int_{\Omega} \frac{\gamma}{\det(\Theta)} |U'|^2 \\
- \int_{t=a}^{b} \int_{\Omega} \langle U', \mathrm{Div}[\overline{\mathcal{T}}'(U')] \rangle &= \int_{t=a}^{b} \int_{\Omega} \overline{\mathcal{T}}(U') .. DU - \int_{t=a}^{b} \int_{\Gamma} \langle U', \overline{\mathcal{T}}'(U') n \rangle \, .
\end{aligned}
$$

Therefore, we have the following "equipartition" identity.

$$
\begin{aligned}
&(U', U'_t)_{L^2_\Theta(\Omega)} \Big|_0^T - \int_0^T \|U'_t\|^2_{L^2_\Theta(\Omega)} + \int_0^T |U'|^2_{1,\Omega} - \int_0^T \int_\Gamma \langle U', \overline{\mathcal{T}}'(U') n \rangle \\
&+ \gamma \int_0^T \|U'\|^2_{L^2_\Theta(\Omega)} + \frac{\alpha}{2} \|U'\|^2_{L^2_\Theta(\Omega)} \Big|_0^T = 0
\end{aligned}
\tag{4.5}
$$

4.2.2 Preliminary Inequalities:

Let $c > 0$, to be specified later. If we multiply (4.5) by c and add (4.3) and (4.4), we arrive at the "fundamental identity":

$$\begin{aligned}
& E(T) + 2\nu \int_0^T \|w'\|^2_{\mathbf{H}^1(\Omega_f)} + \alpha \int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)} + c \int_0^T |U'|^2_{1,\Omega} + c\gamma \int_0^T \|U'\|^2_{L^2_\Theta(\Omega)} \\
= \; & E(0) + \int_0^T \int_\Gamma \langle w', p'n - 2\nu\varepsilon(w')n\rangle + \int_0^T \int_\Gamma \langle U_t' + cU', \overline{\mathcal{T}}'(U')n\rangle + c\int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)} \\
& - \int_0^T \int_{\Omega_f} \langle w', (Dw')w + (Dw)w'\rangle \\
& - c(U', U_t')_{L^2_\Theta(\Omega)}\Big|_0^T - c\frac{\alpha}{2}\|U'\|^2_{L^2_\Theta(\Omega)}\Big|_0^T.
\end{aligned} \tag{4.6}$$

The main strategy for the proof of Theorem 4.2.1 is to apply Lemma 4.3.4 to an inequality derived from (4.6).

So, in order to apply Lemma 4.3.4 to (4.6), we will estimate "unwanted" terms on the right-hand side of the equation via any of the following:

- multiples of $E(0)$ (or $E_{U'}(0)$ which is smaller),
- *negative* multiples of $E(T)$ or $E_{U'}(T)$,
- trace products $\int_\Gamma \langle \overline{\mathcal{T}}'(U')n, U_t'\rangle$ and $\int_\Gamma \langle \overline{\mathcal{T}}'(U')n, U'\rangle$,
- integrals over Ω_f.

In the second phase, we will invoke the boundary conditions of (4.1) and estimate all the terms remaining on the right-hand side in terms of multiples of $E(0)$ or of fractional multiples of the terms on the left-hand side.

First, consider $c\int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)}$ from the right-hand side of (4.6). If $c \le \alpha$, say $c = \alpha/2$, then we could absorb the term into the corresponding integral on the left-hand side, but

that, as we will see later in Remark 4.2.1, indirectly places additional restrictions on the viscosity. Instead, we rewrite this kinetic energy term using (4.4):

$$c\int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)} = \frac{c}{\alpha}\left(E_{U'}(0) - E_{U'}(T)\right) + \frac{c}{\alpha}\int_0^T\int_\Gamma \langle U_t', \overline{\mathcal{T}}'(U')n\rangle\,. \tag{4.7}$$

Since $\alpha > 0$ and $\gamma > 0$ in the elasticity equation, use Lemma 4.3.1 to see that

$$\begin{aligned}
-c(U', U_t')_{L^2_\Theta(\Omega)}\Big|_0^T - c\frac{\alpha}{2}\|U'\|^2_{L^2_\Theta(\Omega)}\Big|_0^T &\leq c\left[\frac{1}{2}\|U'\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2}\|U_t'\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2}\alpha\|U_t'\|^2_{L^2_\Theta(\Omega)}\right]_T \\
&\quad + c\left[\frac{1}{2}\|U'\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2}\|U_t'\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2}\alpha\|U_t'\|^2_{L^2_\Theta(\Omega)}\right]_0 \\
&\leq c(1+\alpha)\left[\frac{1}{2}\|U'\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2}\|U_t'\|^2_{L^2_\Theta(\Omega)}\right]_T \\
&\quad + c(1+\alpha)\left[\frac{1}{2}\|U'\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2}\|U_t'\|^2_{L^2_\Theta(\Omega)}\right]_0 \\
&\leq \frac{c(1+\alpha)}{\gamma}\left[E_{U'}(0) + E_{U'}(T)\right].
\end{aligned}$$

for γ small. Note that for large γ, we can replace $\frac{c(1+\alpha)}{\gamma}$ with $\frac{c(1+\alpha+\gamma)}{\gamma}$ that still works for the rest of the stability argument.

This last estimate, however, brings in the term $E_{U'}(T)$, so we further invoke (4.4) to estimate $E_{U'}(T)$ from above and obtain

$$-c(U', U_t')_{L^2_\Theta(\Omega)}\Big|_0^T - c\frac{\alpha}{2}\|U'\|^2_{L^2_\Theta(\Omega)}\Big|_0^T \leq \frac{c(1+\alpha)}{\gamma}\left(E_{U'}(0) + E_{U'}(0) + \int_0^T\int_\Gamma \langle U_t', \overline{\mathcal{T}}'(U')n\rangle\right). \tag{4.8}$$

Applying (4.7) and (4.8) to (4.6), we are left with

$$
\begin{aligned}
& E(T) + 2\nu \int_0^T \|w'\|^2_{\mathbf{H}^1(\Omega_f)} + c \int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)} + c \int_0^T |U'|^2_{1,\Omega} + c\gamma \int_0^T \|U'\|^2_{L^2_\Theta(\Omega)} \\
\leq \; & E(0) + \int_0^T \int_\Gamma \langle w', p'n - 2\nu\varepsilon(w')n \rangle \\
& + \int_0^T \int_\Gamma \left\langle c\left(\frac{1+\alpha}{\gamma} + \frac{1}{\alpha}\right) U_t' + U_t' + cU', \overline{\mathcal{T}}'(U')n \right\rangle \\
& - \int_0^T \int_{\Omega_f} \langle w', (Dw')w + (Dw)w' \rangle \\
& + \left(\frac{2c(1+\alpha)}{\gamma} + \frac{c}{\alpha}\right) E_{U'}(0) - \frac{c}{\alpha} E_{U'}(T)\,.
\end{aligned}
\tag{4.9}
$$

At this point, the first phase of the estimates has been completed and we are ready to invoke the boundary conditions. We label, for a shorthand

$$
C_{c,\alpha,\gamma} = c\left(\frac{1+\alpha}{\gamma} + \frac{1}{\alpha}\right) + 1,
$$

and rewrite the integrals over Γ on the right-hand side of (4.9) using the boundary

conditions of our system (4.1):

$$
\begin{aligned}
& \int_0^T \int_\Gamma \langle w', p'n - 2\nu\varepsilon(w')n\rangle + \int_0^T \int_\Gamma \left\langle C_{c,\alpha,\gamma} U_t' + cU', \overline{\mathcal{T}}'(U')n \right\rangle \\
= & \int_0^T \int_\Gamma \langle w', \mathcal{B}(U') - \overline{\mathcal{T}}'(U')n\rangle \\
& + C_{c,\alpha,\gamma} \int_0^T \int_\Gamma \langle w' + (Dw)U' - K\overline{\mathcal{T}}'(U')n, \overline{\mathcal{T}}'(U')n\rangle + c\int_0^T \int_\Gamma \langle U', \overline{\mathcal{T}}'(U')n\rangle \\
= & \int_0^T \int_\Gamma \left\langle (C_{c,\alpha,\gamma} - 1)\, w', \overline{\mathcal{T}}'(U')n \right\rangle + \int_0^T \int_\Gamma \langle w', \mathcal{B}(U')\rangle \\
& + C_{c,\alpha,\gamma} \int_0^T \int_\Gamma \langle (Dw)U', \overline{\mathcal{T}}'(U')n\rangle \\
& - C_{c,\alpha,\gamma} K \int_0^T \int_\Gamma |\overline{\mathcal{T}}'(U')n|^2 + \int_0^T \int_\Gamma c\langle U', \overline{\mathcal{T}}'(U')n\rangle \,.
\end{aligned}
\tag{4.10}
$$

In (4.9) we drop the term $-(c/\alpha)E_{U'}(T)$ (since $E_{U'}(T) \geq 0$), replace $E_{U'}(0)$ by the larger total energy $E(0)$, and plug in (4.10). We are left with the following:

$$
\begin{aligned}
& E(T) + 2\nu \int_0^T \|w'\|^2_{\mathbf{H}^1(\Omega_f)} + \alpha \int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)} + c\int_0^T |U'|^2_{1,\Omega} + c\gamma \int_0^T \|U'\|^2_{L^2_\Theta(\Omega)} \\
& + C_{c,\alpha,\gamma} K \int_0^T \int_\Gamma |\overline{\mathcal{T}}'(U')n|^2 \\
\leq & \left(1 + \frac{2c(1+\alpha)}{\gamma} + \frac{c}{\alpha}\right) E(0) - \int_0^T \int_{\Omega_f} \langle w', (Dw')w + (Dw)w'\rangle \\
& + \int_0^T \int_\Gamma \left\langle (C_{c,\alpha,\gamma} - 1)\, w', \overline{\mathcal{T}}'(U')n \right\rangle + \int_0^T \int_\Gamma \langle w', \mathcal{B}(U')\rangle \\
& + \int_0^T \int_\Gamma \langle (c + C_{c,\alpha,\gamma} Dw)U', \overline{\mathcal{T}}'(U')n\rangle \,.
\end{aligned}
\tag{4.11}
$$

4.2.3 Stabilization Estimate:

Recall that the well-posedness result of the total linearization asserts (a fortiori) that the $C^1(\overline{\Omega})$ norm of the steady regime velocity w is small, whereas the viscosity ν must stay

above some positive minimal value. In line with this, we need an additional constraint that ν is large enough (or w is small enough) so that there exists $\eta \in (0,1)$ for which

$$\left| \int_0^T \int_{\Omega_f} \langle w', (Dw')w + (Dw)w' \rangle \right| \leq \eta(2\nu) \int_0^T \|w'\|^2_{\mathbf{H}^1(\Omega_f)}.$$

This is the only other condition on viscosity ν on top of the restrictions imposed by well-posedness theorem. We justify the existence of such an η with Lemma 4.3.3. For small w and Dw, we are granted a small value of k in the lemma.

Choice of c Parameter: Recall that c was a parameter of our choice. Also, operator $\mathcal{B}$ is a tangential boundary operator with variable coefficients (dependent on the steady regime (w, φ, p)), which is continuous $\mathbf{H}^1(\Omega) \to \mathbf{H}^{-1/2}(\Gamma)$. Hence we can choose

$$c = c(\nu, \alpha, \gamma, w, \varphi, p) > 0$$

large enough so that the following inequality on the trace integrals from the right-hand side of (4.11) holds (by means of the Schwartz and Young and trace inequalities).

$$\begin{aligned} & \left| \int_0^T \int_\Gamma \left\langle (C_{c,\alpha,\gamma} - 1)\, w', \overline{\mathcal{T}}'(U')n \right\rangle \right| + \left| \int_0^T \int_\Gamma \langle w', \mathcal{B}(U') \rangle \right| \\ & + \left| \int_0^T \int_\Gamma \langle (c + C_{c,\alpha,\gamma} Dw)U', \overline{\mathcal{T}}'(U')n \rangle \right| \\ \leq\ & (1-\eta)\nu \int \|w'\|^2_{\mathbf{H}^1(\Omega_f)} + \frac{c}{2} \left(\int_0^T |U'|^2_{1,\Omega} + \gamma \int_0^T \|U'\|^2_{L^2_\Theta(\Omega)} \right) \\ & + C_2(c, \alpha, \gamma) \int_0^T \int_\Gamma |\overline{\mathcal{T}}'(U')n|^2 \end{aligned} \tag{4.12}$$

To justify the existence of this $c > 0$, we consider individual terms.

$$\begin{aligned}
&\left|\int_0^T \int_\Gamma \langle (C_{c,\alpha,\gamma} - 1)w', \overline{\mathcal{T}}'(U')n\rangle\right| \\
\leq\ & \int_0^T \int_\Gamma \left(\frac{1}{2}\epsilon |w'|^2\right) + \int_0^T \int_\Gamma \left(\frac{1}{2\epsilon}(C_{c,\alpha,\gamma} - 1)|\overline{\mathcal{T}}'(U')n|^2\right)
\end{aligned}$$

using Lemma 4.3.1 for ϵ to be specified later. Then, using the trace inequality for a fixed $C_{\text{trace}} > 0$,

$$\begin{aligned}
&\left|\int_0^T \int_\Gamma \langle (C_{c,\alpha,\gamma} - 1)w', \overline{\mathcal{T}}'(U')n\rangle\right| \\
\leq\ & \int_0^T \int_\Gamma \left(\frac{1}{2}\epsilon |w'|^2\right) + \int_0^T \int_\Gamma \left(\frac{1}{2\epsilon}(C_{c,\alpha,\gamma} - 1)|\overline{\mathcal{T}}'(U')n|^2\right) \\
\leq\ & \frac{1}{2}\epsilon C_{\text{trace}} \int_0^T \|w'\|^2_{H^1(\Omega_f)} + \frac{1}{2\epsilon}(C_{c,\alpha,\gamma} - 1)\int_0^T \int_\Gamma \left(|\overline{\mathcal{T}}'(U')n|^2\right).
\end{aligned} \tag{4.13}$$

For the second term in (4.12), we can proceed in a similar way as the first term. Apply Lemma 4.3.1.

$$\begin{aligned}
\int_0^T \int_\Gamma \langle w', \mathcal{B}(U')\rangle &\leq \frac{1}{2}\epsilon \int_0^T \int_\Gamma |w'|^2 + \frac{1}{2\epsilon}\int_0^T \int_\Gamma |\mathcal{B}(U')|^2 \\
&\leq \frac{1}{2}\epsilon \int_0^T \int_\Gamma |w'|^2 + \frac{1}{2\epsilon} C_{\mathcal{B}} \int_0^T \|U'\|^2_{H^1(\Omega)}
\end{aligned}$$

for some fixed $C_{\mathcal{B}} > 0$ using the boundedness of the $\mathcal{B}$ operator from Proposition 2.2.2.

Continue the estimation with the trace inequality.

$$
\begin{aligned}
\int_0^T \int_\Gamma \langle w', \mathcal{B}(U')\rangle &\leq \frac{1}{2}\epsilon \int_0^T \int_\Gamma |w'|^2 + \frac{1}{2\epsilon} C_{\mathcal{B}} \int_0^T \|U'\|^2_{H^1(\Omega)} \\
&\leq \frac{1}{2}\epsilon C_{\text{trace}} \int_0^T \|w'\|^2_{H^1(\Omega_f)} + \frac{1}{2\epsilon} C_{\mathcal{B}} \int_0^T \|U'\|^2_{H^1(\Omega)} \\
&\leq \frac{1}{2}\epsilon C_{\text{trace}} \int_0^T \|w'\|^2_{H^1(\Omega_f)} \\
&\quad + \frac{1}{2\epsilon} C_{\mathcal{B}} C_{\text{equiv}} \left(\int_0^T |U'|^2_{1,\Omega} + \gamma \int_0^T \|U'\|^2_{L^2_\Theta(\Omega)} \right)
\end{aligned}
\tag{4.14}
$$

from equivalence of $H^1(\Omega)$ norms.

Lastly, for the final term on the left in (4.12), we can apply the triangle inequality and Lemma 4.3.1.

$$
\begin{aligned}
&\left| \int_0^T \int_\Gamma \langle (c + C_{c,\alpha,\gamma} Dw) U', \overline{\mathcal{T}}'(U')n \rangle \right| \\
\leq\ & \left| \int_0^T \int_\Gamma \langle c U', \overline{\mathcal{T}}'(U')n \rangle \right| + \left| \int_0^T \int_\Gamma \langle C_{c,\alpha,\gamma}(Dw) U', \overline{\mathcal{T}}'(U')n \rangle \right| \\
\leq\ & \int_0^T \int_\Gamma \frac{1}{2}\epsilon |U'|^2 + \int_0^T \int_\Gamma \frac{1}{2\epsilon} c |\overline{\mathcal{T}}'(U')n|^2 + \left| \int_0^T \int_\Gamma \langle C_{c,\alpha,\gamma}(Dw) U', \overline{\mathcal{T}}'(U')n \rangle \right| \\
\leq\ & \int_0^T \int_\Gamma \frac{1}{2}\epsilon |U'|^2 + \int_0^T \int_\Gamma \frac{1}{2\epsilon} c |\overline{\mathcal{T}}'(U')n|^2 \\
& + \int_0^T \int_\Gamma \frac{1}{2}\epsilon C_{Dw} |U'|^2 + \int_0^T \int_\Gamma \frac{1}{2\epsilon} C_{c,\alpha,\gamma} |\overline{\mathcal{T}}'(U')n|^2
\end{aligned}
$$

The final inequality above uses Lemma 4.3.2. Next, apply the trace inequality and com-

bine like terms.

$$
\begin{aligned}
&\left|\int_0^T \int_\Gamma \langle (c + C_{c,\alpha,\gamma} Dw)U', \overline{\mathcal{T}}'(U')n\rangle\right| \\
\leq\ & \frac{1}{2}\epsilon(C_{Dw}+1)C_{\text{trace}} \int_0^T \|U'\|^2_{H^1(\Omega)} + \frac{1}{2\epsilon}\left(c + C_{c,\alpha,\gamma}\right)\int_0^T \int_\Gamma |\overline{\mathcal{T}}'(U')n|^2 \\
\leq\ & \frac{1}{2}\epsilon(C_{Dw}+1)C_{\text{trace}}C_{\text{equiv}}\left(\int_0^T |U'|^2_{1,\Omega} + \gamma\int_0^T \|U'\|^2_{L^2_\Theta(\Omega)}\right) \\
& + \frac{1}{2\epsilon}\left(c + C_{c,\alpha,\gamma}\right)\int_0^T \int_\Gamma |\overline{\mathcal{T}}'(U')n|^2
\end{aligned}
\tag{4.15}
$$

again using the equivalence of norms on $H^1(\Omega)$.

Combining the results from (4.13), (4.14), and (4.15) we see that

$$
\begin{aligned}
&\left|\int_0^T \int_\Gamma \left\langle \left(C_{c,\alpha,\gamma} - 1\right) w', \overline{\mathcal{T}}'(U')n\right\rangle\right| + \left|\int_0^T \int_\Gamma \langle w', \mathcal{B}(U')\rangle\right| \\
&+ \left|\int_0^T \int_\Gamma \langle (c + C_{c,\alpha,\gamma} Dw)U', \overline{\mathcal{T}}'(U')n\rangle\right| \\
\leq\ & \epsilon C_{\text{trace}} \int_0^T \|w'\|^2_{H^1(\Omega_f)} \\
&+ \left(\frac{1}{2\epsilon}C_{\mathcal{B}}C_{\text{equiv}} + \frac{1}{2}\epsilon(C_{Dw}+1)C_{\text{trace}}C_{\text{equiv}}\right)\left(\int_0^T |U'|^2_{1,\Omega} + \gamma\int_0^T \|U'\|^2_{L^2_\Theta(\Omega)}\right) \\
&+ \left(\frac{1}{\epsilon}C_{c,\alpha,\gamma} + \frac{1}{2\epsilon}\right)\left(\int_0^T \int_\Gamma |\overline{\mathcal{T}}'(U')n|^2\right)
\end{aligned}
\tag{4.16}
$$

So, first choose $\epsilon > 0$ small enough that $\epsilon C_{\text{trace}} \leq (1-\eta)\nu$. Next, based on the value of ϵ, choose our unspecified parameter c large enough that

$$
\frac{1}{2\epsilon}C_{\mathcal{B}}C_{\text{equiv}} + \frac{1}{2}\epsilon(C_{Dw}+1)C_{\text{trace}}C_{\text{equiv}} \leq \frac{c}{2}\,.
$$

Remark 4.2.1. *This is the point in the analysis that choosing $c = \alpha/2$ forces extra restrictions on the viscosity ν. With $c = \alpha/2$, the choice of ϵ is restricted, hence the*

viscosity must be larger to absorb the $\|w'\|^2_{H^1(\Omega_f)}$ *integral.*

Finally, define $C_2(c,\alpha,\gamma)$ by

$$C_2(c,\alpha,\gamma) = \frac{1}{\epsilon} C_{c,\alpha,\gamma} + \frac{1}{2\epsilon}\,.$$

This completes the justification of (4.12).

With this inequality in mind, and looking back at (4.11), we see that we need the boundary feedback coefficient K large enough to satisfy

$$K \geq \frac{C_2(c,\alpha,\gamma)}{C_{c,\alpha,\gamma}}.$$

In this case, the last term of (4.12) can be absorbed into the damping term $C_{c,\alpha,\gamma} K \int_0^T \int_\Gamma |\overline{\mathcal{T}}'(U')n|^2$ on the left of (4.11). Therefore (4.11) becomes

$$\begin{aligned} &E(T) + (1-\eta)\nu \int_0^T \|w'\|^2_{\mathbf{H}^1(\Omega_f)} + \alpha \int_0^T \|U_t'\|^2_{L^2_\Theta(\Omega)} + \frac{c}{2}\int_0^T |U'|^2_{1,\Omega} + \frac{c\gamma}{2}\int_0^T \|U'\|^2_{L^2_\Theta(\Omega)} \\ &\leq \left(1 + \frac{2c(1+\alpha)}{\gamma} + \frac{c}{\alpha}\right) E(0). \end{aligned}$$

The integral terms on the left-hand side of the latter dominate (a multiple of) the integral of the total energy $E(t)$. So, we have arrived at the following inequality with parameters for $C > 0$, $\delta > 0$, independent of the solution itself,

$$E(T) + \delta \int_0^T E(t)dt \leq CE(0) \quad \text{for any} \quad T > 0. \tag{4.17}$$

To complete the proof of exponential stability, we need only apply Lemma 4.3.4. Note that we may repeat the argument in this chapter replacing all time integrals over $[0,T]$ with time integrals over $[a,b]$ for any $0 \leq a \leq b$. This allows us to apply Lemma 4.3.4

immediately.

4.3 Important Lemmas

The following lemmas have been used in the preceeding sections of this chapter.

Lemma 4.3.1. *(Generalized Young's Inequality) Suppose that u and v are column vectors of the same size. Then for $\langle u, v\rangle = u^* v$ and $|w|^2 = \langle w, w\rangle$,*

$$\langle u, v\rangle \leq \frac{1}{2}\epsilon|u|^2 + \frac{1}{2\epsilon}|v|^2$$

for any $\epsilon > 0$. Furthermore, this generalizes to

$$\begin{aligned}
(U_1, U_2)_{L^2(\Omega)} &\leq \frac{1}{2}\epsilon\|U_1\|^2_{L^2(\Omega)} + \frac{1}{2\epsilon}\|U_2\|^2_{L^2(\Omega)} \\
&\textit{and} \\
(U_1, U_2)_{L^2_\Theta(\Omega)} &\leq \frac{1}{2}\epsilon\|U_1\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2\epsilon}\|U_2\|^2_{L^2_\Theta(\Omega)} \\
&\textit{and} \\
(U_1, U_2)_{L^2(\Gamma)} &\leq \frac{1}{2}\epsilon\|U_1\|^2_{L^2(\Gamma)} + \frac{1}{2\epsilon}\|U_2\|^2_{L^2(\Gamma)}
\end{aligned}$$

Proof. We can apply the Cauchy-Schwartz inequality to see

$$\langle u, v\rangle = u^* v \leq |u||v|$$

Finally, apply the scalar version of Young's inequality for any $\epsilon > 0$ to see

$$\langle u, v\rangle \leq |u||v| \leq \frac{\epsilon}{2}|u|^2 + \frac{1}{2\epsilon}|v|^2$$

To show the generalization,

$$\begin{aligned}
(U_1, U_2)_{L^2_\Theta(\Omega)} &= \int_\Omega \frac{1}{\det(\Theta)} U_1^* U_2 \\
&= \int_\Omega \frac{1}{\det(\Theta)} \langle U_1, U_2 \rangle \\
&\leq \int_\Omega \frac{1}{\det(\Theta)} \left(\frac{1}{2}\epsilon \langle U_1, U_1 \rangle + \frac{1}{2\epsilon} \langle U_2, U_2 \rangle \right) \\
&= \frac{1}{2}\epsilon \|U_1\|^2_{L^2_\Theta(\Omega)} + \frac{1}{2\epsilon} \|U_2\|^2_{L^2_\Theta(\Omega)}
\end{aligned}$$

The $L^2(\Omega)$ and $L^2(\Gamma)$ inequalities are similar. □

Lemma 4.3.2. *Suppose $V, W \in L^2_\Theta(\Omega)$ are vectors and $A : L^2_\Theta(\Omega) \to L^2_\Theta(\Omega)$ is a bounded linear operator where $\|A\|_{L^\infty(\Omega)} < k$. Then*

$$(V, AW)_{L^2_\Theta(\Omega)} \leq c \left(\|V\|^2_{L^2_\Theta(\Omega)} + \|W\|^2_{L^2_\Theta(\Omega)} \right)$$

for some constant multiple c of k. Similar results hold for $L^2(\Omega)$, $L^2(\Omega_f)$, and $L^2(\Gamma)$.

Proof.

$$\begin{aligned}
(V, AW)_{L^2_\Theta(\Omega)} &= \int_\Omega \frac{1}{\det(\Theta)} V^* AW \\
&\leq \int_\Omega \frac{1}{\det(\Theta)} |V||AW|
\end{aligned}$$

using the Cauchy-Schwartz inequality where $|\cdot|$ is the usual vector norm in $\mathbb{R}^n$. So,

$$(V, AW)_{L^2_\Theta(\Omega)} \leq \int_\Omega \frac{1}{\det(\Theta)} |V||A||W|$$

where $|A|$ is the matrix 2-norm of the operator A. Using the finite dimensionality of $\mathbb{R}^n$, we can consider the equivalent ∞-norm of A. There exists some fixed $k_1 > 0$ where $|A| \leq k_1 |A|_\infty$. Furthermore since $\|A\|_{L^\infty(\Omega)} < k$, there exists a fixed $k_2 > 0$ such that $|A|_\infty \leq k_2 k$. Thus

$$\begin{aligned}
(V, AW)_{L^2_\Theta(\Omega)} &\leq k_2 k \int_\Omega \frac{1}{\det(\Theta)} |V||W| \\
&\leq \frac{1}{2} k_2 k \int_\Omega \frac{1}{\det(\Theta)} \left(|V|^2 + |W|^2\right) \\
&= c \left(\|V\|^2_{L^2_\Theta(\Omega)} + \|W\|^2_{L^2\Theta(\Omega)} \right)
\end{aligned}$$

for $c = \frac{1}{2} k_2 k$. □

Lemma 4.3.3. *Suppose we have w' and w as defined in Chapter 2 where $\|Dw\|_{L^\infty(\Omega_f)} < k$ and $\|w\|_{L^\infty(\Omega_f)} < k$. Then there exists a $C > 0$ proportional to k such that*

$$\int_{\Omega_f} \langle w', Dw'w + Dww' \rangle \leq C \|w'\|^2_{H^1(\Omega_f)} .$$

Proof. Notice $\int_{\Omega_f} \langle w', (Dw')w + (Dw)w' \rangle = \int_{\Omega_f} \langle w', (Dw')w \rangle + \int_{\Omega_f} \langle w', (Dw)w' \rangle$. We will handle each term independently.

Using lemma 4.3.2, there exists a $c_1 > 0$ such that

$$\begin{aligned}
\int_{\Omega_f} \langle w', (Dw)w' \rangle &\leq c_1 \left(\|w'\|^2_{L^2(\Omega_f)} + \|w'\|^2_{L^2(\Omega_f)} \right) \\
&= 2c_1 \|w'\|^2_{L^2(\Omega_f)} \\
&\leq 2c_1 \|w'\|^2_{L^2(\Omega_f)} + 2c_1 \int_{\Omega_f} |Dw'|^2 \\
&\leq c_2 \int_{\Omega_f} \varepsilon(w') .. \varepsilon(w') .
\end{aligned}$$

using the equivalence of $H^1_{\Gamma_f}(\Omega_f)$ norms in Proposition 2.2.1 where c_2 is a constant multiple of c_1.

Next, use the $\mathbb{R}^n$ version of the Cauchy-Schwartz inequality to see

$$\begin{aligned}\int_{\Omega_f} \langle w', (Dw')w\rangle &= \int_{\Omega_f} (w')^*(Dw')w \\ &\leq \int_{\Omega_f} |w'||(Dw')w| \\ &\leq \int_{\Omega_f} |w'||Dw'||w|\,.\end{aligned}$$

Then we may follow a similar argument found in the proof of lemma 4.3.2 to see

$$\begin{aligned}\int_{\Omega_f} |w'||Dw'||w| &\leq k\int_{\Omega_f} |w'||Dw'| \\ &\leq \frac{1}{2}k\int_{\Omega_f} \left(|w'|^2 + |Dw'|^2\right) \\ &= \frac{1}{2}k\|w'\|^2_{H^1(\Omega_f)}\,.\end{aligned}$$

Then, choose $C > 0$ such that $\frac{1}{2}C = \max\{\frac{1}{2}k, c_2\}$. Either way, C is a multiple of k. Therefore when we add our inequalities,

$$\int_{\Omega_f} \langle w', (Dw')w + (Dw)w'\rangle \leq \frac{1}{2}C\|w'\|^2_{H^1(\Omega_f)} + \frac{1}{2}C\|w'\|^2_{H^1(\Omega)} = C\|w'\|^2_{H^1(\Omega)}$$

□

Note that as we follow the proof of lemma 4.3.3, as k decreases, the choice of C will also decrease.

Lemma 4.3.4. *Suppose $f : [0, \infty) \to [0, \infty)$ is continuous and for all $0 \leq a \leq b$*

$$f(b) + c_1 \int_{t=a}^{b} f(t)dt \leq c_2 f(a)$$

for some $c_1, c_2 > 0$. Then there exist $C, \delta > 0$ such that for all $t \in [0, \infty)$

$$f(t) \leq Ce^{-\delta t}$$

Proof. Since $f(t) \geq 0$ for all $t > 0$ and $c_1 > 0$, we have $f(b) \leq c_2 f(a)$ for all $a \leq b$. Rewrite this in terms of t to see that $(1/c_2)f(b) \leq f(t)$ for all $t \leq b$. Then from the hypothesis inequality,

$$\begin{aligned} f(b) + \frac{c_1}{c_2} \int_{t=a}^{b} f(b) &\leq c_2 f(a) \\ \Rightarrow f(b) + \frac{c_1}{c_2}(b-a)f(b) &\leq c_2 f(a) \\ \left(1 + \frac{c_1}{c_2}(b-a)\right) f(b) &\leq c_2 f(a) \end{aligned}$$

for all $0 \leq a \leq b$. If $a = 0$ and b is large, we have

$$f(b) \leq \frac{c_2}{1 + (c_1/c_2)b} f(0) = c_3 f(0) \tag{4.18}$$

where $0 < c_3 < 1$. We can prove by induction that for all $m \in \mathbb{N}$,

$$f(mb) \leq c_3^m f(0) \tag{4.19}$$

The $m = 1$ case is already complete. Assume that $f((m-1)b) \leq c_3^{m-1} f(0)$. Then

$$f(mb) \leq \left(\frac{c_2}{1 + (c_1/c_2)(mb - (m-1)b)} \right) f((m-1)b) = c_3 f((m-1)b) \leq c_3^m f(0)$$

using (4.18) and the induction hypothesis. Therefore for all $m \in \mathbb{N}$, $f(mb) \leq c_3^m f(0)$.

Let $t \in [0, \infty)$ be fixed. Then there exists an $m_t \in \mathbb{N}$ such that $(m_t - 1)b \leq t < m_t b$. So

$$\begin{aligned} f(t) &\leq c_2 f((m_t - 1)b) \\ &\leq c_2 c_3^{(m_t - 1)} f(0) \\ &= \frac{c_2}{c_3} c_3^{m_t} f(0) \\ &\leq \frac{c_2}{c_3} c_3^{t/b} f(0) \end{aligned}$$

since $0 < c_3 < 1$ and $t/b < m_t$. Further since $0 < c_3 < 1$, we may choose a $\delta > 0$ such that $e^{-\delta} = c_3^{1/b}$. Thus for any $t \in [0, \infty)$ with $C = \frac{c_2}{c_3} f(0) > 0$,

$$f(t) \leq C e^{-\delta t}$$

$\square$

Chapter 5

Conclusions and Future Work

Overall, there is a great deal of value in studying the total linearization developed in [11, 12]. Here, we summarize the important conclusions explained earlier in this work as well as point out future work in this area that could be done.

For the numerical sensitivity analysis, the primary observations are the following:

(i) The total linearization is more sensitive with respect to the lower-order terms on the common boundary which depend on the curvature of the interface.

(ii) Large zero-order terms on the boundary and large velocity of the steady regime around which linearization is performed lead to singularities independently of the size of the viscosity (confirming the observations in [9]).

(iii) Larger viscosity renders the condition numbers of the matrices in the discretization more robust with respect to the parameter change.

(iv) Within the admissible parameter range, the FEM convergence rates in a test case do not seem to be influenced by the dynamics on the interface and match (slightly better than) those derived in [7].

All these considerations, of course, do not apply to the standard coupling of linear Stokes flow and system of linear elasticity, where the boundary terms arising from the linearization of the common moving boundary are absent.

When considering the question of feedback stability, the total linearization also provides some interesting conclusions. First, unlike the classical Stokes-elasticity coupling, the total linearization appears to be strictly ω-dissipative. This understanding led to considering feedback not only on the boundary interface as in the Stokes-elasticity, but also in the interior of the elastic solid. With the resulting dissipative system (4.1) (along with an additonal assumption on the viscosity of the fluid beyond the well-posedness assumptions), exponential stability is achieved. The analysis in Chapter 4 also differs from the classical Stokes-elasticity feedback stabilization in that the coefficient for interface damping depends on the size of the steady regime solution that the total linearization is centered about, among other parameters in the damped system.

The next milestones in our understanding of the discussed hydro-elasticity model include the following projects:

(i) Implement a time-stepping solver to simulate the evolution of the fluid-solid interaction for the prototype system.

(ii) Test the impact of the damping terms (interior viscous on the solid and the interface feedback) on the stability of the prototype system. Here we would expect to recover exponential decay rates.

(iii) Apply the numerical scheme to the original system instead of the prototype model. To this end we will use an elements with curved boundaries to model the outer and the interface boundaries. The state around which linearization is performed (described by an arbitrary divergence free function in the prototype system) will be

replaced by an analytic solution to a simple nonlinear steady regime, e.g., steady circulation around a cylinder. At this point we will be able to compare our simulations to the Stokes-elasticity model and to the available experimental data.

(iv) It is known that the Stokes-elasticity model is exponentially stable in the presence of the interface feedback alone, without any dissipation in the solid, provided the geometry of the domain satisfies certain conditions. The question whether the interior feedback on the solid ($\alpha, \gamma > 0$) is always necessary for the exponential stability of the total linearization remains open. This aspect will be investigated numerically with the help of the above tools. This problem will also be studied theoretically by employing suitable generalizations of weighted multiplier techniques in order to obtain a bound on the potential energy of the solid in terms of the damping term $|\overline{\mathcal{T}}'(U')|$ on the boundary.

Bibliography

[1] G. Avalos and R. Triggiani. The coupled PDE system arising in fluid/structure interaction. I. Explicit semigroup generator and its spectral properties. In *Fluids and waves*, volume 440 of *Contemp. Math.*, pages 15–54. Amer. Math. Soc., Providence, RI, 2007.

[2] George Avalos and Matthew Dvorak. A new maximality argument for a coupled fluid-structure interaction, with implications for a divergence-free finite element method. *Appl. Math. (Warsaw)*, 35(3):259–280, 2008.

[3] George Avalos and Roberto Triggiani. Boundary feedback stabilization of a coupled parabolic-hyperbolic Stokes-Lamé PDE system. *J. Evol. Equ.*, 9(2):341–370, 2009.

[4] George Avalos and Roberto Triggiani. Semigroup well-posedness in the energy space of a parabolic-hyperbolic coupled Stokes-Lamé PDE system of fluid-structure interaction. *Discrete Contin. Dyn. Syst. Ser. S*, 2(3):417–447, 2009.

[5] George Avalos and Roberto Triggiani. Backwards uniqueness of the C_0-semigroup associated with a parabolic-hyperbolic Stokes-Lamé partial differential equation system. *Trans. Amer. Math. Soc.*, 362(7):3535–3561, 2010.

[6] H. Beirão da Veiga. On the existence of strong solutions to a coupled fluid-structure evolution problem. *Journal of Mathematical Fluid Mechanics*, 6(1):21–52, 2004.

[7] M. Bercovier and O. Pironneau. Error estimates for finite element method solution of the Stokes problem in the primitive variables. *Numer. Math.*, 33(2):211–224, 1979.

[8] Lorena Bociu, Steven Derochers, and Daniel Toundykov. Linearized hydro-elasticity: A numerical study. *Evolution Equations and Control Theory*, 5(4):533–559, 2016.

[9] Lorena Bociu, Daniel Toundykov, and Jean-Paul Zolésio. Well-posedness analysis for the total linearization of a fluid-elasticity interaction. *SIAM J. Math. Anal.*, 47(3):1958–2000, 2015.

[10] Lorena Bociu and Jean-Paul Zolésio. Existence for the linearization of a steady state fluid/nonlinear elasticity interaction. *Discrete Contin. Dyn. Syst.*, (Dynamical systems, differential equations and applications. 8th AIMS Conference. Suppl. Vol. I):184–197, 2011.

[11] Lorena Bociu and Jean-Paul Zolésio. Linearization of a coupled system of nonlinear elasticity and viscous fluid. In *Modern aspects of the theory of partial differential equations*, volume 216 of *Oper. Theory Adv. Appl.*, pages 93–120. Birkhäuser/Springer Basel AG, Basel, 2011.

[12] Lorena Bociu and Jean-Paul Zolésio. Sensitivity analysis for a free boundary fluid-elasticity interaction. *Evol. Equ. Control Theory*, 2(1):55–79, 2013.

[13] Muriel Boulakia. Existence of weak solutions for the three-dimensional motion of an elastic structure in an incompressible fluid. *J. Math. Fluid Mech.*, 9(2):262–294, 2007.

[14] Jacques Chazarain and Alain Piriou. Caractérisation des problèmes mixtes hyperboliques bien posés. *Ann. Inst. Fourier (Grenoble)*, 22(4):193–237, 1972.

[15] Philippe G. Ciarlet. *Linear and nonlinear functional analysis with applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013.

[16] H. Cohen and S. I. Rubinow. Some mathematical topics in biology. In *Proc. Symp. on System Theory*, pages 321–337. Polytechnic Press, New York, 1965.

[17] Carlos. Conca, Jacques. Planchard, Bernadette. Thomas, and Robert Dautray. *Problèmes mathématiques en couplage fluide-structure: applications aux faisceaux tubulaires*. Editions Eyrolles, Paris, 1994.

[18] Carlos Conca, Jorge San Martín H., and Marius Tucsnak. Existence of solutions for the equations modelling the motion of a rigid body in a viscous fluid. *Comm. Partial Differential Equations*, 25(5-6):1019–1042, 2000.

[19] Daniel Coutand and Steve Shkoller. The interaction between quasilinear elastodynamics and the Navier-Stokes equations. *Arch. Ration. Mech. Anal.*, 179(3):303–352, 2006.

[20] M. C. Delfour and Jean-Paul Zolésio. *Shapes and geometries*, volume 22 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2011. Metrics, analysis, differential calculus, and optimization.

[21] B. Desjardins and M. J. Esteban. Existence of weak solutions for the motion of rigid bodies in a viscous fluid. *Arch. Ration. Mech. Anal.*, 146(1):59–71, 1999.

[22] B. Desjardins, M. J. Esteban, C. Grandmont, and P. Le Tallec. Weak solutions for a fluid-elastic structure interaction model. *Rev. Mat. Complut.*, 14(2):523–538, 2001.

[23] J. Donea, S. Giuliani, and J.P. Halleux. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 33(1–3):689 – 723, 1982.

[24] Thierry Fanion, Miguel Fernández, and Patrick le Tallec. Deriving adequate formulations for fluid-structure interaction problems: from ALE to transpiration. In *Fluid-structure interaction*, Innov. Tech. Ser., pages 51–79. Kogan Page Sci., London, 2003.

[25] Eduard Feireisl. On the motion of rigid bodies in a viscous incompressible fluid. *J. Evol. Equ.*, 3(3):419–441, 2003. Dedicated to Philippe Bénilan.

[26] Miguel Ángel Fernández and Patrick Le Tallec. Linear stability analysis in fluid-structure interaction with transpiration. I. Formulation and mathematical analysis. *Comput. Methods Appl. Mech. Engrg.*, 192(43):4805–4835, 2003.

[27] Miguel Ángel Fernández and Patrick Le Tallec. Linear stability analysis in fluid-structure interaction with transpiration. II. Numerical analysis and applications. *Comput. Methods Appl. Mech. Engrg.*, 192(43):4837–4873, 2003.

[28] Miguel Ángel Fernández and Marwan Moubachir. An exact block-Newton algorithm for solving fluid-structure interaction problems. *C. R. Math. Acad. Sci. Paris*, 336(8):681–686, 2003.

[29] G. P. Galdi. *An introduction to the mathematical theory of the Navier-Stokes equations*. Springer Monographs in Mathematics. Springer, New York, second edition, 2011. Steady-state problems.

[30] A. R. Galper and T. Miloh. Motion stability of a deformable body in an ideal fluid with applications to the N spheres problem. *Phys. Fluids*, 10(1):119–130, 1998.

[31] Adelina Georgescu. *Hydrodynamic stability theory*, volume 9 of *Mechanics: Analysis*. Martinus Nijhoff Publishers, Dordrecht, 1985. Translated from the Romanian, Translation edited by David Sattinger.

[32] C. Geuzaine and J.F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[33] Céline Grandmont. Existence for a three-dimensional steady state fluid-structure interaction problem. *J. Math. Fluid Mech.*, 4(1):76–94, 2002.

[34] Céline Grandmont and Yvon Maday. Existence for an unsteady fluid-structure interaction problem. *M2AN Math. Model. Numer. Anal.*, 34(3):609–636, 2000.

[35] Céline Grandmont and Yvon Maday. Fluid-structure interaction: a theoretical point of view. In *Fluid-structure interaction*, Innov. Tech. Ser., pages 1–22. Kogan Page Sci., London, 2003.

[36] Max D. Gunzburger, Hyung-Chun Lee, and Gregory A. Seregin. Global existence of weak solutions for viscous incompressible flows around a moving rigid body in three dimensions. *J. Math. Fluid Mech.*, 2(3):219–266, 2000.

[37] Kenneth C. Hall and William S. Clark. Linearized euler predictions of unsteady aerodynamic loads in cascades. *AIAA Journal*, 31(3):540–550, 2016/08/07 1993.

[38] Kenneth C. Hall and Edward F. Crawley. Calculation of unsteady flows in turbomachinery using the linearizedeuler equations. *AIAA Journal*, 27(6):777–787, 2016/08/07 1989.

[39] K.-H. Hoffmann and V. N. Starovoitov. On a motion of a solid body in a viscous fluid. Two-dimensional case. *Adv. Math. Sci. Appl.*, 9(2):633–648, 1999.

[40] Thomas J. R. Hughes, Wing Kam Liu, and Thomas K. Zimmermann. Lagrangian-Eulerian finite element formulation for incompressible viscous flows. *Comput. Methods Appl. Mech. Engrg.*, 29(3):329–349, 1981.

[41] Mihaela Ignatova, Igor Kukavica, Irena Lasiecka, and Amjad Tuffaha. On well-posedness and small data global existence for an interface damped free boundary fluid-structure model. *Nonlinearity*, 27(3):467–499, 2014.

[42] Mitsuru Ikawa. A mixed problem for hyperbolic equations of second order with non-homogeneous Neumann type boundary condition. *Osaka J. Math.*, 6:339–374, 1969.

[43] S. Kesavan. *Topics in functional analysis and applications*. John Wiley & Sons Inc., New York, 1989.

[44] M. Lesoinne and Charbel Farhat. Re-engineering of an aeroelastic code for solving eigen problems in all flight regimes. In *4th European Computational Fluid Dynamics Conference, Athens, Greece*, pages 1052–1061, 1998.

[45] Michel Lesoinne, Marcus Sarkis, Ulrich Hetmaniuk, and Charbel Farhat. A linearized method for the frequency analysis of three-dimensional fluid/structure interaction problems in all flow regimes. *Computer Methods in Applied Mechanics and Engineering*, 190(24–25):3121 – 3146, 2001. Advances in Computational Methods for Fluid-Structure Interaction.

[46] M. J. Lighthill. On displacement thickness. *J. Fluid Mech.*, 4:383–392, 1958.

[47] G. Mortchéléwicz. Application of linearized euler equations to flutter. In *85th AGARD SMP Meeting, Aalborg, Denmark*, 1997.

[48] B. Palmerio. A two-dimensional FEM adaptive moving-node method for steady Euler flow simulations. *Computer Methods in Applied Mechanics and Engineering*, 71(3):315 – 340, 1988.

[49] A. Pazy. *Semigroups of linear operators and applications to partial differential equations*, volume 44. Springer-Verlag, 1983.

[50] Serge Piperno and Charbel Farhat. Design of efficient partitioned procedures for the transient solution of aeroelastic problems. In *Fluid-structure interaction*, Innov. Tech. Ser., pages 23–49. Kogan Page Sci., London, 2003.

[51] P. Raj and B. Harris. Using surface transpiration with an Euler method for cost-effective aerodynamic analysis. In *AIAA 24th Applied Aerodynamics Conference, 93-3506, Monterey, Canada*, 1993.

[52] Reiko Sakamoto. *Hyperbolic boundary value problems*. Cambridge University Press, Cambridge, 1982. Translated from the Japanese by Katsumi Miyahara.

[53] Jorge Alonso San Martín, Victor Starovoitov, and Marius Tucsnak. Global weak solutions for the two-dimensional motion of several rigid bodies in an incompressible viscous fluid. *Arch. Ration. Mech. Anal.*, 161(2):113–147, 2002.

[54] James Serrin. Mathematical principles of classical fluid mechanics. In *Handbuch der Physik (herausgegeben von S. Flügge), Bd. 8/1, Strömungsmechanik I (Mitherausgeber C. Truesdell)*, pages 125–263. Springer-Verlag, Berlin-Göttingen-Heidelberg, 1959.

[55] Takéo Takahashi and Marius Tucsnak. Global strong solutions for the two-dimensional motion of an infinite cylinder in a viscous fluid. *J. Math. Fluid Mech.*, 6(1):53–77, 2004.

[56] Nomura Takashi and Thomas J.R. Hughes. An arbitrary Lagrangian-Eulerian finite element method for interaction of fluid and a rigid body. *Computer Methods in Applied Mechanics and Engineering*, 95(1):115 – 138, 1992.

[57] P. Le Tallec and J. Mouro. Fluid structure interaction with large structural displacements. *Computer Methods in Applied Mechanics and Engineering*, 190(24–25):3039 – 3067, 2001. Advances in Computational Methods for Fluid-Structure Interaction.

[58] J. Tambača, M. Kosor, S. Čanić, and D. Paniagua. Mathematical modeling of vascular stents. *SIAM J. Appl. Math.*, 70(6):1922–1952, 2010.

[59] Michael E. Taylor. *Tools for PDE*, volume 81 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2000. Pseudodifferential operators, paradifferential operators, and layer potentials.

[60] T. Wick and W. Wollner. On the differentiability of fluid-structure interaction problems with respect to the problem data. Technical Report 2014-16, RICAM, 2014.

APPENDIX

Appendix A

Matlab Codes

This appendix shows the Matlab codes used in the numerical analysis for the total linearization found in Chapter 3.

The main code is found first in main.m, and the rest of the codes follow alphabetically. The main.m code first calls getMesh.m (which calls load_gmsh2.m) to gather required information from a non-uniform mesh. This mesh is specified by the solidfluid.msh file generated by the gmesh software [32]. Higher mesh levels are handled similarly with larger .msh files. Next, the stiffness matrix for the extensions into the solid domain is calculated in calcExtenStiff.m. Then, depending on user input for the desired type of problem to solve, main.m calls SolveEigenfunction.m, calcExtenProb.m, calcVUvec.m, and/or calcUbarExt.m to finish solving the appropriate extension problems into the solid domain. With this information, main.m calls buildsystem.m to build the overarching Babuška-Brezzi matrix system. Finally, the code calls AdjustFluidBdry.m to account for the outer boundary conditions and outputs desired information on coefficient vectors, eigenvalues, and condition numbers.

Throughout this process, the appropriate input functions for the finite element frame-

work are called. This includes the piecewise quadratic basis functions, Phi.m, their derivatives, DiffPhi.m, the linear basis functions, Eta.m, as well as the functions inputted on the right hand side of the maximality system (2.27): w1.m, U1.m, and V1.m. Finally, the fluid velocity that the system was linearized about is simulated by wplain.m. The choice for wplain.m is reflected in lamDw.m which outputs $(\lambda I - Dw)$ for use in the extension problems.

In all instances where integrals over mesh elements need to be calculated numerically, the files getMapInfo.m, getGlobalNodeNum.m, and getTangVec.m are called as needed. These integrals are performed over a reference triangle, hence information on the proper affine mapping is required. The getMapInfo.m code gathers relevant information on the affine mapping xK from the reference triangle to the mesh element by calling xKmap.m, xKJac.m, xKinvMap.m, and DxKinvMap.m for the mapping, its Jacobian, the inverse mapping, and the derivative of the inverse mapping respectively. The getTangVec.m code provides the unit tangent vector on the boundary interface in the counter-clockwise direction. The quadrature is calculated with a user defined precision using quadTri.m. Finally, getGlobalNodeNum.m outputs the global node number for a node for proper entry placement of the integral contribution. Line integrals over the boundary interface are handled similarly with quadLine.m where quadLine.m integrates on mesh edges and calls getMapInfo.m for the affine xK mapping to transfer basis functions to the mesh.

The SolveEigenfunction.m code calculates an approximation for the coefficient vector of U' in the eigenfunction test case from Section 3.2.3. This approximation is then fed into the remainder of the algorithm serving as U_1. Specifically, it is used as an input for calcUbarExt.m which solves for $D_{S,\lambda^2}[\lambda U_1, (\lambda I - Dw)^{-1} U_1]$ in the eigenfunction case (where $V_1 = \mathbf{0}$).

The calcExtenProb.m code creates a library of FEM coefficient vector solutions to

the extension problems $D_{S,\lambda^2}[0,\varphi]$ and $D_{S,\lambda^2}[0,(\lambda I - Dw)^{-1}\varphi]$. Similarly, calcVUvec.m calculates a FEM coefficient vector for $D_{S,\lambda^2}[V_1 + \lambda U_1, (\lambda - Dw)^{-1}U_1]$.

In buildsystem.m, the code constructs the overarching Babuška-Brezzi matrix system. To complete this task, the code calls alambdaEx.m and GEx.m to incorporate the extension terms in the $a_\lambda(\cdot,\cdot)$ product and $G(\cdot)$ operator respectively.

The errorExp.m code takes in a vector of errors and outputs the exponent of decay and plots a graph showing how the exponent of decay changes with respect to mesh level.

A.1 main.m

```
function [out,L2fluid,H1fluid,coef,alpha,beta,vec1,uprime,Ubar,S,L2,
L2norm_alpha,L2norm_beta_error] = main( meshlevel,testflag,const )
%==================================================================
%Code for the the intermediate elasticity problem
%  Details:
%      meshlevel = 1,2,3,...,7 is the desired mesh refinement
%                 level.
%      const = vector of parameter constants to use in the
%b            calculations
%        const = [lambda Lambda viscosity e1 e2 e3 epsilon]
%      testflag = flag to determine whether to use test problem
%                or not
%        testflag=0 means use the eigenvalue test problem
%        testflag=1 means use the analytic solution test problem
%        testflag=2 means only looking at eigenvalues and
```

```
%        condition numbers with nonzero wplain.m and Dw
%        testflag = 3 means use the second analytic solution
%                 test problem
%      out = vector of condition numbers, minimum eigenvalues, a
%           record of inputs leading to those results, errors
%        out=[ 3 condition #'s, 3 eigenvalues, const, meshlevel,
%            L2norm_alpha_error, H1norm_alpha_error, ...
%            L2norm_beta_error, H1_uprime_error ]
%      coef = [alpha; beta] the coefficient vectors for w and pi
%
%
% Note: We exclude cases where the mesh in the solid domain does
% not allow a triangular element to have two edges on the interface
%=================================================================

%=========================================================
%   Initial Processes
%=========================================================
%clc
close all
clear Mdpts msh solperimeter solidpts S preS solperimeter
clear solidpts fluidpts
clear NODECO M L ELNODE nMdpts outerbdrynodes solidbdrynodes
clear nsolperimeter
clear nsolidpts nfluidpts ELsolid nELsolid ELfluid nELfluid
clear edgetri
```

```
clear nedgetri searchtriangles otherDim
more off

addpath('Mesh Files')
tic;

disp('      ---------------------------------------------')
disp('     |   Solving Intermediate Elasticity Problem |')
disp('      ---------------------------------------------')
disp(' ')
disp('Beginning Calculations:')
disp('----------------------')


%==================================================================
%   Global Variables and Important Constants
%==================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information;
                      % see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
```

```
global solperimeter  % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter % Number of nodes on the solid boundary
global solidpts      % List of global node numbers on Solid Domain
global nsolidpts     % Number of nodes in the solid domain
global fluidpts      % List of global node numbers on Fluid Domain
global nfluidpts     % Number of nodes in the fluid domain
global ELsolid       % Global node numbers of solid domain elements
global nELsolid      % Number of triangular elements in solid domain
global ELfluid       % Global node numbers of fluid domain elements
global nELfluid      % Number of triangular elements in solid domain
global edgetri       % List of triangular elements with two nodes on solid
   boundary
global nedgetri      % Number of triangular elements with two nodes
                     % on solid boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim      % Number of fluid vertices (also # of global
                          % Eta's)
global lambda        % Dissipativity constant given in problem
global pressure      % The constant pressure function
global T             % Number of local basis functions Phi
                     % (12; 6 in each coord)
global T2            % Number of local basis functions Eta
                     % (6; 3 in each coord)

global e1 e2 e3   % Epsilon constants for intermediate problem
global Lambda     % Parameter introduced to guarantee ellipticity
```

```
global mu lam     % Lame constants
global nu         % Viscosity Constant
global kappa      % Coefficient of Theta mapping
%-------------------------------------------------------------------------

T=12;     %We are using 6 phi_i's, make a basis for H_0^1(Omega)
          %  Functions of form Phi=[phi;0] and Phi=[0;phi] make a
          %  basis for [H_0^1(Omega)]^2
T2=3;     %We are using 3 eta_i's

%--------------------
%const=[lambda Lambda nu e1 e2 e3 epsilon]=input vector of constants
lambda=const(1);
Lambda=const(2);
nu=const(3);
e1=const(4);
e2=const(5);
e3=const(6);
epsilon=const(7);
%--------------------
kappa=1+epsilon;  % Theta will be taken to be
                  % (1+epsilon)*Id=kappa*Id


lam=1;        %Lame constant (notated lambda in paper);
              % beware confusion
```

```
              % with the dissipitivity lambda above
mu=1;         %Lame constant
pressure=1;   %constant pressure used in the eigenfunction problem
if testflag==1
   pressure=0;
   disp('Using pressure=0 in the analytic solution #1 test case...')
end
if testflag==3
   pressure=e2;
   disp('Using pressure=e2 in the analytic solution #2 test case...')
   if e1~=0
      error('Error: e1 must be 0 in the testflag=3 case...')
   end
end
%{
lambda=5;     %Constant from dissipativity argument
Lambda=5;     %constant depending on lambda
nu=1;         %viscosity constant nu>0
e1=0.1;       %epsilon constants; they should be small
e2=0.1;       %compared to lambda
e3=0.1;
epsilon=0.1;
%}
```

```
%=====================================================================
%   Gather Mesh and Boundary Information
%=====================================================================
[msh,Mdpts,cons,solidpts,fluidpts,solperimeter,...
   outerbdrynodes,ELsolid,ELfluid,edgetri]=getMesh(meshlevel);
M=msh.nbNod;                % Number of vertices in mesh
NODECO=msh.POS(1:M,1:2);    % Vertex Coordinates
L=msh.nbTriangles;          % Number of elements (triangular)
ELNODE=msh.TRIANGLES(1:L,1:4); % Global node number for vertices
nELsolid=cons(1);           % Number of elements in solid domain
nELfluid=cons(2);           % Number of elements in fluid domain
otherDim=cons(3);           % Number of fluid vertices
                            % (w/out mdpts)
nsolidpts=cons(4);          % Number of nodes in solid domain
nfluidpts=cons(5);          % Number of nodes in fluid domain
                            % (w/mdpts)
nMdpts=cons(6);             % Number of all midpoints in mesh
```

```
nsolperimeter=cons(7);       % Number of nodes on Gamma_S,
                             % solid bdry
nedgetri=cons(8);            % Number of elements in solid
                             % touching bdry


%=====================================================================
%   Calculate Extension Stiffness Matrix
%=====================================================================
[H1solid,preS,S,trueH1solid,L2]=calcExtenStiff();
% H1solid = matrix formed by (Phi_i,Phi_j)_{H^1} with the equivalent
% H^1 norm
%   - used as a shortcut to taking integrals
%   - lambda^2(u,v)_L2+(u,v)_H1 = u^* H1solid v
% preS = true stiffness matrix formed by a_S(Phi_i,Phi_j) without
%   replacing boundary equations with the identity
%   - needed to solve the non-homogeneous Dirichlet BC problem
%     D_{S,lambda}
% S = stiffness matrix with boundary equations replaced by identity
% L2 = L2 mass matrix over the solid domain with quadratic basis
% elements


%=====================================================================
%   Solve Eigenfunction Test Case
%=====================================================================
```

```
% Here we solve for Ubar leading to the eigenfunction of the overall
% system in the testflag=0 case
if testflag==0
    [U1bar]=SolveEigenfunction(testflag);
    rawU1bar=U1bar; %rename here to make inputs easier in
                                %buildsystem.m
end
% Ubar is the coefficient vector to the eigenfunction problem:
%                        S(U) = 0
%   Tbarprime(U)*normal - B(U) = -pressure*normal
% The solution [U;0;0] is an eigenfunction of the original
% dissipativity equation



%=====================================================================
%    Solve Extension Problems
%=====================================================================
[AllDPhi, AllDPsi]=calcExtenProb(preS,S,testflag);

if testflag==0    %Eigenfunction Test Case
    VUvec=calcUbarExt(preS,S,U1bar,testflag);
    rawV1bar=zeros(size(U1bar));
elseif testflag==1 %Analytic Solution Test Case #1
    [VUvec,rawU1bar,rawV1bar]=calcVUvec(preS,S,testflag);
elseif testflag==2 %Only looking for eigenvalues and condition
                        %numbers
```

```
    VUvec=zeros(2*nsolidpts,1);
    rawU1bar=zeros(2*nsolidpts,1);
    rawV1bar=zeros(2*nsolidpts,1);
elseif testflag==3 %Analytic Solution Test Case #2
    [VUvec,rawU1bar,rawV1bar]=calcVUvec(preS,S,testflag);
elseif testflag==4    %Only looking for eigenvalues and
                      %condition numbers
                      %with variable weighting on e2
    VUvec=zeros(2*nsolidpts,1);
    rawU1bar=zeros(2*nsolidpts,1);
    rawV1bar=zeros(2*nsolidpts,1);
end
% AllDPhi = matrix of coefficient vector columns
%   - each column is the coefficient vector of
%     sol=D_{S,lambda^2}( 0, Phi )
% AllDPsi = matrix of coefficient vector columns
%   - each col is the coef vec of
%     sol=D_{S,lambda^2}(0,inv(lambda-Dw)Phi)
% VUvec= coef vec of
%        D_{S,lambda^2}(V1+lambda U1,inv(lambda-Dw)U1)
% U1bar = coef vec of U1(p)???
% V1bar = coef vec of V1(p)???
disp('Finished solving extension problems.')


%=====================================================================
```

```
%   Build the Overarching Babushka Brezzi System of Equations
%=================================================================
[A,B,G,A1,L2fluid,smallL2fluid,H1fluid,...
vec1,U1bar,V1bar]=buildsystem(...
                                  H1solid,AllDPhi,...
                                  AllDPsi,VUvec,L2,...
                                  rawU1bar,rawV1bar,testflag);
%build component matrices and relevant mass matrices


zilch=zeros(otherDim,otherDim);
BigMatrix=[A B; B' zilch]; %This is the overarching stiffness matrix
                        % w/out adjusting for the









% Gather Condition Numbers and Eigenvalue Data
%condBig=condest(BigMatrix);
%condA1=condest(A1);
%condA=condest(A);
condBig=0; condA1=0; condA=0;
```

```
disp('Done finding all condition numbers...')
disp('Edit: did not calculate condition numbers b/c they are not
 needed')

%eigBig=eigs(BigMatrix+BigMatrix',1,'sa');
eigBig=0; %we don't need the minimum eigenvalue for BB matrix

eigA=eigs(A+A',1,'sa');
%eigA1=eigs(A1+A1',1,'sa');
eigA1=0; %don't need the minimum eigenvalue for A1 matrix
disp('Done finding all minimum eigenvalues...')
%}

[BigMatrix,G]=AdjustFluidBdry(A,B,G); %Adjust system for fluid bdry

% Display Data
fprintf('\n\nUsing Input:\n -- const =[%d %d %d %d %d %d %d]\n',...
 const)
fprintf(' -- Meshlevel = %d\n',meshlevel)
fprintf('Our system has condition numbers and minimum
   eigenvalues:\n')
fprintf(' -- cond(BigMatrix) = %d',condBig)
fprintf('\n -- cond(A) = %d',condA)
fprintf('\n -- cond(A1) = %d\n',condA1)
fprintf(' -- mineig(BigMatrix+BigMatrix^*) = %d',eigBig)
fprintf('\n -- mineig(A+A^*) = %d',eigA)
```

```
fprintf('\n -- mineig(A1+A1^*) = %d\n',eigA1)
%{
if meshlevel>3
  fprintf('Warning: Mesh level above 3 means eig and cond not
         accurate\n')
end
%}
out=[condBig condA condA1 eigBig eigA eigA1 const meshlevel];



%=====================================================================
%   Solve System for Coefficient Vector
%=====================================================================
coef=BigMatrix\G;

resid_error=BigMatrix*coef-G;
disp(' ')
disp(['The absolute residual error from solving [A B;B^T 0]*coef=G
   is '...
    ,num2str(norm(resid_error))]);
alpha=coef(1:2*nfluidpts,1);
beta=coef(2*nfluidpts+1:end,1);
```

```
%=====================================================================
%   Analyze and Output Results
%=====================================================================
max_alpha=max(abs(alpha));
avg_beta=mean(beta);
fprintf('\nCoef=[alpha; beta] Results:\n')
fprintf(' -- max_alpha = %d\n -- avg_beta = %d \n\n',max_alpha,
avg_beta)

L2norm_alpha=alpha'*L2fluid*alpha;
L2norm_alpha=sqrt(L2norm_alpha);



%L2norm_alpha_dvorak=alpha'*L2dvorak*alpha; %using the mass matrix
%from dvorak's code; this checks to be the same as using hte L2fluid
%mass matrix defined in this code
%L2norm_alpha_dvorak=sqrt(L2norm_alpha_dvorak);



H1norm_alpha=alpha'*H1fluid*alpha;
H1norm_alpha=sqrt(H1norm_alpha);
L2norm_beta=beta'*smallL2fluid*beta;
L2_beta_error=(beta-vec1)'*smallL2fluid*(beta-vec1);
L2_beta_error=sqrt(L2_beta_error);
H1_uprime_error=(uprime-U1bar)'*trueH1solid*(uprime-U1bar);
H1_uprime_error=sqrt(H1_uprime_error);
```

```
fprintf(' -- L2norm_alpha_error = %d',L2norm_alpha)
fprintf('\n -- H1norm_alpha_error = %d',H1norm_alpha)
%fprintf('\n -- L2norm_beta = %d',L2norm_beta)
fprintf('\n -- L2_beta_error = %d',L2_beta_error)
fprintf('\n -- H1_uprime_error = %d \n',H1_uprime_error)

out=[out L2norm_alpha, H1norm_alpha, L2_beta_error,H1_uprime_error];
%out=[out L2norm_alpha, H1norm_alpha, L2_beta_error,H1_uprime_error,
   L2norm_alpha_dvorak];

%=====================================================================
%   Closing Processes
%=====================================================================
endtime=toc;
minutes=(1/60)*endtime;
disp(' ')
string=['Done! Elapsed time was: '...
    ,num2str(endtime), ' seconds (',num2str(minutes),' minutes)'];
disp(string)
fprintf('\n\n')



save('tempA','A')


end
```

A.2 AdjustFluidBdry.m

```
function [ BigMatrix,G ] = AdjustFluidBdry( A,B,G )
%Adjust the system for the outer fluid boundary by replacing portions of
%the system with the identity


%==========================================================================
%   Global Variables and Important Constants
%==========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
```

```
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
    progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
    coord)
global T2             % Number of local basis functions Eta (6; 3 in each
    coord)

global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % GO BACK AND DEFINE LAMBDA LATER
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % GO BACK AND DEFINE LATER

%==========================================================================


%Replace certain rows of the matrix with the identity since f is 0 on outer
%bdry Gamma_f
disp('Adjusting system to account for fluid boundary...')
```

```
zip=zeros(2*nfluidpts,1);
zip2=zeros(1,otherDim);

for k=1:length(outerbdrynodes)
    place=outerbdrynodes(k);
    place=find(place==fluidpts);
    place2=place+nfluidpts;
    A(place,:)=zip';
    A(:,place)=zip; %Not necessary to replace the column with all zeros, but
        it doesn't hurt
    A(place,place)=1;
    A(place2,:)=zip';
    A(:,place2)=zip; %Not necessary to replace the column with all zeros
    A(place2,place2)=1;
    G(place,1)=0;
    G(place2,1)=0;
    B(place,:)=zip2;
    B(place2,:)=zip2;
end
for k=1:nMdpts
    if Mdpts(k,3)==1 && Mdpts(k,8)==2; %if midpoint is on bdry & in fluid
        place=k+otherDim; %midpoint nodes in fluid are listed after 1,2,..
            otherDim vertex nodes
        place2=place+nfluidpts;
        A(place,:)=zip';
        A(:,place)=zip'; %no need to replace column with 0, but doesn't hurt
```

```
        A(place,place)=1;
        A(place2,:)=zip';
        A(:,place2)=zip; %no need to replace column with 0, but doesn't hurt
        A(place2,place2)=1;
        G(place,1)=0;
        G(place2,1)=0;
        B(place,:)=zip2;
        B(place2,:)=zip2;
    end
end



zilch=zeros(otherDim,otherDim);
BigMatrix=[A B; B' zilch]; %This is the overarching stiffness matrix



end
```

A.3 alambdaEx.m

```
function [ out ] = alambdaEx( ii,iwhichhalf,jj,...
                            jwhichhalf,H1,AllDPhi,AllDPsi,testflag )
%Function to add in the a_S(u,v)+lambda*< u,v >_{L^2_Theta} terms in the
%a_lambda bilinear form. The coefficient vectors for the extension
%problems are given in AllDPhi=D_{S,lambda^2}(0,Phi) solutions
%and AllDPsi=D_{S,lambda^2}(0,inv(lambda+Dw)Phi) solutions
```

```
%==========================================================================
%   Global Variables and Important Constants
%==========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
```

```
global lambda         % Constant given in problem (adjust to progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
    coord)
global nsolperimeter  % Number of nodes on the solid boundary
global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % Parameter introduced to ensure ellipticity
global lambda mu      % Lame constants
global nu             % Viscosity
global kappa          % Coefficient for Theta mapping


%-----------------------------------
% Get the correct solution vectors
%-----------------------------------
i=find(ii==solperimeter);
j=find(jj==solperimeter); %get which column from database for solution vec

if iwhichhalf==2; %if second coordinate of Phi is nonzero
    i=i+nsolperimeter;
end
if jwhichhalf==2;
    j=j+nsolperimeter;
end

iVec=AllDPsi(:,i); %i input corresponds to D_{S,lambda^2}(0,inv(lambda-e3)*
    Phi_i)
```

```
jVec=AllDPhi(:,j); %j input corresponds to D_{S,lambda^2}(0,Phi_j)



%---------------------------------
% Take the appropriate integrals
%---------------------------------

%Shortcut to use matrix product for the first three terms of a_S; i.e. the
%alternative H1 product plus lambda^2*L2 product
firstterms=iVec'*H1*jVec;

%Take the appropriate boundary integral
out=0;
nodes=zeros(1,6);
for q=1:nedgetri
    nodes(1,1:3)=ELNODE(edgetri(q,1),1:3);

    [row,col]=find(searchtriangles==edgetri(q,1));
    localnum(1,1)=Mdpts(row(1),2*col(1)+2);
    localnum(2,1)=Mdpts(row(2),2*col(2)+2);
    localnum(3,1)=Mdpts(row(3),2*col(3)+2);

    nodes(1,4)=row(find(4==localnum))+M;
    nodes(1,5)=row(find(5==localnum))+M;
    nodes(1,6)=row(find(6==localnum))+M;
```

```
[J,DxKinv,xK,xKinv]=getMapInfo(edgetri(q,2),2);
%abs(jacobian) & inv(D(affine map)), affine map, inverse affine map

% GET COORDINATES TO INTEGRATE OVER
ver=nodes(1:3); %pull the vertices of the element

%Find which two vertices are on the solid bdry
test(1)=isempty(find(ver(1)==solidbdrynodes)); %test=0 => on bdry
test(2)=isempty(find(ver(2)==solidbdrynodes));
test(3)=isempty(find(ver(3)==solidbdrynodes));
whichnodes=find(min(test)==test);
%find which two entries of test are 0
starting=ver(whichnodes(1));
%global node number to start integration on
finishing=ver(whichnodes(2));
%global node number to finish integration on

if starting<=M %if we are looking at a vertex
    p1=NODECO(starting,1:2); %get coordinate pair
    p1=p1';
else            %if we are looking at a midpoint
    p1=Mdpts(starting-M,1:2); %get coordinate pair
    p1=p1';
end
if finishing<=M %if we are looking at a vertex
    p2=NODECO(finishing,1:2); %get coordinate pair
```

```
    p2=p2';
else            %if we are looking at a midpoint
    p2=Mdpts(finishing-M,1:2); %get coordinate pair
    p2=p2';
end

%Make sure we don't need to swap the order of p1 and p2
% here we assume nodes are given in counterclockwise order from
% getMesh.m
loc1=find(starting==ver);
loc2=find(finishing==ver);
if loc2-loc1~=1; %swap needed
    temp=p1;
    p1=p2;
    p2=temp;
end

tang=p2-p1;
tang=(1/norm(tang))*tang; %get tangent vector

n1=find(nodes(1,1)==solidpts); % use global solid node numbers
n2=find(nodes(1,2)==solidpts); % to assign coef's to nodes
n3=find(nodes(1,3)==solidpts);
n4=find(nodes(1,4)==solidpts);
n5=find(nodes(1,5)==solidpts);
n6=find(nodes(1,6)==solidpts);
```

```
n7=n1+nsolidpts;

n8=n2+nsolidpts;

n9=n3+nsolidpts;

n10=n4+nsolidpts;

n11=n5+nsolidpts;

n12=n6+nsolidpts;


%D_{S,lambda^2}(0,inv(lambda-e3)*Phi)=iSol

iSol=@(p) iVec(n1)*Phi(1,p)+iVec(n2)*Phi(2,p)+iVec(n3)*Phi(3,p)+...

    iVec(n4)*Phi(4,p)+iVec(n5)*Phi(5,p)+iVec(n6)*Phi(6,p)+...

    iVec(n7)*Phi(7,p)+iVec(n8)*Phi(8,p)+iVec(n9)*Phi(9,p)+...

    iVec(n10)*Phi(10,p)+iVec(n11)*Phi(11,p)+iVec(n12)*Phi(12,p);


%D_{S,lambda^2}(0,Phi)=jSol

jSol=@(p) jVec(n1)*Phi(1,p)+jVec(n2)*Phi(2,p)+jVec(n3)*Phi(3,p)+...

    jVec(n4)*Phi(4,p)+jVec(n5)*Phi(5,p)+jVec(n6)*Phi(6,p)+...

    jVec(n7)*Phi(7,p)+jVec(n8)*Phi(8,p)+jVec(n9)*Phi(9,p)+...

    jVec(n10)*Phi(10,p)+jVec(n11)*Phi(11,p)+jVec(n12)*Phi(12,p);


%D(jSol)  (including DxKinv!!)

DjSol=@(p) (jVec(n1)*DiffPhi(1,p)+jVec(n2)*DiffPhi(2,p)...

    +jVec(n3)*DiffPhi(3,p)+jVec(n4)*DiffPhi(4,p)...

    +jVec(n5)*DiffPhi(5,p)+jVec(n6)*DiffPhi(6,p)+...

    jVec(n7)*DiffPhi(7,p)+jVec(n8)*DiffPhi(8,p)...

    +jVec(n9)*DiffPhi(9,p)+jVec(n10)*DiffPhi(10,p)...

    +jVec(n11)*DiffPhi(11,p)+jVec(n12)*DiffPhi(12,p))*DxKinv;
```

```
    %Apply Boundary Operator B to iSol: B(jSol)=BjSol
    if testflag~=4
        BjSol=@(p) e1*jSol(xKinv(p))+e2*DjSol(xKinv(p))*tang;
    end
    if testflag==4
        BjSol=@(p) e1*jSol(xKinv(p))...
            +e2*(p(1)^2+p(2)^2)*DjSol(xKinv(p))*tang;
    end

    %Output Integrand
    integ=@(p) (-1)*BjSol(p)'*iSol(xKinv(p));
    out=quadLine(integ,p1,p2)+out;

end



out=firstterms+out;





end
```

A.4 buildsystem.m

```
function [A,B,G,A1,L2fluid,smallL2fluid,H1fluid,vec1,U1bar,V1bar]=buildsystem(
    H1solid,...
```

```
                                        AllDPhi,AllDPsi,VUvec,...
                                        L2,rawU1bar,rawV1bar,testflag)
%Function to build the overarching finite element system

%==========================================================================
%   Global Variables and Important Constants
%==========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
```

```
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
    progress in time)
global pressure       % The constant pressure function
global T              % Number of local basis functions Phi (12; 6 in each
    coord)
global T2             % Number of local basis function Eta (6; 3 in each coord
    )

global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % Parameter introduced to guarantee ellipticity
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % Coefficient of Theta mapping


%==========================================================================
%   Define Important Products
%==========================================================================
    function out=Frob(q,r,s,p,DxKinv)
        %Frobenius product between elastic stress tensors
        %e(u)..e(v)=out (lowercase epsilons; i.e. no kappa's yet for
```

```
        intermediate problem)

    ephi=@(k,p) (.5)*(DiffPhi(k,p)*DxKinv+DxKinv'*DiffPhi(k,p)'); %
        symmetric stress tensor

    out=trace(ephi(r,p)'*ephi(s,p)); %Frobenius product

end

function [out,A1out]=Aproduct(q,r,s)
    %Product a_lambda(phi_i, phi_j) to build A submatrix

    %Get Jacobian and inv(derivative of affine map) from reference
    %element to mesh element (fluid domain)
    [J,DxKinv,xK]=getMapInfo(q,1); %abs(jacobian) & inv(D(affine map))
      %xK is the affine map

    integA1=@(p) (Lambda+lambda)*Phi(r,p)'*Phi(s,p); %first term;
                                          %Phi is 2x1 vector valued
    integA2=@(p) 2*nu*Frob(q,r,s,p,DxKinv);          %second term
    integA3=@(p) Phi(s,p)'*DiffPhi(r,p)*DxKinv*w1(xK(p)); %third term;
                      %the rest of terms are bdry terms, so omit here

    integA=@(p) J*(integA1(p)+integA2(p)+integA3(p));
    out=quadTri(integA); %integ value to output if node not on sol bdry
```

```
    integA1out=@(p) (lambda*Phi(r,p)'*Phi(s,p)+integA2(p)+integA3(p))*J;
    A1out=quadTri(integA1out);
end

function out=Bproduct(q,r,s)
    %Product b(eta_j,phi_i) to build B submatrix
    %Get Jacobian and inv(derivative of affine map) from reference
    %element (fluid domain)
    [J,DxKinv,xK]=getMapInfo(q,1); %abs(jacobian) & inv(D(affine map))
    integB=@(p) J*(-Eta(s,p))*trace(DiffPhi(r,p)*DxKinv);
           %term with trace is div(Phi)
    out=quadTri(integB);
end

function out=Gproduct(q,r)
    %Functional F(phi_i) to build nonzero portion of G vector on r.h.s.

    %Get Jacobian and inv(derivative of affine map) from reference
    %element (fluid domain)
    [J,DxKinv,xK]=getMapInfo(q,1); %abs(jacobian) & inv(D(affine map))

    integC=@(p) J*w1(xK(p))'*Phi(r,p);
    %w1(p) is given
    % notice that all other terms are bdry terms, so omit for now

    out=quadTri(integC);
```

```
end

function out=EtaL2(q,r,s)
    %Product to create the L2 mass matrix for the pressure term
    % It needs to use linear basis elements, Eta's
    [J,DxKinv,xK]=getMapInfo(q,1);
    integl2=@(p) J*Eta(r,p)'*Eta(s,p);
    out=quadTri(integl2);
end

function out=H1fluidProduct(q,r,s)
    %Build the mass matrix for the H1 norm over the fluid
    [J,DxKinv,xK]=getMapInfo(q,1); %abs(jacobian) & inv(D(affine map))

    % Typical H^1 norm mass matrix
    %integ1=@(p) Phi(r,p)'*Phi(s,p);
    %integ2=@(p) trace(DxKinv'*DiffPhi(r,p)'*DiffPhi(s,p)*DxKinv);
    %integH1=@(p) J*(integ1(p)+integ2(p));
    %out=quadTri(integH1);

    % Equivalent H^1 norm mass matrix given by Prop 4.1 on pg.16
    integH1=@(p) J*Frob(q,r,s,p,DxKinv);
    out=quadTri(integH1);
end

function out=L2fluidProduct(q,r,s)
```

```
        %Build the mass matrix for the L2 norm over the fluid (using
        %quadratic basis functions)
        [J,DxKinv,xK]=getMapInfo(q,1); % 1 corresponds to the fluid domain

        integL2=@(p) J*Phi(r,p)'*Phi(s,p);
        out=quadTri(integL2);
    end

%==========================================================================
%   Begin Building the Overarching System
%==========================================================================
fprintf('\n\nBuilding the overarching linear system...\n');
%We want to build overarching system:
% [A B; B^T 0] *coef = [G; 0]
% (since the right hand vector is 0 in bottom entries, just allocate full
% length for G vector as all zeros)


% A=zeros(2*nfluidpts,2*nfluidpts);
% B=zeros(2*nfluidpts,otherDim);

A = sparse(zeros(2*nfluidpts,2*nfluidpts)); % matrix for AProduct(u,v)
B = sparse(zeros(2*nfluidpts,otherDim)); % matrix for BProduct(u,v)
                        % otherDim is number of fluid vertices;
                        % i.e. the total number of Eta basis functions
G = zeros(2*nfluidpts+otherDim,1); % the load vector on r.h.s.
```

```
A1=A;                    % matrix for a_lambda^1 product
L2fluid=A;               % mass matrix for L2 norm over fluid
H1fluid=A;               % mass matrix for H1 norm over fluid
linearL2fluid=A;         % mass matrix for L2 norm over fluid w/linear fn
vec1=zeros(otherDim,1); % projection of pressure into L2 linear

string=[' - Computing products for ',num2str(nELfluid),...
    ' triangles and ',num2str(T),' basis functions each...'];
disp(string)

%Loop with all terms except the extension terms in the A Product
% Cycle through each fluid triangular element
% Here I am assuming that all fluid elements are listed first in ELNODE
for q=1:nELfluid
    disp([' -- for element ', num2str(q),' of ',num2str(nELfluid)]);
    for r=1:T    %cycle through each Phi_i; r is local basis number

        %Get  i=row of S to add product to (global node number in only
        %      fluid domain)
        %   ii=global node number (not just in fluid domain)
        %   iwhichhalf= 1 or 2 based on which coordinate Phi_i is nonzero
        %   The 1 corresponds to using the fluid domain
        [i,ii,iwhichhalf]=getGlobalNodeNum(q,r,1);

        %--------------- Loop for matrix A --------------------------------
        for s=1:T %cycle thru columns; use symmetry to stop at s=r
```

```
    [j,jj,jwhichhalf]=getGlobalNodeNum(q,s,1);
    [a,a1]=Aproduct(q,r,s);
    A(i,j)=A(i,j)+a;
    A1(i,j)=A1(i,j)+a1; %matrix using only a_lambda^(1) product

end

%--------------- Loop for matrix B --------------------------------
for s=1:T2 %cycle thru columns; B is not symmetric so no shortcut
    %also, s is the degree of freedom associated with the Eta's
    [j,jj,jwhichhalf]=getGlobalNodeNum(q,s,1);

    b=Bproduct(q,r,s);
    B(i,j)=B(i,j)+b;
end

%--------------- Loop for Mass Matrices ---------------------------
for s=1:r %cycle thru columns
    [j,jj,jwhichhalf]=getGlobalNodeNum(q,s,1);
    h1=H1fluidProduct(q,r,s);
    H1fluid(i,j)=H1fluid(i,j)+h1; %update mass matrix entry
    H1fluid(j,i)=H1fluid(i,j); %use symmetry

    l2=L2fluidProduct(q,r,s);
    L2fluid(i,j)=L2fluid(i,j)+l2; %update mass matrix entry
    L2fluid(j,i)=L2fluid(i,j); %use symmetry
```

```
        end

        %--------------- Loop for Vector G --------------------------------
        c=Gproduct(q,r);
        G(i)=G(i)+c;
    end
end       %END OF FIRST LOOP TO BUILD OVERARCHING MATRIX SYSTEM



%Build L2 mass matrix over fluid domain using linear basis functions
for q=1:nELfluid
    [J,DxKinv,xK]=getMapInfo(q,1);
    for r=1:T2
        [i,ii,iwhichhalf]=getGlobalNodeNum(q,r,1);
        integralvec1=@(p) J*pressure*Eta(r,p);
        v1=quadTri(integralvec1);
        vec1(i,1)=vec1(i,1)+v1; %vec1 here is used to get the L2 projection
                                %of the constant pressure function

        for s=1:T2
            [j,jj,jwhichhalf]=getGlobalNodeNum(q,s,1);
            l2=EtaL2(q,r,s);
            linearL2fluid(i,j)=linearL2fluid(i,j)+l2;
        end
    end
end
```

```
smallL2fluid=linearL2fluid(1:otherDim,1:otherDim); %L2 mass matrix (vertex
    nodes)
vec1=inv(smallL2fluid)*vec1; %multiply by inverse to get actual projection

if testflag~=0 %Solve for U1 and V1 L2 projection
    U1bar=L2\rawU1bar;
    V1bar=L2\rawV1bar;
else %testflag=0 Case where U1bar was already found
    U1bar=rawU1bar;
    V1bar=rawV1bar;
end


%==========================================================================
%   Incorporate Extension Terms
%==========================================================================
%Loop to take care of the extension terms in the A-Product
% Since the products are going to be the same for the second coordinate
% nonzero basis vectors, I am using the product from the first coordinate
% nonzero basis vectors in both places

fprintf('Incorporating Extension Terms...\n')
disp(' - Computing the products of extensions into the solid domain...')
for index=1:nsolperimeter %cycle through each solid bdry node
    string=[' -- for solid boundary node ',num2str(index),...
```

```
    ' of ',num2str(nsolperimeter)];
disp(string)
k=solperimeter(index);   %get the global node number of bdry node
place=find(k==fluidpts); %get where to place the product
for index2=1:nsolperimeter % inner loop to pair Phi_k and Phi_k2
    k2=solperimeter(index2);
    place2=find(k2==fluidpts);

    value1=alambdaEx(k,1,k2,1,H1solid,AllDPhi,AllDPsi,testflag);
    value2=alambdaEx(k,2,k2,1,H1solid,AllDPhi,AllDPsi,testflag);
    value3=alambdaEx(k,1,k2,2,H1solid,AllDPhi,AllDPsi,testflag);
    value4=alambdaEx(k,2,k2,2,H1solid,AllDPhi,AllDPsi,testflag);

    A(place,place2)=A(place,place2)+value1;
    A(place+nfluidpts,place2)=A(place+nfluidpts,place2)+value2;
    A(place,place2+nfluidpts)=A(place,place2+nfluidpts)+value3;
    A(place+nfluidpts,place2+nfluidpts)=A(place+nfluidpts,...
        place2+nfluidpts)+value4;
end

G(place,1)=G(place,1)+GEx(k,1,VUvec,AllDPhi,H1solid,...
    U1bar,V1bar,L2,testflag);
% Extension with nonzero first coord ^^
G(place+nfluidpts,1)=G(place+nfluidpts,1)...
         +GEx(k,2,VUvec,AllDPhi,H1solid,U1bar,V1bar,L2,testflag);
% Extension with nonzero second coord ^^
```

```
end




end
```

A.5 calcExtenProb.m

```
function [ AllDPhi,AllDPsi ] = calcExtenProb( preS,S,testflag )
%-------------------------------------------------------------------------
% Function to create databases for coefficient vectors of solutions to
% extension problems.
% AllDPhi = matrix of coefficient vector columns
%  - each column is the coefficient vector of sol=D_{S,lambda^2}( 0, Phi )
% AllDPsi = matrix of coefficient vector columns
%  - each col is the coef vec of sol=D_{S,lambda^2}(0,inv(lambda-Dw)Phi)
%-------------------------------------------------------------------------

%=========================================================================
%   Global Variables and Important Constants
%=========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
```

```
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
    progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
    coord)

global AllDPhi        % Library of all solutions to D_{S,lambda^2}(0,Phi) in
    solid
global AllDPsi        % Library of all solutions to D_{S,lambda^2}(0,inv(
```

```
   lambda+Dw)*Phi) in solid
global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % Parameter introduced to ensure ellipticity
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % Coefficient for Theta mapping


%==========================================================================
%   Solve Extension Problems
%==========================================================================

%Solve for all the Dphi=coef of D_{lambda^2,S}(0,Phi)
%--------------------------------------------------------------------------
%  Solve extension problem with 0 as r.h.s. and Phi on bdry
%
%  Create a library of Dphi solution vectors
%--------------------------------------------------------------------------
%    Dimensions are 2*nsolidpts x 2*nsolperimeter
%      This is because each coef vector is 2*nsolidpts tall
%      and we need to have a solution vector for each node
%      on the solid bdry for each coordinate.
fprintf(['Creating database of coef vectors for D_{S,lambda^2}(0,Phi)'...
   '\n (solved two at a time)\n'])

AllDPhi=zeros(2*nsolidpts,2*nsolperimeter);
```

```
%cycle through each solperimeter node and record solution
for t=1:nsolperimeter %Go thru each node on Gamma_S
    disp([' -- for node ',num2str(t),' and ',num2str(t+nsolperimeter),...
        ' of ',num2str(2*nsolperimeter)])
    gamma1=zeros(2*nsolidpts,1); %Initialize bdry vectors
    gamma2=gamma1;        %gamma1 is for Phi's w/nonzero in first coord.
    place1=find(solperimeter(t)==solidpts);
    place2=place1+nsolidpts;

    gamma1(place1,1)=1;        % gamma2 for second coord. nonzero
    gamma2(place2,1)=1;

    rhs1=(-1)*preS*gamma1; %right hand side; 0-(S w/out id) * gamma1
    rhs2=(-1)*preS*gamma2; %right hand side; 0-(S w/out id) * gamma2

    %Replace bdry entries of rhs's with 0
    for k=1:nsolperimeter
        place=find(solperimeter(k)==solidpts);
        place2=place+nsolidpts;
        rhs1(place,1)=0;
        rhs1(place2,1)=0;
        rhs2(place,1)=0;
        rhs2(place2,1)=0;
    end

    Dphi1=S\rhs1;     %Solve system with S w/identity
```

```
    Dphi1=Dphi1+gamma1; %Correct for b.c.'s by adding back bdry vector
    Dphi2=S\rhs2;
    Dphi2=Dphi2+gamma2;

    AllDPhi(:,t)=Dphi1; %Store coef vectors in library
    AllDPhi(:,t+nsolperimeter)=Dphi2;
end




%Solve for all the Dpsi=coef of D_{S,lambda^2}(0,(lambda-Dw)^{-1}(p)'*Phi(p))
%-------------------------------------------------------------------------
%   Solve extension problem with 0 as r.h.s. and inv(lambda-Dw)*Phi on bdry
%
% For the intermediate problem, see lamDw.m for the working definition of
% inv(lambda-Dw)(p)
%-------------------------------------------------------------------------
fprintf(['Creating database of coef vectors for D_{S,lambda^2}'...
    '(0,inv(lambda-Dw)Phi) \n (solved two at a time)\n'])
AllDPsi=zeros(2*nsolidpts,2*nsolperimeter);

 %cycle through each solperimeter node and record solution
for t=1:nsolperimeter %Go thru each node on Gamma_Solid
    disp([' -- for node ',num2str(t),' and ',num2str(t+nsolperimeter),...
        ' of ',num2str(2*nsolperimeter)])
    gamma1=zeros(2*nsolidpts,1); %Initialize bdry vectors
```

```
gamma2=gamma1;              %gamma1 is for Phi's w/nonzero in first coord.
place1=find(solperimeter(t)==solidpts);
place2=place1+nsolidpts;



if solperimeter(t)<=M %If we have a vertex node
    coord=NODECO(solperimeter(t),1:2); %coordinates of point to plug into
        function below
else %if we have a midpoint node
    location=solperimeter(t)-M; %which row is this midpoint node in Mdpts
        database
    coord=Mdpts(location,1:2); %coordinates of point to plug into function
        below
end

lamDwMat=lamDw(coord',testflag);

if det(lamDwMat)==0
  error('\n e3=%d chosen such that inv(lambda*I-Dw) does not exist',e3)
end

g1=lamDwMat\[1;0];
g2=lamDwMat\[0;1];

gamma1(place1,1)=g1(1,1);
gamma1(place2,1)=g1(2,1);
```

```
    gamma2(place1,1)=g2(1,1);
    gamma2(place2,1)=g2(2,1);

    rhs1=(-1)*preS*gamma1; %right hand side; 0-(S w/out id) * gamma1
    rhs2=(-1)*preS*gamma2; %right hand side; 0-(S w/out id) * gamma2

    %Replace bdry entries of rhs's with 0
    for k=1:nsolperimeter
        place=find(solperimeter(k)==solidpts);
        place2=place+nsolidpts;
        rhs1(place,1)=0;
        rhs1(place2,1)=0;
        rhs2(place,1)=0;
        rhs2(place2,1)=0;
    end

    Dpsi1=S\rhs1;     %Solve system with S w/identity
    Dpsi1=Dpsi1+gamma1; %Correct for b.c.'s by adding back bdry vector
    Dpsi2=S\rhs2;
    Dpsi2=Dpsi2+gamma2;

    AllDPsi(:,t)=Dpsi1; %Store coef vectors in library
    AllDPsi(:,t+nsolperimeter)=Dpsi2;
end

end
```

A.6 calcExtenStiff.m

```
function [ H1,preS,S,trueH1,L2 ] = calcExtenStiff()
%==========================================================================
%Function to calculate the extension stiffness matrix, S; along with
%its variations H1 and preS
% H1 is used for: lambda^2(u,v)_{L^2} + (u,v)_{alternative H^1}= u^* H1 v
% preS = stiffness matrix without replacing any equations with identity
%==========================================================================



%==========================================================================
%   Global Variables and Important Constants
%==========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
```

```
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
    progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
    coord)
global AllDPhi        % Library of all solutions to D_{S,lambda^2}(0,Phi) in
    solid
global AllDPsi        % Library of all solutions to D_{S,lambda^2}(0,inv(
    lambda+Dw)*Phi) in solid

global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % GO BACK AND DEFINE LAMBDA LATER
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % GO BACK AND DEFINE LATER
```

```
%==========================================================================
%   Define Necessary Bilinear Forms
%==========================================================================

    function out=Frob(q,r,s,p,DxKinv)
        %Frobenius product between elastic stress tensors
        %e(u)..e(v)=out (lowercase epsilons; i.e. no kappa's yet for
            intermediate problem)

        %[J,DxKinv,xK]=getMapInfo(q,t); %J=jacobian, DxKinv=(DxK)^-1, xK=map
            from ref to mesh triangle, 2 means solid domain



        ephi=@(k,pt) (.5)*(DiffPhi(k,pt)*DxKinv+DxKinv'*DiffPhi(k,pt)'); %
            symmetric stress tensor

        out=trace(ephi(r,p)'*ephi(s,p)); %Frobenius product

    end

    function [out,h1,l2]=EXprod(q,r,s)
        %Function to build the stiffness matrix in the extension
        %problems, preS. S is the same matrix except partially replaced by
        %identity on boundary entries
```

```
% Essentially we have a_S(Phi_r, Phi_s) here w/out bdry integral
% term

[J,DxKinv,xK,xKinv]=getMapInfo(q,2);
%J=jacobian, DxKinv=(DxK)^-1, xK=map from ref to mesh triangle
%note the 2 indicates we want the solid domain

EXinteg1=@(p) (1/kappa^2)*Phi(r,p)'*Phi(s,p);
EXinteg2=@(p) (1/kappa^2)*( ...
    lam*kappa^4*trace(DiffPhi(r,p)*DxKinv)...
    *trace(DiffPhi(s,p)*DxKinv)+2*mu*kappa^4*Frob(q,r,s,p,DxKinv));
    %notice that the pg.22 code may have a typo; there should be a
    %2 in front of the mu constant
EXinteg3=@(p) (lam+mu)*(kappa^2-1)*trace(DxKinv'*DiffPhi(r,p)'...
    *DiffPhi(s,p)*DxKinv);
% using trace is another way to write Frob matrix product
EXinteg4=@(p) lambda^2*(1/kappa^2)*Phi(r,p)'*Phi(s,p);
% the lambda^2(Phi_r,Phi_s)_Theta of Prop 6.7

%The boundary line integral is omitted from this product. It will
%be addressed in a separate loop.

EXinteg=@(p) J*(EXinteg1(p)+EXinteg2(p)+EXinteg3(p)+EXinteg4(p));

out=quadTri(EXinteg); %the interior terms of a_S(Phi_r,Phi_s)
```

```
    h1integ=@(p) J*(kappa^2*EXinteg1(p)...
        +trace(DxKinv'*DiffPhi(r,p)'*DiffPhi(s,p)*DxKinv));
    h1=quadTri(h1integ); %product to build H1 mass matrix over solid

    l2integ=@(p) J*Phi(r,p)'*Phi(s,p);
    l2=quadTri(l2integ); %product to build L2 mass matrix over solid
end

function out=EXprodBdry(q,r,s,ii,jj)
    %the boundary line integral portion of a_S(Phi_r,Phi_s) from global
    % node ii to global node jj
    %Function to build the stiffness matrix in the extension problems,
    %preS. S is the same matrix except partially replaced by the
    %identity on boundary entries
    %  ii, jj are global node numbers over whole domain
    %  q is global element number over solid domain
    %  r and s are degrees of freedom
    %
    % out = { B(Phi_r), Phi_s }_{solid bdry}
    % B(V)= e1*V + e2*DV*tangent_vec

    % I am assuming the element triangle vertices are given in
        counterclockwise
    % order.
    [tang,p1,p2]=getTangVec(ii,jj);
    %get tangent vector,begin/end coord, segment length
```

```
        [J,DxKinv,xK,xKinv]=getMapInfo(q,2);
        %jacobian, inv(Dmap), map (ref to mesh), inverse map (mesh to ref)

        Bphi=@(p) e1*Phi(s,xKinv(p))+e2*DiffPhi(s,xKinv(p))*DxKinv*tang;
        %e1 and e2 are global constants given above; relatively small

        integ=@(p) (-1)*Bphi(p)'*Phi(r,xKinv(p));

        out=quadLine(integ,p1,p2);

    end





%{
%     function out=testb(p)
%         x=p(1); y=p(2);
%         out=[x.^2*y.^2; (x-1/3)*x*y*y];
%     end
%     function out=testfunct(q,r)
%         [J,DxKinv,xK]=getMapInfo(q,2);
%         integz=@(p) J*testb(xK(p))'*Phi(r,p);
%         out=quadTri(integz);
%     end
%}
```

```
%==========================================================================
%   Calculate Extension Stiffness Matrix
%==========================================================================
disp('Building stiffness matrix for use in extensions in solid domain...')
disp([' - Computing products for ',num2str(nELsolid),' triangles and ',...
    num2str(T),' basis functions each...'])
%S=sparse(zeros(2*nsolidpts,2*nsolidpts)); %initialize stiffness matrix for
    extension problem
S=sparse(zeros(2*nsolidpts,2*nsolidpts));
trueH1=S;
L2=zeros(2*nsolidpts,2*nsolidpts);



%Integrate over each mesh element in the solid to build stiffness mat. S
% Here I am assuming that the solid elements are listed last from getMesh.m
for q=1:nELsolid   %cycle through each mesh element in the solid domain
    string=[' -- for element ', num2str(q),' of ',num2str(nELsolid)];
    disp(string)
    for r=1:T      %cycle through each Phi_i; r is local basis number


        %Get  i=row of S to add product to (global node number in only
        %      solid domain)
        %   ii=global node number (not just in solid domain)
        %   iwhichhalf= 1 or 2 based on which coordinate Phi_i is nonzero
        [i,ii,iwhichhalf]=getGlobalNodeNum(q,r,2);
            %ii and iwhichhalf are irrelevant for this section of code.
```

```
            %they will come up later building the overarching system.
            % the 2 corresponds to the solid domain

        for s=1:r %cycle through columns; use symmetry to stop at s=r
            [j,jj,jwhichhalf]=getGlobalNodeNum(q,s,2);

            [value,h1,l2]=EXprod(q,r,s); %get value of appropriate integrals

            S(i,j)=S(i,j)+value;     %add that value to entry in S
            S(j,i)=S(i,j);

            trueH1(i,j)=trueH1(i,j)+h1; %add entry to get actual matrix for
                usual H1 product
            trueH1(j,i)=trueH1(i,j);

            L2(i,j)=L2(i,j)+l2; %add entry to get actual matrix for L2 product
            L2(j,i)=L2(i,j);
        end
    end
end
H1=S; %save the portion of the extension stiffness matrix that corresponds
      %  to the lambda^2+H^1 product of two functions
      %  i.e. <u,v>_{H^1}+lambda^2*(1/kappa^2)<u,v>_{L^2}
      %           = coef_u' *H1 *coef_v;
      %  where <u,v>_{H^1} is the equivalent inner product for the
      %  usual H^1 product given on pg.21 of the final draft
```

```
%-------------------------------------------------
% Incorporate Boundary Line Integral into S
disp(' - Incorporating solid boundary integrals...')
for k=1:nedgetri %cycle through each element in solid touching solid bdry
    disp([' -- for element ',num2str(k),' of ',...
        num2str(nedgetri),' on solid boundary'])
    qs=edgetri(k,2); %qs is the global solid element number
    ver=ELNODE(edgetri(k),1:3); %pull the vertices of the element
    [J,DxKinv,xK,xKinv]=getMapInfo(qs,2);

    %Find which two vertices are on the solid bdry
    %  We are assuming the only options are 0 vertices on boundary or 2
    %  vertices on boundary. In reality 1 and 3 vertices on boundary are
    %  possible. This restricts the types of allowable meshes this code
    %  can use. At some point, go back and put the other two cases in this
    %  loop.
    test(1)=isempty(find(ver(1)==solidbdrynodes)); %test=0 means on bdry
    test(2)=isempty(find(ver(2)==solidbdrynodes));
    test(3)=isempty(find(ver(3)==solidbdrynodes));
    whichnodes=find(min(test)==test); %find which two entries of test are 0
    starting=ver(whichnodes(1)); %global node number to start integration on
    finishing=ver(whichnodes(2)); %global node number to finish integration on

    triangle=ELsolid(qs,1:3);
```

```
    c=getLocalPhi(test); %get the local node numbers on the solid bdry;
                         %based on which vertices were on solid bdry

    %Start the Integration Loops
    % Note that c is a vector of integers, so below we are looping from
    % index=c(1), index=c(2), index=c(3),..., index=c(6)
    for r=c
        [i,ii,iwhichhalf]=getGlobalNodeNum(qs,r,2);
        for s=c
            [j,jj,jwhichhalf]=getGlobalNodeNum(qs,s,2);

            value=EXprodBdry(qs,r,s,starting,finishing);
            S(i,j)=S(i,j)+value;
        end
    end

end
%-------------------------------------------------
preS=S; % save the extension stiffness matrix S without replacing with id



%Replace Boundary Portions of the Stiffness Matrix with Identity
disp(' - Adjusting for Dirichlet Boundary Conditions...')
zilch=zeros(2*nsolidpts,1);
for k=1:nsolperimeter
    bdry1=solperimeter(k); %pick the kth node on the boundary
```

```
    bdry1=find(bdry1==solidpts);
    bdry2=bdry1+nsolidpts; %also replace the corresponding other coordinate
    S(bdry1,:)=zilch; %replace the row and column with zeros
    S(:,bdry1)=zilch'; %not necessary to replace column with zeros, but it won
        't hurt
    S(bdry1,bdry1)=1;   %put a 1 at the intersection
    S(bdry2,:)=zilch;
    S(:,bdry2)=zilch';   %not necessary to replace column with zeros, but it
        won't hurt
    S(bdry2,bdry2)=1;
end
disp('Done building extension stiffness matrix')



end
```

A.7 calcUbarExt.m

```
function [ VUvec ] = calcUbarExt( preS,S,Ubar,testflag )
%-------------------------------------------------------------------------
%Given the coefficient vector of numerical eigenfunction Ubar, this
%function will calculate the vector:
%  D_{S,lambda^2}[ lambda*Ubar, inv(lambda-Dw)*Ubar ]
%which solves the Dirichlet problem
%  lambda^2*V + S(V) = lambda*Ubar      on solid domain
%                V  = inv(lambda-Dw)*Ubar on solid bdry
%-------------------------------------------------------------------------
```

```
%==========================================================================
%   Global Variables and Important Constants
%==========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
```

```
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
   progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
   coord)
global T2             % Number of local basis functions Eta (6; 3 in each
   coord)

global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % GO BACK AND DEFINE LAMBDA LATER
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % GO BACK AND DEFINE LATER


%==========================================================================
%  Build Gamma Vector
%==========================================================================
% gamma is the vector used to correct for non-homogeneous Dirichlet
% conditions

disp('Solving for D_{S,lambda^2}[ V1+lambda*U1, inv(lambda-Dw)*U1 ]')

gamma=zeros(2*nsolidpts,1); %initialize gamma
for k=1:nsolidpts
```

```
    ii=solidpts(k);

    if ii<=M %If we have a vertex node
        coord=NODECO(ii,1:2); %coordinates of point to plug into function below
    else %if we have a midpoint node
        location=ii-M; %which row is this midpoint node in Mdpts database
        coord=Mdpts(location,1:2); %coordinates of point to plug into function
            below
    end

    lamDwMat=lamDw(coord',testflag);

    % Get value of inv(lambda-Dw)*Ubar to plug in for gamma
    %lamDwMat*gammapoint = Ubarpoint = [Ubar(k,1);Ubar(k+nsolidpts,1)]
    gammapoint=lamDwMat\[Ubar(k,1);Ubar(k+nsolidpts,1)];

    gamma(k,1)=gammapoint(1);
    gamma(k+nsolidpts,1)=gammapoint(2);
end

rhs=lambda*Ubar-preS*gamma; %here we are assuming that V1(p)=0 from the
    eigenfunction problem

% Adjust rhs for Dirichlet conditions (zero out bdry entries)
for k=1:nsolperimeter
    bdry1=solperimeter(k); %pick the kth node on the boundary
```

```
    bdry1=find(bdry1==solidpts);
    bdry2=bdry1+nsolidpts; %also replace in the corresponding other coordinate

    rhs(bdry1,1)=0;
    rhs(bdry2,1)=0;
end

VUvec=S\rhs;  %solving using the usual finite element method
VUvec=VUvec+gamma;



end
```

A.8 calcVUvec.m

```
function [ VUvec,rawU1bar,rawV1bar ] = calcVUvec( preS,S,testflag,L2fluid )
%-------------------------------------------------------------------------
%Solve for VUvec=coef of D_{S,lambda^2}(V1+lambda U1,(lambda-Dw)^{-1}U1)
%-------------------------------------------------------------------------

%=========================================================================
%   Global Variables and Important Constants
%=========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
```

```
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
    progress in time)
global pressure       % The constant pressure function
global T              % Number of local basis functions Phi (12; 6 in each
    coord)
global T2             % Number of local basis functions Eta (6; 3 in each
```

```
    coord)

global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % Parameter introduced to guarantee ellipticity
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % Coefficient of Theta mapping
%-------------------------------------------------------------------------

%=========================================================================
%Solve for VUvec=coef of D_{S,lambda^2}(V1+lambda U1,(lambda-Dw)^{-1}U1)
%=========================================================================
%   Solve extension problem with V1+lambda*U1 as r.h.s.
%   and inv(lambda-Dw)*U1 on bdry
%-------------------------------------------------------------------------
disp(['Solving for coef vector of',...
    'D_{S,lambda^2}(V1+lambda U1,inv(lambda-Dw)U1)'])
VU=@(p) V1(p,testflag)+lambda*U1(p,testflag);
rhs=zeros(2*nsolidpts,1); %allocate space for right hand side vector
rawU1bar=zeros(2*nsolidpts,1); %allocate space for U1bar projection vector
rawV1bar=rawU1bar; %allocate space for V1bar projection vector
for q=1:nELsolid  % for each solid element, calculate right hand side
    [J,DxKinv,xK]=getMapInfo(q,2);
    for r=1:T   % pick a DOF (local over the element)
        % Compute the index of the global basis function
        % from element number and local DOF
```

```
        [i,ii,iwhichhalf]=getGlobalNodeNum(q,r,2);

        intgr=@(p) J*(1/kappa)*(VU(xK(p))'*Phi(r,p));
        intgrU1bar=@(p) J*(1/kappa)*U1(xK(p),testflag)'*Phi(r,p);
        intgrV1bar=@(p) J*(1/kappa)*V1(xK(p),testflag)'*Phi(r,p);

        value=quadTri(intgr);
        rhs(i,1)=rhs(i,1)+value;
        rawU1bar(i,1)=rawU1bar(i,1)+quadTri(intgrU1bar);
        rawV1bar(i,1)=rawV1bar(i,1)+quadTri(intgrV1bar);
    end
end




gamma=zeros(2*nsolidpts,1);
%allocate space for non-homogeneous Dirichlet BC's vector
for k=1:nsolperimeter %Evaluate Boundary function over solid bdry nodes
    i=find(solperimeter(k)==solidpts);
    %find global solid node number of given bdry node for placement

    if solperimeter(k)<=M %If we have a vertex node
        coord=NODECO(solperimeter(k),1:2);
        %coordinates of point to plug into U1 function below
    else %if we have a midpoint node
        place=solperimeter(k)-M;
```

```
        %which row is this midpoint node in Mdpts database
        coord=Mdpts(place,1:2);
        %coordinates of point to plug into U1 function below
    end

    lamDwMat=lamDw(coord',testflag);
    value=lamDwMat\U1(coord',testflag);

    gamma(i,1)=value(1,1);
    gamma(i+nsolidpts,1)=value(2,1);

    %compareUbarU1=norm([rawU1bar(i,1);rawU1bar(i+nsolidpts)]-U1(coord',
        testflag))
end

%Adjust rhs for Dirichlet bdry: rhs <-- (rhs-(S w/out Id)*gamma)
rhs=rhs-preS*gamma;

for k=1:nsolperimeter %replace bdry entries of rhs with 0
    kk=solperimeter(k);
    place=find(kk==solidpts);
    rhs(place,1)=0;
    rhs(place+nsolidpts,1)=0;
end
VUvec=S\rhs;
VUvec=VUvec+gamma;
```

```
end
```

A.9 DiffPhi.m

```
function [ out ] = DiffPhi( m,p )
%==========================================================================
%Gradient of the Phi function
%  Diff(Phi)=[ phi1_x phi1_y; phi2_x phi2_y ]
%      where Phi=[phi1; phi2]
%
%  There are six scalar functions (three for vertices and three for
%  midpoints). Since our function is vector valued in R^2, we take these six
%  scalar basis functions to generate twelve Phi's
%==========================================================================

x=p(1); y=p(2); %Rename the variables for readability

if m==1
   out=[4*x+4*y-3, 4*x+4*y-3; 0, 0];
elseif m==2
   out=[4*x-1, 0; 0, 0];
elseif m==3
   out=[0, 4*y-1; 0, 0];
elseif m==4
```

```
    out=[4*y, 4*x; 0, 0];
elseif m==5
    out=[-4*y, -4*(x+2*y-1); 0, 0];
elseif m==6
    out=[-4*(2*x+y-1), -4*x; 0, 0];

    %End of first form basis functions. Begin second form
elseif m==7
    out=[0, 0; 4*x+4*y-3, 4*x+4*y-3];
elseif m==8
    out=[0, 0; 4*x-1, 0];
elseif m==9
    out=[0, 0; 0, 4*y-1];
elseif m==10
    out=[0, 0; 4*y, 4*x];
elseif m==11
    out=[0, 0; -4*y, -4*(x+2*y-1)];
elseif m==12
    out=[0, 0; -4*(2*x+y-1), -4*x];
end









end
```

A.10 DxKinvMap.m

```
function retval = DxKinvMap(p1,p2,p3)
% The Differential of the map XKinvMap. It is independent of p.
    x1 =p1(1); y1=p1(2);
    x2 =p2(1); y2=p2(2);
    x3 =p3(1); y3=p3(2);
    retval = [y3-y1, x1- x3; y1-y2,x2-x1]/xKJac(p1,p2,p3);
end
```

A.11 errorExp.m

```
function [ out,points,vals ] = errorExp( errorTable )
%Given a column vector of errors, this function outputs the final exponent
%of decay and plots a graph showing how the exponent of decay changes with
%mesh level

displayPlots=1; %Change to ~=1 to suppress plotting

% Assumes that h_{n+1} = h_n/2

len=size(errorTable);
len=len(1)-1;
points=linspace(1,len,len); % 1, 2, 3, ... len : just the number of
    approximations
vals=zeros(len,1);
for i=1:(len)
```

```
    err1=errorTable(i);
    err2=errorTable(i+1);
    vals(i)=log(err1/err2)/log(2); % h1/h2 = 2
end
if displayPlots==1
    fprintf('Using: %s\n',inputname(1))
    disp(['Last Exponent Estimate: alpha ~ ',num2str(vals(len))]);
    figure;
    plot(points,vals,'-*');
end

out=vals(end);



end
```

A.12 Eta.m

```
function [out]=Eta(m,p)
    %local basis functions Eta_m(x,y), m=1,2,3
    %Defined on reference triangle (0,0) (1,0) (0,1)
    %  3 vertices; basis functions are 1 on one of them and 0 on rest

    x=p(1);
    y=p(2);
    if m==1        %vertex 1
        out=1-x-y;
```

```
    elseif m==2    %vertex 2
        out=x;
    else           %vertex 3
        out=y;
    end
end
```

A.13 getGlobalNodeNum.m

```
function [k, kk, kwhichhalf]=getGlobalNodeNum(q,dof,domain)
%==========================================================================
%Calculates the global node number of a node (midpoint or vertex) based on
%the global element number q and degree of freedom dof
%  k = global node number when only considering the given domain
%      domain=1 corresponds to fluid domain Omega_F
%      domain=2 corresponds to solid domain Omega_S
%  kk= global node number when considering full domain Omega
%  kwhichhalf = 1 or 2 based on which coordinate Phi_r will be nonzero in
%==========================================================================

global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solperimeter   % List of Vertices & Midpoints on Solid bdry
```

```
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global nsolidnodes    % Already defined?
global searchtriangles % Submatrix of Mdpts; contains local node information
global S              % Stiffness matrix for the extension problems w/identity
global preS           % Stiffness matrix for the ext. problems w/out identity
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Constant given in problem (adjust to progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
   coord)
global Dv1            % Coefficient vector for D_lambda(v1star)
global Avec           % Coefficient vector for A_lambda(v2star+lambda*v1star)
global nsolperimeter  % Number of nodes on the solid boundary
global AllDPhi        % Library of all solutions to D_lambda(Phi) in solid

% GO BACK LATER AND GET RID OF A TON OF THESE GLOBAL VARIABLES

%-----------------------------------------------------------------------
%Definitions based on desired domain:
if domain==1 %if we are considering the fluid domain
```

```
    Elem=ELfluid;
    nodes=fluidpts;
    nnodes=nfluidpts;
    triangle=q;   %the global element number to use to find midpoints
elseif domain==2 %if we are considering the solid domain
    Elem=ELsolid;
    nodes=solidpts;
    nnodes=nsolidpts;
    triangle=q+L-nELsolid; %the global elem number to use to find midpts; here
        we assume that solid elements are listed last
else
    error('Error in requested domain. Use 1 for fluid domain and 2 for solid
       domain.')
end




% Informaiton about the midpoints in that element

% searchtriangles contains pairs of element numbers sharing midpoint
% these element numbers are ordered as follows
%  fluid; fluid; ... fluid; solid; solid; solid;....
[row,col]=find(triangle==searchtriangles);
    %row above will give the rows of the midpoint database (i.e. the
    %global node numbers of the midpoint nodes) associated with the
```

```
    %element q. col gives the column where there is a match.
    %We need both because some elements share midpoints, hence we have
    %two columns in Mdpts with global element numbers listed for each
    %midpoint.
    %
    %We are assuming that the fluid elements are
    %listed first in the given mesh file.

% row(1),..row(3) are global midpoint numbers in the system
localnum(1,1)=Mdpts(row(1),2*col(1)+2);
localnum(2,1)=Mdpts(row(2),2*col(2)+2);
localnum(3,1)=Mdpts(row(3),2*col(3)+2); %localnum will always be 4;5;6 in some
     order
                                        %Equation comes from pulling
                                        %the 4th or 6th column from
                                        %Mdpts depending on which
                                        %column in searchtriangles the
                                        %global element number was
                                        %found.


%-----------------------------------------------------------------------
%Find node numbers:
if dof<=3
```

```
    kk=Elem(q,dof); % global node number in the system
    k=find(kk==nodes); % global node number in the domain
    kwhichhalf=1;
elseif dof<=6
    k_in_localnum=find(dof==localnum); %find which place dof is in localnum
    kk=row(k_in_localnum)+M; % kk returns global node # of the midpoint.
        %Add M b/c there are M vertices not included in Mdpts matrix
    k=find(kk==nodes);
    kwhichhalf=1;



elseif dof<=9
    kk=Elem(q,dof-6);  % repeat above, but looking at
                          % r-6 to be sure 1<=r<=6 for local node numbers
    k=find(kk==nodes)+nnodes; % since r>6, shift to bottom right of S
    kwhichhalf=2;
else
    k_in_localnum=find(dof-6==localnum);
    kk=row(k_in_localnum)+M;
    k=find(kk==nodes)+nnodes;
    kwhichhalf=2;
end
```

```
end
```

A.14 getLocalPhi.m

```
function [ c ] = getLocalPhi( test )
%Function that outputs the local node numbers for the Phi's that are on the
%solid boundary
% input: test= triple of two 0's and one 1.
%  ELNODE = v1 v2 v3 flag & the zeros for test correspond to the nodes of
%  the nodes on the solid bdry

c(1,1:2)=find(0==test); % c1 & c2 local nodes (vertices) on the bdry
c(1,3)=3+find(1==test); % c3 is local node num of midpoint between c1 and c2
c(1,4:6)=c(1,1:3)+6;  % for second coordinate basis functions, just add six to
     everything



end
```

A.15 getMapInfo.m

```
function [ J,DxKinv,xK,xKinv ] = getMapInfo( q,domain )
%Function to calculate:
%   J=abs(jacobian)
%   DxKinv = inv(derivative of the affine map xK) (a 2x2 matrix)
%       xK: ref element --> mesh element
%       xK(p) = M_2x2 * p + [x1; y1] where M=[x2-x1 x3-x1; y2-y1 y3-y1]
```

```
% domain=1 corresponds to fluid domain Omega_F
% domain=2 corresponds to solid domain Omega_S

global M
global NODECO
global L
global ELNODE
global Mdpts
global outerbdrynodes
global solperimeter
global solidpts
global nsolidpts
global ELsolid
global nELsolid
global ELfluid
global nELfluid
global nsolidnodes
global searchtriangles
global otherDim
global lambda
global T



%Define Elem based on desired domain
```

```
if domain==1 %if we are considering the fluid domain
    Elem=ELfluid;
elseif domain==2 %if we are considering the solid domain
    Elem=ELsolid;
end

%Get the appropriate map information
nodes=Elem(q,1:3); % get full domain global node numbers of q-element vertices
     p1-p3
p1=NODECO(nodes(1),:); % p1=(x1,y1)
p2=NODECO(nodes(2),:); % p2=(x2,y2)
p3=NODECO(nodes(3),:); % p3=(x3,y3)

% The map xK takes reference element "K_t" to element number l "K"
xK=@(p)xKmap(p1,p2,p3,p)'; %L in our usual notation
jacobian=xKJac(p1,p2,p3); %det(D L)
% The differential of the inverse
DxKinv = DxKinvMap(p1,p2,p3); %inv(DL)=D(inv(L)) since DL is constant
                            %DxKinv= inv(M) from description above

xKinv=@(p) xKinvMap(p1,p2,p3,p)';
J=abs(jacobian);



end
```

A.16 getMesh.m

```
function [O1,O2,O3,O4,O5,O6,O7,O8,O9,O10]=getMesh(input)
%Function to gather mesh information from gmesh files
%---------------------------
% DEFINE SHORT-NAME OUTPUTS:
%---------------------------
%  O1=msh;            Mesh file from gmesh
%  O2=Mdpts;          Midpoint Database
%  O3=cons;           Various important constants
%  O4=solidpts;       Nodes (vertices and midpoints) on solid domain
%  O5=fluidpts;       Nodes (vertices and midpoints) on fluid domain
%  O6=solperimeter;   Nodes (vertices and midpoints) on solid/fluid bdry
%  O7=outerbdrynodes; Vertices on the outer boundary
%  O8=ELsolid;        Global element numbers of vertices on solid domain
%  O9=ELfluid;        Global element numbers of vertices on fluid domain
%  O10=edgetri;       List of global element #'s in solid touching bdry
%-------------------------------------------------------------------------

global searchtriangles
global solidbdrynodes

disp('Gathering mesh information...')
%Choose particular mesh to use:
if input==1
   usemesh='solidfluid.msh';
elseif input==2
```

```
    usemesh='solidfluid2.msh';
elseif input==3
    usemesh='solidfluid3.msh';
elseif input==4
    usemesh='solidfluid4.msh';
elseif input==5
    usemesh='solidfluid5.msh';
elseif input==6
    usemesh='solidfluid6.msh';
elseif input==7
    usemesh='solidfluid7.msh';
elseif input==-1                % <----debugging with mesh by hand
    usemesh='fixedhandmesh1.msh';
elseif input==-2                % <----debugging with mesh by hand
    usemesh='fixedhandmesh2.msh';
else
    usemesh='solidfluid.msh';
end
disp([' - Using Mesh:  ',usemesh])



msh = load_gmsh2(usemesh,[1 2]); % [1,2] requests variables for lines and
    triangles only

 M=msh.nbNod;                  % Number of nodes in mesh
 NODECO=(1/3)*msh.POS(1:M,1:2);     % NODE COordinates
```

```
msh.POS(1:M,1:2)=NODECO;

L=msh.nbTriangles;            % Number of elements (triangular)
ELNODE=msh.TRIANGLES(1:L,1:4); % Global node numbers for all elements;
    element domain
chtype=ELNODE(1,end);         % get tag for domain type; assume first type
    listed is fluid
k=0;
j=0;
for i=1:L                     % get a list of elements in each domain
    if ELNODE(i,4)~=chtype
        k=k+1;
        ELsolid(k,1:3)=ELNODE(i,1:3); %coordinates of elements in solid domain
    else
        j=j+1;
        ELfluid(j,1:3)=ELNODE(i,1:3); %coordinates of elements in fluid domain
    end
end
nELsolid=k;                   % number of elements in the solid domain
nELfluid=j;                   % number of elements in the fluid domain

% The following information can be used to subdivide the boundary into
    segments
%global L1
%L1=msh.nbLines;              % Number of element edges on the natural
    boundary
```

```
%global BDRYEDGES
BDRYEDGES=msh.LINES;          % EDGES (node, node, ...)

[nBdryNodes,dum]=size(BDRYEDGES); % total number of nodes on a boundary
k=1;
k2=1;
chtype=BDRYEDGES(1,end);
for i=1:nBdryNodes
    if BDRYEDGES(i,end)==chtype
        outerbdrynodes(k,1)=BDRYEDGES(i,1);
        k=k+1;
    else
        solidbdrynodes(k2,1)=BDRYEDGES(i,1);
        k2=k2+1;
    end
end

%----------------------
%Determine fluid/solid nodes
%----------------------
 nfluidpts=0;
 nsolidpts=0;
 temp=ELfluid(:,1:3);
 temp2=ELsolid(:,1:3);
 for k=1:M
     check=isempty(find(k==temp));
```

```
    if check==0
        nfluidpts=nfluidpts+1;
        fluidpts(nfluidpts)=k;
    end
    check=isempty(find(k==temp2));
    if check==0
        nsolidpts=nsolidpts+1;
        solidpts(nsolidpts)=k;
    end
 end
 otherDim=nfluidpts;
 nsolidnodes=nsolidpts;
%-------------------
% Midpoint Database
%-------------------
   function [out]=mdptcalc(v1,v2)
       % Gives midpoint (x,y) coordinates between global nodes v1 and v2
       % also places a flag for the domain
       %  out(3)=0 means pt in interior of solid
       %  out(3)=1 means pt on solid bdry
       %  out(3)=2 means pt outside solid domain not on solid bdry
       v1coord=NODECO(v1,1:2);
       v2coord=NODECO(v2,1:2);
       out(1)=.5*(v1coord(1)+v2coord(1));
       out(2)=.5*(v1coord(2)+v2coord(2));
       out(3)=0;
```

```
        check1=isempty(find(v1==fluidpts)); %is v1 in fluid domain, 0=yes,1=no
        check2=isempty(find(v2==fluidpts)); %is v2 in fluid domain, 0=yes,1=no
        if check1+check2==0 %if both vertices are in fluid domain
            index1=find(v1==solidbdrynodes); % where is v1 on solid bdry if at
                all?
            index2=find(v2==solidbdrynodes);
            check3=isempty(index1);
            check4=isempty(index2);
            if check3+check4==0 %if both vertices are on the solid bdry
                if abs(index1-index2)==1 || abs(index1-index2)==length(
                    solidbdrynodes)-1
                    out(3)=1;
                end
            else
                out(3)=2;
            end
        end
    end

disp('Generating midpoint database...')
disp(' ')
%Mdpts= [x-coord, y-coord, bdry?, local node #1, global elem #1, local node
    #2,
        % global element #2, fluid/solid]
%  loctype is from 1-6 depending on what node is across
%  midpoint could be used twice, so have 2 loctype and node2
```

```
% If a point is in both domains, notated as 1, fluid=2, solid=0
 tol=10^(-8);
 Mdpts=zeros(3*M,8);  %pre-allocate space since no more than triple the nodes
    for mdpts
 Mdpts(:,3)=ones(3*M,1); %assume midpoint is on boundary until proven not
 nMdpts=1; % will count the number of unique midpoints in the mesh
 for i=1:L % cycle through each triangle (can we skip solid domain midpts?)
    v1=ELNODE(i,1); v2=ELNODE(i,2); v3=ELNODE(i,3);
    mid=zeros(3,4);
    mid(1,1:3)=mdptcalc(v2,v3); %organize midpoints in same way as local basis
        phi's
    mid(2,1:3)=mdptcalc(v3,v1);
    mid(3,1:3)=mdptcalc(v1,v2);
    mid(:,4)=[4;5;6];
    %Check to see if any of the above are repeats
    for j=1:3
       vec=ones(nMdpts,1);
       diff_x=abs(mid(j,1)*vec-Mdpts(1:nMdpts,1));
       diff_y=abs(mid(j,2)*vec-Mdpts(1:nMdpts,2));
       check=min(diff_x+diff_y);
       if check>tol
          Mdpts(nMdpts,1:2)=mid(j,1:2);
          Mdpts(nMdpts,4)=mid(j,4);
          Mdpts(nMdpts,5)=i;
          Mdpts(nMdpts,8)=mid(j,3);
          nMdpts=nMdpts+1;
```

```
        else
           [dum,place]=min(diff_x+diff_y);
           Mdpts(place,6:7)=[j+3,i];
           Mdpts(place,3)=0;       %mark that your midpoint isn't on bdry
        end
    end
end
nMdpts=nMdpts-1;
Mdpts=Mdpts(1:nMdpts,1:8);

%----------------------------------------
%Determine the fluid/solid midpoints
%----------------------------------------
temp=Mdpts(:,8);
count=0;
clear solperimeter
for k=1:nMdpts
    if temp(k)~=0
        nfluidpts=nfluidpts+1;
        fluidpts(nfluidpts)=k+M;
    end
    if temp(k)~=2
        nsolidpts=nsolidpts+1;
        solidpts(nsolidpts)=k+M;
        if temp(k)==1
            count=count+1;
```

```
            solperimeter(count,1)=k+M;
        end
    end
 end
 solperimeter=[solidbdrynodes; solperimeter];
 nsolperimeter=length(solperimeter);



 %Build a list of global node numbers that intersect the solid boundary,
 %edgetri
edgetri=zeros(0,2);
count=0;
chtype=ELNODE(1,end);         % get tag for domain type; assume first type
   listed is fluid
for q=1:L
   if ELNODE(q,end)~=chtype  % if we are looking at an element in solid
       domain
       count=count+1;
       v1=ELNODE(q,1);
       test1=isempty(find(v1==solidbdrynodes)); %see if any of the vertices
           are on solid bdry
       v2=ELNODE(q,2);
       test2=isempty(find(v2==solidbdrynodes)); %test=0 means yes on the bdry
       v3=ELNODE(q,3);
       test3=isempty(find(v3==solidbdrynodes));
       if test1+test2+test3==1 % if two are on bdry
```

```
            edgetri=[edgetri;[q,count]]; %record the global element number and
                solid global element number
        end
    end
end
nedgetri=length(edgetri); % number of solid elements on bdry
```

```
 cons=[nELsolid,nELfluid,otherDim,nsolidpts,nfluidpts,nMdpts,nsolperimeter,
     nedgetri];

 searchtriangles=[Mdpts(:,5),Mdpts(:,7)]; %Global element numbers for each
     midpoint


 % DEFINE SHORT-NAME OUTPUTS:
 O1=msh;
 O2=Mdpts;
 O3=cons;
 O4=solidpts;
```

```
 O5=fluidpts;
 O6=solperimeter;
 O7=outerbdrynodes;
 O8=ELsolid;
 O9=ELfluid;
 O10=edgetri;

end
```

A.17 getTangVec.m

```
function [ tang,p1,p2,leng ] = getTangVec( ii,jj )
%Function to help in taking line integrals in the boundary term of the
%extension problem
% leng = length between p1 and p2
% tang = tangent vector between global nodes ii and jj

global M              % Number of vertex elements in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Number of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global solidbdrynodes % List of Vertices on Solid Bdry
global solidpts       % List of global node numbers on Solid Domain
```

```
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global searchtriangles % Submatrix of Mdpts; contains local node information
global S              % Stiffness matrix for the extension problems w/identity
global preS           % Stiffness matrix for the ext. problems w/out identity
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Constant given in problem (adjust to progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
    coord)
global nsolperimeter  % Number of nodes on the solid boundary
global AllDPhi        % Library of all solutions to D_lambda(Phi) in solid
global Lambda         % GO BACK AND DEFINE LAMBDA LATER
global mu             % GO BACK AND DEFINE LATER
global nu             % GO BACK AND DEFINE LATER


% Get coordinates, p1 and p2, of global nodes ii and jj
ii=mod(ii,M+nMdpts); %to get coordinates, we should look at the
                     %first coord set of nodes
jj=mod(jj,M+nMdpts);
```

```
if ii<=M %if we are looking at a vertex
    pii=NODECO(ii,1:2); %get coordinate pair
    pii=pii';
else            %if we are looking at a midpoint
    pii=Mdpts(ii-M,1:2); %get coordinate pair
    pii=pii';
end
if jj<=M %if we are looking at a vertex
    pjj=NODECO(jj,1:2); %get coordinate pair
    pjj=pjj';
else            %if we are looking at a midpoint
    pjj=Mdpts(jj-M,1:2); %get coordinate pair
    pjj=pjj';
end


%Determine counterclockwise ordering of nodes ii and jj w/coordinate pii
%and pjj
%  Vertex nodes are listed in counterclockwise order in solidbdrynodes
locii=find(ii==solidbdrynodes); %find where ii and jj fall in solidbdrynodes
locjj=find(jj==solidbdrynodes);
if locjj-locii==1 %if order in solidbdrynodes is ii then jj; typical order
    p1=pii;
    p2=pjj;
else              %if ii and jj are not successive entries; opposite order
    p1=pjj;
```

```
    p2=pii;
    %disp('SWAP ORDER')
end
```

```
tang=p2-p1;       % tangent vector for line segment from p1 to p2
leng=norm(tang);
tang=(1/leng)*tang; % normalize the tangent vector
```

```
% % % Define the needed maps and constants
% % DL=[ p2(1)-p1(1), p1(2)-p2(2); p2(2)-p1(2), p2(1)-p1(1) ]; %rotate and
    rescale reference interval (0,1)
% % DLinv=inv(DL);           %inverse mapping
% % leng=norm(p2-p1);        %length of interval
% % xL=@(p) DL*p+p1;         %shift interval p1 units
% % xLinv=@(p) DLinv*(p-p1); %inverse mapping of xL
```

```
end
```

A.18 GEx.m

```
function [ out ] = GEx( ii,iwhichhalf,VUvec,AllDPhi,H1,U1bar,V1bar,L2,...
                        testflag)
%Incorporates the extension terms into the G functional that makes up the
%right hand side of the overarching system of equations

%==========================================================================
%   Global Variables and Important Constants
%==========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
```

```
global edgetri        % List of triangular elements with two nodes on solid
    boundary
global nedgetri       % Number of triangular elements with two nodes on solid
    boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
    progress in time)
global T              % Number of local basis functions Phi (12; 6 in each
    coord)
global T2             % Number of local basis functions Eta (6; 3 in each
    coord)

global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % GO BACK AND DEFINE LAMBDA LATER
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % GO BACK AND DEFINE LATER


%==========================================================================
%   Interior Terms
%==========================================================================
i=find(ii==solperimeter);

if iwhichhalf==2
```

```
    i=i+nsolperimeter;
end

iVec=AllDPhi(:,i); % this is the appropriate D_{S,lambda^2}[0,Phi_ii]

interiorterms=(-1)*VUvec'*H1*iVec; %notice the (-1) is already factored in

interiorterms=interiorterms+(1/kappa^2)*(V1bar+lambda*U1bar)'*L2*iVec;
%interiorterms=interiorterms+(1/kappa^2)*lambda*Ubar'*L2*iVec;



%add in the final term listed on pg.36 of the paper

%==========================================================================
%   Boundary Integral Terms
%==========================================================================

value=0;
nodes=zeros(1,6);
for q=1:nedgetri
    nodes(1,1:3)=ELNODE(edgetri(q,1),1:3);

    [row,col]=find(searchtriangles==edgetri(q,1));
    localnum(1,1)=Mdpts(row(1),2*col(1)+2);
    localnum(2,1)=Mdpts(row(2),2*col(2)+2);
    localnum(3,1)=Mdpts(row(3),2*col(3)+2);
```

```
nodes(1,4)=row(find(4==localnum))+M;
nodes(1,5)=row(find(5==localnum))+M;
nodes(1,6)=row(find(6==localnum))+M;

[J,DxKinv,xK,xKinv]=getMapInfo(edgetri(q,2),2);
%abs(jacobian) & inv(D(affine map)), affine map, inverse affine map

% GET COORDINATES TO INTEGRATE OVER
ver=nodes(1:3); %pull the vertices of the element

%Find which two vertices are on the solid bdry
test(1)=isempty(find(ver(1)==solidbdrynodes)); %test=0 means on bdry
test(2)=isempty(find(ver(2)==solidbdrynodes));
test(3)=isempty(find(ver(3)==solidbdrynodes));
whichnodes=find(min(test)==test);
%find which two entries of test are 0
starting=ver(whichnodes(1));
%global node number to start integration on
finishing=ver(whichnodes(2));
%global node number to finish integration on

if starting<=M %if we are looking at a vertex
    p1=NODECO(starting,1:2); %get coordinate pair
    p1=p1';
else            %if we are looking at a midpoint
```

```
    p1=Mdpts(starting-M,1:2); %get coordinate pair
    p1=p1';
end
if finishing<=M %if we are looking at a vertex
    p2=NODECO(finishing,1:2); %get coordinate pair
    p2=p2';
else            %if we are looking at a midpoint
    p2=Mdpts(finishing-M,1:2); %get coordinate pair
    p2=p2';
end

%Make sure we don't need to swap the order of p1 and p2
% here we assume nodes are given in counterclockwise order from
% getMesh.m
loc1=find(starting==ver);
loc2=find(finishing==ver);
if loc2-loc1~=1; %swap needed
    temp=p1;
    p1=p2;
    p2=temp;
end

tang=p2-p1;
tang=(1/norm(tang))*tang; %get tangent vector

n1=find(nodes(1,1)==solidpts); % use global solid node numbers
```

```
n2=find(nodes(1,2)==solidpts); % to assign coef's to nodes
n3=find(nodes(1,3)==solidpts);
n4=find(nodes(1,4)==solidpts);
n5=find(nodes(1,5)==solidpts);
n6=find(nodes(1,6)==solidpts);
n7=n1+nsolidpts;
n8=n2+nsolidpts;
n9=n3+nsolidpts;
n10=n4+nsolidpts;
n11=n5+nsolidpts;
n12=n6+nsolidpts;

%D_{S,lambda^2}(0,Phi)=iSol
iSol=@(p) iVec(n1)*Phi(1,p)+iVec(n2)*Phi(2,p)+iVec(n3)*Phi(3,p)+...
    iVec(n4)*Phi(4,p)+iVec(n5)*Phi(5,p)+iVec(n6)*Phi(6,p)+...
    iVec(n7)*Phi(7,p)+iVec(n8)*Phi(8,p)+iVec(n9)*Phi(9,p)+...
    iVec(n10)*Phi(10,p)+iVec(n11)*Phi(11,p)+iVec(n12)*Phi(12,p);

%VU = function with VUvec coefficients
VU=@(p) VUvec(n1)*Phi(1,p)+VUvec(n2)*Phi(2,p)+VUvec(n3)*Phi(3,p)+...
    VUvec(n4)*Phi(4,p)+VUvec(n5)*Phi(5,p)+VUvec(n6)*Phi(6,p)+...
    VUvec(n7)*Phi(7,p)+VUvec(n8)*Phi(8,p)+VUvec(n9)*Phi(9,p)+...
    VUvec(n10)*Phi(10,p)+VUvec(n11)*Phi(11,p)+VUvec(n12)*Phi(12,p);
%Derivatives of VU above (including DxKinv!!)
DVU=@(p) (VUvec(n1)*DiffPhi(1,p)+VUvec(n2)*DiffPhi(2,p)...
    +VUvec(n3)*DiffPhi(3,p)+VUvec(n4)*DiffPhi(4,p)...
```

```
        +VUvec(n5)*DiffPhi(5,p)+VUvec(n6)*DiffPhi(6,p)+...
        VUvec(n7)*DiffPhi(7,p)+VUvec(n8)*DiffPhi(8,p)...
        +VUvec(n9)*DiffPhi(9,p)+VUvec(n10)*DiffPhi(10,p)...
        +VUvec(n11)*DiffPhi(11,p)+VUvec(n12)*DiffPhi(12,p))*DxKinv;



    %Apply Boundary Operator B to iSol: B(iSol)=BiSol
    if testflag~=4
        BVU=@(p) e1*VU(xKinv(p))+e2*DVU(xKinv(p))*tang;
    end
    if testflag==4
        BVU=@(p) e1*VU(xKinv(p))...
            +e2*(p(1)^2+p(2)^2)*DVU(xKinv(p))*tang;
    end

    %Output Integrand
    integ=@(p) BVU(p)'*iSol(xKinv(p));
    %we are incorporating the two negatives as a positive for the BVU term

    value=quadLine(integ,p1,p2)+value;
end

out=interiorterms+value;
```

```
end
```

A.19 lamDw.m

```
function [ out ] = lamDw( p,testflag )
% used to calculate inv(lambda*I - Dw) in the extension problems
%  if you change Dw here, don't forget to change w=wplain.m

global lambda     %lambda from dissipativity (not the Lame constant)
global e3         %epsilon constant

x=p(1); y=p(2); %get the x and y values from input p

% IF YOU CHANGE DW, YOU MUST CHANGE w=wplain.m ALSO!!
%Dw=e3*[1 2*y; 2*x 1]; %using w=wplain=e3*[ x+y^2; y+x^2 ] %no big changes
%Dw=e3*[2*x -1; 1 2*y]; %using w=wplain=e3*[ x^2-y; y^2+x ]



if testflag==0;
   Dw=[0 0; 0 0]; %Eigenfunction test problem
elseif testflag==1
   Dw=[0 0; 0 0]; %analytic solution test problem
elseif testflag==2 %case for testing sensitivity of e3
   %Dw=e3*1000*[x*y*(x - 1)*(x - 1/3)*(y - 1)*(y - 1/3)*(y - 2/3)...
   %  + x*y*(x - 1)*(x - 2/3)*(y - 1)*(y - 1/3)*(y - 2/3)...
   %  + x*y*(x - 1/3)*(x - 2/3)*(y - 1)*(y - 1/3)*(y - 2/3)...
```

```
    %  + y*(x - 1)*(x - 1/3)*(x - 2/3)*(y - 1)*(y - 1/3)*(y - 2/3),...
    %  x*y*(x - 1)*(x - 1/3)*(x - 2/3)*(y - 1)*(y - 1/3)...
    %  + x*y*(x - 1)*(x - 1/3)*(x - 2/3)*(y - 1)*(y - 2/3)...
    %  + x*y*(x - 1)*(x - 1/3)*(x - 2/3)*(y - 1/3)*(y - 2/3)...
    %  + x*(x - 1)*(x - 1/3)*(x - 2/3)*(y - 1)*(y - 1/3)*(y - 2/3); 0 0];
    Dw=e3*[0 1-2*y; 1-2*x 0];
elseif testflag==3 %second analytic solution test problem
    Dw=[0 0; 0 0];
elseif testflag==4 %case for testing sensitivity of e2 with variable coefs
    Dw=e3*[0 1-2*y; 1-2*x 0];
end



out=lambda*eye(2)-Dw;
```

A.20 load_gmsh2.m

```
function msh = load_gmsh2(filename, which)

%% Reads a mesh in msh format, version 1 or 2

% Usage:
% To define all variables m.LINES, m.TRIANGLES, etc
% (Warning: This creates a very large structure. Do you really need it?)
%           m = load_gmsh4('a.msh')
%
% To define only certain variables (for example TETS and HEXS)
```

```
%           m = load_gmsh4('a.msh', [ 5 6])
%
% To define no variables (i.e. if you prefer to use m.ELE_INFOS(i,2))
%           m = load_gmsh4('a.msh', -1)
%           m = load_gmsh4('a.msh', [])
%
% Copyright (C) 2007 JP Moitinho de Almeida (moitinho@civil.ist.utl.pt)
% and R Lorphevre(r(point)lorphevre(at)ulg(point)ac(point)be)
%
% based on load_gmsh.m supplied with gmsh-2.0

% Structure msh always has the following elements:
%
% msh.MIN, msh.MAX - Bounding box
% msh.nbNod - Number of nodes
% msh.nbElm - Total number of elements
% msh.nbType(i) - Number of elements of type i (i in 1:19)
% msh.POS(i,j) - j'th coordinate of node i (j in 1:3, i in 1:msh.nbNod)
% msh.ELE_INFOS(i,1) - id of element i (i in 1:msh.nbElm)
% msh.ELE_INFOS(i,2) - type of element i
% msh.ELE_INFOS(i,3) - number of tags for element i
% msh.ELE_INFOS(i,4) - Dimension (0D, 1D, 2D or 3D) of element i
% msh.ELE_TAGS(i,j) - Tags of element i (j in 1:msh.ELE_INFOS(i,3))
% msh.ELE_NODES(i,j) - Nodes of element i (j in 1:k, with
%                     k = msh.NODES_PER_TYPE_OF_ELEMENT(msh.ELE_INFOS(i,2)))
%
```

```
% These elements are created if requested:
%
% msh.nbLines - number of 2 node lines
% msh.LINES(i,1:2) - nodes of line i
% msh.LINES(i,3) - tag (WHICH ?????) of line i
%
% msh.nbTriangles - number of 2 node triangles
% msh.TRIANGLES(i,1:3) - nodes of triangle i
% msh.TRIANGLES(i,4) - tag (WHICH ?????) of triangle i
%
% Etc

% These definitions need to be updated when new elemens types are added to
    gmsh
%
% msh.Types{i}{1} Number of an element of type i
% msh.Types{i}{2} Dimension (0D, 1D, 2D or 3D) of element of type i
% msh.Types{i}{3} Name to add to the structure with elements of type i
% msh.Types{i}{4} Name to add to the structure with the number of elements of
    type i
%

nargchk(1, 2, nargin);

msh.Types = { ...
    { 2, 1, 'LINES', 'nbLines'}, ... % 1
```

```
    { 3, 2, 'TRIANGLES', 'nbTriangles'}, ...
    { 4, 2, 'QUADS', 'nbQuads'}, ...
    { 4, 3, 'TETS', 'nbTets'}, ...
    { 8, 3, 'HEXAS', 'nbHexas'}, ... %5
    { 6, 3, 'PRISMS', 'nbPrisms'}, ...
    { 5, 3, 'PYRAMIDS', 'nbPyramids'}, ...
    { 3, 1, 'LINES3', 'nbLines3'}, ...
    { 6, 2, 'TRIANGLES6', 'nbTriangles6'}, ...
    { 9, 2, 'QUADS9', 'nbQuads9'}, ... % 10
    { 10, 3, 'TETS10', 'nbTets10'}, ...
    { 27, 3, 'HEXAS27', 'nbHexas27'}, ...
    { 18, 3, 'PRISMS18', 'nbPrisms18'}, ...
    { 14, 3, 'PYRAMIDS14', 'nbPyramids14'}, ...
    { 1, 0, 'POINTS', 'nbPoints'}, ... % 15
    { 8, 3, 'QUADS8', 'nbQuads8'}, ...
    { 20, 3, 'HEXAS20', 'nbHexas20'}, ...
    { 15, 3, 'PRISMS15', 'nbPrisms15'}, ...
    { 13, 3, 'PYRAMIDS13', 'nbPyramids13'}, ...
};

ntypes = length(msh.Types);

if (nargin==1)
    which = 1:ntypes;
else
    if isscalar(which) && which == -1
```

```
        which = [];
    end
end

% Could check that "which" is properlly defined....

fid = fopen(filename, 'r');
fileformat = 0; % Assume wrong file

tline = fgetl(fid);
if (feof(fid))
    disp (sprintf('Empty file: %s', filename));
    msh = [];
    return;
end

%% Read mesh type
if (strcmp(tline, '$NOD'))
    fileformat = 1;
elseif (strcmp(tline, '$MeshFormat'))
    fileformat = 2;
    tline = fgetl(fid);
    if (feof(fid))
        disp (sprintf('Syntax error (no version) in: %s', filename));
        fileformat = 0;
    end
```

```
    [ form ] = sscanf( tline, '%f %d %d');
    if (form(1) ~= 2.2)
        disp (sprintf('Unknown mesh format: %s', tline));
        fileformat = 0;
    end
    if (form(2) ~= 0)
        disp ('Sorry, this program can only read ASCII format');
        fileformat = 0;
    end
    fgetl(fid);   % this should be $EndMeshFormat
    if (feof(fid))
        disp (sprintf('Syntax error (no $EndMeshFormat) in: %s', filename));
        fileformat = 0;
    end
    tline = fgetl(fid);  % this should be $Nodes
    if (feof(fid))
        disp (sprintf('Syntax error (no $Nodes) in: %s', filename));
        fileformat = 0;
    end
end

if (~fileformat)
    msh = [];
    return
end
```

```
%% Read nodes

if strcmp(tline, '$NOD') || strcmp(tline, '$Nodes')
    msh.nbNod = fscanf(fid, '%d', 1);
    aux = fscanf(fid, '%g', [4 msh.nbNod]);
    msh.POS = aux(2:4,:)';
    numids = max(aux(1,:));
    IDS = zeros(1, numids);
    IDS(aux(1,:)) = 1:msh.nbNod; % This may not be an identity
    msh.MAX = max(msh.POS);
    msh.MIN = min(msh.POS);
    fgetl(fid); % End previous line
    fgetl(fid); % Has to be $ENDNOD $EndNodes
else
    disp (sprintf('Syntax error (no $Nodes/$NOD) in: %s', filename));
    fileformat = 0;
end

%% Read elements
tline = fgetl(fid);
if strcmp(tline,'$ELM') || strcmp(tline, '$Elements')
    msh.nbElm = fscanf(fid, '%d', 1);
    % read all info about elements into aux (it is faster!)
    aux = fscanf(fid, '%g', inf);
    start = 1;
    msh.ELE_INFOS = zeros(msh.nbElm, 4); % 1 - id, 2 - type, 3 - ntags, 4 -
```

```
    Dimension
msh.ELE_NODES = zeros(msh.nbElm,6); % i - Element, j - ElNodes
if (fileformat == 1)
    ntags = 2;
else
    ntags = 3; % just a prediction
end
msh.ELE_TAGS = zeros(msh.nbElm, ntags); % format 1: 1 - physical number, 2
     - geometrical number
                                        % format 2: 1 - physical number, 2 -
                                            geometrical number, 3 - mesh
                                            partition number
msh.nbType = zeros(ntypes,1);
for I = 1:msh.nbElm
    if (fileformat == 2)
        finnish = start + 2;
        msh.ELE_INFOS(I, 1:3) = aux(start:finnish);
        ntags = aux(finnish);
        start = finnish + 1;
        finnish = start + ntags -1;
        msh.ELE_TAGS(I, 1:ntags) = aux(start:finnish);
        start = finnish + 1;
    else
        finnish = start + 1;
        msh.ELE_INFOS(I, 1:2) = aux(start:finnish);
        start = finnish + 1; % the third element is nnodes, which we get
```

```
            from the type
        msh.ELE_INFOS(I, 3) = 2;
        finnish = start + 1;
        msh.ELE_TAGS(I, 1:2) = aux(start:finnish);
        start = finnish + 2;
    end
    type = msh.ELE_INFOS(I, 2);
    msh.nbType(type) = msh.nbType(type) + 1;
    msh.ELE_INFOS(I, 4) = msh.Types{type}{2};
    nnodes = msh.Types{type}{1};
    finnish = start + nnodes - 1;
    msh.ELE_NODES(I, 1:nnodes) = IDS(aux(start:finnish));
    start=finnish + 1;
end
fgetl(fid); % Has to be $ENDELM or $EndElements
else
    disp (sprintf('Syntax error (no $Elements/$ELM) in: %s', filename));
    fileformat = 0;
end

if (fileformat == 0)
    msh = [];
end

fclose(fid);
```

```
%% This is used to create the explicit lists for types of elements

for i = which
    if (~isempty(msh.Types{i}{3}))
        cmd = sprintf('msh.%s=msh.nbType(%d);', msh.Types{i}{4}, i);
        eval(cmd);
        % Dimension
        cmd = sprintf('msh.%s=zeros(%d,%d);', msh.Types{i}{3}, msh.nbType(i),
            msh.Types{i}{1}+1);
        eval(cmd);
        % Clear nbType for counting, next loop will recompute it
        msh.nbType(i) = 0;
    end
end

for i = 1:msh.nbElm
    type = msh.ELE_INFOS(i,2);
    if (find(which == type))
        if (~isempty(msh.Types{type}{3}))
            msh.nbType(type) = msh.nbType(type)+1;
            aux=[ msh.ELE_NODES(i,1:msh.Types{type}{1}), msh.ELE_TAGS(i,1) ];
            cmd = sprintf('msh.%s(%d,:)=aux;', msh.Types{type}{3}, msh.nbType(
                type));
            eval(cmd);
        end
    end
```

```
end

return;
```

A.21 Phi.m

```
function [ out ] = Phi( m,p )
%==========================================================================
%Basis functions for [H_0^1(\Omega)]^2 (full set of 12 for reference
%triangle)
%  Detailed explanation goes here
%==========================================================================

x=p(1); y=p(2); %Rename the input variables for readability

if m==1
    out(1,1)=(1-x-y)*(2*(1-x-y)-1);
    out(2,1)=0;
elseif m==2
    out(1,1)=x*(2*x-1);
    out(2,1)=0;
elseif m==3
    out(1,1)=y*(2*y-1);
    out(2,1)=0;
elseif m==4
    out(1,1)=4*x*y;
    out(2,1)=0;
```

```
elseif m==5
    out(1,1)=4*(1-x-y)*y;
    out(2,1)=0;
elseif m==6
    out(1,1)=4*(1-x-y)*x;
    out(2,1)=0;

%End of [Phi ; 0] basis functions. Begin [0 ; Phi] basis functions.
elseif m==7
    out(2,1)=(1-x-y)*(2*(1-x-y)-1);
    out(1,1)=0;
elseif m==8
    out(2,1)=x*(2*x-1);
    out(1,1)=0;
elseif m==9
    out(2,1)=y*(2*y-1);
    out(1,1)=0;
elseif m==10
    out(2,1)=4*x*y;
    out(1,1)=0;
elseif m==11
    out(2,1)=4*(1-x-y)*y;
    out(1,1)=0;
else %m==12
    out(2,1)=4*(1-x-y)*x;
    out(1,1)=0;
```

```
end




end
```

A.22 quadLine.m

```
function [ out ] = quadLine( f,p1,p2 )
% Basic Five Point Gaussian Quadrature Rule
%  Inputs a function f(p) to be integrated from p1 to p2
%  Outputs value of the line integral from coord p1 to coord p2 of f(p)

% Usual Nodes:
n=[.046910077, .2307653449, .5, .7692346551, .953089923]; %spaced for integ
   from 0 to 1
% Usual Weights:
w=[.1184634425, .2393143352, .2844444444, .2393143352, .1184634425]; %weighted
    for integ from 0 to 1

tang=p2-p1; %tangent vector
leng=norm(tang); %length of tangent vector
tang=(1/leng)*tang; %normalize tangent vector

t1=p1+n(1)*leng*tang;
t2=p1+n(2)*leng*tang;
```

```
t3=p1+n(3)*leng*tang;
t4=p1+n(4)*leng*tang;
t5=p1+n(5)*leng*tang;



integrand=@(t) f(p1+(p2-p1)*t)*leng;



% %
% % checkintegralvalue=checkintegralvalue
% % t1_t5=[t1 t2 t3 t4 t5];

% out=w(1)*f(t1)+w(2)*f(t2)+w(3)*f(t3)... %weighted sum of terms
%     +w(4)*f(t4)+w(5)*f(t5);
% out=out*leng;

out=quadv(integrand,0,1);

end
```

A.23 quadTri.m

```
function retval=quadTri(func)
% Use quadratures to integrate a function over a
% reference triangle with vertices (0,0), (1,0), (0,1)
%
% Source:
```

```
% http://people.sc.fsu.edu/~jburkardt/datasets/quadrature_rules_tri/
    quadrature_rules_tri.html
% (make the list of points and weights persistent across calls)

% Higher Precision Schemes
%{
%TOMS706_37, order 37, degree of precision 13, a rule from ACM TOMS algorithm
    #706.
persistent qXYW37
qXYW37=[0.33333333333333333333333333333333,    0.33333333333333333333333333333333,
       0.05173976606574413355517914542 2;
0.95027566292410556545035208952 0,     0.02486216853794721727482395523 9,
    0.00800779955556480159780412346 0;
0.02486216853794721727482395523 9,     0.95027566292410556545035208952 0,
    0.00800779955556480159780412346 0;
0.02486216853794721727482395523 9,     0.02486216853794721727482395523 9,
    0.00800779955556480159780412346 0;
0.17161491492383534755630479555 1,     0.41419254253808232622184760221 4,
    0.04686889898182164482322673207 1;
0.41419254253808232622184760221 4,     0.17161491492383534755630479555 1,
    0.04686889898182164482322673207 1;
0.41419254253808232622184760221 4,     0.41419254253808232622184760221 4,
    0.04686889898182164482322673207 1;
0.53941224367719044026309298551 1,     0.23029387816140477986845350724 4,
    0.04659094018397648796036177007 0;
0.23029387816140477986845350724 4,     0.53941224367719044026309298551 1,
```

```
   0.04659094018397648796036177007 0;
0.23029387816140477986845350724 4,     0.23029387816140477986845350724 4,
   0.04659094018397648796036177007 0;
0.77216003667653256175028557011 3,     0.11391998166173371912485721494 3,
   0.03101694331379638140764622013 1;
0.11391998166173371912485721494 3,     0.77216003667653256175028557011 3,
   0.03101694331379638140764622013 1;
0.11391998166173371912485721494 3,     0.11391998166173371912485721494 3,
   0.03101694331379638140764622013 1;
0.00908539994983535388357296474 0,     0.49545730002508232305821351763 2,
   0.01079161273663127362317824013 6;
0.49545730002508232305821351763 2,     0.00908539994983535388357296474 0,
   0.01079161273663127362317824013 6;
0.49545730002508232305821351763 2,     0.49545730002508232305821351763 2,
   0.01079161273663127362317824013 6;
0.06227729030588699349708364052 7,     0.46886135484705650325145817972 7,
   0.03219553424243161881941448220 5;
0.46886135484705650325145817972 7,     0.06227729030588699349708364052 7,
   0.03219553424243161881941448220 5;
0.46886135484705650325145817972 7,     0.46886135484705650325145817972 7,
   0.03219553424243161881941448220 5;
0.02207628965362440514244687693 1,     0.85130650417434855038945767222 3,
   0.01544583421070158381769290005 3;
0.02207628965362440514244687693 1,     0.12661720617202709693316364791 8,
   0.01544583421070158381769290005 3;
0.85130650417434855038945767222 3,     0.02207628965362440514244687693 1,
```

```
    0.015445834210701583817692900053;
0.851306504174348550389457672223,     0.126617206172027096933163647918,
    0.015445834210701583817692900053;
0.126617206172027096933163647918,     0.022076289653624405142446876931,
    0.015445834210701583817692900053;
0.126617206172027096933163647918,     0.851306504174348550389457672223,
    0.015445834210701583817692900053;
0.018620522802520968955913511549,     0.689441970728591295496647976487,
    0.017822989923178661888748319485;
0.018620522802520968955913511549,     0.291937506468887771754472382212,
    0.017822989923178661888748319485;
0.689441970728591295496647976487,     0.018620522802520968955913511549,
    0.017822989923178661888748319485;
0.689441970728591295496647976487,     0.291937506468887771754472382212,
    0.017822989923178661888748319485;
0.291937506468887771754472382212,     0.018620522802520968955913511549,
    0.017822989923178661888748319485;
0.291937506468887771754472382212,     0.689441970728591295496647976487,
    0.017822989923178661888748319485;
0.096506481292159228736516560903,     0.635867859433872768286976979827,
    0.037038683681384627918546472190;
0.096506481292159228736516560903,     0.267625659273967961282458816185,
    0.037038683681384627918546472190;
0.635867859433872768286976979827,     0.096506481292159228736516560903,
    0.037038683681384627918546472190;
0.635867859433872768286976979827,     0.267625659273967961282458816185,
```

```
   0.03703868368138462791854647219 0;
0.26762565927396796128245881618 5,     0.09650648129215922873651656090 3,
   0.03703868368138462791854647219 0;
0.26762565927396796128245881618 5,     0.63586785943387276828697697982 7,
   0.03703868368138462791854647219 0];

%TOMS612_28, order 28, degree of precision 11, a rule from ACM TOMS algorithm
   #612.
persistent qXYW28
qXYW28=[ 0.33333333333333333,  0.333333333333333333, 0.08797730116222190;
  0.9480217181434233,   0.02598914092828833,  0.008744311553736190;
  0.02598914092828833,  0.9480217181434233,   0.008744311553736190;
  0.02598914092828833,  0.02598914092828833,  0.008744311553736190;
  0.8114249947041546,   0.09428750264792270,  0.03808157199393533;
  0.09428750264792270,  0.8114249947041546,   0.03808157199393533;
  0.09428750264792270,  0.09428750264792270,  0.03808157199393533;
  0.01072644996557060,  0.4946367750172147,   0.01885544805613125;
  0.4946367750172147,   0.01072644996557060,  0.01885544805613125;
  0.4946367750172147,   0.4946367750172147,   0.01885544805613125;
  0.5853132347709715,   0.2073433826145142,   0.07215969754474100;
  0.2073433826145142,   0.5853132347709715,   0.07215969754474100;
  0.2073433826145142,   0.2073433826145142,   0.07215969754474100;
  0.1221843885990187,   0.4389078057004907,   0.06932913870553720;
  0.4389078057004907,   0.1221843885990187,   0.06932913870553720;
  0.4389078057004907,   0.4389078057004907,   0.06932913870553720;
  0.6779376548825902,   0.04484167758913055,  0.04105631542928860;
```

```
  0.6779376548825902,  0.27722066752827925, 0.04105631542928860;
  0.04484167758913055, 0.6779376548825902,  0.04105631542928860;
  0.04484167758913055, 0.27722066752827925, 0.04105631542928860;
  0.27722066752827925, 0.6779376548825902,  0.04105631542928860;
  0.27722066752827925, 0.04484167758913055, 0.04105631542928860;
  0.8588702812826364,  0.00000000000000000, 0.007362383783300573;
  0.8588702812826364,  0.1411297187173636,  0.007362383783300573;
  0.0000000000000000,  0.8588702812826364,  0.007362383783300573;
  0.0000000000000000,  0.1411297187173636,  0.007362383783300573;
  0.1411297187173636,  0.8588702812826364,  0.007362383783300573;
  0.1411297187173636,  0.0000000000000000,  0.007362383783300573];

%TOMS612_19, order 19, degree of precision 9, a rule from ACM TOMS algorithm
   #612.
persistent qXYW19
qXYW19=[ 0.33333333333333331,        0.33333333333333331,
   9.71357962827961025E-002;
  2.06349616025259287E-002,    0.48968251919873701,        3.13347002271398278
     E-002;
  0.48968251919873701,          2.06349616025259287E-002,  3.13347002271398278
     E-002;
  0.48968251919873701,    0.48968251919873701,      3.13347002271398278E-002;
  0.12582081701412900,     0.43708959149293553,     7.78275410047754301E-002;
  0.43708959149293553,        0.12582081701412900,        7.78275410047754301
     E-002;
  0.43708959149293553,         0.43708959149293553,       7.78275410047754301
```

```
    E-002;
  0.62359292876193562,          0.18820353561903219,       7.96477389272090969
    E-002;
  0.18820353561903219,    0.62359292876193562,      7.96477389272090969E-002;
  0.18820353561903219,     0.18820353561903219,     7.96477389272090969E-002;
  0.91054097321109406,      4.47295133944529688E-002,      2.55776756586981006
    E-002;
  4.47295133944529688E-002,   0.91054097321109406,         2.55776756586981006
    E-002;
  4.47295133944529688E-002,    4.47295133944529688E-002,    2.55776756586981006
    E-002;
  0.74119859878449801,    3.68384120547362581E-002, 4.32835393772893970E-002;
  0.74119859878449801,     0.22196298916076573,     4.32835393772893970E-002;
  3.68384120547362581E-002,    0.74119859878449801,        4.32835393772893970
    E-002;
  3.68384120547362581E-002,    0.22196298916076573,        4.32835393772893970
    E-002;
  0.22196298916076573,      0.74119859878449801,           4.32835393772893970
    E-002;
  0.22196298916076573,      3.68384120547362581E-002,      4.32835393772893970
    E-002];
%}


% STRANG7, order 7, degree of precision 5.
persistent qXYW7
```

```
if isempty(qXYW7) %only initialize qXYW7 if not already stored
qXYW7=[0.33333333333333333, 0.33333333333333333, 0.22500000000000000;
  0.79742698535308720, 0.10128650732345633, 0.12593918054482717;
  0.10128650732345633, 0.79742698535308720, 0.12593918054482717;
  0.10128650732345633, 0.10128650732345633, 0.12593918054482717;
  0.05971587178976981, 0.47014206410511505, 0.13239415278850616;
  0.47014206410511505, 0.05971587178976981, 0.13239415278850616;
  0.47014206410511505, 0.47014206410511505, 0.13239415278850616];
end

persistent qRule
qRule=qXYW7;
persistent qPts
dummy=size(qRule);
qPts=dummy(1);

%{
retval=0;
for i=1:qPts
    func(qRule(i,1:2));
    retval=retval+func(qRule(i,1:2))*qRule(i,3);
end
%}

%
%Instead of For loop above, speed process by hard coding
```

```
 retval = (func(qRule(1,1:2))*qRule(1,3)...
    +func(qRule(2,1:2))*qRule(2,3)+func(qRule(3,1:2))*qRule(3,3)...
    +func(qRule(4,1:2))*qRule(4,3)+func(qRule(5,1:2))*qRule(5,3)...
    +func(qRule(6,1:2))*qRule(6,3)+func(qRule(7,1:2))*qRule(7,3));
%}
%Steven's additional code to get the correct integral value:
retval=.5*retval;


end
```

A.24 solidfluid.msh

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
25
1 0 0 0
2 3 0 0
3 3 3 0
4 0 3 0
5 1 1 0
6 2 1 0
7 2 2 0
8 1 2 0
9 0.9999999999960245 0 0
10 1.999999999996158 0 0
```

```
11 3 0.9999999999960245 0
12 3 1.999999999996158 0
13 2.000000000001941 3 0
14 1.00000000000281 3 0
15 0 2.000000000001941 0
16 0 1.00000000000281 0
17 2.499999999998079 0.4999999999980123 0
18 0.4999999999980123 0.5000000000014051 0
19 2.50000000000097 2.49999999999903 0
20 0.5000000000009703 2.50000000000097 0
21 0.4999999999990297 1.500000000002375 0
22 1.500000000002375 2.50000000000097 0
23 1.499999999996091 0.4999999999980123 0
24 2.500000000001988 1.499999999996091 0
25 1.5 1.5 0
$EndNodes
$Elements
52
1 1 2 9 1 1 9
2 1 2 9 1 9 10
3 1 2 9 1 10 2
4 1 2 9 2 2 11
5 1 2 9 2 11 12
6 1 2 9 2 12 3
7 1 2 9 3 3 13
8 1 2 9 3 13 14
```

9 1 2 9 3 14 4

10 1 2 9 4 4 15

11 1 2 9 4 15 16

12 1 2 9 4 16 1

13 1 2 10 5 5 6

14 1 2 10 6 6 7

15 1 2 10 7 7 8

16 1 2 10 8 8 5

17 2 2 15 13 2 17 10

18 2 2 15 13 1 18 16

19 2 2 15 13 3 19 12

20 2 2 15 13 4 20 14

21 2 2 15 13 9 10 23

22 2 2 15 13 11 12 24

23 2 2 15 13 5 16 18

24 2 2 15 13 6 10 17

25 2 2 15 13 8 14 20

26 2 2 15 13 5 21 16

27 2 2 15 13 8 22 14

28 2 2 15 13 7 12 19

29 2 2 15 13 13 14 22

30 2 2 15 13 15 16 21

31 2 2 15 13 9 23 18

32 2 2 15 13 5 8 21

33 2 2 15 13 7 22 8

34 2 2 15 13 11 24 17

```
35 2 2 15 13 5 23 6
36 2 2 15 13 6 24 7
37 2 2 15 13 6 23 10
38 2 2 15 13 7 24 12
39 2 2 15 13 13 22 19
40 2 2 15 13 15 21 20
41 2 2 15 13 3 13 19
42 2 2 15 13 4 15 20
43 2 2 15 13 1 9 18
44 2 2 15 13 2 11 17
45 2 2 15 13 7 19 22
46 2 2 15 13 8 20 21
47 2 2 15 13 5 18 23
48 2 2 15 13 6 17 24
49 2 2 16 14 5 6 25
50 2 2 16 14 5 25 8
51 2 2 16 14 6 7 25
52 2 2 16 14 7 8 25
$EndElements
```

A.25 SolveEigenfunction.m

```
function [ Ubar ] = SolveEigenfunction( testflag )
%-------------------------------------------------------------------------
%Function to numerically solve the problem:
%                              S(Ubar) = 0              on solid domain
% Tbar_prime(Ubar) dot normal - B(Ubar) = -pressure*normal on solid bdry
```

```
%Essentially solve the system:
%   a_S(Ubar,W)=(0,W)_H1 - { normal,W }_Gamma
%   from p.26 in the paper, the S-Neumann problem
%This coefficient vector for Ubar will be used as part of the eigenfunction
%[Ubar; 0; 0] to our original system. With this eigenfunction vector, we
%can show that the rest of the code is consistent with this solution.
%-------------------------------------------------------------------------

%=========================================================================
%   Global Variables and Important Constants
%=========================================================================
global M              % Number of element vertices in mesh
global NODECO         % Vertex Coordinates
global L              % Number of triangular elements in mesh
global ELNODE         % Global node numbers of mesh elements
global Mdpts          % Database of Midpoint information; see getMesh.m
global nMdpts         % Numer of Midpoint nodes
global outerbdrynodes % List of Vertices on [0,1]x[0,1]
global solidbdrynodes % List of Vertices on Solid Bdry
global solperimeter   % List of Vertices & Midpoints on Solid Bdry
global nsolperimeter  % Number of nodes on the solid boundary
global solidpts       % List of global node numbers on Solid Domain
global nsolidpts      % Number of nodes in the solid domain
global fluidpts       % List of global node numbers on Fluid Domain
global nfluidpts      % Number of nodes in the fluid domain
global ELsolid        % Global node numbers of solid domain elements
```

```
global nELsolid       % Number of triangular elements in solid domain
global ELfluid        % Global node numbers of fluid domain elements
global nELfluid       % Number of triangular elements in solid domain
global edgetri        % List of triangular elements with two nodes on solid
   boundary
global nedgetri       % Number of triangular elements with two nodes on solid
   boundary
global searchtriangles % Submatrix of Mdpts; contains local node information
global otherDim       % Number of fluid vertices (also # of global Eta's)
global lambda         % Dissipativity constant given in problem (adjust to
   progress in time)
global pressure       % The constant pressure function
global T              % Number of local basis functions Phi (12; 6 in each
   coord)
global T2             % Number of local basis functions Eta (6; 3 in each
   coord)

global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % Parameter introduced to ensure ellipticity
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % Coefficient for Theta mapping


%=========================================================================
%   Test Case Error Flag
```

```
%==========================================================================
% Test case is assuming that lambda=1
if testflag==0
    if lambda~=1
        error('Input lambda must equal 1 in the eigenfunction test case')
    end
end

%==========================================================================
%   Define Products
%==========================================================================
    function out=Frob(q,r,s,p,DxKinv)
        %Frobenius product between elastic stress tensors
        % e(u)..e(v)=out (lowercase epsilons;
        % i.e. no kappa's yet for intermediate problem)

        %Symmetric Stress Tensor
        ephi=@(k,p) (.5)*(DiffPhi(k,p)*DxKinv+DxKinv'*DiffPhi(k,p)');

        out=trace(ephi(r,p)'*ephi(s,p)); %Frobenius Product

    end

    function out=a_Sinterior(q,r,s)
        %Function to build the stiffness matrix in the extension
        %problems, preS. S is the same matrix except partially replaced by
```

```
    %identity on boundary entries
    % Essentially we have a_S(Phi_r, Phi_s) here w/out bdry integral
    % term

    [J,DxKinv,xK,xKinv]=getMapInfo(q,2);
        %J=jacobian, DxKinv=(DxK)^-1, xK=map from ref to mesh triangle
        %note the 2 indicates we want the solid domain

    EXinteg1=@(p) (1/kappa^2)*Phi(r,p)'*Phi(s,p);
    EXinteg2=@(p) (1/kappa^2)*( ...
        lam*kappa^4*trace(DiffPhi(r,p)*DxKinv)*...
        trace(DiffPhi(s,p)*DxKinv)+2*mu*kappa^4*Frob(q,r,s,p,DxKinv));
        %notice that the pg.22 code may have a typo; there should be a
        %2 in front of the mu constant
    EXinteg3=@(p) (lam+mu)*(kappa^2-1)*trace(DxKinv'*DiffPhi(r,p)'...
        *DiffPhi(s,p)*DxKinv);
        % using trace is another way to write Frob matrix product

    %The boundary line integral is omitted from this product. It will
    %be addressed in a separate loop.

    EXinteg=@(p) J*(EXinteg1(p)+EXinteg2(p)+EXinteg3(p));

    out=quadTri(EXinteg); %the interior terms of a_S(Phi_r,Phi_s)
end
```

```
function out=a_Sbdry(q,r,s,ii,jj)
    %the boundary line integral portion of a_S(Phi_r,Phi_s) from global
    % node ii to global node jj
    %Function to build the stiffness matrix in the extension problems,
    %preS. S is the same matrix except partially replaced by the
    %identity on boundary entries
    %  ii, jj are global node numbers over whole domain
    %  q is global element number over solid domain
    %  r and s are degrees of freedom
    %
    % out = { B(Phi_r), Phi_s }_{solid bdry}
    % B(V)= e1*V + e2*DV*tangent_vec

    % I am assuming the element triangle vertices are given in
    % counterclockwise order.
    [tang,p1,p2,leng]=getTangVec(ii,jj); %get tangent vector,
                                         %begin/end coord,
                                         %segment length
    [J,DxKinv,xK,xKinv]=getMapInfo(q,2);
    %jacobian, inv(Dmap), map (ref to mesh), inverse map (mesh to ref)

    Bphi=@(p) e1*Phi(s,xKinv(p))+e2*DiffPhi(s,xKinv(p))*DxKinv*tang;
    %e1 and e2 are global constants given above; relatively small

    lineinteg=@(p) (-1)*Bphi(p)'*Phi(r,xKinv(p));
```

```
        out=quadLine(lineinteg,p1,p2);
    end

    function out=rhsbdry(q,r,ii,jj)
        [J,DxKinv,xK,xKinv]=getMapInfo(q,2);
        [tang,p1,p2,leng]=getTangVec(ii,jj);
        normal=[0 1; -1 0]*tang;
        %normal vector is the tangent vector rotated pi/2 clockwise

        integ=@(p) (-1)*pressure*normal'*Phi(r,xKinv(p));
        out=quadLine(integ,p1,p2);
    end


%==========================================================================
%   Build System of Equations
%==========================================================================
% Notice that (0,Phi)_H1=0, so we only need to worry about calculating
% -{normal,Phi)_Gamma

disp(' ')
disp('Solving Eigenfunction Problem...')
%Initialize stiffness matrix and rhs vector
Stiff=sparse(zeros(2*nsolidpts,2*nsolidpts));
rhs=zeros(2*nsolidpts,1);
```

```
disp(' - Taking integrals over interior...')
%Integrate over each mesh element in the solid to build stiff. mat. Stiff
% Here I am assuming that the solid elements are listed last from getMesh.m
for q=1:nELsolid   %cycle through each mesh element in the solid domain
    string=[' -- for element ', num2str(q),' of ',num2str(nELsolid)];
    disp(string)
    for r=1:T      %cycle through each Phi_i; r is local basis number

        %Get  i=row of S to add product to (global node number in only
        %      solid domain)
        %   ii=global node number (not just in solid domain)
        %   iwhichhalf= 1 or 2 based on which coordinate Phi_i is nonzero
        [i,ii,iwhichhalf]=getGlobalNodeNum(q,r,2);
            %ii and iwhichhalf are irrelevant for this section of code.
            %they will come up later building the overarching system.
            % the 2 corresponds to the solid domain

        for s=1:r %cycle through columns; use symmetry to stop at s=r
            [j,jj,jwhichhalf]=getGlobalNodeNum(q,s,2);

            value=a_Sinterior(q,r,s); %get value of appropriate integrals

            Stiff(i,j)=Stiff(i,j)+value; %add that value to entry in S
            Stiff(j,i)=Stiff(i,j);
        end
```

```
    end
end

disp(' - Taking line integrals...')
%-LOOP TO BUILD RHS AND LINE INTEGRALS OF SYSTEM:
for k=1:nedgetri %cycle through each element in solid touching solid bdry
    disp([' -- for element ',num2str(k),' of ',...
        num2str(nedgetri),' on solid boundary'])
    qs=edgetri(k,2); %qs is the global solid element number
    ver=ELNODE(edgetri(k),1:3); %pull the vertices of the element
    [J,DxKinv,xK,xKinv]=getMapInfo(qs,2);

    %Find which two vertices are on the solid bdry
    %  We are assuming the only options are 0 vertices on boundary or 2
    %  vertices on boundary. In reality 1 and 3 vertices on boundary are
    %  possible. This restricts the types of allowable meshes this code
    %  can use. At some point, go back and put the other two cases in this
    %  loop.
    test(1)=isempty(find(ver(1)==solidbdrynodes)); %test=0 means on bdry
    test(2)=isempty(find(ver(2)==solidbdrynodes));
    test(3)=isempty(find(ver(3)==solidbdrynodes));
    whichnodes=find(min(test)==test); %find which two entries of test are 0
    starting=ver(whichnodes(1)); %global node number to start integration
    finishing=ver(whichnodes(2)); %global node number to finish integration

    triangle=ELsolid(qs,1:3);
```

```
    c=getLocalPhi(test); %get the local node numbers on the solid bdry;
                         %based on which vertices were on solid bdry

    %Start the Integration Loops
    % Note that c is a vector of integers, so below we are looping from
    % index=c(1), index=c(2), index=c(3),..., index=c(6)
    for r=c
        [i,ii,iwhichhalf]=getGlobalNodeNum(qs,r,2);

        value=rhsbdry(qs,r,starting,finishing);
        rhs(i,1)=rhs(i,1)+value; %incorporate this value into rhs vector
        for s=c
            [j,jj,jwhichhalf]=getGlobalNodeNum(qs,s,2);

            value=a_Sbdry(qs,r,s,starting,finishing);
            Stiff(i,j)=Stiff(i,j)+value;
        end

    end
end


%==========================================================================
%   Solve System
%==========================================================================
% Unlike Dirichlet problem, we do not need to replace equations with Id
```

```
Ubar=Stiff\rhs;



fprintf('\n')



end
```

A.26 U1.m

```
function [ out ] = U1( p,testflag )
%Input function of intermediate elasticity problem

global lambda %dissipativity constant
x=p(1); y=p(2);

%U1 is zero on the solid bdry and has 0 derivative on the solid bdry
if testflag==1
    out(1,1)=10^6*(x-1/3)^2*(x-2/3)^2*(y-1/3)^2*(y-2/3)^2;
    out(2,1)=0;
elseif testflag==3
    out(1,1)= y;
    out(2,1)= -x;
end

% In both testflag=1 and testflag=3 cases, U'=lambda* U1 for the proposed
% analytical solution U'
```

```
out=lambda*out;

end
```

A.27 V1.m

```
function [ out ] = V1( p,testflag )
%Input function for intermediate elasticity problem

%==========================================================================
%   Global Variables and Important Constants
%==========================================================================
global e1 e2 e3       % Epsilon constants for intermediate problem
global Lambda         % Parameter introduced to guarantee ellipticity
global mu lam         % Lame constants
global nu             % Viscosity Constant
global kappa          % Coefficient of Theta mapping
%--------------------------------------------------------------------------

x=p(1); y=p(2);

% out(1,1)=0;
% out(2,1)=0;

% In both of the testflag=1 case and testflag=3 case, V1=S(U') for the
% proposed solution U'
```

```
if testflag==1
out(1,1)=1000000*(x - 1/3)^2*(x - 2/3)^2*(y - 1/3)^2*(y - 2/3)^2 ...
    - kappa^2*((2000000*kappa^2*mu*(9*x^2 - 9*x + 2)^2*(54*y^2 - 54*y + 13))
        /729 ...
+ (2000000*kappa^2*(lam + 2*mu)*(54*x^2 - 54*x + 13)*(9*y^2 - 9*y + 2)^2)/729
    ...
+ (2000000*(kappa^2 - 1)*(lam + mu)*(9*x^2 - 9*x + 2)^2*(54*y^2 - 54*y + 13))
    /729 ...
+ (2000000*(kappa^2 - 1)*(lam + mu)*(54*x^2 - 54*x + 13)*(9*y^2 - 9*y + 2)^2)
    /729);
out(2,1)= -(4000000*kappa^4*(lam + mu)*(18*x^3 - 27*x^2 + 13*x - 2)*(18*y^3 -
    27*y^2 + 13*y - 2))/81;
elseif testflag==3
    out(1,1)= y;
    out(2,1)= -x;
end
```

A.28 w1.m

```
function [ out ] = w1( p )
%Input function for r.h.s. of original maximality system

out=[0; 0];

end
```

A.29 wplain.m

```
function [ out ] = wplain( p,testflag )
%Input function for the intermediate elasticity problem
global e3

x=p(1); y=p(2);



if testflag==0 %Eigenfunction problem test case
    out=e3*[1; 1];
elseif testflag==1 %analytic solution test case
    out=e3*[1;1];
elseif testflag==2 %epsilon sensitivity case
    %out=e3*1000*[x*(1-x)*(1/3-x)*(2/3-x)*y*(1-y)*(1/3-y)*(2/3-y); 0];
    out=e3*[ (1-y)*y; (1-x)*x ];
elseif testflag==3 %second analytic solution test case
    out=e3*[1; 1];
elseif testflag==4 %epsilon sensitivity case with variable e2 coef
    out=e3*[ (1-y)*y; (1-x)*x ];
end





end
```

A.30 xKinvMap.m

```
function retval = xKinvMap(p1,p2,p3,p)
% Maps point p from the ELEMENT "K" with primary nodes p1,p2,p3 TO the %
    REFERENCE triangle "K_t"
    x = p(1); y = p(2);
    x1 =p1(1); y1=p1(2);
    x2 =p2(1); y2=p2(2);
    x3 =p3(1); y3=p3(2);
    Dinv=1/((x2-x1)*(y3-y1) - (y2-y1)*(x3-x1));
    retval = Dinv*[(y3-y1)*(x-x1) - (x3-x1)*(y-y1); -(y2-y1)*(x-x1) + (x2-x1)
        *(y-y1)];
end
```

A.31 xKJac.m

```
function retval = xKJac(p1,p2,p3)
% Jacobian of the map xK
% Map xK takes point p FROM the REFERENCE triangle "K_t" TO the ELEMENT %"K"
    with primary nodes p1,p2,p3
    x1 =p1(1); y1=p1(2);
    x2 =p2(1); y2=p2(2);
    x3 =p3(1); y3=p3(2);
    retval = (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);
end
```

A.32 xKmap.m

```
function retval = xKmap(p1,p2,p3,p)
% Map xK takes point p FROM the REFERENCE triangle "K_t" TO the ELEMENT %"K"
% with primary nodes p1,p2,p3, in that order for local numbering and %
    orientation

%   (0,0) <---> (x1,y1) <--> local nd 1 <--- PhiV1
%   (1,0) <---> (x2,y2) <--> local nd 2 <--- PhiV2
%   (0,1) <---> (x3,y3) <--> local nd 3 <--- PhiV3
    x = p(1); y = p(2);
    x1 =p1(1); y1=p1(2);
    x2 =p2(1); y2=p2(2);
    x3 =p3(1); y3=p3(2);
    retval = [(x2-x1)*x+(x3-x1)*y+x1, (y2-y1)*x+(y3-y1)*y+y1];
end
```