Computational Approaches to Improving Room Heating and Cooling for Energy Efficiency in Buildings

Brian K. McBee

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mathematics

John A. Burns, Chair
Jeffrey T. Borggaard
Eugene M. Cliff
Lizette Zietsman

August 25, 2011
Blacksburg, Virginia

Keywords: Building Energy Efficiency, Boundary Control, Boussinesq, COMSOL®, Finite Elements

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Computational Approaches to Improving Room Heating and Cooling for Energy Efficiency in Buildings

Brian K. McBee

(ABSTRACT)

With a nation-wide aim toward reducing operational energy costs in buildings, it is important to understand the dynamics of controlled heating, cooling, and air circulation of an individual room, the "One-Room Model Problem." By understanding how one most efficiently regulates a room's climate, one can use this knowledge to help develop overall best-practice power reduction strategies. A key toward effectively analyzing the "One-Room Model Problem" is to understand the capabilities and limitations of existing commercial tools designed for similar problems. In this thesis we develop methodology to link commercial Computational Fluid Dynamics (CFD) software (COMSOL®) with standard computational mathematics software (MATLAB®), and design controllers that apply inlet airflow and heating or cooling to a room and investigate their effects. First, an appropriate continuum model, the Boussinesq System, is described within the framework of this problem. Next, abstract and weak formulations of the problem are described and tied to a Finite Element Method (FEM) approximation as implemented in the interface between COMSOL® and MATLAB.® A methodology is developed to design Linear Quadratic Regulator (LQR) controllers and associated functional gains in MATLAB® which can be implemented in COMSOL.® These "closed-loop" methods are then tested numerically in COMSOL® and compared against "open-loop" and average state closed-loop controllers.

Dedication

To my Father, a former High School math teacher who made me believe that math was interesting. To my Mother, who instilled in me a love of learning new things and a desire to pursue formal education. To my wife for her constant patience, love, and support; not just with my graduate studies, but through military assignments, deployments, and the general chaos that surrounds a military family. To my children, for making it all worth it. Most importantly, to my Heavenly Father, who has watched over and helped me my entire life.

Acknowledgments

I would like to thank all of the people I have come to know at ICAM for their direct help and support. It feels more like a family than the cutting-edge organization that it really is. This is my third graduate program, and none of the others even come close in support or creating an environment for learning and collaboration. Each member of my committee was extremely accessible, treating me like a part of the ICAM family, and providing me with many critical insights. My fellow students were also lifesavers in offering encouragement and freely providing me with samples of their own abundance of knowledge, from assisting me with software, to helping me finally "get it" with some principle of Mathematics. I would especially like to thank WeiWei Hu for all of her theoretical help with the subject of this dissertation, and lastly, my advisor, Prof. John Burns for his wisdom, instruction, and patient willingness to put up with an old rusty-at-math Air Force officer trying to expand his knowledge.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The United States Air Force (USAF) has become extremely diligent in its stewardship of energy consumption. A recent USAF regulatory document, Air Force Instruction 90-1701, states that "As the Department of Defense's (DoD) largest energy consumer, the Air Force is taking a leadership role in developing energy strategies, visions, and programs to reduce energy consumption across missions [1]."

One of the Air Force's primary consumption goals is to "reduce installation energy intensity by 3% per annum" and by regulation, "new buildings, large capital investments, and renovations in an existing building shall employ the most energy efficient designs, systems, and controls that are life cycle cost-effective [1]."

It is well known that extremely energy-efficient buildings can be built from the ground up. In fact, the U.S. Department of Energy (DOE) maintains a "Zero Energy Buildings Database" that lists eight commercial buildings within the United States alone that produce at least as much energy as they use [22]. Every year, the DoD constructs new buildings and they can certainly use state-of-the-art practices to maximize the efficiency of these building. However, many construction projects are simple renovations of existing buildings. When making these renovations with a limited budget, priorities must be established that include safety, and functionality for mission accomplishment, placing further limits on what can and cannot be done to reduce net energy consumption. Therefore, careful analysis should be conducted that takes all building energy usage into account to determine which modifications will produce the greatest effect at the lowest cost.

For most buildings, there are two main sources of energy that come at a direct regular monetary cost: electrical power and natural-gas/propane. These sources are used for building maintenance, lighting, security, functional equipment operation, waste management, water circulation/heating/cooling, and HVAC (Heating, Ventilation, and Air Conditioning). A significant portion of the energy costs in a typical building is HVAC.

There are many components of a typical HVAC system. As an example, Tashtoush [51] et al. discuss an "air-conditioned room and air-handling unit (AHU), which consists of a

fan, cooling and dehumidifying coil, heating coil, humidifier, filter and ductwork". Certainly, modeling an entire HVAC system for a multi-room structure, capturing all of the physics and dynamics involved, is a complex task. However, knowledge of how to reduce energy consumption should begin with understanding how air is heated, cooled, and circulated at the room level and then designing sub-optimal control schemes that help reduce energy consumption by reducing the temperature gradients within the useable portions of the room. Although this is only a first step, it is an essential component of the overall building control design process.

1.2 Existing Research

An enormous amount of work has been done finding ways to reduce building energy consumption. However less has been done in the arena of applying computational fluid dynamics (CFD) specifically to the problem of control. Most of the relevant CFD work surveyed deals with numerical experimentation, rather than using mathematical tools to discover "optimal" parameters. For example, Lu et al. [38] used CFD to simulate airflow/temperature in a room to track pollutant dynamics. They found that the CFD results correlated reasonably well with measured experiments. Similarly, Yang et al. [56] used the finite volume CFD software package FLUENT® to predict airflow and ventilation effects in efforts to maintain a clean hospital ward. Their approach was to design several ventilation schemes and choose the scheme of least cost. Sinha et al. [45] also used CFD (again finite volume) to compare discrete vent configuration cases, but also ran simulations for different fixed Reynolds Re and Grashof Gr Number values. Freire et al. [18] studied the problem of optimizing thermal comfort and energy savings using model-based predictive control where their controlled input was to apply power to the HVAC device. Tashtoush et al. [51] analyzed an HVAC system and used Proportional Integral Derivative (PID) control on components of the AHU. One of their assumptions was that air in a given zone is completely mixed, not taking into account the full physical dynamics of temperature distribution due to convection and diffusion. Finally, van Schijndel [53] used COMSOL® linked with SIMULINK® to model various facets of building and room air climate, however these studies included only limited use of SIMULINK® controllers.

Of course, there is considerable research on understanding the dynamics and control of fluid flow which is typically modeled by the Navier-Stokes Equations. There are numerous good texts on these subjects (see for example [16], [19], [20], [25], [30], [36], [52]) . More specific to the physical model of the problem at hand, there are a number of articles on boundary control of the Boussinesq system. Although these articles do not specifically reference applications for heating and cooling a room, they provide some basic background for the computational work in this thesis.

Burns et al. [14] implemented feedback boundary temperature control on a fluid contained in a circular pipe standing in a vertical plane. In this case, because of the geometry involved, they were able to make some elegant simplifications of the system. They formu-

lated a linear quadratic regulator problem (LQR), established the existence of an optimal controller, and developed numerical methods to solve the LQR problem. Ito and Ravindran [24] analyzed the usage of boundary temperature control to minimize flow vorticity in systems where fluid flows are generated from application of heat. They provided existence results and necessary conditions of optimality. They also used a finite element approximation and constructed a discrete optimality system. Hinze and Matthes [21] discussed a control problem for the Boussinesq equations and applied their results to crystal growth. In this case, they used both boundary and distributed controls of temperature and velocity force on the interior of the domain. They gave results for Model Predictive Control and Instantaneous Control. Lee has coauthored a number of papers concerning optimal control of the Boussinesq equations. For instance, Lee and Choi [33] specifically addressed linear feedback control of the Boussinesq equations. In this case, they set up controls for the "body forces of the velocity and temperature field". In [34], Lee and Kim discussed optimal boundary temperature control, with mixed boundary conditions. In this case, one aim was to minimize costs in temperature gradients, but only on the boundary (i.e. avoiding hot spots). Within the domain, velocity and vorticity gradients were minimized. Their approach was to derive the optimality system and then solve the adjoint equations using state equations from the previous time step.

In almost all the work related to this problem, computational work has been accomplished either with author-designed research code, tailored for the specific problem at hand, or simulations only with commercial physical modeling software. However, little has been done to integrate commercial software for controller design for the problem. In this thesis we investigate the applicability of combining CFD simulation software (COMSOL®) with control design software (MATLAB®) to produce a practical design. In addition, we conduct a numerical study of the high-fidelity closed-loop system. A theoretical framework for the well-posedness and existence of the control laws may be found in the recent papers [23], and will be used as a starting point for the numerical work in this thesis. Thus, this thesis is best described as research in computational science. The main objective is to use mathematical and numerical analysis to provide insight into problems of design and control, rather than generate a new theoretical framework.

1.3 A One Room Model Problem

We focus on the specific problem of designing feedback controllers by using a CFD code to derive a linear model. This linear model is used for various control designs. In theory, this approach could lead to the development of holistic control design and in particular, this approach would provide a general framework to address different problems.

In this vein, the first aim of this thesis is to specify an appropriate physical/mathematical continuum model for the dynamics of heating and cooling a simplified room-like domain. Figure 1.1 is a simple example of a 3D version of such a domain. Air enters a room exclusively via velocity and temperature-scale-driven inlets on the boundary and exits freely from outlet

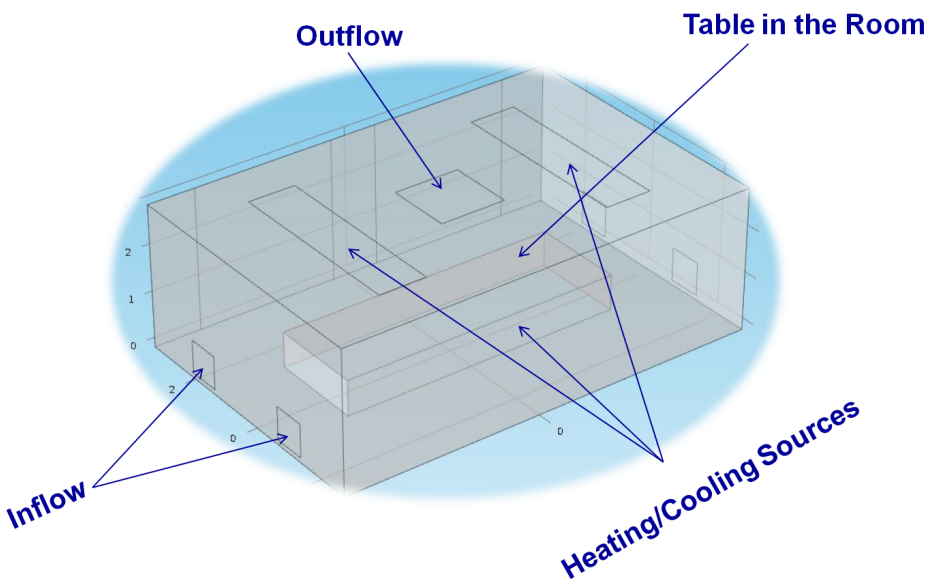

Figure 1.1: Example Room

vents also on the boundary. Other portions of the boundary may consist of heating or cooling sources from disturbances such as exterior windows, lights, electrical equipment, or people. The boundary is considered to be impermeable and fully-insulated. Heat is distributed throughout the room over time via simultaneous convection and diffusion.

With a physics-based model in hand, the overall goal of this thesis is to specify a discrete approximation of this model using a readily-available commercial software package, COMSOL®. Then using COMSOL's® built-in ability to link with MATLAB®, we design a state-space feedback inflow/temperature controller that helps to reduce the average temperature gradients in the room over time and space. Though the scope of this thesis does not address all the physics involved with air circulation and heating in a room, one goal of this effort is to provide a "proof of concept" that illustrates how one can link the flexibility and countless design options of commercial physical/CFD modeling software with tailored, separately-designed controllers using traditional computational mathematics software, such as MATLAB®. The goal is to create a useful tool in analyzing how to apply controls and minor modifications to existing ventilation systems that help reduce energy costs.

1.4 Thesis Outline

In Chapter 2, we derive a basic mathematical and physical model to formulate a meaningful control problem. This model is represented by the Boussinesq sytem of coupled nonlinear parabolic partial differential equation (PDE) with Dirichlet boundary conditions on the controlled portion of the boundary, Neumann-like conditions on the outflow portion of the

boundary, and Dirichlet-velocity or Neumann-temperature conditions on the rest of the boundary. Here we introduce a cost function associated with minimizing squared differences in temperature and velocity over time. For completeness, we briefly provide the abstract formulation of this problem.

In Chapter 3, we discuss the discrete approximation of the model used by COMSOL®, namely the Finite Element method. We review the weak formulation of the PDE system and use the finite element method to derive an approximation of the model from Chapter 2 represented as a system of ordinary differential equations (ODE). Then, to illustrate the FEM, we turn to a much simpler linear 2D model, the diffusion equation with Dirichlet boundary control. LQR control is then introduced for the associated discretized system. We finally discuss how one would similarly implement LQR control of the discretized physical model from Chapter 2 by using the linearized version of this model.

In Chapter 4, we conduct COMSOL®-linked-with-MATLAB® simulations using a coarsely discretized physical model. We describe an implementation of the scheme from Chapter 3 between COMSOL® and MATLAB®, where a classical state system is imported from COMSOL® into MATLAB® along with all of the information necessary to use this system in state feedback design. In particular, we compute the "functional gains," that define the feedback operators. We use these techniques to explore steering a uniform room temperature to a target temperature as well as steering a domain with a boundary disturbance back to a uniform room temperature. Since these experiments are proof-of-concept we use model parameters that keep the results computationally simple, but we demonstrate how application of tailored closed-loop control with proper tuning of cost parameters can provide encouraging results.

In Chapter 5 we conclude by providing an overview of our results and discuss potential future work with the eventual goal of designing a practical controller to assist in engineering design.

Chapter 2

Physical Model and Problem Statement

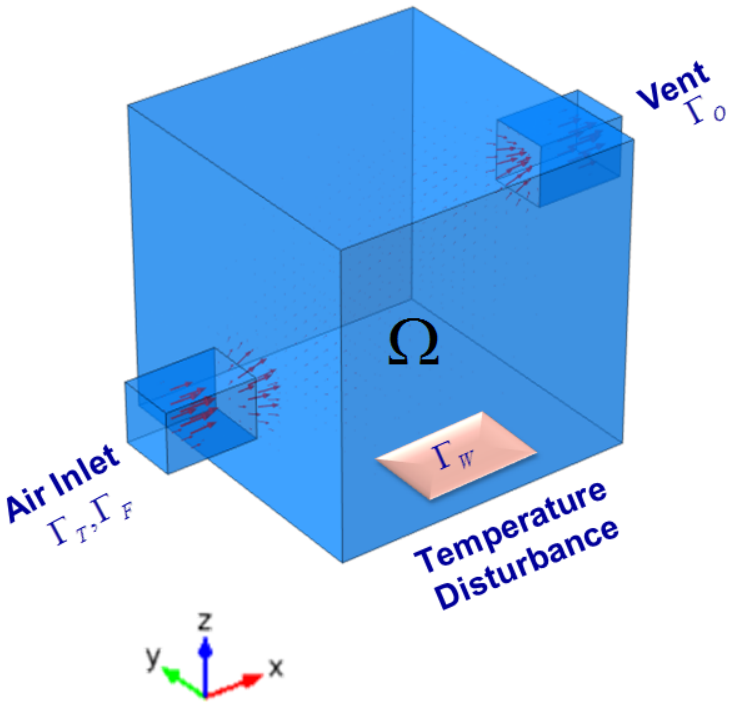

Figure 2.1: Example Domain in 3D

Consider the simple domain Ω defined by the room in Figure 2.1. We assume the gravitational force $\mathbf{g}$ is oriented in the negative vertical $\mathbf{z}$ direction. The boundary of the room is defined by the walls, floors, and ceiling and represents a piecewise smooth (PWS) boundary Γ divided into smooth, disjoint subsets. Portions of the boundary will be used as temperature-regulated heat or cooling sources Γ_T, controlled forced air inlets (may or may not coincide

with heat sources) Γ_F, outflow vents Γ_O, and possibly exterior windows or other sources of temperature disturbance with time-varying temperature fluctuations Γ_W.

Although the model for this system is well-known, we review the basic steps of its derivation found in Doering and Gibbon [16] with some additional insight from Rubio [44]. This derivation will help in the process of setting up the boundary control systems we will analyze.

2.1 A Basic Continuum Model and The Boussinesq Equations

To derive the equations it is important to review the *convective derivative*. For a given function of space and time $\mathbf{f}(\mathbf{x},t) \in \mathbb{R}^d, d = 2$ or 3[1] (for the derivations in this chapter, we use $d = 3$), the convective derivative is the rate of change in f for a fluid element that's position is dependent on time, $\mathbf{x}(t)$. In particular, if $\mathbf{x}(t) = (x(t), y(t), z(t))^T$ represents the position of a fluid particle at time t, then

$$\begin{aligned}\frac{d\mathbf{f}(\mathbf{x}(t),t)}{dt} &= \begin{pmatrix} \frac{\partial}{\partial t}f_1 + \frac{\partial}{\partial x}f_1\dot{x} + \frac{\partial}{\partial y}f_1\dot{y} + \frac{\partial}{\partial z}f_1\dot{z} \\ \frac{\partial}{\partial t}f_2 + \frac{\partial}{\partial x}f_2\dot{x} + \frac{\partial}{\partial y}f_2\dot{y} + \frac{\partial}{\partial z}f_2\dot{z} \\ \frac{\partial}{\partial t}f_3 + \frac{\partial}{\partial x}f_3\dot{x} + \frac{\partial}{\partial y}f_3\dot{y} + \frac{\partial}{\partial z}f_3\dot{z} \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial}{\partial t}f_1 \\ \frac{\partial}{\partial t}f_2 \\ \frac{\partial}{\partial t}f_3 \end{pmatrix} + \begin{pmatrix} v_1\frac{\partial}{\partial x}f_1 \\ v_1\frac{\partial}{\partial x}f_2 \\ v_1\frac{\partial}{\partial x}f_3 \end{pmatrix} + \begin{pmatrix} v_2\frac{\partial}{\partial y}f_1 \\ v_2\frac{\partial}{\partial y}f_2 \\ v_2\frac{\partial}{\partial y}f_3 \end{pmatrix} + \begin{pmatrix} v_3\frac{\partial}{\partial z}f_1 \\ v_3\frac{\partial}{\partial z}f_2 \\ v_3\frac{\partial}{\partial z}f_3 \end{pmatrix} \\ &= \frac{\partial}{\partial t}\mathbf{f} + \begin{pmatrix} (\mathbf{v}\cdot\nabla)f_1 \\ (\mathbf{v}\cdot\nabla)f_2 \\ (\mathbf{v}\cdot\nabla)f_3 \end{pmatrix} \triangleq \frac{\partial}{\partial t}\mathbf{f} + (\mathbf{v}\cdot\nabla)\mathbf{f}. \end{aligned} \tag{2.1.1}$$

where $\mathbf{v} := (v_1, v_2, v_3)^T$, the velocity. Thus, the term $\frac{d\mathbf{f}(\mathbf{x}(t),t)}{dt} \triangleq \frac{\partial}{\partial t}\mathbf{f} + (\mathbf{v}\cdot\nabla)\mathbf{f}$ is called the convective derivative and will be used throughout the thesis.

2.1.1 The Continuity Equation

Consider an infinitesimal fluid element centered at the point $\mathbf{x}(t) = (x(t), y(t), z(t))^T$ at time t. Its volume $\delta V = \delta x \delta y \delta z$ will change with time as follows,

$$\frac{d\delta V(t)}{dt} = \frac{d\delta x(t)}{dt}\delta y(t)\,\delta z(t) + \frac{d\delta y(t)}{dt}\delta x(t)\,\delta z(t) + \frac{d\delta z(t)}{dt}\delta x(t)\,\delta y(t). \tag{2.1.2}$$

[1] We take the convention that **bold** functions or variables represent vectors or matrices.

For a given component direction of the fluid element, the rate of change in element length for that direction (e.g. $\frac{d}{dt}\delta x$) is the difference in component velocities halfway from the element center. Thus

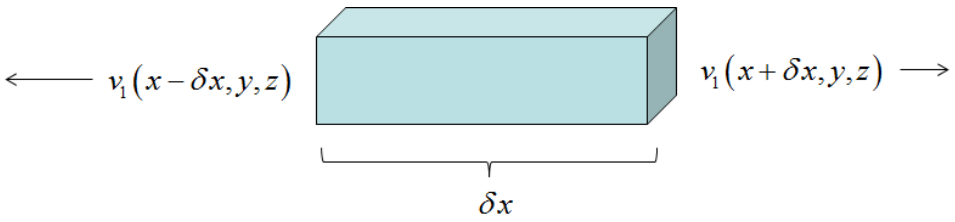

Figure 2.2: Rate of Element Length Change in the e_1 Direction

$$\underbrace{\frac{d\delta x}{dt} = v_1\left(x+\frac{\delta x}{2},y,z\right) - v_1\left(x-\frac{\delta x}{2},y,z\right) = \frac{\partial v_1}{\partial x}\delta x}_{\text{similar for } \delta y,\delta z}. \tag{2.1.3}$$

Equations (2.1.2) and (2.1.3) imply that

$$\frac{d}{dt}\delta V = (\nabla\cdot\mathbf{v})\,\delta V. \tag{2.1.4}$$

Let δm represent the mass of the fluid element and let $\rho \triangleq {}^{\delta m}\!/_{\delta V}$, represent density. Now, conservation of mass, i.e. ${}^{d\delta m}\!/_{dt} = 0$, (2.1.1) and (2.1.4) yield the well-known *continuity equation.*

$$\begin{aligned}\frac{d\rho}{dt} &= \frac{d}{dt}\frac{\delta m}{\delta V} = \frac{\delta V\frac{d}{dt}\delta m - \delta m\frac{d}{dt}\delta V}{(\delta V)^2}\\ &= -\frac{\delta m}{(\delta V)^2}\frac{d}{dt}\delta V = -\rho\frac{1}{\delta V}\frac{d}{dt}\delta V =\\ &= -\rho\,(\nabla\cdot\mathbf{v})\end{aligned} \tag{2.1.5}$$

which implies that

$$\begin{aligned}\frac{d\rho}{dt} + \rho\,(\nabla\cdot\mathbf{v}) &= \frac{\partial\rho}{\partial t} + (\mathbf{v}\cdot\nabla)\,\rho + \rho\,(\nabla\cdot\mathbf{v})\\ &= \frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0.\end{aligned} \tag{2.1.6}$$

2.1.2 Newtons Second Law, Momentum, and the Velocity Field

According to Newton's second law of motion, the rate of change of momentum (i.e. *mass* $\times$ *velocity*) on a fluid element in motion is equal to the net force applied on the fluid element.

Therefore,

$$\frac{d}{dt}\left(\delta m\,\mathbf{v}\right) = \underbrace{\mathbf{v}\frac{d}{dt}\delta m}_{=0} + \delta m\frac{d}{dt}\mathbf{v} = \delta m\left[\frac{\partial \mathbf{v}}{\partial t} + \left(\mathbf{v}\cdot\nabla\right)\mathbf{v}\right], \tag{2.1.7}$$

which yields

$$\frac{d\mathbf{v}}{dt} = \left[\frac{\partial \mathbf{v}}{\partial t} + \left(\mathbf{v}\cdot\nabla\right)\mathbf{v}\right] = \frac{1}{\delta m}\delta\mathbf{F}. \tag{2.1.8}$$

In this model, $\delta\mathbf{F}$ will have several components: net pressure ($\delta\mathbf{F}_p$), net shear stress ($\delta\mathbf{F}_s$), and buoyancy/gravitational force ($\delta\mathbf{F}_b$) due to changes in air temperature (T) causing changes in density.

2.1.3 Pressure

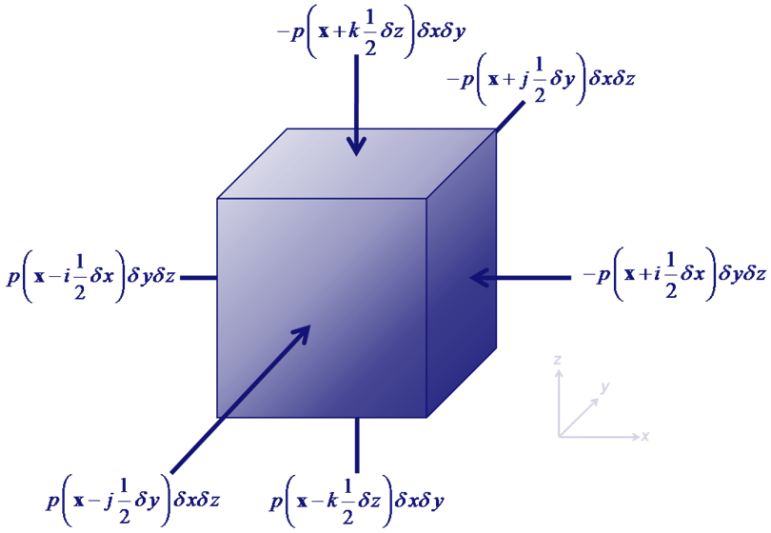

Figure 2.3: Pressure Forces on Element

The first fluid force we will consider is due to pressure. Doering and Gibbon [16] define this force as "the magnitude of the force per unit area, or normal stress, imposed on elements of the fluid from neighboring elements." Given an infinitesimal fluid element centered at $\mathbf{x}$ and oriented with the standard Cartesian basis $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$, the net force due to pressure, $\delta\mathbf{F}_p = (F_{p1}, F_{p2}, F_{p2})^T$, at the point $\mathbf{x}$ can be thought of as the sum of forces normal to each face of the element (Figure 2.3). Thus, we have

$$\delta \mathbf{F}_p = \begin{pmatrix} \delta F_{p1} \\ \delta F_{p2} \\ \delta F_{p2} \end{pmatrix} = \begin{pmatrix} \left[p\left(\mathbf{x} - i\frac{1}{2}\delta x\right) - p\left(\mathbf{x} + i\frac{1}{2}\delta x\right) \right] \delta y \delta z \\ \left[p\left(\mathbf{x} - j\frac{1}{2}\delta y\right) - p\left(\mathbf{x} + j\frac{1}{2}\delta y\right) \right] \delta x \delta z \\ \left[p\left(\mathbf{x} - k\frac{1}{2}\delta z\right) - p\left(\mathbf{x} + k\frac{1}{2}\delta z\right) \right] \delta x \delta y \end{pmatrix}$$

$$= \begin{pmatrix} \dfrac{p\left(\mathbf{x} - i\frac{1}{2}\delta x\right) - p\left(\mathbf{x} + i\frac{1}{2}\delta x\right)}{\delta x} \\ \dfrac{p\left(\mathbf{x} - j\frac{1}{2}\delta y\right) - p\left(\mathbf{x} + j\frac{1}{2}\delta y\right)}{\delta y} \\ \dfrac{p\left(\mathbf{x} - k\frac{1}{2}\delta z\right) - p\left(\mathbf{x} + k\frac{1}{2}\delta z\right)}{\delta z} \end{pmatrix} \delta V$$

$$\cong -\delta V \nabla p = -\frac{\delta m}{\rho} \nabla p. \tag{2.1.9}$$

2.1.4 The Force from Shear Stress

The next force to consider is due to the shear stress on the fluid element (shear stress can also be applied to a solid). This force is composed of frictional forces associated with differing velocities of neighboring points within the fluid. In a Newtonian fluid such as air in a building (where force applied on the fluid has no effect on viscosity), the shear stress is a direction-invariant (isotropic) linear function of the fluid's rate of strain tensor $\mathbf{R}$ which Doering and Gibbon [16] describe as "that controlling the evolution of the relative positions of points in a fluid element" (See Figure 2.4. For two points in a fluid separated by $\delta \mathbf{x}(t)$), one has

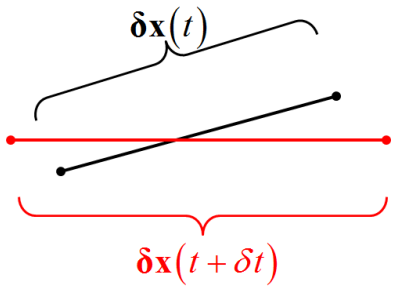

Figure 2.4: Infinitesimal Evolution of Relative Position Between Points

$$\begin{aligned}
\frac{d}{dt}\left\|\delta\mathbf{x}\left(t\right)\right\|_2^2 &= \frac{d}{dt}\left\langle\delta\mathbf{x}\left(t\right),\delta\mathbf{x}\left(t\right)\right\rangle \\
&= \left\langle\delta\mathbf{x}\left(t\right),\frac{d}{dt}\delta\mathbf{x}\left(t\right)\right\rangle + \left\langle\frac{d}{dt}\delta\mathbf{x}\left(t\right),\delta\mathbf{x}\left(t\right)\right\rangle \\
&= \left\langle\delta\mathbf{x}\left(t\right),\left(\mathbf{v}\left(\mathbf{x}\left(t\right)+\delta\mathbf{x}\left(t\right)\right)-\mathbf{v}\left(\mathbf{x}\left(t\right)\right)\right)\right\rangle + \left\langle\left(\mathbf{v}\left(\mathbf{x}\left(t\right)+\delta\mathbf{x}\left(t\right)\right)-\mathbf{v}\left(\mathbf{x}\left(t\right)\right)\right),\delta\mathbf{x}\left(t\right)\right\rangle \\
&\cong \left\langle\delta\mathbf{x}\left(t\right),\text{Jacobian}\left[\mathbf{v}\left(\delta\mathbf{x}\left(t\right)\right)\right]\delta\mathbf{x}\left(t\right)\right\rangle + \left\langle\text{Jacobian}\left[\mathbf{v}\left(\delta\mathbf{x}\left(t\right)\right)\right]\delta\mathbf{x}\left(t\right),\delta\mathbf{x}\left(t\right)\right\rangle \\
&= \delta\mathbf{x}(t)^T\left(\text{Jacobian}\left[\mathbf{v}\left(\delta\mathbf{x}\left(t\right)\right)\right]+\left(\text{Jacobian}\left[\mathbf{v}\left(\delta\mathbf{x}\left(t\right)\right)\right]\right)^T\right)\delta\mathbf{x}\left(t\right) \\
&= (\delta\mathbf{x})^T\mathbf{R}\delta\mathbf{x}
\end{aligned}$$

where

$$R_{ij} = R_{ji} = \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}. \tag{2.1.10}$$

Any isotropic linear function of $\mathbf{R}$ (including the stress tensor $\mathbf{T}$) is representable as

$$\mathbf{T} = \alpha\mathbf{R} + \beta\text{Tr}\left(\mathbf{R}\right)\mathbf{I}. \tag{2.1.11}$$

where α and β are scalars that are determined by material properties which will be addressed later.

The $ij'th$ component of a stress tensor reflects the force per unit area in the $j'th$ directions acting on a face which is normal to the $i'th$ direction.[2] With reference to the Cartesian coordinate system, keeping in mind Newton's third law and adopting the convention that force directions are oriented on the rear, left, and bottom of each infinitesimal fluid element, the net infinitesimal force acting on an element centered at $\mathbf{x}\left(t\right)$, is $\delta\mathbf{F}_s = \left(F_{s1}, F_{s2}, F_{s3}\right)^T$ where

$$\begin{aligned}
F_{sj} &= \begin{cases} \left[T_{1j}\left(x+\delta x/2,y,z\right)-T_{1j}\left(x-\delta x/2,y,z\right)\right]\delta y\delta z \\ +\left[T_{2j}\left(x,y+\delta y/2,z\right)-T_{2j}\left(x,y-\delta y/2,z\right)\right]\delta x\delta z \\ +\left[T_{3j}\left(x,y,z+\delta z/2\right)-T_{3j}\left(x,y,z-\delta z/2\right)\right]\delta x\delta y \end{cases} \\
&\cong \frac{\partial}{\partial x}T_{1j}\left(\mathbf{x}\right)\delta x\delta y\delta z + \frac{\partial}{\partial y}T_{2j}\left(\mathbf{x}\right)\delta y\delta x\delta z + \frac{\partial}{\partial z}T_{3j}\left(\mathbf{x}\right)\delta z\delta x\delta y \\
&= \nabla\cdot\begin{pmatrix} T_{1j} \\ T_{2j} \\ T_{3j} \end{pmatrix}\delta V \text{ is } \nabla\cdot\mathbf{T}\delta V.
\end{aligned} \tag{2.1.12}$$

[2]Here i or j refer to the standard basis $\mathbf{e}_i$ or $\mathbf{e}_j$ directions respectively.

Therefore,

$$\begin{aligned}\delta\mathbf{F}_s &= \delta V \nabla\cdot\mathbf{T} = \delta V \nabla\cdot\left(\alpha\mathbf{R}+\beta Tr\left(\mathbf{R}\right)\mathbf{I}\right)\\ &= \delta V\left[\alpha\nabla\cdot\left(\nabla\mathbf{v}+\left(\nabla\mathbf{v}\right)^T\right)+2\beta\nabla\cdot\left(\left(\nabla\cdot\mathbf{v}\right)\mathbf{I}\right)\right]\\ &= \delta V\left[\alpha\left(\Delta\mathbf{v}+\nabla\cdot\left(\nabla\mathbf{v}\right)^T\right)+2\beta\nabla\cdot\left(\left(\nabla\cdot\mathbf{v}\right)\mathbf{I}\right)\right]\\ &= \delta V\left[\alpha\Delta\mathbf{v}+\left(\alpha+2\beta\right)\left[\nabla\cdot\left(\left(\nabla\cdot\mathbf{v}\right)\mathbf{I}\right)\right]\right]\\ &= \delta V\left[\alpha\Delta\mathbf{v}+\left(\alpha+2\beta\right)\left[\nabla\left(\nabla\cdot\mathbf{v}\right)\right]\right]. \end{aligned} \tag{2.1.13}$$

Equations (2.1.9) and (2.1.13), allow for a more specific representation of (2.1.8). In particular,

$$\delta\mathbf{F} = \delta\mathbf{F}_b - \frac{\delta m}{\rho}\nabla p + \delta V\left[\alpha\Delta\mathbf{v}+\left(\alpha+2\beta\right)\left[\nabla\left(\nabla\cdot\mathbf{v}\right)\right]\right] \tag{2.1.14}$$

implies

$$\frac{1}{\delta m}\delta\mathbf{F} = -\frac{1}{\rho}\nabla p + \frac{\alpha}{\rho}\Delta\mathbf{v} + \frac{\alpha+2\beta}{\rho}\left[\nabla\left(\nabla\cdot\mathbf{v}\right)\right] + \frac{\delta\mathbf{F}_b}{\delta m}, \tag{2.1.15}$$

which yields

$$\frac{\partial\mathbf{v}}{dt} + \left(\mathbf{v}\cdot\nabla\right)\mathbf{v} = -\frac{1}{\rho}\nabla p + \mathbf{v}\Delta\mathbf{v} + \frac{\mathbf{F}_b}{\rho} + \frac{\alpha+2\beta}{\rho}\left[\nabla\left(\nabla\cdot\mathbf{v}\right)\right]. \tag{2.1.16}$$

Consequently, one has

$$\rho\frac{\partial\mathbf{v}}{dt} + \rho\left(\mathbf{v}\cdot\nabla\right)\mathbf{v} = \nabla\cdot\left(-pI+\mu\left(\nabla\mathbf{v}+\left(\nabla\mathbf{v}\right)^T\right)\right) + \mathbf{F}_b + 2\beta\left[\nabla\left(\nabla\cdot\mathbf{v}\right)\right]. \tag{2.1.17}$$

Here the parameter ν is known as the "kinematic viscosity", $\mu = \rho\nu$ is the "dynamic viscosity" and $\mathbf{F}_b$ is the net force per unit volume of buoyancy/gravity.

Equation (2.1.17) combined with the continuity equation (2.1.6) yields the well-known *Navier-Stokes equations*,

$$\frac{\partial\mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v} = -\frac{1}{\rho}\nabla p + \nu\Delta\mathbf{v} + \frac{\alpha+2\beta}{\rho}\left[\nabla\left(\nabla\cdot\mathbf{v}\right)\right] + \frac{\mathbf{F}_b}{\rho} \tag{2.1.18}$$

$$\frac{\partial\rho}{\partial t} + \nabla\cdot\left(\rho\mathbf{v}\right) = 0. \tag{2.1.19}$$

2.1.5 Vertical Force: Gravity Versus Buoyancy

Archimedes principle states that the net upward (away from gravity) force on an object is equal to the local gravitational acceleration constant multiplied by the difference between the mass of the object and the mass of the fluid displaced by the object. In the case of a thermal fluid element, such as air, density (unit mass) differences can be explained in terms of temperature differences. Therefore, for the purposes of the one-room model problem, $\mathbf{F}_b$

is represented as being purely dependent on the temperature state T. In the 3D case, this leads to the expression

$$\mathbf{F}_b = -\mathbf{e}_3 \frac{\delta m}{\delta V} g = -\mathbf{e}_3 \frac{g}{\delta V} \delta m = \mathbf{e}_3 g \rho_0 \frac{\partial V}{\partial T} (T - T_0) = g \rho_0 \alpha_T (T - T_0) \mathbf{e}_3 \tag{2.1.20}$$

Here g is the magnitude of gravitational acceleration, $\alpha_T \triangleq \frac{\partial V}{\partial T}$ is the thermal expansion coefficient, ρ_0 is a reference density (e.g. average density of air at room temperature in a local geographic region), T_0 is a reference temperature (e.g. room temperature), and $\mathbf{e}_3$ is the standard basis vector, in 3D, corresponding to the "up" direction. Equation (2.1.17) now becomes

$$\frac{\partial \mathbf{v}}{dt} + (\mathbf{v} \cdot \nabla) \mathbf{v} = -\frac{1}{\rho_0} \nabla p + \nu \Delta \mathbf{v} + \frac{\alpha + 2\beta}{\rho_0} [\nabla (\nabla \cdot \mathbf{v})] + g \alpha_T (T - T_0) \mathbf{e}_3. \tag{2.1.21}$$

2.1.6 The Heat Equation

From equation (2.1.21), we see that the velocity field in a thermal fluid is dependent on the temperature distribution throughout the domain. More importantly, knowledge of the temperature distribution is critical to the one-room problem where temperature is the primary state one wishes to control. Therefore, we briefly review the heat equation using lecture notes from Sun [50].

Fourier's law of heat conduction states that in "an isotropic medium, flows in the direction in which temperature decreases most rapidly and the amount of heat flowing in this direction is proportional to the rate of change of temperature in that direction." With this in mind, we consider our domain Ω. For our particular problem, we assume that heat can only be added or removed across a portion of the boundary Γ. We define $T : \Omega \times [0, \infty) \to \mathbb{R}$ as temperature and the heat flux as $\mathbf{q}(\mathbf{x}, t) \triangleq -k \nabla T$, where k is the thermal conductivity of the medium in Ω. If we define $h(\mathbf{x}, t)$ as the total heat generated per unit time per unit volume, then the total heat per unit time is

$$\int_\Omega h(\mathbf{x}, t) \, d\mathbf{x}, \tag{2.1.22}$$

and the heat flowing into Ω through Γ is

$$\int_\Gamma \left\langle \mathbf{q}(\mathbf{x}, t), \underset{\text{unit normal}}{-\mathbf{n}} \right\rangle d\mathbf{s} = \int_\Gamma \langle k \nabla T, \mathbf{n} \rangle \, d\mathbf{s} \underset{\text{by div thm}}{=} \int_\Omega \nabla \cdot k \nabla T d\mathbf{x}. \tag{2.1.23}$$

This heat flow balances the induced rate of change in temperature, $\frac{\partial T(\mathbf{x}, t)}{\partial t}$ as follows:

$$\int_\Omega \frac{\partial T(\mathbf{x}, t)}{\partial t} C_p \rho d\mathbf{x} = \int_\Omega \nabla \cdot k \nabla T d\mathbf{x}, \tag{2.1.24}$$

where C_p is the specific heat of the medium. Because Ω could be any volume, including a domain of a uniform fluid in motion, it follows that for a dynamic fluid,

$$\frac{d}{dt}T\left(\mathbf{x}\left(t\right),t\right)=\frac{\partial}{\partial t}T\left(\mathbf{x}\left(t\right),t\right)+\mathbf{v}\left(\mathbf{x}\left(t\right)\right)\cdot\nabla T\left(\mathbf{x}\left(t\right),t\right)=\kappa\Delta T, \tag{2.1.25}$$

where $\kappa=\frac{k}{C_p\rho}$ is called the "thermal diffusivity" and C_p is called the "heat capacity."

2.1.7 Boussinesq Assumptions and Basic Model

The machinery is now in place to introduce the assumptions that constitute the Boussinesq approximations. These assumptions are intuitive for modeling the evolution of temperature distributions in a typical pressure-neutral room. First it is assumed that the fluid is incompressible, $\nabla\cdot\mathbf{v}=0$, and constant density ρ_0 except for the density differences implied in the buoyancy force term, $\mathbf{F}_b$. Secondly, the heat generated from viscous dissipation (frictional heat) is neglected. Using (2.1.20), the Navier-Stokes equations ((2.1.18)) are coupled to the heat equation. This yields the so-called Boussinesq approximation of a thermal fluid and is modeled by the system

$$\nabla\cdot\mathbf{v}=0, \tag{2.1.26}$$

$$\text{implies }\frac{\partial\mathbf{v}}{\partial t}+\mathbf{v}\cdot\nabla\mathbf{v}=-\frac{1}{\rho_0}\nabla p+\nu\Delta\mathbf{v}+g\alpha_T\left(T-T_0\right)\mathbf{e}_3 \tag{2.1.27}$$

$$\frac{\partial T}{\partial t}+\mathbf{v}\cdot\nabla T=\kappa\Delta T. \tag{2.1.28}$$

Here we note that

$$\nabla\cdot\left(pI\right)=\begin{pmatrix}\frac{\partial}{\partial x}p\\ \frac{\partial}{\partial y}p\\ \frac{\partial}{\partial z}p\end{pmatrix}=\nabla p, \tag{2.1.29}$$

and

$$\nabla\cdot\left(\nabla\mathbf{v}\right)^T=\begin{pmatrix}\frac{\partial}{\partial x}\frac{\partial v_1}{\partial x}+\frac{\partial}{\partial y}\frac{\partial v_2}{\partial x}+\frac{\partial}{\partial z}\frac{\partial v_3}{\partial x}\\ \frac{\partial}{\partial x}\frac{\partial v_1}{\partial y}+\frac{\partial}{\partial y}\frac{\partial v_2}{\partial y}+\frac{\partial}{\partial z}\frac{\partial v_3}{\partial y}\\ \frac{\partial}{\partial x}\frac{\partial v_1}{\partial z}+\frac{\partial}{\partial y}\frac{\partial v_2}{\partial z}+\frac{\partial}{\partial z}\frac{\partial v_3}{\partial z}\end{pmatrix}=\begin{pmatrix}\frac{\partial}{\partial x}\left(\nabla\cdot\mathbf{v}\right)\\ \frac{\partial}{\partial y}\left(\nabla\cdot\mathbf{v}\right)\\ \frac{\partial}{\partial z}\left(\nabla\cdot\mathbf{v}\right)\end{pmatrix}=0. \tag{2.1.30}$$

We close this subsection by restating the Boussinesq system in concise form. We also provide a table of likely coefficient values as they would apply to the "One-Room Model Problem." Thus, we focus on the system

$$\Sigma_B\begin{cases}\frac{\partial\mathbf{v}}{dt}+\left(\mathbf{v}\cdot\nabla\right)\mathbf{v}=\frac{1}{\rho_0}\nabla\cdot\left(-pI+\mu\left(\nabla\mathbf{v}+\left(\nabla\mathbf{v}\right)^T\right)\right)+g\alpha_T\left(T-T_0\right)\mathbf{e}_3,\\ \frac{\partial T}{\partial t}+\left(\mathbf{v}\cdot\nabla\right)T=\kappa\Delta T,\\ \operatorname{div}\mathbf{v}=\nabla\cdot\mathbf{v}=0,\end{cases} \tag{2.1.31}$$

where typical parameters are given in Table 2.1.

Name	Description	Value
ρ_0	Density	$1.18\frac{\text{kg}}{\text{m}^3}$
g	|Gravitational Acceleration|	$9.81\frac{m}{s^2}$
μ	Viscosity	$1.81 \times 10^{-5}\ \frac{\text{kg}}{\text{ms}}$
α_T	Coef. of Thermal Expansion	$3.38 \times 10^{-3}\ \frac{1}{\text{K}}$
T_0	Room Temperature	293.15°K
κ	Thermal Diffusivity	$2.08 \times 10^{-5}\frac{\text{m}^2}{\text{s}}$
Re	Reynolds Number	2.46×10^5
Gr	Grashof Number	7.61×10^{10}
Pr	Prandtl Number	880.6
C_p	Specific Heat	$1012\ \frac{\text{J}}{\text{kg K}}$
k	Thermal Conductivity	$0.0257\frac{\text{W}}{\text{m K}}$

Table 2.1: Boussinesq Coefficient Values

2.2 Boundary Conditions and Control

We turn now to the boundary conditions for the Boussinesq equations. For the purposes of this thesis, control will be introduced through these boundary conditions for both velocity and temperature. Each type will be treated separately, then they will be recombined with the introduction of control on portions of the boundary.

2.2.1 Velocity Boundary Conditions

Assume that air flow velocity on the boundary $\mathbf{v}_\Gamma$ can be characterized by mixed Dirichlet (essential) and Neumann-type (natural) conditions. The piecewise-connected portion of the boundary with essential conditions ($\Gamma_{E\mathbf{v}}$) consists of walls and velocity controlled inlets, disjoint from which is the piecewise-connected portion of the boundary with natural conditions ($\Gamma_{N\mathbf{v}}$). The boundary Γ is connected and is the union of these two sets.

$$\Gamma = \Gamma_{E\mathbf{v}} + \Gamma_{N\mathbf{v}}; \tag{2.2.1}$$

This specification of velocity is reasonable for the one-room problem. For instance, the velocity boundary of a room is mostly surrounded by walls. As a fluid particle grows in

proximity to a rigid wall, the greater will be the influence of shear forces from the wall so that in the limit, the velocity will theoretically be zero. This is observed in river flow near a bank. Another example, given by Layton [30], discusses how dust or pollen can fall from a tree onto a car. Though the pollen could easily be brushed off, it remains on the sides of the car when traveling at high speeds, thus indicating that despite the high relative wind between the car and the air at a distance, the air is at least very close to zero in proximity of the pollen. An interesting artifact of this phenomena is that slow fluid near the boundary has lessening deceleration effects on the fluid as the fluid moves further away from the wall, until there is almost no effect at all. A simple 2D COMSOL® example illustrates this principle (See Figure 2.5).

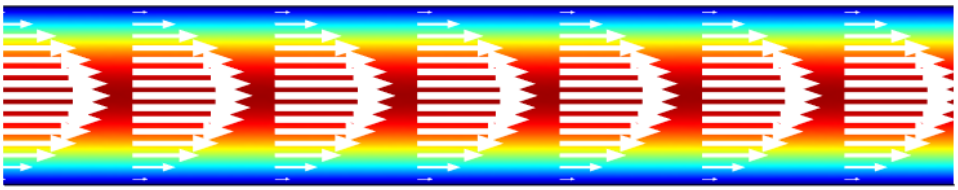

Figure 2.5: Example of No-slip Condition in a Narrow Domain

This is a steady-state model with an unrealistic viscosity, for application to a room problem, but it illustrates the effects of walls on a fluid. It also indicates that a specified parabolic-type function that is zero on the boundary edges would be reasonable, physically, for the inflow boundary condition and that the appropriate way to control airflow would be simply to increase or decrease the amplitude of this parabolic function (possibly in two directions to represent the notion of directing the air flow angle as well; as is manually accomplished with vents in an automobile).

Therefore, we represent $\mathbf{v}_{\Gamma_{E_\mathbf{v}}}$ as a continuous, piecewise-smooth (PWS) function that is zero everywhere except the inlet, which is characterized by an amplitude-controlled parabola or parabolic-type function in the case of 3D and the outlet, with the so-called "do-nothing" or "natural" no-stress outflow condition. Hence we assume

$$\mathbf{v}_{\Gamma_{E_\mathbf{v}}} = \mathbf{h}(\mathbf{x})\, u_\mathbf{v}(t), \tag{2.2.2}$$

$$\left[-p\mathbf{I} + \mu\left(\nabla \mathbf{v}_{\Gamma_{N_\mathbf{v}}} + (\nabla \mathbf{v}_{\Gamma_{N_\mathbf{v}}})^T\right)\right] \cdot \mathbf{n} = 0, \tag{2.2.3}$$

where $h(\mathbf{x})$ is a given "parabolic-like" function to be described later. Elmen et. al [17] remark that this "natural", no-stress outflow condition ensures that the fluid divergence-free assumption remains consistent at the boundary (see (2.2.5)). Therefore, the boundary of any admissible flow field will be assumed to be PWS. Since $\nabla \cdot \mathbf{v}(t, \mathbf{x}) = 0$ on $\mathbf{x} \in \Omega$, it follows that

$$0 = \int_\Omega \nabla \cdot \mathbf{v} d\mathbf{x} \tag{2.2.4}$$

which, by Green's Theorem, implies

$$\int_{\Gamma} [\mathbf{v} \cdot \mathbf{n}] \, dS = 0. \tag{2.2.5}$$

This requires that the air mass and volume flowing into the domain should be equal to the air mass and volume flowing out. We then define an admissible set of flow field boundary functions as

$$\mathcal{J}_{\Gamma} = \left\{ \begin{array}{l} \mathbf{w} : \Gamma \times \ [0, t_f] \to \mathbb{R}^3 \\ \text{s.t. } \mathbf{w} \text{ is } \mathbf{PWS}, (2.2.2) \text{ - } (2.2.5) \text{ hold.} \end{array} \right\}, \tag{2.2.6}$$

and we require that $\mathbf{v}|_{\Gamma} \in \mathcal{J}_{\Gamma}$.

2.2.2 Temperature Boundary Conditions

Again, from the perspective of renovating a room without changing the placement of heat or cooling sources, we begin with the simple case of representing temperature on the boundary as the disjoint union of mixed conditions. Natural (Neumann-type), or flux conditions, are assumed for fully-insulated walls, convection-dominated outlets (Γ_{NT}), and possibly disturbances (Γ_{DT}) from windows, lights, electrical equipment, etc. Dirichlet-type heating or cooling sources are assumed for controlled heating or cooling input (Γ_{ET}) as well as uncontrolled disturbances. In particular, we assume that

$$(1) \quad \Gamma = \Gamma_{NT} + \Gamma_{ET} + \Gamma_{DT}, \tag{2.2.7}$$

$$(2) \quad \mathbf{n} \cdot (k \nabla T_{\Gamma_{NT}}) = 0, \tag{2.2.8}$$

$$(3) \quad T_{\Gamma_{ET}} = h_T(\mathbf{x}) \, u_T(t), \tag{2.2.9}$$

$$(4) \quad T_{\Gamma_{DT}} = h_{DT}(\mathbf{x}) \, d(t) \ \text{ or } \mathbf{n} \cdot (k \nabla T_{\Gamma_{DT}}) = \hat{h}_{DT}(\mathbf{x}) \, \hat{d}(t). \tag{2.2.10}$$

Here $h_T(\mathbf{x})$ represent a specified non-zero heat-source temperature profile on the boundary (typically constant unity) and $u_T(t)$ represents a time-dependent controlled amplification factor of this profile. The boundary condition (2.2.10) represents a (possibly time dependent disturbance along the boundary Γ_{DT}. For engineering design, the disturbance portions of the boundary could be represented by, for example, smooth, interpolated, time-dependent weather data relative the the thermal resistance (or R-value) of the room, which is the "ratio of temperature difference across an insulator and the heat flux" [55]. For the purposes of this thesis, we set the walled portion of the boundary as zero flux, which represents perfect insulation where there is no change in temperature across the boundary. Again, for engineering design, the flux would be set according to the R-value and, timed measurements of typical temperature differences between inside and outside. Setting the flux to zero is also a reasonable representation of heat on the velocity outlet with the assumption that very little heat diffusion will flow from a fluid particle at the moment it crosses the boundary. In

a real building, the actual boundary flux would be determined by heating (radiant, etc.) on the outside walls.

2.3 Abstract Formulations

Though not the focus of this thesis, we finish this chapter by briefly mentioning the abstract formulations of the Boussinesq system. With a bit more detail, we restate (2.1.31).

$$\frac{\partial \mathbf{v}(t,\mathbf{x})}{\partial t} + (\mathbf{v}(t,\mathbf{x})\cdot\nabla)\mathbf{v}(t,\mathbf{x}) = \nu\nabla^2\mathbf{v}(t,\mathbf{x}) + g\alpha_T T(t,\mathbf{x}) - \nabla\frac{1}{\rho_0}p(t,\mathbf{x}) + f(t,\mathbf{x}), \quad (2.3.1)$$

$$\nabla\cdot\mathbf{v}(t,\mathbf{x}) = 0 \;\; \mathbf{x}\in\Omega, \quad (2.3.2)$$

$$\frac{\partial T(t,\mathbf{x})}{\partial t} + \mathbf{v}(t,\mathbf{x})\cdot\nabla T(t,\mathbf{x}) = \kappa\Delta T(t,\mathbf{x}) + h(t,\mathbf{x}) \;\; \mathbf{x}\in\Omega, \quad (2.3.3)$$

Here, $\Omega \subset R^d,\ d = 2,3$, is an open, piecewise smooth, bounded and connected domain with boundary Γ. The variables $(t,\mathbf{x}) \in (0,\infty)\times\mathbb{R}^d$, $\mathbf{v}(t,\mathbf{x})$ is the velocity vector, $p(t,\mathbf{x})$ is the pressure and $T(t,\mathbf{x})$ is the fluid temperature. As mentioned earlier, the control inputs will occur through boundary conditions (Dirichlet) as described above. In particular, there will be a finite number of control inputs m so that the control space is $\mathbb{R}^m$.[3]

Feedback stabilization of Boussinesq equations has been considered in ([3], [33], [35], [44], [54]), where control is assumed over the entire boundary. Recently, Raymond ([39], [41]) has considered control of the Navier-Stokes equations with partial boundary control. We shall considered mixed boundary control conditions of Dirichlet and Neumann or Robin type. A theoretical motivation based on the Navier-Stokes equations can be found in [23] and [40]. The coupled system (2.3.1) - (2.3.3) can be written as an abstract non-linear system of the form

$$\dot{z}(t) = \mathcal{A}z(t) + \mathcal{F}(z(t)) + \mathcal{B}u(t), \quad z(0) = z_0, \quad \in \mathbb{Z} \quad (2.3.4)$$

where $z(t,x) = (T(t,x), \mathbf{v}(t,x))^T$ so that $z(t,\cdot)$ represents the physical state of the thermal fluid at time t. The state space $\mathbb{Z}$ is a Hilbert space and is a subset of

$$H = L_2(\Omega)\times H_{div}(\Omega), \quad (2.3.5)$$

where $H_{div}(\Omega) \subseteq L_2(\Omega)$ is the subspace of divergent free vector fields on Ω. The precise definition of $\mathbb{Z}$ is rather complex and depends on the type of boundary controls to be used. We shall not discuss the details of how $\mathbb{Z}$ is constructed since this is not required for the computational study here. The important point in the CFD implementation is to realize that the

[3]For the purposes of this thesis, $m = 2$ or $m = 3$)

divergence free condition is approximated in COMSOL® and the abstract formulation (2.3.4) serves mostly as a guide to the development of practical computational tools for controller design. However, rigorous convergence proofs can be found in [23] and understanding the precise spaces are essential to establishing the theoretical basis for the computational work presented here. In particular, we observe that the operator $\mathcal{A}$ has the block form

$$\mathcal{A} = \begin{bmatrix} A_T & A_{vT} \\ A_{Tv} & A_v \end{bmatrix} \triangleq \begin{bmatrix} \kappa\Delta & -\nabla T(t,\mathbf{x})\cdot \\ g\alpha_T & \nu\Delta \end{bmatrix}, \tag{2.3.6}$$

and the nonlinear operator has the form

$$\mathcal{F}(z(t)) = \mathcal{F}\left(\begin{bmatrix} T(t,\mathbf{x}) \\ \mathbf{v}(t,\mathbf{x}) \end{bmatrix}\right) = \begin{bmatrix} 0 \\ -(\mathbf{v}(t,\mathbf{x})\cdot\nabla)\mathbf{v}(t,\mathbf{x}) \end{bmatrix} \tag{2.3.7}$$

The control input operator $\mathcal{B}:\mathbb{R}^m \to \mathbb{Z}$ has the form

$$\mathcal{B}u(t) = \mathcal{B}\begin{bmatrix} u_T(t) \\ u_v(t) \end{bmatrix} = \begin{bmatrix} \mathcal{B}_T & 0 \\ 0 & \mathcal{B}_v \end{bmatrix}\begin{bmatrix} u_T(t) \\ u_v(t) \end{bmatrix}, \tag{2.3.8}$$

where $\mathcal{B}_T$ and $\mathcal{B}_v$ are appropriate boundary operators (see [23]). Thus, the insight provided by the state space formulation provides practical information about what one must compute.

For example, we use linear control theory to design a controller of the form

$$u(t) = -\mathcal{K}z(t),\ t \in (0,\infty), \tag{2.3.9}$$

where the feedback operator $\mathcal{K}$ is a bounded linear operator on Ω from $\mathbb{Z}$ to $\mathbb{R}^m$. Since $\mathbb{Z} \subseteq L_2(\Omega)\times H_{div}(\Omega)$ one can apply the Riesz representation theorem to show that there exists kernels

$$k_{i,T}(\mathbf{x})\ ,\ k_{i,u}(\mathbf{x})\ ,\ k_{i,v}(\mathbf{x})$$

such that

$$u_i(t) = -\int_\Omega k_{i,T}(\mathbf{x})\,T(t,\mathbf{x})d\mathbf{x} - \int_\Omega k_{i,u}(\mathbf{x})u(t,\mathbf{x})d\mathbf{x} - \int_\Omega k_{i,v}(\mathbf{x})v(t,\mathbf{x})d\mathbf{x}, \tag{2.3.10}$$

for $i = 1,2,\ldots,m$. The kernels $k_{i,T}(\mathbf{x}) \in L_2(\Omega)$ and $k_{i,u}(\mathbf{x})\ ,\ k_{i,v}(\mathbf{x}) \in H_{div}(\Omega)$ are called *functional gains*.

One computational challenge is to determine how one can combine COMSOL® with MATLAB® to "compute" these functional gains. In addition, once these functional gains have been computed, a second challenge is simulating the closed loop system which now involves a non-local boundary condition.

We shall use linearization about a set point (equilibrium) and design an LQR controller based on the linearization. Thus, the basic approach, as suggested by Hu [23], can be defined in the following steps:

1. Find a steady-state solution z_e, such that

$$\mathcal{A}z_e + \mathcal{F}(z_e) = 0. \tag{2.3.11}$$

   It is clear that in a real room, there may not be a steady-state solution. However, for this study, we make this simplifying assumption.

2. Denote $x(t) = z(t) - z_e$ and obtain the translated nonlinear problem around z_e

$$\dot{x}(t) = \mathcal{A}_e x(t) + \mathcal{F}(x(t)) + \mathcal{B}(t)u(t), \quad x(0) = z_0 - z_e; \tag{2.3.12}$$

3. Linearize about z_e so that in an appropriate state space $\mathcal{X}$

$$\dot{x}(t) = \mathcal{A}_e x(t) + \mathcal{B}(t)u(t); \tag{2.3.13}$$

4. Apply LQR control to the linearized system (2.3.13) by minimizing a quadratic cost function of the form

$$J(u) = \int_0^\infty \left[\langle Qx(t), x(t)\rangle_X + \langle Ru(t), u(t)\rangle_{\mathbb{R}^m}\right] dt, \tag{2.3.14}$$

   to obtain the controller in the feedback form $u(t) = -\mathcal{K}x(t)$.

Based on this strategy, [4], [5], [6] obtain a stabilizing Riccati-based, Dirichlet boundary controller for Navier-Stokes equations, $d = 2, 3$ with tangential control only. Hu [23], Raymond [40] et al, and Badra [3] have extended this to the full Boussinesq system with more realistic and general boundary control terms. The most challenging issue arises from a combination of $d = 3$ with Dirichlet boundary conditions as discussed in [6]. In an approximate setting, we shall implement the above steps with existing CFD and MATLAB® software.

Finally we note that (except in the 2D case) the problem of establishing the well-posedness of the abstract non-linear Boussinesq system (2.3.4) remains an open question (even for the Navier-Stokes equations). However, well-posedness of the linearized system is well understood (see [23],[40]) and easily follows from the corresponding results for the Navier-Stokes equations (see [46], [47]).

Chapter 3

Finite Element Discretization and Boundary Control

The smoothness requirements of the Boussinesq equations, as originally introduced in the previous chapter, can be far too restrictive for the current problem, especially when attempting to solve a finite-dimensional problem that adequately approximates the solutions. Thus, one must introduce weak solutions. The weak formulation also provides the mathematical framework needed to derive Finite Element Methods (FEMs) and to analyze their properties. There are numerous texts that cover weak formulation and its role in FEM (see, for example, [13], [17], [20], [30], [43] and [49]). In this chapter, we summarize the basic ideas and definitions. This will provide some understanding of how COMSOL® sets up and solves the Boussinesq equations. Moreover, the inclusion of boundary control terms as unknown control.

As noted in the introduction, one goal of this study is to illustrate how an existing commercial CFD package, with the proper interface and functionality, can be leveraged to develop control design tools. The presentation below follows the developments in the references (including usage of much of the same symbology) on the subject provided by Gunzburger [20], Elman et al. [17], Reddy and Gartling [42], Braess [13], [32], and Layton [30]. However, we also present slight variations in these formulations so that we are consistent with the COMSOL® implementations as well.

3.1 Function Space Definitions

The first step in providing a weak formulation is to define proper function spaces in which to pose the problem. For the most part, these spaces are rather standard, but are included for completeness.

The Hilbert Space $L^2(\Omega) = L^2(\Omega;\mathbb{R})$ with its associated inner product $(\ ,\ )$ is defined by

$$L^2(\Omega) \triangleq \left\{ g(\cdot) : \Omega \to \mathbb{R} : \int_\Omega |g(\mathbf{x})|^2 dx < +\infty \right\} \tag{3.1.1}$$

where

$$(g(\cdot), h(\cdot)) \triangleq \int_\Omega g(\mathbf{x})\, h(\mathbf{x})\, dx.^1$$

The Sobolev Space $H^1(\Omega)$, and its associated norm $\|\ \|_1$ and inner product $(\ ,\ )$ are defined by

$$H^1(\Omega;\mathbb{R}) \triangleq \left\{ g(\cdot) \in L^2(\Omega) : \|g\|_1 < +\infty \right\}, \tag{3.1.2}$$

where

$$(g(\cdot), h(\cdot))_1 = (g(\cdot), h)(\cdot) + \int_\Omega \langle \nabla g(\mathbf{x}), \nabla h(\mathbf{x}) \rangle\, d\mathbf{x},$$

and

$$\|g(\cdot)\|_1 = \sqrt{(g(\cdot), g(\cdot))_1}.$$

We also need the space $H^1_{g0}(\Omega)$ of $H^1(\Omega)$ defined by

$$H^1_{g0}(\Omega) = \left\{ g(\cdot) \in H^1(\Omega) : g(\mathbf{x})|_{\Gamma_{Eg}} = 0 \right\}, \tag{3.1.3}$$

and the dual space $H^{-1}(\Omega)$, which is the space of bounded linear functionals (specified as a Riesz map) on $H^1_0(\Omega)$ with its associated norm $\|l\|_{-1}$. Thus,

$$H^{-1}(\Omega) = \left\{ l(\cdot) : (l(\cdot), g(\cdot)) < \infty \quad \forall\, g(\cdot) \in H^1_0(\Omega) \right\}, \tag{3.1.4}$$

and

$$\|l(\cdot)\|_{-1} = \sup_{g \in H^1_0(\Omega) : \|g\|_1 = 1} (l(\cdot), g(\cdot)). \tag{3.1.5}$$

For the case of the Boussinesq system, we must consider spaces of vector functions consisting of velocity, temperature, and pressure. In particular, we set

$$\begin{gathered} \mathbf{L}^2(\Omega) = \left\{ \mathbf{g}(\cdot) \in L^2(\Omega) \times ... \times L^2(\Omega) : \|\mathbf{g}(\cdot)\| < \infty \right\}; \\ (\mathbf{g}(\cdot), \mathbf{h}(\cdot)) = \sum_{i=1}^{d} (g_i(\cdot), h_i(\cdot)), \end{gathered} \tag{3.1.6}$$

$$\begin{gathered} \mathbf{H}^1(\Omega) = \left\{ \mathbf{g}(\cdot) \in H^1(\Omega) \times ... \times H^1(\Omega) : \|\mathbf{g}(\cdot)\|_1 < \infty \right\}, \\ (\mathbf{g}(\cdot), \mathbf{h}(\cdot))_1 = \sum_{i=1}^{d} (g(\cdot)_i, h(\cdot)_i)_1 \end{gathered} \tag{3.1.7}$$

[1] Note that $\|g(\cdot)\| = \sqrt{(g(\cdot), g(\cdot))}$

$$\begin{aligned}\mathbf{H}^{-1}(\Omega) &= \left\{H^{-1}(\Omega) \times ... \times H^{-1}(\Omega)\right\}, \\ \|\phi(\cdot)\|_{-1} &= \sup_{\mathbf{g}\in\mathbf{H}_0^1(\Omega):\|\phi\|_1=1} (\phi(\cdot), \mathbf{g}(\cdot)),\end{aligned} \tag{3.1.8}$$

$$\mathbf{L}^2_{\mathbf{g}0}(\Omega) = \left\{\mathbf{g} \in \mathbf{L}^2(\Omega) : \mathbf{g}(\mathbf{x})|_{\Gamma_{E\mathbf{g}}} = 0\right\}, \tag{3.1.9}$$

$$\mathbf{H}^1_{\mathbf{g}0}(\Omega) = \left\{\mathbf{g} \in \mathbf{H}^1(\Omega) : \mathbf{g}(\mathbf{x})|_{\Gamma_{E\mathbf{g}}} = 0\right\}, \tag{3.1.10}$$

and

$$\mathbf{H}_{div} = \left\{\mathbf{g}(\cdot) \in \mathbf{H}^1(\Omega) : \nabla \cdot \mathbf{g}(\mathbf{x}) = 0, \quad \text{for } \mathbf{x} \in \Omega\right\}. \tag{3.1.11}$$

The above spaces are adequate for space-dependent functions, but must be somehow extended to ensure finite energy over time. Therefore, given a Hilbert space $\mathbf{H}$ with norm $\|\cdot\|_{\mathbf{H}}$, we define the space of strongly measurable maps $L^2(0, t_f; \mathbf{H})$, with its associated norm $\|g\|_{L^2(0,t_f;\mathbf{H})}$, as

$$\begin{aligned}L^2(0, t_f; \mathbf{H}) &= \left\{g : [0, t_f] \to \mathbf{H} : \|g\|_{L^2(0,t_f;\mathbf{H})} < \infty\right\} \\ \|g\|_{L^2(0,t_f;\mathbf{H})} &= \left(\int_0^{t_f} \|g(t)\|^2_{\mathbf{H}}\, dt\right)^{\frac{1}{2}}.\end{aligned} \tag{3.1.12}$$

Similarly,

$$\begin{aligned}L^\infty(0, t_f; \mathbf{H}) &= \left\{g : [0, t_f] \to \mathbf{H} : \|g\|_{L^\infty(0,t_f;\mathbf{H})} < \infty\right\} \\ \|g\|_{L^\infty(0,t_f;\mathbf{H})} &= \operatorname*{ess\,sup}_{0<t<t_f} \|g(t)\|_{\mathbf{H}}.\end{aligned} \tag{3.1.13}$$

3.2 Weak Form of the Boussinesq System

Although the derivation of the weak form is rather standard, we review the process since inclusion of Dirichlet boundary control requires additional nonstandard treatment. We divide the discussion of (2.1.26) into the equations and conditions that express air flow (Incompressible Navier-Stokes Equations), followed by equations for heat flow (Energy Equation).

3.2.1 Incompressible Navier-Stokes Equations

Consider the Incompressible Navier-Stokes Equations with control,

$$\frac{\partial \mathbf{v}}{\partial t} = \nabla \cdot \frac{1}{\rho_0}\left(-pI + \mu\left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right)\right) + \mathbf{v} \cdot \nabla \mathbf{v} + \frac{1}{\rho_0}\mathbf{F}, \tag{3.2.1}$$

$$\nabla \cdot \mathbf{v} = 0, \tag{3.2.2}$$

$$\mathbf{v}_{\Gamma_{E\mathbf{v}}} = \mathbf{h}\left(\mathbf{x}\right) u_{\mathbf{v}}\left(t\right), \tag{3.2.3}$$

$$\left[-p\mathbf{I} + \mu\left(\nabla \mathbf{v}_{\Gamma_{N\mathbf{v}}} + (\nabla \mathbf{v}_{\Gamma_{N\mathbf{v}}})^T\right)\right] \cdot \mathbf{n} = 0, \tag{3.2.4}$$

$$\mathbf{v}\left(\mathbf{x}, 0\right) = \mathbf{v}_0\left(\mathbf{x}\right). \tag{3.2.5}$$

To derive an associated weak-form we take the $\mathbf{L}^2$ inner product of the momentum equation (3.2.1) with any adequately smooth nonzero vector "test" function $\varphi \in \mathbf{H}^1_{\mathbf{v}0}\left(\Omega\right)$, so that

$$\begin{aligned}
\left(\frac{\partial \mathbf{v}}{\partial t}, \varphi\right) &= \left(\nabla \cdot \frac{1}{\rho_0}\left(-pI + \mu\left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right)\right) - \mathbf{v} \cdot \nabla \mathbf{v} + \frac{1}{\rho_0}\mathbf{F}_b, \varphi\right) \\
&= \int_\Omega \left\langle \nabla \cdot \frac{1}{\rho_0}\left(\underbrace{-pI + \mu\left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right)}_{\text{Define as } \mathcal{T}}\right) - \mathbf{v} \cdot \nabla \mathbf{v} + \frac{1}{\rho_0}\mathbf{F}_b, \varphi \right\rangle d\mathbf{x} \\
&= \frac{1}{\rho_0}\int_\Omega \left[\left[\nabla \cdot \mathcal{T}\right] \cdot \varphi\right] d\mathbf{x} - \int_\Omega \left[\mathbf{v} \cdot \nabla \mathbf{v} \cdot \varphi\right] d\mathbf{x} + \frac{1}{\rho_0}\int_\Omega \left[\mathbf{F}_b \cdot \varphi\right] d\mathbf{x}.
\end{aligned} \tag{3.2.6}$$

Observe that

$$\begin{aligned}
\nabla \cdot (\mathcal{T} \cdot \varphi) &= \nabla \cdot \left[\varphi_1 \begin{pmatrix} \mathcal{T}_{11} \\ \mathcal{T}_{12} \\ \mathcal{T}_{13} \end{pmatrix} + \varphi_2 \begin{pmatrix} \mathcal{T}_{21} \\ \mathcal{T}_{22} \\ \mathcal{T}_{23} \end{pmatrix} + \varphi_3 \begin{pmatrix} \mathcal{T}_{31} \\ \mathcal{T}_{32} \\ \mathcal{T}_{33} \end{pmatrix}\right] \\
&= \left[\nabla\varphi_1 \cdot \begin{pmatrix} \mathcal{T}_{11} \\ \mathcal{T}_{12} \\ \mathcal{T}_{13} \end{pmatrix} + \nabla\varphi_2 \cdot \begin{pmatrix} \mathcal{T}_{21} \\ \mathcal{T}_{22} \\ \mathcal{T}_{23} \end{pmatrix} + \nabla\varphi_3 \cdot \begin{pmatrix} \mathcal{T}_{31} \\ \mathcal{T}_{32} \\ \mathcal{T}_{33} \end{pmatrix}\right] \\
&+ \left[\varphi_1\nabla \cdot \begin{pmatrix} \mathcal{T}_{11} \\ \mathcal{T}_{12} \\ \mathcal{T}_{13} \end{pmatrix} + \varphi_2\nabla \cdot \begin{pmatrix} \mathcal{T}_{21} \\ \mathcal{T}_{22} \\ \mathcal{T}_{23} \end{pmatrix} + \varphi_3\nabla \cdot \begin{pmatrix} \mathcal{T}_{31} \\ \mathcal{T}_{32} \\ \mathcal{T}_{33} \end{pmatrix}\right] \\
&= \mathcal{T} : \nabla\varphi + \left[\nabla \cdot \mathcal{T}\right] \cdot \varphi,
\end{aligned} \tag{3.2.7}$$

where, given two $N \times N$ matrices, $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{A} : \mathbf{B} \triangleq \sum_{i,j=1}^{N} A_{ij}B_{ij}$.

Equation (3.2.7) implies

$$(\nabla \cdot \mathcal{T}) \cdot \varphi = \nabla \cdot (\mathcal{T} \cdot \varphi) - \mathcal{T} : \nabla\varphi.$$

Thus,

$$\int_{\Omega} [(\nabla \cdot \mathcal{T}) \cdot \varphi] \, d\mathbf{x} = \int_{\Omega} [(\nabla \cdot \mathcal{T}) \cdot \varphi] \, d\mathbf{x} - \int_{\Omega} [\mathcal{T} : \nabla\varphi] \, d\mathbf{x},$$

which, by the Divergence Theorem, implies

$$\int_{\Omega} [(\nabla \cdot \mathcal{T}) \cdot \varphi] \, d\mathbf{x} = \int_{\Gamma} [(\mathcal{T} \cdot \varphi) \cdot \mathbf{n}] \, d\mathbf{x} - \int_{\Omega} [\mathcal{T} : \nabla\varphi] \, d\mathbf{x}.$$

Hence, we have

$$\int_{\Omega} [[\nabla \cdot \mathcal{T}] \cdot \varphi] \, d\mathbf{x} = \int_{\Gamma} [(\mathcal{T} \cdot \varphi) \cdot \mathbf{n}] \, d\mathbf{x} - \int_{\Omega} \left[\mu \left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) : \nabla\varphi - p\nabla \cdot \varphi\right] d\mathbf{x}. \quad (3.2.8)$$

Therefore (3.2.6) and (3.2.8), along with the divergence-free (incompressibility) condition and homogeneous Neumann condition yields

$$\begin{aligned} \int_{\Omega} \left[\frac{\partial \mathbf{v}}{\partial t} \cdot \varphi\right] d\mathbf{x} = &- \frac{1}{\rho_0} \int_{\Omega} \left[\mu \left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) : \nabla\varphi\right] d\mathbf{x} + \frac{1}{\rho_0} \int_{\Omega} p\nabla \cdot \varphi d\mathbf{x} \\ &- \int_{\Omega} \mathbf{v} \cdot \nabla\mathbf{v} \cdot \varphi d\mathbf{x} + \frac{1}{\rho_0} \int_{\Omega} \mathbf{F}_b \cdot \varphi d\mathbf{x}, \end{aligned} \quad (3.2.9)$$

and

$$\int_{\Omega} \phi\nabla \cdot \mathbf{v} d\mathbf{x} = 0, \text{ given any } \phi \in H^1_{\mathbf{v}0}, \ \varphi \in \mathbf{H}^1_{\mathbf{v}0}. \quad (3.2.10)$$

It is obvious that for solutions, $\mathbf{v}, p$, of (3.2.1), the equations (3.2.9), (3.2.10) hold for all choices of $\varphi \in \mathbf{H}^1_{\mathbf{v}0}$ and $\phi \in H^1_0$. Therefore, the weak form of the problem is defined by:

Given $\mathbf{v}(\mathbf{x}, 0) = \mathbf{v}_0 \in \mathbf{H}_{div}$ find

$$\mathbf{v} \in L^2\left(0, t_f; \mathbf{H}^1(\Omega)\right) \cap L^\infty\left(0, t_f; \mathbf{H}^1(\Omega)\right); p \in L^2\left(0, t_f; L^2_0(\Omega)\right) \quad (3.2.11)$$

such that

$$\begin{aligned}
&\int_{\Omega}\left[\frac{\partial \mathbf{v}}{\partial t}\cdot\varphi\right]d\mathbf{x}\\
&=-\frac{1}{\rho_0}\int_{\Omega}\left[\mu\left(\nabla\mathbf{v}+(\nabla\mathbf{v})^T\right):\nabla\varphi\right]d\mathbf{x}\\
&+\frac{1}{\rho_0}\int_{\Omega}p\nabla\cdot\varphi d\mathbf{x}-\int_{\Omega}\mathbf{v}\cdot\nabla\mathbf{v}\cdot\varphi d\mathbf{x}+\frac{1}{\rho_0}\int_{\Omega}\mathbf{F}_b\cdot\varphi d\mathbf{x},\\
&\int_{\Omega}\phi\nabla\cdot\mathbf{v}d\mathbf{x}=0,
\end{aligned}\tag{3.2.12}$$

for all

$$\varphi\in\mathbf{H}^1_{\mathbf{v}0}(\Omega),\ \phi\in L^2_{\mathbf{v}0}(\Omega).\tag{3.2.13}$$

To obtain a boundary control operator, after assuming more regular basis functions, another integration by parts is performed on (3.2.12) to obtain a "very weak form". In other words, given $\mathbf{v}(\mathbf{x},0)=\mathbf{v}_0\in\mathbf{H}^2_{div}$ find

$$\mathbf{v}\in L^2\left(0,t_f;\mathbf{H}^2(\Omega)\right)\cap L^\infty\left(0,t_f;\mathbf{H}^2(\Omega)\right);\ p\in L^2\left(0,t_f;L^2_0(\Omega)\right),\tag{3.2.14}$$

such that

$$\begin{aligned}
&\int_{\Omega}\left[\frac{\partial \mathbf{v}}{\partial t}\cdot\varphi\right]d\mathbf{x}\\
&=\frac{\mu}{\rho_0}\int_{\Omega}\langle\mathbf{v},\Delta\varphi\rangle\,d\mathbf{x}+\frac{1}{\rho_0}\int_{\Omega}p\nabla\cdot\varphi d\mathbf{x}-\int_{\Omega}(\mathbf{v}\cdot\nabla)\,\mathbf{v}\cdot\varphi d\mathbf{x}\\
&+\frac{1}{\rho_0}\int_{\Omega}\mathbf{F}_b\cdot\varphi d\mathbf{x}-u_{\mathbf{v}}(t)\frac{1}{\rho_0}\int_{\Gamma_{\mathbf{v}0}}\mathbf{h}_{\mathbf{v}}(\mathbf{x})\cdot\frac{\partial\varphi}{\partial\mathbf{n}}d\mathbf{x},
\end{aligned}\tag{3.2.15}$$

and

$$\int_{\Omega}\phi\nabla\cdot\mathbf{v}d\mathbf{x}=0,$$

for all

$$\phi\in L^2_{\mathbf{v}0},\ \varphi\in\mathbf{H}^2\cap\mathbf{H}^1_{\mathbf{v}0}$$

The very weak formulation clearly brings out a boundary operator on the control, however it does not provide for a practical or easy discrete approximation because of the additional smoothness requirement on the test functions. We will address this issue later.

3.2.2 The Energy Equation

Consider the system

$$\frac{\partial T}{\partial t} = \kappa \Delta T - \mathbf{v} \cdot \nabla T, \tag{3.2.16}$$

with

$$\mathbf{n} \cdot (k \nabla T_{\Gamma_{NT}}) = 0, \tag{3.2.17}$$

$$T_{\Gamma_{ET}} = h_T(\mathbf{x})\, u_T(t), \tag{3.2.18}$$

and

$$T(\mathbf{x}, 0) = T_0(\mathbf{x}). \tag{3.2.19}$$

Similarly to the Navier-Stokes equation, one defines a weak form of the heat convection/diffusion portion of the Boussinesq equations. Assume that conditions make T smooth enough that (3.2.16)-(3.2.19) is well-defined. Take the L^2 inner product of the energy equation with any adequately smooth nonzero "test" function $\phi_T \in H^1_{T0}$, so that

$$\begin{aligned}\left(\frac{\partial T}{\partial t}, \phi_T\right) &= (\kappa \Delta T - \mathbf{v} \cdot \nabla T, \phi_T) \\ &= \kappa \int_\Omega \phi_T \left[\nabla \cdot \nabla T\right] d\mathbf{x} - \int_\Omega \left[(\mathbf{v} \cdot \nabla T)\, \phi_T\right] d\mathbf{x}.\end{aligned} \tag{3.2.20}$$

Apply integration by parts on the first integral in (3.2.20) to obtain

$$\int_\Omega \phi_T \left[\nabla \cdot \nabla T\right] d\mathbf{x} = \int_\Gamma \left[\frac{\partial T}{\partial \mathbf{n}}\right] \phi_T d\mathbf{x} - \int_\Omega \left[\nabla T \cdot \nabla \phi_T\right] d\mathbf{x}. \tag{3.2.21}$$

Using (3.2.20) and (3.2.21) it follows that

$$\int_\Omega \left[\dot{T} \phi_T\right] d\mathbf{x} = -\kappa \int_\Omega \left[\nabla T \cdot \nabla \phi_T\right] d\mathbf{x} - \int_\Omega \left[(\mathbf{v} \cdot \nabla T)\, \phi_T\right] d\mathbf{x} + \underbrace{\kappa \int_\Gamma \left[\frac{\partial T}{\partial \mathbf{n}}\right] \phi_T d\mathbf{x}}_{=0 \text{ Due to B.C.s}}. \tag{3.2.22}$$

Again, (3.2.2) must hold for all choices of $\phi_T \in H^1_{T0}(\Omega)$. Consequently, the weak form of (3.2.16)-(3.2.19) is given by:

Given

$$T(\mathbf{x}, 0) = T_0 \in H^1(\Omega),$$

find

$$T \in L^2\left(0, t_f; H^1(\Omega)\right) \cap L^\infty\left(0, t_f; H^1(\Omega)\right),$$

such that

$$\int_\Omega \left[\dot{T}\phi_T\right] d\mathbf{x} = -\kappa \int_\Omega \left[\nabla T \cdot \nabla \phi_T\right] d\mathbf{x} - \int_\Omega \left[\left(\mathbf{v} \cdot \nabla T\right)\phi_T\right] d\mathbf{x}.$$

for all

$$\phi_T \in H^1_{T0}(\Omega).$$

As with the Navier-Stokes equations, to obtain a boundary control operator, after assuming more regular basis functions, another integration by parts is performed on the Laplacian term to obtain the "very weak form". In other words, find

$$T \in L^2\left(0, t_f; H^2(\Omega)\right) \cap L^\infty\left(0, t_f; H^2(\Omega)\right),$$

such that

$$\int_\Omega \left[\dot{T}\phi_T\right] d\mathbf{x} = \kappa \int_\Omega \left[T\Delta\phi_T\right] d\mathbf{x} - \int_\Omega \left[\left(\left(\mathbf{v} \cdot \nabla\right) T\right)\phi_T\right] d\mathbf{x} - u_T(t)\,\kappa \int_{\Gamma_{T0}} h_T(\mathbf{x}) \frac{\partial \phi_T}{\partial \mathbf{n}} d\mathbf{x},$$

for all

$$\phi_T \in H^2 \cap H^1_{T0}.$$

3.2.3 The Coupled Problem

Combining the very weak forms from Sections 3.2.1 and 3.2.2, one obtains the full very weak control problem associated with the Boussinesq system. However, for finite element methods, we wish to create discrete approximations using the weak form. A possible way to accomplish this is to add the boundary terms from the very weak form into the weak form. This is done at the discrete level and will be discussed in the following section. However, the problem is stated as:

Find

$$\begin{gathered}\mathbf{v}^h \in L^2\left(0, t_f; \mathbf{H}^1(\Omega)\right) \cap L^\infty\left(0, t_f; \mathbf{H}^1(\Omega)\right),\ p^h \in L^2\left(0, t_f; L^2_0(\Omega)\right);, \\ T^h \in L^2\left(0, t_f; H^1(\Omega)\right) \cap L^\infty\left(0, t_f; H^1(\Omega)\right),\end{gathered} \tag{3.2.23}$$

such that

$$\begin{aligned}
&\int_{\Omega}\left[\frac{\partial \mathbf{v}^h}{\partial t}\cdot\varphi^h\right]d\mathbf{x}\\
&=-\frac{1}{\rho_0}\int_{\Omega}\left[\mu\left(\nabla\mathbf{v}^h+\left(\nabla\mathbf{v}^h\right)^T\right):\nabla\varphi^h\right]d\mathbf{x}+\frac{1}{\rho_0}\int_{\Omega}p\nabla\cdot\varphi^h d\mathbf{x}\\
&-\int_{\Omega}\left(\mathbf{v}^h\cdot\nabla\right)\mathbf{v}^h\cdot\varphi^h d\mathbf{x}+\frac{1}{\rho_0}\int_{\Omega}\mathbf{F}_b^h\cdot\varphi^h d\mathbf{x}-u_{\mathbf{v}}(t)\frac{1}{\rho_0}\int_{\Gamma_{\mathbf{v}0}}\mathbf{h}_{\mathbf{v}}^h(\mathbf{x})\cdot\frac{\partial\varphi^h}{\partial\mathbf{n}}d\mathbf{x},
\end{aligned}\tag{3.2.24}$$

$$\int_{\Omega}\phi^h\nabla\cdot\mathbf{v}^h d\mathbf{x}=0,\tag{3.2.25}$$

and

$$\begin{aligned}
\int_{\Omega}\left[\dot{T}^h\phi_T^h\right]d\mathbf{x}&=-\kappa\int_{\Omega}\left[\nabla T^h\cdot\nabla\phi_T^h\right]d\mathbf{x}-\int_{\Omega}\left[\left(\left(\mathbf{v}^h\cdot\nabla\right)T^h\right)\phi_T^h\right]d\mathbf{x}\\
&-u_T(t)\,\kappa\int_{\Gamma_{Tc}}h_T^h(\mathbf{x})\frac{\partial\phi_T^h}{\partial\mathbf{n}}d\mathbf{x}-\kappa\int_{\Gamma_{T_{DT}}}h_{DT}^h(\mathbf{x})\frac{\partial\phi_T^h}{\partial\mathbf{n}}d\mathbf{x},
\end{aligned}\tag{3.2.26}$$

for all $\phi_T^h\in H_{T0}^1,\ \phi^h\in L_{\mathbf{v}0}^2,\ \varphi^h\in\mathbf{H}_{\mathbf{v}0}^1$. In the case of the related diffusion equation, Lasiecka and Triggiani [29] show that solutions to this finite-dimensional approximation converge to the more regular solutions of the very weak form. Linearization of equations (3.2.23)-(3.2.26) is straight-forward so that one can readily obtain the operators necessary to solve the LQR problem.

3.3 Finite Element Approximation

Given the weak form system, the Finite Element Method (FEM) can be used to produce finite-dimensional approximations of the Boussinesq system. We briefly describe this process below.

3.3.1 Description

Partition Ω into a mesh of simply-connected adjacent faces (called elements), which are formed by simple closed curves, specified by either three (triangles or deformed triangles) or four (rectangles or deformed rectangles) distinct adjacent edges, each of which is specified by two adjacent vertices. This is done in such a way that no vertex is found in the interior of a face or edge. We call the partitioned domain Ω^h, where h is the maximum length of any edge in the partition (see Figure 3.1).

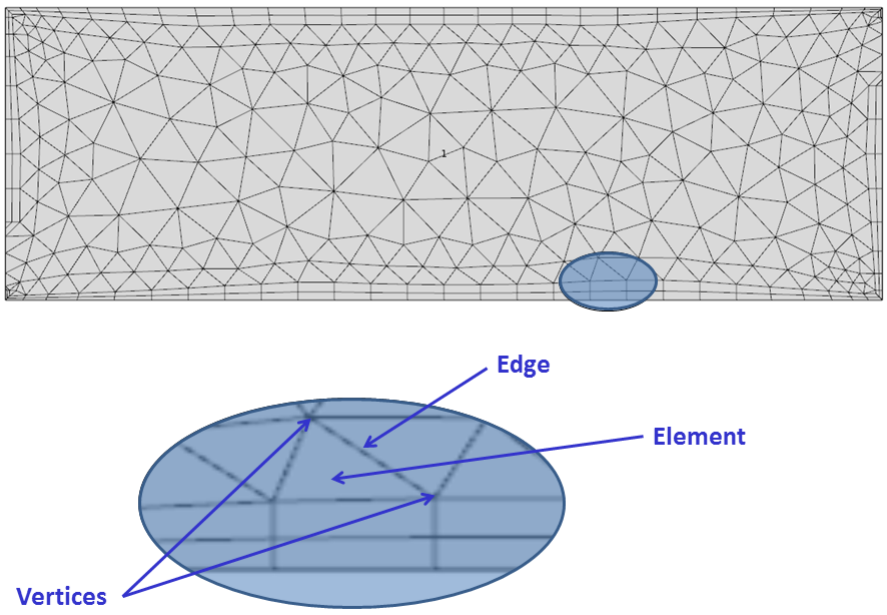

Figure 3.1: Example COMSOL® Meshed Domain Ω^h

The vertex points of Ω^h are called the nodes. In the case of piecewise quadratic calculations (for continuity across edges and to avoid singular systems), one specifies an additional node on every edge of Ω^h as well as possibly (but not necessarily) the interior of every 4-node element of Ω^h. Total count of vertex nodes is denoted by N_{vtx} and the total count of nodes N.

In the case of the Galerkin FEM, the load functions, Dirichlet boundary conditions, initial conditions, and coupling parameters are all spatially interpolated to be exact at each of the vertex nodes where these conditions and functions are specified.[2] For this thesis, these interpolations are represented as linear combinations of independent piecewise linear or piecewise quadratic spatially-dependent polynomials (ϕ_i^h and φ_i^h) with the property that for a given node i, $\phi_i^h(\mathbf{x}_j) = \delta(\mathbf{x}_i - \mathbf{x}_j)$ or $\varphi_i^h(\mathbf{x}_j) = \delta(\mathbf{x}_i - \mathbf{x}_j)$ and that the support of $\phi_i^h(\mathbf{x}_j)$ or $\varphi_i^h(\mathbf{x}_j)$ is limited to the union of faces whose closure contain node i (see Figure 3.2). These basis functions are easily found by stitching together "shape functions", element by element, that are nothing more than the unique Lagrange polynomials, such that for a given node, i, on the closure of a given element, $\varphi_i^h(\mathbf{x}_j) = \delta(\mathbf{x}_i - \mathbf{x}_j)$ for each node, j, on the closure of the same element[3] (see Figure 3.3).[4]

[2]For piecewise quadratic interpolations, we require exactness at the interior nodes as well.

[3]Similar for ϕ_i^h.

[4]'*Each*' element node applies to the case of piecewise quadratic polynomials. Only *vertex* element nodes are used for piecewise linear polynomials.

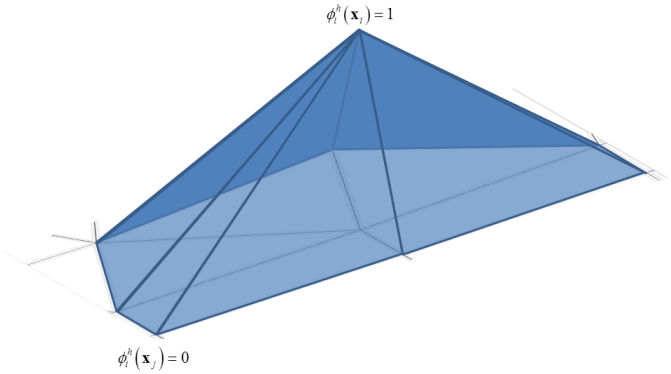

Figure 3.2: Example Linear Basis Function from COMSOL Generated Mesh

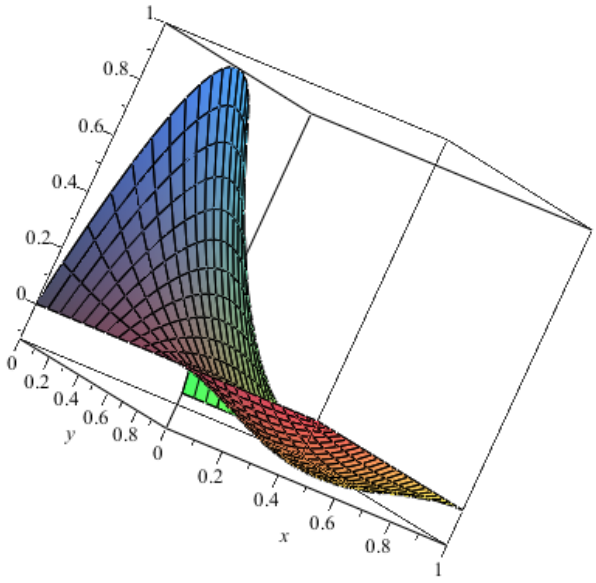

Figure 3.3: Example of Normalized Quadratic Shape Function on a Rectangular Element

Definitions

The FEM can be described in terms of the following sets:

Name		Description
$\mathbf{X}^h$	=	$\left\{ \mathbf{x}_i\big\vert_{i=1}^{N} \in \Omega^h : \mathbf{x}_i \text{ is a node in } \Omega^h \right\}$
$\mathbf{X}_0^h$	=	$\left\{ \mathbf{x}_i \in \mathbf{X}^h : \mathbf{x}_i \ \in \left(\text{interior } \Omega^h\right) \cup \left(\Gamma_N \cap \mathbf{X}^h\right) \right\}$[5]
$\mathbf{X}_{vtx}^h$	=	$\left\{ \mathbf{x}_i \in \mathbf{X}^h : \mathbf{x}_i \text{ is a vertex} \right\}$
$\mathbf{X}_{\Gamma_E}^h$	=	$\left\{ \mathbf{x}_i \in \mathbf{X}^h : \mathbf{x}_i \ \in \Gamma_E \right\}$ [6]
Φ^h	=	$\left\{ \phi_i : \mathbf{x} \to \mathbb{R} : \begin{array}{c} \phi_i\left(\mathbf{x}_j\right) = \delta\left(\mathbf{x}_j - \mathbf{x}_i\right) \text{ for } \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}_{vtx}^h \\ \phi_i\left(\mathbf{x}\right) \in P_1\left(\mathbf{x}\right) \text{ on each element of } \Omega^h \end{array} \right\}$
Φ_0^h	=	$\left\{ \phi_i \in \Phi^h : \mathbf{x}_i \in \mathbf{X}_{vtx}^h \cap \mathbf{X}_0^h \right\}$
$\Phi_{\Gamma_E}^h$	=	$\left\{ \phi_i \in \Phi^h : \mathbf{x}_i \in \mathbf{X}_{vtx}^h \cap \mathbf{X}_{\Gamma_E}^h \right\}$
Ψ^h	=	$\left\{ \varphi_i : \mathbf{x} \to \mathbb{R} : \begin{array}{c} \varphi_i\left(\mathbf{x}_j\right) = \delta\left(\mathbf{x}_j - \mathbf{x}_i\right) \text{ for } \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}^h \\ \varphi_i\left(\mathbf{x}\right) \in P_2\left(\mathbf{x}\right) \text{ on each element of } \Omega^h \end{array} \right\}$
Ψ_0^h	=	$\left\{ \varphi_i \in \Psi^h : \mathbf{x}_i \in \mathbf{X}_0^h \right\}$
$\Psi_{\Gamma_{vE}}^h$	=	$\left\{ \varphi_i \in \Psi^h : \mathbf{x}_i \in \mathbf{X}_{\Gamma_{vE}}^h \right\}$
$\mathcal{PT}^h$	=	$\text{span}\left(\Phi^h\right)$
$\mathcal{PT}_{T0}^h$	=	$\text{span}\left(\Phi_{T0}^h\right)$
$\mathcal{V}^h$	=	$\text{span}\left(\Psi^h\right)$
$\mathcal{V}_0^h$	=	$\text{span}\left(\Psi_{\mathbf{v}0}^h\right)$
$\mathcal{V}_{\Gamma_{vE}}^h$	=	$\text{span}\left(\Psi_{\Gamma_{vE}}^h\right)$
$\mathcal{V}_{\Gamma_{TE}}^h$	=	$\text{span}\left(\Phi_{\Gamma_{TE}}^h\right)$

Table 3.1: Sets Needed to Construct Finite Element Models

3.3.2 Projection of the Weak Form onto a Finite-Dimensional Space

Now we are ready to approximate each of the functions and conditions from Section 3.2.3. Following the approach used by COMSOL®, we convert the essential portion of the boundary into a natural boundary condition with a boundary constraint that provides for specification of the normal component as a Lagrange multiplier function. This method of representing boundary conditions is explained by Braess [13].

Also note that the velocity discretizations use piecewise polynomials one degree higher

than the pressure discretizations. Specifically, quadratic velocity with linear pressure on triangular and/or quadrilateral elements has been shown to satisfy the the so-called "div-stability condition" which ensure that the finite-dimensional weak divergence-free constraint approaches the infinite-dimensional constraint as element size approaches zero [20]. For simplicity, we approximate the temperature discretizations with the same basis functions as the pressure discretizations.

Coefficient Matrices

We begin by defining key coefficient matrices needed to concisely represent the approximate (finite-dimensional) weak form[7]. First, we must have notation for the finite dimensional subspaces we use. These are

$$N \triangleq \dim \mathcal{V}^h,\ N_{\mathcal{PT}^h} \triangleq \dim \mathcal{PT}^h,\ N_{\mathcal{V}^h_{\Gamma_{vE}}} \triangleq \dim \mathcal{V}^h_{\Gamma_{vE}}, N_{\mathcal{V}^h_{\Gamma_{TE}}} \triangleq \dim \mathcal{V}^h_{\Gamma_{TE}}. \tag{3.3.1}$$

Next we list the matrices that define the finite-dimensional systems of ordinary differential equations. In particular, let

$$\underset{N\times N}{\mathbf{M}} = \int_\Omega \underset{N\times 1}{\Psi^h(\mathbf{x})} \underset{1\times N}{\left(\Psi^h(\mathbf{x})\right)^T} d\mathbf{x}, \quad \underset{N_{\mathcal{PT}^h}\times N_{\mathcal{PT}^h}}{\mathbf{M}_T} = \int_\Omega \underset{N_{\mathcal{PT}^h}\times 1}{\Phi^h(\mathbf{x})} \underset{1\times N_{\mathcal{PT}^h}}{\left(\Phi^h(\mathbf{x})\right)^T} d\mathbf{x}, \tag{3.3.2}$$

$$\underset{N\times N}{\mathbf{C}\left(\mathbf{v}^h\right)} = \int_\Omega \underset{N\times 1}{\Psi^h} \left[\left(\underset{1\times N}{\left(\Psi^h\right)^T} \underset{N\times 1}{V_1(t)} \right) \underset{1\times N}{\frac{\partial \left(\Psi^h\right)^T}{\partial x}} + \left(\underset{1\times N}{\left(\Psi^h\right)^T} \underset{N\times 1}{V_2(t)} \right) \underset{1\times N}{\frac{\partial \left(\Psi^h\right)^T}{\partial y}} \right] d\mathbf{x}, \tag{3.3.3}$$

$$\underset{N\times N_{\mathcal{PT}^h}}{\mathbf{Q}_x} = \frac{1}{\rho_0} \int_\Omega \underset{N\times 1}{\frac{\partial \Psi^h}{\partial x}} \underset{1\times N_{\mathcal{PT}^h}}{\left(\Phi^h\right)^T} d\mathbf{x}, \quad \underset{N\times N_{\mathcal{PT}^h}}{\mathbf{Q}_y} = \frac{1}{\rho_0} \int_\Omega \underset{N\times 1}{\frac{\partial \Psi^h}{\partial y}} \underset{1\times N_{\mathcal{PT}^h}}{\left(\Phi^h\right)^T} d\mathbf{x}, \tag{3.3.4}$$

$$\underset{N\times N}{\mathbf{K}_{xx}} = \frac{\mu}{\rho_0} \int_\Omega \underset{N\times 1}{\frac{\partial \Psi^h}{\partial x}} \underset{1\times N}{\left(\frac{\partial \Psi^h}{\partial x}\right)^T} d\mathbf{x}, \ \underset{N\times N}{\mathbf{K}_{xy}} = \frac{\mu}{\rho_0} \int_\Omega \underset{N\times 1}{\frac{\partial \Psi^h}{\partial x}} \underset{1\times N}{\left(\frac{\partial \Psi^h}{\partial y}\right)^T} d\mathbf{x}, \tag{3.3.5}$$

$$\underset{N\times N}{\mathbf{K}_{yx}} = \frac{\mu}{\rho_0} \int_\Omega \underset{N\times 1}{\frac{\partial \Psi^h}{\partial y}} \underset{1\times N}{\left(\frac{\partial \Psi^h}{\partial x}\right)^T} d\mathbf{x}, \ \underset{N\times N}{\mathbf{K}_{yy}} = \frac{\mu}{\rho_0} \int_\Omega \underset{N\times 1}{\frac{\partial \Psi^h}{\partial y}} \underset{1\times N}{\left(\frac{\partial \Psi^h}{\partial y}\right)^T} d\mathbf{x}, \tag{3.3.6}$$

$$\underset{N\times N_{\mathcal{PT}^h}}{\mathbf{B}} = g\alpha_T \int_\Omega \underset{N\times 1}{\Psi^h} \underset{1\times N_{\mathcal{PT}^h}}{\Phi^h} d\mathbf{x}, \ \mathbf{F}_{ref} = g\alpha_T \int_\Omega \underset{N\times 1}{\Psi^h} T_0 d\mathbf{x}, \tag{3.3.7}$$

$$\underset{N_{\mathcal{PT}^h}\times N_{\mathcal{PT}^h}}{\mathbf{L}} = \kappa \int_\Omega \left[\underset{N_{\mathcal{PT}^h}\times 1}{\frac{\partial \Phi^h}{\partial x}} \underset{1\times N_{\mathcal{PT}^h}}{\left(\frac{\partial \Phi^h}{\partial x}\right)^T} + \underset{N_{\mathcal{PT}^h}\times 1}{\frac{\partial \Phi^h}{\partial y}} \underset{1\times N_{\mathcal{PT}^h}}{\left(\frac{\partial \Phi^h}{\partial y}\right)^T} \right] d\mathbf{x}, \tag{3.3.8}$$

[7] Our notations mostly follow those of Reddy and Gartling [42].

and

$$\underset{N_{\mathcal{PT}^h}\times N_{\mathcal{PT}^h}}{\mathbf{D}\left(\mathbf{v}^h\right)} = \int_{\Omega} \underset{N_{\mathcal{PT}^h}\times 1}{\Phi^h} \left[\left(\underset{1\times N}{\left(\Psi^h\right)^T} \underset{N\times 1}{\mathbf{V}_1\left(t\right)} \right) \underset{1\times N_{\mathcal{PT}^h}}{\left(\frac{\partial \Phi^h}{\partial x}\right)^T} + \left(\underset{1\times N}{\left(\Psi^h\right)^T} \underset{N\times 1}{\mathbf{V}_2\left(t\right)} \right) \underset{1\times N_{\mathcal{PT}^h}}{\left(\frac{\partial \Phi^h}{\partial y}\right)^T} \right] d\mathbf{x}. \tag{3.3.9}$$

Weak Formulation and Finite-Dimensional Model

We introduce a typical FEM approximation of the Boussinesq system weak form, used by COMSOL® , that does not include a representation of the boundary control operator derived from the very weak form. We will show by example that with proper handling of the boundary conditions in the discrete system, one can obtain a finite-dimensional boundary control operator that approximates the infinite-dimensional "B" operator.

A FEM approximation of the Boussinesq system weak form can be formed in terms of the previous matrices.[8] This approximation is to find

$$\mathbf{v}^h\left(\mathbf{x},t\right) \in \mathcal{V}^h \times \mathcal{V}^h = \begin{pmatrix} \sum_{j=1}^{N} v_1^h\left(\mathbf{x}_j,t\right)\varphi_j\left(\mathbf{x}\right) \\ \sum_{j=1}^{N} v_1^h\left(\mathbf{x}_j,t\right)\varphi_j\left(\mathbf{x}\right) \end{pmatrix} = \begin{pmatrix} \underset{1\times N}{\left(\Psi^h\left(\mathbf{x}\right)\right)^T} \underset{N\times 1}{\mathbf{V}_1\left(t\right)} \\ \left(\Psi^h\left(\mathbf{x}\right)\right)^T \mathbf{V}_2\left(t\right) \end{pmatrix};\ \mathbf{x}_j \in \mathbf{X}^h, \tag{3.3.10}$$

$$T^h\left(\mathbf{x},t\right) \in \mathcal{PT}^h = \sum_{j=1}^{N_{\mathcal{PT}^h}} T^h\left(\mathbf{x}_j,t\right)\phi_j\left(\mathbf{x}\right) = \underset{1\times N_{\mathcal{PT}^h}}{\left(\Phi^h\left(\mathbf{x}\right)\right)^T} \underset{N_{\mathcal{PT}^h}\times 1}{\mathbf{T}\left(t\right)};\ \mathbf{x}_j \in \mathbf{X}_{vtx}^h, \tag{3.3.11}$$

$$p^h\left(\mathbf{x},t\right) \in \mathcal{PT}^h = \sum_{j=1}^{N_{\mathcal{PT}^h}} p^h\left(\mathbf{x}_j,t\right)\phi_j\left(\mathbf{x}\right) = \underset{1\times N_{\mathcal{PT}^h}}{\left(\Phi^h\left(\mathbf{x}\right)\right)^T} \underset{N_{\mathcal{PT}^h}\times 1}{\mathbf{P}\left(t\right)};\ \mathbf{x}_j \in \mathbf{X}_{vtx}^h, \tag{3.3.12}$$

and

$$\lambda^h\left(\mathbf{x},t\right), \lambda_T^h\left(\mathbf{x},t\right), \tag{3.3.13}$$

[8] Below COMSOL® doesn't really address constraints and associated Lagrange multipliers until the fully discrete system is assembled.

such that the weak-form momentum equation approximation is

$$\begin{aligned}\int_{\Omega}\left[\dot{\mathbf{v}}^h\cdot\varphi_i^h\right]d\mathbf{x} &= \begin{pmatrix}\mathbf{M} & 0\\ 0 & \mathbf{M}\end{pmatrix}\begin{pmatrix}\dot{\mathbf{V}}_1(t)\\ \dot{\mathbf{V}}_2(t)\end{pmatrix}\\ &= -\frac{1}{\rho_0}\int_{\Omega}\left[\mu\left(\nabla\mathbf{v}^h+\left(\nabla\mathbf{v}^h\right)^T\right):\nabla\varphi_i^h\right]d\mathbf{x}+\frac{1}{\rho_0}\int_{\Omega}p^h\nabla\cdot\varphi_i^h d\mathbf{x}\\ &\quad-\int_{\Omega}\mathbf{v}^h\cdot\nabla\mathbf{v}^h\cdot\varphi_i^h d\mathbf{x}+\frac{1}{\rho_0}\int_{\Omega}\mathbf{F}\left(T^h\right)_b\cdot\varphi_i^h d\mathbf{x}+\frac{1}{\rho_0}\int_{\Gamma_{\mathbf{v}N}}\left[\left(\underbrace{\mathbf{n}^T\mathcal{T}}_{=0}\varphi_i^h\right)\right]ds\\ &\quad+\frac{1}{\rho_0}\int_{\Gamma_{\mathbf{v}E}}\left[\lambda^h(\mathbf{x},t)\,\varphi_i^h\right]ds\end{aligned} \tag{3.3.14}$$

or

$$\begin{aligned}&\begin{pmatrix}\mathbf{M} & 0\\ 0 & \mathbf{M}\end{pmatrix}\begin{pmatrix}\dot{\mathbf{V}}_1(t)\\ \dot{\mathbf{V}}_2(t)\end{pmatrix}\\ &= -\begin{pmatrix}2\mathbf{K}_{xx}+\mathbf{K}_{yy} & \mathbf{K}_{yx}\\ \mathbf{K}_{xy} & \mathbf{K}_{xx}+2\mathbf{K}_{yy}\end{pmatrix}\begin{pmatrix}\mathbf{V}_1(t)\\ \mathbf{V}_2(t)\end{pmatrix}+\begin{pmatrix}\mathbf{Q}_x\\ \mathbf{Q}_y\end{pmatrix}\mathbf{P}(t)\\ &\quad-\begin{pmatrix}\mathbf{C}\left(\mathbf{v}^h\right) & 0\\ 0 & \mathbf{C}\left(\mathbf{v}^h\right)\end{pmatrix}\begin{pmatrix}\mathbf{V}_1(t)\\ \mathbf{V}_2(t)\end{pmatrix}+\begin{pmatrix}0\\ \mathbf{BT}(t)\end{pmatrix}-\begin{pmatrix}0\\ \mathbf{F}_{ref}\end{pmatrix}\\ &\quad+\underbrace{\frac{1}{\rho_0}\int_{\Gamma_{\mathbf{v}E}}\left[\lambda^h(\mathbf{x},t)\,\varphi_i^h\right]ds}_{\text{Discretized Later}}.\end{aligned} \tag{3.3.15}$$

Also, the weak-form heat equation approximation is

$$\begin{aligned}\int_{\Omega}\left[\dot{T}^h\phi_i^h\right]d\mathbf{x} &= \mathbf{M}_T\dot{\mathbf{T}}(t)\\ &= -\kappa\int_{\Omega}\left[\nabla T^h\cdot\nabla\phi_i^h\right]d\mathbf{x}-\int_{\Omega}\left[\left(\mathbf{v}\cdot\nabla T^h\right)\phi_i^h\right]d\mathbf{x}+\kappa\int_{\Gamma_{TE}}\left[\left(\lambda_T^h(\mathbf{x},t)\,\phi_i^h\right)\right]ds\end{aligned} \tag{3.3.16}$$

$$= -\mathbf{LT}(t)-\mathbf{D}\left(\mathbf{v}^h\right)\mathbf{T}(t)+\underbrace{\kappa\int_{\Gamma_{TE}}\left[\left(\lambda_T^h(\mathbf{x},t)\,\phi_i^h\right)\right]ds}_{\text{Discretized Later}}. \tag{3.3.17}$$

Likewise, the weak-form incompressibility approximation is given by

$$\frac{1}{\rho_0}\int\limits_{\Omega}\phi_i^h\nabla\cdot\mathbf{v}^h d\mathbf{x}=\frac{1}{\rho_0}\left(\begin{array}{cc}\int\limits_{\Omega}\underset{N_{\mathcal{P}\mathcal{T}^h}\times 1}{\Phi^h}\underset{1\times N}{\left(\frac{\partial\Psi^h}{\partial x}\right)^T}d\mathbf{x} & \int\limits_{\Omega}\underset{N_{\mathcal{P}\mathcal{T}^h}\times 1}{\Phi^h}\underset{1\times N}{\left(\frac{\partial\Psi^h}{\partial y}\right)^T}d\mathbf{x}\end{array}\right)\underset{2N\times 1}{\left(\begin{array}{c}\mathbf{V}_1(t)\\ \mathbf{V}_2(t)\end{array}\right)}$$

$$=\left(\begin{array}{cc}(\mathbf{Q}_x)^T & (\mathbf{Q}_y)^T\end{array}\right)\left(\begin{array}{c}\mathbf{V}_1(t)\\ \mathbf{V}_2(t)\end{array}\right)=0. \tag{3.3.18}$$

The boundary constraints are expressed as

$$\mathbf{v}^h(\mathbf{x}_j,t)=\left(\begin{array}{c}\underset{1\times N}{\left(\Psi^h(\mathbf{x}_j)\right)^T}\underset{N\times 1}{\mathbf{V}_1(t)}\\ \underset{1\times N}{\left(\Psi^h(\mathbf{x}_j)\right)^T}\underset{N\times 1}{\mathbf{V}_2(t)}\end{array}\right)=u_{\mathbf{v}}(t)\,\mathbf{h}_{\mathbf{v}}(\mathbf{x}_j)\,;\,\mathbf{x}_j\in\mathbf{X}^h_{\Gamma_{\mathbf{v}E}}, \tag{3.3.19}$$

and

$$T^h(\mathbf{x}_j,t)=\underset{1\times N_{\mathcal{P}\mathcal{T}^h}}{\left(\Phi^h(\mathbf{x}_j)\right)^T}\underset{N_{\mathcal{P}\mathcal{T}^h}\times 1}{\mathbf{T}(t)}=u_T(t)\,h_T(\mathbf{x}_j)\,;\;\;\mathbf{x}_j\in\Gamma_{\mathbf{v}E}\cap\mathbf{X}^h_{vtx}, \tag{3.3.20}$$

$$T^h(\mathbf{x}_j,t)=\underset{1\times N_{\mathcal{P}\mathcal{T}^h}}{\left(\Phi^h(\mathbf{x}_j)\right)^T}\underset{N_{\mathcal{P}\mathcal{T}^h}\times 1}{\mathbf{T}(t)}=h_{DT}(\mathbf{x}_j,t)\,;\;\;\mathbf{x}_j\in\Gamma_{DT}\cap\mathbf{X}^h_{vtx}. \tag{3.3.21}$$

We now restate equations (3.3.2)-(3.3.20) as a constrained, nonlinear system in matrix form,

$$\left(\begin{array}{cccc}\mathbf{M}&&&\\&\mathbf{M}&&\\&&\mathbf{M}_T&\\&&&0\end{array}\right)\left(\begin{array}{c}\dot{\mathbf{V}}_1(t)\\ \dot{\mathbf{V}}_2(t)\\ \dot{\mathbf{T}}(t)\\ \dot{\mathbf{P}}(t)\end{array}\right)$$

$$+\left(\begin{array}{cccc}\mathbf{C}\left(\mathbf{v}^h\right)+2\mathbf{K}_{xx}+\mathbf{K}_{yy} & \mathbf{K}_{yx} & 0 & -\mathbf{Q}_x\\ \mathbf{K}_{xy} & \mathbf{C}\left(\mathbf{v}^h\right)+\mathbf{K}_{xx}+2\mathbf{K}_{yy} & -\mathbf{B} & -\mathbf{Q}_y\\ 0 & 0 & \mathbf{D}\left(\mathbf{v}^h\right)+\mathbf{L} & 0\\ (-\mathbf{Q}_x)^T & (-\mathbf{Q}_y)^T & 0 & 0\end{array}\right)\left(\begin{array}{c}\mathbf{V}_1(t)\\ \mathbf{V}_2(t)\\ \mathbf{T}(t)\\ \mathbf{P}(t)\end{array}\right)$$

$$-\left(\begin{array}{cccc}(\mathbf{N}_{\mathbf{v}^h{}_{11}})^T & 0 & 0 & 0\\ 0 & (\mathbf{N}_{\mathbf{v}^h{}_{22}})^T & 0 & 0\\ 0 & 0 & (\mathbf{N}_{T^h})^T & 0\\ 0 & 0 & 0 & 0\end{array}\right)\left(\begin{array}{c}\mathbf{\Lambda}_1(t)\\ \mathbf{\Lambda}_2(t)\\ \mathbf{\Lambda}_T(t)\\ 0\end{array}\right)+\left(\begin{array}{c}0\\ \mathbf{F}_{ref}\\ 0\\ 0\end{array}\right)=0, \tag{3.3.22}$$

and

$$\begin{pmatrix} \mathbf{N}_{\mathbf{v}^h{}_{11}} & 0 & 0 \\ 0 & \mathbf{N}_{\mathbf{v}^h{}_{22}} & 0 \\ 0 & 0 & \mathbf{N}_{T^h} \end{pmatrix} \begin{pmatrix} \mathbf{V}_1(t) \\ \mathbf{V}_2(t) \\ \mathbf{T}(t) \end{pmatrix} - \begin{pmatrix} u_1^h(t)\, g_{v_1}\left(\mathbf{X}^h_{\Gamma_{\mathbf{v}E}}\right) \\ u_1^h(t)\, g_{v_2}\left(\mathbf{X}^h_{\Gamma_{\mathbf{v}E}}\right) \\ u_2^h(t)\, g_T\left(\mathbf{X}^h_{\Gamma_{\mathbf{T}E}}\right) \end{pmatrix} = 0. \tag{3.3.23}$$

Here the boundary constraints on the coefficients are defined by

$$\mathbf{N}_{\mathbf{v}^h} = \underbrace{\begin{pmatrix} \left(\Psi^h(\mathbf{x}_j)\right)^T_{\mathbf{x}_j \in \mathbf{X}^h_{\Gamma_{\mathbf{v}E}}} & \\ & \left(\Psi^h(\mathbf{x}_j)\right)^T_{\mathbf{x}_j \in \mathbf{X}^h_{\Gamma_{\mathbf{v}E}}} \end{pmatrix}}_{2N_{\mathcal{V}^h_{\Gamma_{vE}}} \times 2N} \tag{3.3.24}$$

and

$$\mathbf{N}_{\mathbf{v}^h} \begin{pmatrix} \underset{N\times 1}{\mathbf{V}_1(t)} \\ \underset{N\times 1}{\mathbf{V}_2(t)} \end{pmatrix} = u_{\mathbf{v}}(t)\, \mathbf{h}_{\mathbf{v}}(\mathbf{x}_j) \text{ where } \mathbf{x}_j \in \mathbf{X}^h_{\Gamma_{\mathbf{v}E}}. \tag{3.3.25}$$

Also,

$$\mathbf{N}_{T^h} = \underset{\left(N_{\mathcal{V}^h_{\Gamma_{TE}}} + N_{\mathcal{V}^h_{\Gamma_{DE}}}\right) \times N_{\mathcal{P}\mathcal{T}^h}}{\left(\Phi^h(\mathbf{x}_j)\right)^T}, \tag{3.3.26}$$

and

$$\mathbf{N}_{T^h}\mathbf{T}(t) = \begin{cases} u_T(t)\, h_T(\mathbf{x}_j) \text{ where } \mathbf{x}_j \in \mathbf{X}^h_{\Gamma_{TE}} \\ h_{DT}(\mathbf{x}_j) \text{ where } \mathbf{x}_j \in \mathbf{X}^h_{\Gamma_{DE}} \end{cases}. \tag{3.3.27}$$

Because some of the dependent variables in these equations are already specified as Dirichlet boundary conditions (some with control), two techniques can be used to approximate boundary control operators which will be discussed later.

3.3.3 Example: 2D Heat Equation

For an example we can use to compare with COMSOL® results, we now take a simple subset of the system, the 2D heat equation with no convection. We construct a 1×1 square domain Ω with an evenly-spaced mesh composed of 0.2×0.2 rectangular elements (see Figure 3.4). Besides the parabolic temperature source, in red, on the left boundary, the rest of the boundary is fully insulated. First, we model the discrete approximation of the dynamics without any type of control. For convenience, we will number the nodes like COMSOL®.

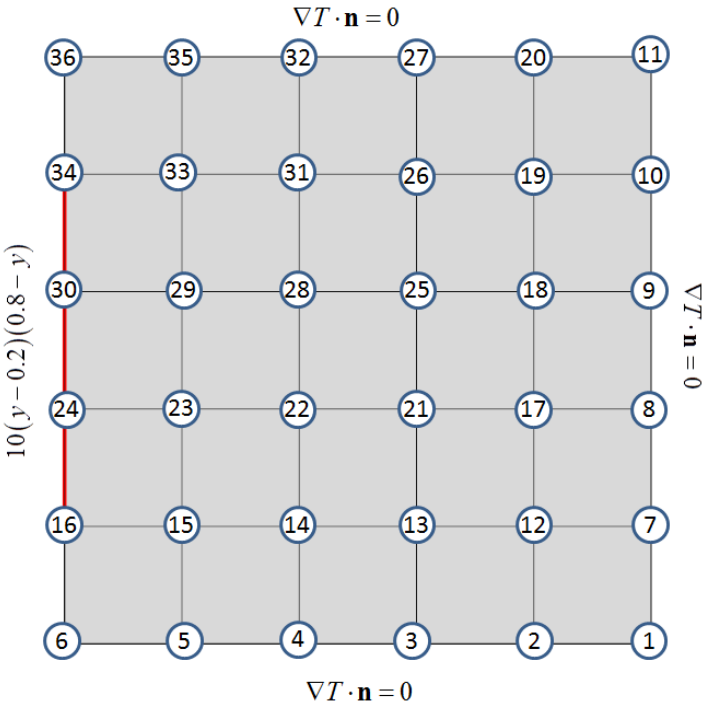

Figure 3.4: Simple Heat Equation Conditions

The discretized equations use the following shape functions (expressed in canonical form) for each element (see Figure 3.5) that are the building blocks for the nodal piecewise-bilinear basis functions.. Here $sh_{LL} : [0, 0.2] \times [0, 0.2] \to \mathbb{R}$ is the bilinear function such that $sh_{LL}(\xi, \eta) = 1$ at the lower-left corner (LL stands for lower left) and $sh_{LL}(\xi, \eta) = 0$ at the other three corners of the element. sh_{LR}, sh_{UR}, and sh_{UR} are similarly defined where LR is "lower-right", UL is "upper-left" and UR is "upper-right".

Therefore, it follows that

$$\mathbf{M}_T \dot{\mathbf{T}}(t) = -\mathbf{L}\mathbf{T}(t) + (\mathbf{N}_{T^h})^T \mathbf{\Lambda}_T(t), \tag{3.3.28}$$

$$\mathbf{N}_{T^h} \mathbf{T}(t) = g_T\left(\mathbf{X}^h_{\Gamma_{TE}}\right), \tag{3.3.29}$$

$$\underset{36\times 4}{\mathbf{N}_{T^h}} = \underset{j=1...36}{\begin{pmatrix} \delta(\mathbf{x}_j - \mathbf{x}_{16}) \\ \delta(\mathbf{x}_j - \mathbf{x}_{24}) \\ \delta(\mathbf{x}_j - \mathbf{x}_{30}) \\ \delta(\mathbf{x}_j - \mathbf{x}_{34}) \end{pmatrix}}, \; g_T\left(\mathbf{X}^h_{\Gamma_{TE}}\right) = \begin{pmatrix} 0 \\ 0.8 \\ 0.8 \\ 0 \end{pmatrix} \tag{3.3.30}$$

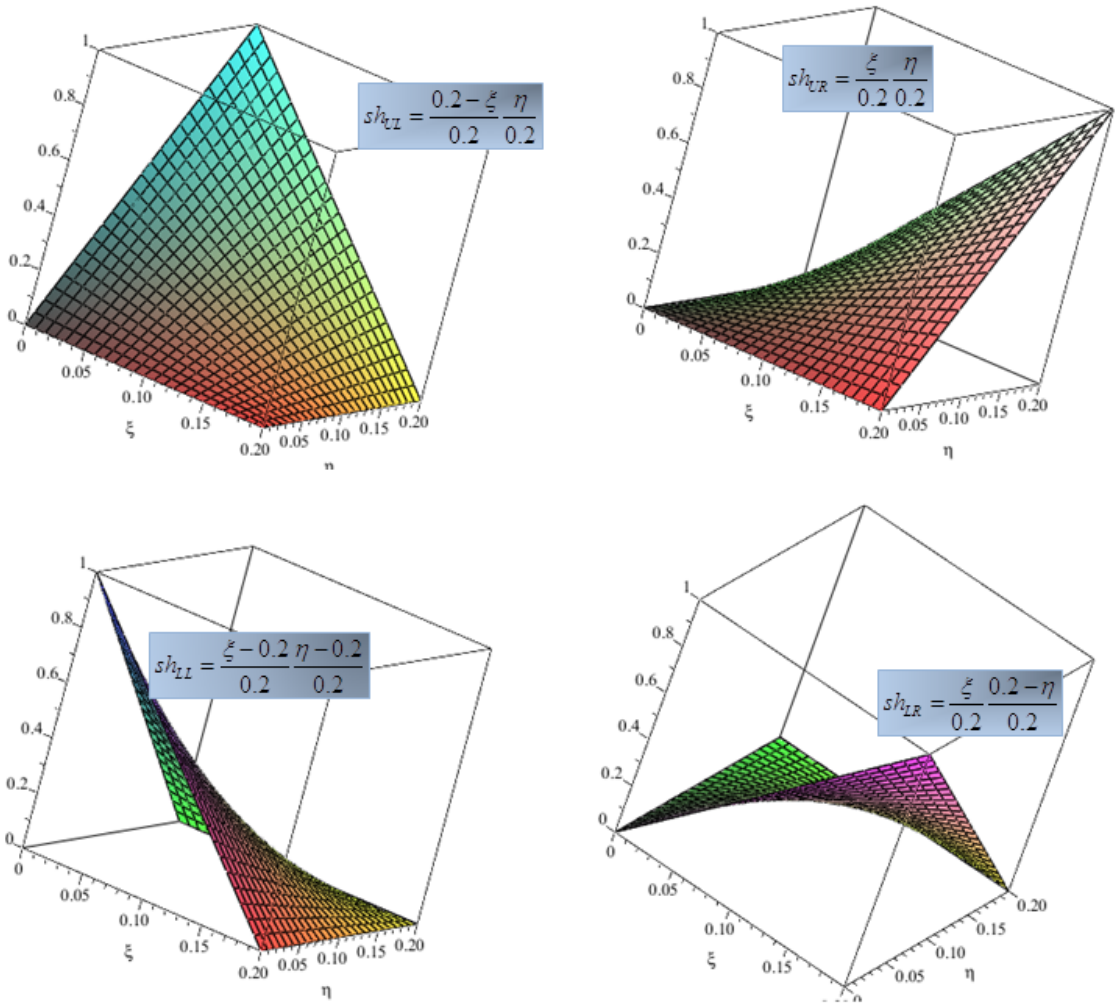

Figure 3.5: Element Shape Functions Conditions

Observe that (see Figure 3.4)

$$\underset{36\times 36}{\mathbf{M}_T} = \int_{\Omega} \underset{36\times 1}{\Phi^h(\mathbf{x})} \underset{1\times 36}{\left(\Phi^h(\mathbf{x})\right)^T} d\mathbf{x} = \left(\int_{\Omega_{ij}} \phi_i(\mathbf{x})\,\phi_j(\mathbf{x})\,d\mathbf{x}\right)_{i,j=1}^{36} \tag{3.3.31}$$

where $\Omega_{ij} = \text{support}\,\phi_i \cap \text{support}\,\phi_j$. Therefore, (referring back to Figure 3.5), one can easily compute

$$
\begin{aligned}
M_{T_{22,23}} &= \int_0^{0.2}\int_0^{0.2} \left(sh_{LL}\left(\xi,\eta\right)\right)\left(sh_{LR}\left(\xi,\eta\right)\right) d\xi d\eta \\
&+ \int_0^{0.2}\int_0^{0.2} \left(sh_{UL}\left(\xi,\eta\right)\right)\left(sh_{UR}\left(\xi,\eta\right)\right) d\xi d\eta \\
&= \int_0^{0.2}\int_0^{0.2} \frac{\xi-0.2}{0.2}\frac{\eta-0.2}{0.2}\frac{\xi}{0.2}\frac{0.2-\eta}{0.2} d\xi d\eta + \int_0^{0.2}\int_0^{0.2} \frac{0.2-\xi}{0.2}\frac{\eta}{0.2}\frac{\xi}{0.2}\frac{\eta}{0.2} d\xi d\eta \\
&\approx 0.00444444, \qquad (3.3.32)
\end{aligned}
$$

and

$$
\begin{aligned}
&M_{T_{22,22}} \\
&= \int_0^{0.2}\int_0^{0.2} \left(sh_{LL}\left(\xi,\eta\right)\right)\left(sh_{LL}\left(\xi,\eta\right)\right) d\xi d\eta + \int_0^{0.2}\int_0^{0.2} \left(sh_{LR}\left(\xi,\eta\right)\right)\left(sh_{LR}\left(\xi,\eta\right)\right) d\xi d\eta \\
&+ \int_0^{0.2}\int_0^{0.2} \left(sh_{UR}\left(\xi,\eta\right)\right)\left(sh_{UR}\left(\xi,\eta\right)\right) d\xi d\eta + \int_0^{0.2}\int_0^{0.2} \left(sh_{UL}\left(\xi,\eta\right)\right)\left(sh_{UL}\left(\xi,\eta\right)\right) d\xi d\eta
\end{aligned}
\qquad (3.3.33)
$$

which implies

$$
\begin{aligned}
&M_{T_{22,22}} \\
&= \int_0^{0.2}\int_0^{0.2} \left(\frac{\xi-0.2}{0.2}\frac{\eta-0.2}{0.2}\right)^2 d\xi d\eta + \int_0^{0.2}\int_0^{0.2} \left(\frac{\xi}{0.2}\frac{0.2-\eta}{0.2}\right)^2 d\xi d\eta \\
&+ \int_0^{0.2}\int_0^{0.2} \left(\frac{\xi}{0.2}\frac{\eta}{0.2}\right)^2 d\xi d\eta + \int_0^{0.2}\int_0^{0.2} \left(\frac{0.2-\xi}{0.2}\frac{\eta}{0.2}\right)^2 d\xi d\eta \\
&\approx 0.01777778.
\end{aligned}
$$

Similarly,

$$\begin{aligned}\underset{36\times 36}{\mathbf{L}} &= \int_{\Omega} \left[\underset{36\times 1}{\frac{\partial \Phi^h}{\partial x}} \underset{1\times 36}{\left(\frac{\partial \Phi^h}{\partial x} \right)^T} + \underset{36\times 1}{\frac{\partial \Phi^h}{\partial y}} \underset{1\times 36}{\left(\frac{\partial \Phi^h}{\partial y} \right)^T} \right] d\mathbf{x} \\ &= \left(\int_{\Omega_{ij}} \frac{\partial \phi_i(\mathbf{x})}{\partial x} \frac{\partial \phi_j(\mathbf{x})}{\partial x} + \frac{\partial \phi_i(\mathbf{x})}{\partial y} \frac{\partial \phi_j(\mathbf{x})}{\partial y} d\mathbf{x} \right)_{i,j=1}^{36},\end{aligned} \tag{3.3.34}$$

so that, for example (again referring to figures 3.4 and 3.5),

$$\begin{aligned}L_{35,31} &= \int_0^{0.2}\int_0^{0.2} \left[\frac{\partial \left(sh_{UL}(\xi,\eta) \right)}{\partial \xi} \frac{\partial \left(sh_{LR}(\xi,\eta) \right)}{\partial \xi} + \frac{\partial \left(sh_{UL}(\xi,\eta) \right)}{\partial \eta} \frac{\partial \left(sh_{LR}(\xi,\eta) \right)}{\partial \eta} \right] d\xi d\eta \\ &= \int_0^{0.2}\int_0^{0.2} \left[\frac{-\eta}{0.04} \frac{0.2-\eta}{0.04} + \frac{-\xi}{0.04} \frac{0.2-\xi}{0.04} \right] d\xi d\eta \\ &\approx -0.33333333.\end{aligned} \tag{3.3.35}$$

In the next chapter, we discuss COMSOL® implementation of the finite element method on this problem

3.4 LQR Boundary Control

Keeping in mind the infinite-dimensional LQR control design steps from (2.3.11) to (2.3.14), we discuss equivalent steps to obtain an approximating LQR control for the discrete system.

3.4.1 Description

We consider an approximate LQR problem defined on a finite element model. Because the heat equation is already linear, this process is straight forward, however we will see that the FEM approximation of the Boussinesq system requires additional work to acquire the needed matrices to solve a discrete LQR problem. Regardless, after linearization about an ideal state and application of boundary constraints to provide a nonsingular linear system, we have the following discrete state-space problem and result that applies to any finite-dimensional approximating system.

Find $\mathbf{u}^*(\cdot) \in \mathbf{L}^2(0,\infty;\mathbb{R}^2)$ such that

$$J^h(\mathbf{u}^*(\cdot)) \leqslant J^h(\mathbf{u}(\cdot)) \tag{3.4.1}$$

where

$$J^h\left(\mathbf{u}\left(t\right)\right)=\int_0^\infty\left[\left(\mathbf{x}^h\left(t\right)\right)^T\ \mathcal{Q}^h\mathbf{x}^h\left(t\right)+\left(\mathbf{u}(t)\right)^T\mathbf{R}\mathbf{u}(t)\right]dt, \tag{3.4.2}$$

and

$$\dot{\mathbf{x}}^h\left(t\right)=A^h\mathbf{x}^h\left(t\right)+B^h\mathbf{u}(t),\ \mathbf{x}^h\left(0\right)=\mathbf{x}_0^h. \tag{3.4.3}$$

where A^h, B^h, and $\mathcal{Q}^h$ are finite element approximations of the infinite-dimensional operators.

It is well-known (see, for example, [2], [12], [15], [28], [31], [37], [48]) that (3.4.1) and (3.4.3) has a unique solution if $\left(A^h,B^h\right)$ is stabilizable and $\left(A^h,\left(\mathcal{Q}^h\right)^{\frac{1}{2}}\right)$ is detectable. If so, then it has the form of state feedback

$$\mathbf{u}^*\left(t\right)=-\mathbf{K}_{opt}^h\mathbf{x}^{h*}\left(t\right), \tag{3.4.4}$$

where $\mathbf{K}_{opt}^h$ is given by

$$\mathbf{K}_{opt}^h=\mathbf{R}^{-1}\left(B^h\right)^T\mathbf{P}_{opt}^h, \tag{3.4.5}$$

and $\mathbf{P}_{opt}^h$ is a non-negative definite solution to the Algebraic Riccati Equation (ARE)

$$\mathbf{P}^hA^h+\left(A^h\right)^T\mathbf{P}^h-\mathbf{P}^hB^h\mathbf{R}^{-1}\left(B^h\right)^T\mathbf{P}^h+\mathcal{Q}^h=0. \tag{3.4.6}$$

Therefore, by obtaining system matrices, one may readily determine state feedback operators. However, it is important to note that the way COMSOL® handles the Dirichlet boundary condition is not fully consistant with the very weak form of the problem. Thus it is important to understand how the Dirichlet boundary control term is constructed. In order to compare the system matrices that com from COMSOL® to existing methods, we present results for the 1D heat equation.

3.4.2 Discretized Heat Equation

We use the discretized heat diffusion equation (without convection) to discuss how COMSOL® obtains approximations of the A^h and B^h. We compare with the exact algebraic method provided by Rubio [44] and follow her derivations in detail. For this problem, the heat equation is already linear, so one avoids a change of variables by simply assuming that $\tilde{T}\left(\mathbf{x}\right)=0$.

Specifically, we break down the problem of obtaining state matrices from a weak form of the one-dimensional heat equation on the unit interval,

$$\dot{T}\left(x,t\right)=\kappa\frac{\partial^2}{\partial x^2}T\left(x,t\right),\ T\left(0,t\right)=u_1\left(t\right),\ T\left(1,t\right)=u_2\left(t\right), \tag{3.4.7}$$

The weak form of (3.4.7) used by COMSOL® is given by

$$\int_0^1 \frac{\partial}{\partial t} T(t,x)\,\varphi(x)\,dx$$
$$= \kappa\left[-\int_0^1 \frac{\partial}{\partial x} T(t,x)\,\frac{\partial}{\partial x}\varphi(x)\,dx + \frac{\partial}{\partial x} T(t,1)\,\varphi(1) - \frac{\partial}{\partial x} T(t,0)\,\varphi(0)\right], \tag{3.4.8}$$

for all $\varphi(\cdot) \in H^1(\Omega)$. This form, when discretized, prevents the boundary integrals (point values for the 1D case) in the weak form from being canceled out by basis functions that only live in $H_0^1(\Omega)$. This is important for determining boundary input matrices from a finite-dimensional approximation.

The FEM approximations are given by

$$\sum_{j=0}^{N+1} \frac{\partial}{\partial t} T_j^h(t) \int_0^1 \varphi_j^h(x)\,\varphi_i^h(x)\,dx$$
$$= -\kappa \sum_{j=0}^{N+1} T_j^h(t) \int_0^1 \left(\varphi_j^h\right)'(x)\left(\varphi_i^h\right)'(x)\,dx + \kappa T_N^h(t)\left(\varphi_N^h\right)'(1)\left(\varphi_i^h\right)(1) \tag{3.4.9}$$
$$+\kappa T_{N+1}^h(t)\left(\varphi_{N+1}^h\right)'(1)\left(\varphi_i^h\right)(1) - \kappa T_0^h(t)\left(\varphi_0^h\right)'(0)\left(\varphi_i^h\right)(0)$$
$$-\kappa T_1^h(t)\left(\varphi_1^h\right)'(0)\left(\varphi_i^h\right)(0) \text{ for } i = 0, ..., N+1,$$

where

$$\varphi_0^h(x) = \begin{cases} \frac{x_1 - x}{h} & \text{for } 0 \leqslant x < x_1 \\ 0 \text{ otherwise} \end{cases}, \tag{3.4.10}$$

$$\underbrace{\varphi_i^h(x)}_{i=1..N} = \begin{cases} \frac{x - x_{i-1}}{h} & \text{for } x_{i-1} \leqslant x < x_i \\ \frac{x_{i+1} - x}{h} & \text{for } x_i \leqslant x < x_{i+1} \\ 0 \text{ otherwise} \end{cases}, \tag{3.4.11}$$

and

$$\varphi_{N+1}^h(x) = \begin{cases} \frac{x - x_N}{h} & \text{for } x_N \leqslant x < x_{N+1} \\ 0 \text{ otherwise} \end{cases}. \tag{3.4.12}$$

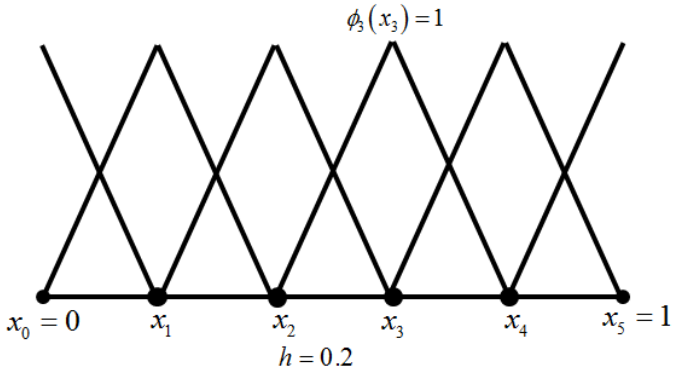

Figure 3.6: Piecewise Linear Basis Functions for 1D Heat Equation

Therefore, the right-hand side of (3.4.9) is

$$\begin{cases} -\kappa \sum_{j=i-1}^{i+1} T_j^h(t) \int_{(i-1)h}^{(i+1)h} \left(\varphi_j^h\right)'(x) \left(\varphi_i^h\right)'(x)\, dx & \text{for } i = 1, ..., N \\ -\kappa \sum_{j=N}^{N+1} T_j^h(t) \int_{1-h}^{1} \left(\varphi_j^h\right)'(x) \left(\varphi_{N+1}^h\right)'(x)\, dx - \frac{\kappa}{h} T_N^h(t) + \frac{\kappa}{h} u_1 & \text{for } i = N+1\,, \\ -\kappa \sum_{j=0}^{1} T_j^h(t) \int_{0}^{h} \left(\varphi_j^h\right)'(x) \left(\varphi_0^h\right)'(x)\, dx - \frac{\kappa}{h} T_1^h(t) + \frac{\kappa}{h} u_0 & \text{for } i = 0 \end{cases} \tag{3.4.13}$$

which equals

$$\begin{cases} \frac{\kappa}{h}\left[\hat{T}_{i-1}^h - 2\hat{T}_i^h + \hat{T}_{i+1}^h\right] \text{ for } i = 1, ..., N \\ \frac{\kappa}{h}\left(-\hat{T}_{N+1}^h\right) + \frac{\kappa}{h} u_1 \text{ for } i = N+1 \\ \frac{\kappa}{h}\left(-\hat{T}_0^h\right) + \frac{\kappa}{h} u_1 \text{ for } i = 0 \end{cases}. \tag{3.4.14}$$

Consequently,

$$\mathbf{K} = \frac{\kappa}{h} \underbrace{\begin{pmatrix} -1 & & & & & \\ 1 & -2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & -2 & 1 & \\ & & & & & -1 \end{pmatrix}}_{(N+2)\times(N+2)}; \quad \mathbf{G} = \frac{\kappa}{h} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 1 \end{pmatrix}}_{(N+2)\times 2}, \tag{3.4.15}$$

and the mass matrix has the form

$$\mathbf{M} = \frac{h}{6} \underbrace{\begin{pmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{pmatrix}}_{(N+2)\times(N+2)}. \tag{3.4.16}$$

The finite dimensional model becomes

$$\mathbf{M}\dot{\mathbf{T}}^h = \mathbf{K}\mathbf{T}^h + \mathbf{G}^h\mathbf{u}, \tag{3.4.17}$$

or since $\mathbf{M}^{-1}$ exists, this becomes

$$\dot{\mathbf{T}}^h = \mathbf{A}^h\mathbf{T}^h + \mathbf{B}^h\mathbf{u}, \tag{3.4.18}$$

where

$$\mathbf{A}^h = \mathbf{M}^{-1}\mathbf{K},\ \mathbf{B}^h = \mathbf{M}^{-1}\mathbf{G}^h \tag{3.4.19}$$

At this stage, the mass matrices $\mathbf{M}$ and the stiffness matrices $\mathbf{K}$ match perfectly between Rubio's method and COMSOL.®

Because $T_0(t)$ and $T_{N+1}(t)$ are specified as $u_0(t)$ and $u_1(t)$ respectively, this system must be reduced to remove $\dot{u}_0(t)$ and $\dot{u}_1(t)$ from the equation. Algebraically, this elimination is straight-forward. The system has the form

$$\begin{aligned}
&\underbrace{\frac{h}{6}\begin{pmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{pmatrix}}_{(N+2)\times(N+2)} \begin{pmatrix} \dot{u}_0 \\ \dot{T}_1^h \\ \vdots \\ \dot{T}_N^h \\ \dot{u}_1 \end{pmatrix} \\
&= \underbrace{\frac{\chi}{h}\begin{pmatrix} -1 & & & & & \\ 1 & -2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & -2 & 1 & \\ & & & & -1 \end{pmatrix}}_{(N+2)\times(N+2)} \begin{pmatrix} u_0 \\ T_1^h \\ \vdots \\ T_N^h \\ u_1 \end{pmatrix} + \underbrace{\frac{\chi}{h}\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 1 \end{pmatrix}}_{(N+2)\times 2} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix}
\end{aligned} \tag{3.4.20}$$

so that

$$\frac{h}{6}\left(2\dot{u}_0 + \dot{T}_1^h\right) = 0, \tag{3.4.21}$$

and

$$\frac{h}{6}\left(\dot{u}_0 + 4\dot{T}_1^h + \dot{T}_2^h\right) = \frac{\chi}{h}\left(u_0 - 2T_1^h + T_2^h\right). \tag{3.4.22}$$

Thus, we have

$$\frac{h}{6}\left(-\frac{1}{2}\dot{T}_1^h + 4\dot{T}_1^h + \dot{T}_2^h\right) = \frac{\chi}{h}\left(u_0 - 2T_1^h + T_2^h\right), \tag{3.4.23}$$

which implies

$$\frac{h}{6}\left(\left(4 - \frac{1}{2}\right)\dot{T}_1^h + \dot{T}_2^h\right) = \frac{\chi}{h}\left(-2T_1^h + T_2^h\right) + \frac{\chi}{h}u_0. \tag{3.4.24}$$

A similar substitution for u_1 produces[9]

$$\mathbf{M}_{red}\dot{\mathbf{T}}_{red}^h = \mathbf{K}_{red}\mathbf{T}_{red}^h + \mathbf{G}_{red}\mathbf{u} \tag{3.4.25}$$

[9]Here we make a small notation abuse and allow that $\mathbf{T}^h$ represent the state, less the boundary terms.

where,

$$\mathbf{M}_{red} = \frac{h}{6} \underbrace{\begin{pmatrix} 4-\frac{1}{2} & 1 & & & & \\ 1 & 4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & 4 & 1 & \\ & & & 1 & 4-\frac{1}{2} \end{pmatrix}}_{N\times N}, \tag{3.4.26}$$

$$\mathbf{K}_{red} = \frac{\chi}{h} \underbrace{\begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & -2 & 1 & \\ & & & 1 & -2 \end{pmatrix}}_{N\times N}, \ \mathbf{G}_{red} = \frac{\chi}{h} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 1 \end{pmatrix}}_{N\times 2}. \tag{3.4.27}$$

Clearly $\mathbf{M}_{red}$ can be inverted to pre-multiply equation (3.4.25) and hence we obtain

$$\dot{\mathbf{x}}^h(t) = \mathbf{A}^h\mathbf{x}^h(t) + \mathbf{B}^h\mathbf{u}(t) \tag{3.4.28}$$

Obtaining System Matrices from COMSOL®

In contrast, the COMSOL® approach is different, but arrives at nearly the same result. As described in the COMSOL® help files, COMSOL® creates its $\mathbf{A}$ and $\mathbf{B}$ matrices from the system

$$\begin{aligned} 0 &= L(\dot{\mathbf{T}}, \mathbf{T}, \mathbf{u}) - \left(\frac{\partial G}{\partial T}\right)^T \Lambda \\ 0 &= G(\mathbf{T}), \end{aligned} \tag{3.4.29}$$

where G, in this case, represents boundary constraints, L is the residual from the last computed solution (or initial conditions) $\mathbf{T}_0$ and Λ is a vector of Lagrange multipliers. The control input, $\mathbf{u}$, is included in the system because COMSOL® provides for the ability to specify variable parameters to automatically compute multiple solutions with varied parameters or to generate a $\mathbf{B}$ matrix in the standard-form state system of equations (which will be discussed later).

In the typical Dirichlet boundary case where $G(\mathbf{T}) \triangleq \mathbf{NT} - g\left(\mathbf{x}_i^h, t\right),\ \mathbf{x}_i^h \in \mathbf{X}_{\Gamma_{TE}}^h$, the system is

$$\begin{aligned} 0 &= L(\dot{\mathbf{T}}, \mathbf{T}, \mathbf{u}) - \mathbf{N}^T\Lambda \\ 0 &= \mathbf{NT} - g\left(\mathbf{x}_i^h, t\right),\ \mathbf{x}_i^h \in \mathbf{X}_{\Gamma_{TE}}^h. \end{aligned} \tag{3.4.30}$$

To solve the system, COMSOL® linearizes L about $\mathbf{T}_0, \dot{\mathbf{T}}_0$ to obtain

$$\begin{aligned} 0 = \left.\frac{\partial L}{\partial \dot{\mathbf{T}}}\right|_{\mathbf{T}=\mathbf{T}_0} \left(\dot{\mathbf{T}} - \dot{\mathbf{T}}_0\right) + \left.\frac{\partial L}{\partial \dot{\mathbf{T}}}\right|_{\mathbf{T}=\mathbf{T}_0} (\mathbf{T} - \mathbf{T}_0) - \mathbf{N}^T \Lambda + L(\mathbf{T}_0, \mathbf{u}_0) \\ 0 = \mathbf{N}(\mathbf{T} - \mathbf{T}_0) - g\left(\mathbf{x}_i^h, 0\right). \end{aligned} \tag{3.4.31}$$

so that $-\frac{\partial L}{\partial \dot{\mathbf{T}}}$ is the mass matrix and $\frac{\partial L}{\partial \mathbf{T}}$ is the stiffness matrix. As expected, in the case of the already-linear 1D heat equation, these matrices coincide perfectly with the mass and stiffness matrices generated by Rubio. However, COMSOL® deals with specified boundary nodal values (both controlled and constant) quite differently when reducing these matrices to coincide with the true degrees of freedom.

The default method COMSOL® uses to solve this linearized system is to do a change of variables on $\mathbf{T}$,

$$\mathbf{T} \triangleq Null\hat{\mathbf{T}} + \mathbf{T}_{part}, \tag{3.4.32}$$

where $Null$ is a matrix whose columns form an orthogonal basis of the null space of $\mathbf{N}$, and $\mathbf{T}_{part}$ is a vector such that $\mathbf{NT_{part}} = g\left(\mathbf{x}_i^h, t\right)$.[10] Typically all this means is that $\mathbf{T}_{part} \triangleq g\left(\mathbf{x}_i^h, t\right)$. Equation (3.4.32) allows for the constraint portion of the system, along with the lagrange multipliers, to be eliminated without solving for them and generates what COMSOL® designates as "eliminated" mass and stiffness matrices:

$$K_e = Null^T K\ Null \tag{3.4.33}$$

$$M_e = Null^T M\ Null. \tag{3.4.34}$$

$$\tag{3.4.35}$$

These matrices are very similar to those produced using the direct algebraic techniques designated by Rubio, however they provide slightly different mass matrices. Using the COMSOL® method on the matrices from (3.4.16)-(3.4.15), one first obtains the COMSOL®-generated null matrix, which is simply all the column vectors corresponding to internal nodes,

$$Null = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.4.36}$$

[10] It is important to note that for the implementation used in this thesis, even though we may have the case that $g\left(\mathbf{x}_i^h, t\right) = uh\left(\mathbf{x}_i^h\right)$, COMSOL® will only see this as a numerical value and not an input parameter.

Therefore, the "eliminated" mass matrix, $\mathbf{M}_e$, is computed to be

$$\mathbf{M}_e = Null^T \mathbf{M} Null = \frac{1}{30} \begin{pmatrix} 4 & 1 & & \\ 1 & 4 & 1 & \\ & 1 & 4 & 1 \\ & & 1 & 4 \end{pmatrix} \begin{pmatrix} \dot{T}_1 \\ \dot{T}_2 \\ \dot{T}_3 \\ \dot{T}_4 \end{pmatrix}, \tag{3.4.37}$$

which is simply $\mathbf{M}$ with the boundary-node-associated rows and columns removed. It is also the mass matrix one would obtain using a finite element method without control if the test functions in the weak form were exclusively elements of $H_0^1(\Omega)$. It differs from $\mathbf{M}_{red}$ on the order of the mesh size h.

The "eliminated" stiffness matrix, $\mathbf{K}_e$ is also formed by eliminating the boundary rows and columns of the full stiffness matrix $\mathbf{K}$, which matches Rubio's direct computations. The control input operator, designated "MB" in COMSOL® , is numerically computed by COMSOL® , from the nonlinear residual vector $L\left(\dot{\mathbf{T}}, \mathbf{T}, \mathbf{u}\right)$ as

$$MB = \frac{\partial L}{\partial \mathbf{u}}, \tag{3.4.38}$$

computed as a finite difference, which also perfectly coincides with $\mathbf{G}_{red}$ for this simple test case.

Despite the mass matrix discrepancy, Rubio [44] proved that $\left\|\mathbf{M}_e^{-1}\mathbf{K}_e\mathbf{u} - \mathbf{M}_{red}^{-1}\mathbf{K}_{red}\mathbf{u}\right\|$ and $\left\|\mathbf{M}_e^{-1}(MB)\,\mathbf{u} - \mathbf{M}_{red}^{-1}\mathbf{G}_{red}\mathbf{u}\right\|$ approach zero as mesh size decreases, which indicates that the COMSOL® matrix creation scheme is adequate to obtain good results for the LQR problem.

3.4.3 Linearized, Discretized Boussinesq System

In the case of the discretized Boussinesq system, we linearize about $(\mathbf{V}_0, \mathbf{T}_0, \mathbf{P}_0)^T$. Let

$$\begin{pmatrix} \tilde{\mathbf{V}}_0 \\ \tilde{\mathbf{T}}_0 \\ \tilde{\mathbf{P}}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{V} - \mathbf{V}_0 \\ \mathbf{T} - \mathbf{T}_0 \\ \mathbf{P} - \mathbf{P}_0 \end{pmatrix} \tag{3.4.39}$$

Then, taking a linear Taylor approximation of the system expressed in (3.3.23) we have the following:

$$
\begin{pmatrix} \mathbf{M} & & & \\ & \mathbf{M} & & \\ & & \mathbf{M}_T & \\ & & & 0 \end{pmatrix} \begin{pmatrix} \dot{\tilde{\mathbf{V}}}_1(t) \\ \dot{\tilde{\mathbf{V}}}_2(t) \\ \dot{\tilde{\mathbf{T}}}(t) \\ \dot{\tilde{\mathbf{P}}}(t) \end{pmatrix} =
$$

$$
- \begin{pmatrix} \tilde{\mathbf{C}}_1(\mathbf{V}_0) + 2\mathbf{K}_{xx} + \mathbf{K}_{yy} & \mathbf{K}_{yx} & 0 & -\mathbf{Q}_x \\ \mathbf{K}_{xy} & \tilde{\mathbf{C}}_2(\mathbf{V}_0) + \mathbf{K}_{xx} + 2\mathbf{K}_{yy} & -\mathbf{B} & -\mathbf{Q}_y \\ \mathbf{D}_1 & \mathbf{D}_2 & \tilde{\mathbf{D}} + \mathbf{L} & 0 \\ (-\mathbf{Q}_x)^T & (-\mathbf{Q}_y)^T & 0 & 0 \end{pmatrix} \begin{pmatrix} \tilde{\mathbf{V}}_1(t) \\ \tilde{\mathbf{V}}_2(t) \\ \tilde{\mathbf{T}}(t) \\ \tilde{\mathbf{P}}(t) \end{pmatrix}
$$

$$
+ \begin{pmatrix} (\mathbf{N}_{\mathbf{v}^h{}_{11}})^T & 0 & 0 & 0 \\ 0 & (\mathbf{N}_{\mathbf{v}^h{}_{22}})^T & 0 & 0 \\ 0 & 0 & (\mathbf{N}_{T^h})^T & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{\Lambda}_1(t) \\ \mathbf{\Lambda}_2(t) \\ \mathbf{\Lambda}_T(t) \\ 0 \end{pmatrix}. \tag{3.4.40}
$$

Here we have five new matrices that are linearizations about $\left(\tilde{\mathbf{V}}_0, \tilde{\mathbf{T}}_0, \tilde{\mathbf{P}}_0\right)^T$ for the convective terms. In the case of zero steady state, these are all zero. The components are given by

$$
\tilde{\mathbf{C}}_1(\mathbf{V}_0) = \int_\Omega \Psi^h (\mathbf{V}_{01})^T \left[\begin{array}{c} \left(\Psi^h \frac{\partial \left(\Psi^h\right)^T}{\partial x} + \frac{\partial \left(\Psi^h\right)}{\partial x} \left(\Psi^h\right)^T \right) \\ + \left(\left(\Psi^h\right)^T \mathbf{V}_{02} \right) \frac{\partial \left(\Psi^h\right)^T}{\partial y} \end{array} \right] d\mathbf{x}, \tag{3.4.41}
$$

$$
\tilde{\mathbf{C}}_2(\mathbf{V}_0) = \int_\Omega \Psi^h (\mathbf{V}_{02})^T \left[\begin{array}{c} \left(\Psi^h \frac{\partial \left(\Psi^h\right)^T}{\partial y} + \frac{\partial \left(\Psi^h\right)}{\partial y} \left(\Psi^h\right)^T \right) \\ + \left(\left(\Psi^h\right)^T \mathbf{V}_{01} \right) \frac{\partial \left(\Psi^h\right)^T}{\partial x} \end{array} \right] d\mathbf{x}, \tag{3.4.42}
$$

$$
\tilde{\mathbf{D}} = \int_\Omega \Phi^h \left[\underset{1\times N_{\mathcal{P}\mathcal{T}^h}}{\left((\mathbf{V}_{01})^T \left(\Psi^h\right) \right) \left(\frac{\partial \Phi^h}{\partial x} \right)^T} + \underset{1\times N_{\mathcal{P}\mathcal{T}^h}}{\left((\mathbf{V}_{02})^T \left(\Psi^h\right) \right) \left(\frac{\partial \Phi^h}{\partial y} \right)^T} \right] d\mathbf{x}, \tag{3.4.43}
$$

$$\mathbf{D}_1 = \int_{\Omega} \left[\Phi^h \Psi^h \underset{1\times N_{\mathcal{PT}^h}}{\left(\frac{\partial \Phi^h}{\partial x} \right)^T} \mathbf{T}_0 \right] d\mathbf{x},\ \mathbf{D}_2 = \int_{\Omega} \left[\Phi^h \Psi^h \underset{1\times N_{\mathcal{PT}^h}}{\left(\frac{\partial \Phi^h}{\partial y} \right)^T} \mathbf{T}_0 \right] d\mathbf{x}. \tag{3.4.44}$$

Again, like with the heat equation, we define $\mathbf{N}_{\mathbf{v}^h}$ as

$$\begin{pmatrix} \mathbf{N}_{\mathbf{v}^h{}_{11}} & 0 \\ 0 & \mathbf{N}_{\mathbf{v}^h{}_{22}} \end{pmatrix}, \tag{3.4.45}$$

and $Null_{\mathbf{v}^h}, Null_{T^h}$ as the matrices who's columns form the null space of $\mathbf{N}_{\mathbf{v}^h}, \mathbf{N}_{T^h}$ respectively. Also, define

$$\mathbf{M}_{\mathbf{v}^h} \triangleq \begin{pmatrix} \mathbf{M} & 0 \\ 0 & \mathbf{M} \end{pmatrix}, \tag{3.4.46}$$

$$\mathbf{K}_{\mathbf{v}^h} \triangleq -\begin{pmatrix} \tilde{\mathbf{C}}_1(\mathbf{V}_0) + 2\mathbf{K}_{xx} + \mathbf{K}_{yy} & \mathbf{K}_{yx} \\ \mathbf{K}_{xy} & \tilde{\mathbf{C}}_2(\mathbf{V}_0) + \mathbf{K}_{xx} + 2\mathbf{K}_{yy} \end{pmatrix}, \tag{3.4.47}$$

and

$$\mathbf{K}_{T^h} = -\left(\tilde{\mathbf{D}} + \mathbf{L} \right). \tag{3.4.48}$$

Finally, we define $\mathbf{Q}$ by

$$\begin{pmatrix} -\mathbf{Q}_x \\ -\mathbf{Q}_y \end{pmatrix}. \tag{3.4.49}$$

which yields

$$\begin{pmatrix} \mathbf{M}_{\mathbf{v}^h} & & \\ & \mathbf{M}_T & \\ & & 0 \end{pmatrix} \begin{pmatrix} \dot{\tilde{\mathbf{V}}}(t) \\ \dot{\tilde{\mathbf{T}}}(t) \\ \dot{\tilde{\mathbf{P}}}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{K}_{\mathbf{v}^h} & \mathbf{B}_{\mathbf{v}^h} & \mathbf{Q} \\ \mathbf{D}_l & \mathbf{K}_{T^h} & 0 \\ \mathbf{Q}^T & 0 & 0 \end{pmatrix} \begin{pmatrix} \tilde{\mathbf{V}}(t) \\ \tilde{\mathbf{T}}(t) \\ \tilde{\mathbf{P}}(t) \end{pmatrix} + \begin{pmatrix} (\mathbf{N}_{\mathbf{v}^h})^T & 0 & 0 \\ 0 & (\mathbf{N}_{T^h})^T & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{\Lambda}_1(t) \\ \mathbf{\Lambda}_T(t) \\ 0 \end{pmatrix}, \tag{3.4.50}$$

where

$$\mathbf{B}_{\mathbf{v}^h} = \begin{pmatrix} 0 \\ \mathbf{B} \end{pmatrix}. \tag{3.4.51}$$

Again, a change of variables and differentiation of the nonlinear system with respect to $\mathbf{u}^h$ provides

$$\begin{pmatrix} \hat{\mathbf{M}}_{\mathbf{v}^h} & & \\ & \hat{\mathbf{M}}_T & \\ & & 0 \end{pmatrix} \begin{pmatrix} \dot{\hat{\mathbf{V}}}(t) \\ \dot{\hat{\mathbf{T}}}(t) \\ \dot{\hat{\mathbf{P}}}(t) \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{K}}_{\mathbf{v}^h} & \hat{\mathbf{B}}_{\mathbf{v}^h} & \hat{\mathbf{Q}} \\ \hat{\mathbf{D}}_l & \hat{\mathbf{K}}_{T^h} & 0 \\ \hat{\mathbf{Q}}^T & 0 & 0 \end{pmatrix} \begin{pmatrix} \hat{\mathbf{V}}(t) \\ \hat{\mathbf{T}}(t) \\ \hat{\mathbf{P}}(t) \end{pmatrix} + \mathbf{B}\mathbf{u}^h \tag{3.4.52}$$

where

$$\begin{aligned} &\begin{pmatrix} \hat{\mathbf{M}}_{\mathbf{v}^h} & & \\ & \hat{\mathbf{M}}_T & \\ & & 0 \end{pmatrix} \\ &= \begin{pmatrix} Null_{\mathbf{v}^h} & & \\ & Null_{T^h} & \\ & & 0 \end{pmatrix}^T \begin{pmatrix} \mathbf{M}_{\mathbf{v}^h} & & \\ & \mathbf{M}_T & \\ & & 0 \end{pmatrix} \begin{pmatrix} Null_{\mathbf{v}^h} & & \\ & Null_{T^h} & \\ & & 0 \end{pmatrix}, \end{aligned} \tag{3.4.53}$$

and

$$\begin{aligned} &\begin{pmatrix} \hat{\mathbf{K}}_{\mathbf{v}^h} & \hat{\mathbf{B}}_{\mathbf{v}^h} & \hat{\mathbf{Q}} \\ 0 & \hat{\mathbf{K}}_{T^h} & 0 \\ \hat{\mathbf{Q}}^T & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} Null_{\mathbf{v}^h} & & \\ & Null_{T^h} & \\ & & 0 \end{pmatrix}^T \begin{pmatrix} \mathbf{K}_{\mathbf{v}^h} & \mathbf{B}_{\mathbf{v}^h} & \mathbf{Q} \\ 0 & \mathbf{K}_{T^h} & 0 \\ \mathbf{Q}^T & 0 & 0 \end{pmatrix} \begin{pmatrix} Null_{\mathbf{v}^h} & & \\ & Null_{T^h} & \\ & & 0 \end{pmatrix}. \end{aligned} \tag{3.4.54}$$

3.4.4 Elimination of the Pressure Term

Considering the singular Mass matrix, Normally, COMSOL®, would solve its eliminated system (including eliminated load vector) as a differential algebraic equation (DAE), however,

one can eliminate the pressure term in the same way COMSOL® eliminates the boundary constraints because the pressure term is really just a Lagrange multiplier associated with the incompressibility constraint in the Boussinesq equations. After this elimination, construction of the state equations from Section 3.4.1 is straight forward. Indeed, Gunzburger [20], in reference to the Navier-Stokes equations by themselves, describes how, in the steady-state case, the discrete linearized system can be written as

$$\begin{pmatrix} \mathbf{K} & \mathbf{N^T} \\ \mathbf{N} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{V} \\ \mathbf{P} \end{pmatrix} = \begin{pmatrix} \mathbf{F} \\ \mathbf{0} \end{pmatrix}, \tag{3.4.55}$$

so that by setting $\mathbf{V} = Null\hat{\mathbf{V}}$, where $Null$ is the matrix consisting of orthogonal vectors that that form the right null space of $\mathbf{N}$, one obtains the system,

$$\left(Null^T \mathbf{K} Null\right) \hat{\mathbf{V}} = Null^T \mathbf{F}. \tag{3.4.56}$$

This concept easily translates to the time-dependent Boussinesq system as well and it can be implemented in MATLAB® using the "null" command on the divergence free constraint matrix, which amounts to computing a singular value decomposition. Unfortunately, this "null" function is extremely slow in MATLAB® , making the process of setting up the preliminaries to solving an "ARE" much more difficult than they need to be. Therefore, we implement an approximation to (3.4.55), also suggested by Gunzburger, that can provide the same result with errors on a similar scale to discretization error. This technique is known as the "penalty method", where (3.4.55) is approximated by

$$\begin{pmatrix} \mathbf{K} & \mathbf{N^T} \\ \mathbf{N} & -\varepsilon \mathbf{M}_p \end{pmatrix} \begin{pmatrix} \mathbf{V} \\ \mathbf{P} \end{pmatrix} = \begin{pmatrix} \mathbf{F} \\ \mathbf{0} \end{pmatrix}, \tag{3.4.57}$$

and ε is on the order of the discretization error associated with the system.[11] Equation(3.4.57) implies that the pressure can then be expressed as

$$\mathbf{P} = \frac{1}{\varepsilon} \mathbf{M}_p^{-1} \mathbf{N} \mathbf{V}, \tag{3.4.58}$$

so that the nonsingular pressure eliminated system is

$$\left(\mathbf{K} + \frac{1}{\varepsilon} \mathbf{N^T} \mathbf{M}_p^{-1} \mathbf{N}\right) \mathbf{V} = \mathbf{F}. \tag{3.4.59}$$

It is straight forward to see that (3.4.59) applies to the time-dependent linearized Boussinesq system as well.

[11] h^2 in our case, where h represents the mesh size [20].

Chapter 4

Numerical Experiments

In this chapter we present the results from several numerical experiments. We begin with a 3D problem to illustrate how one uses COMSOL® to compute steady state solutions and conduct a numerical study of the system's response to a disturbance.

4.1 3D Example

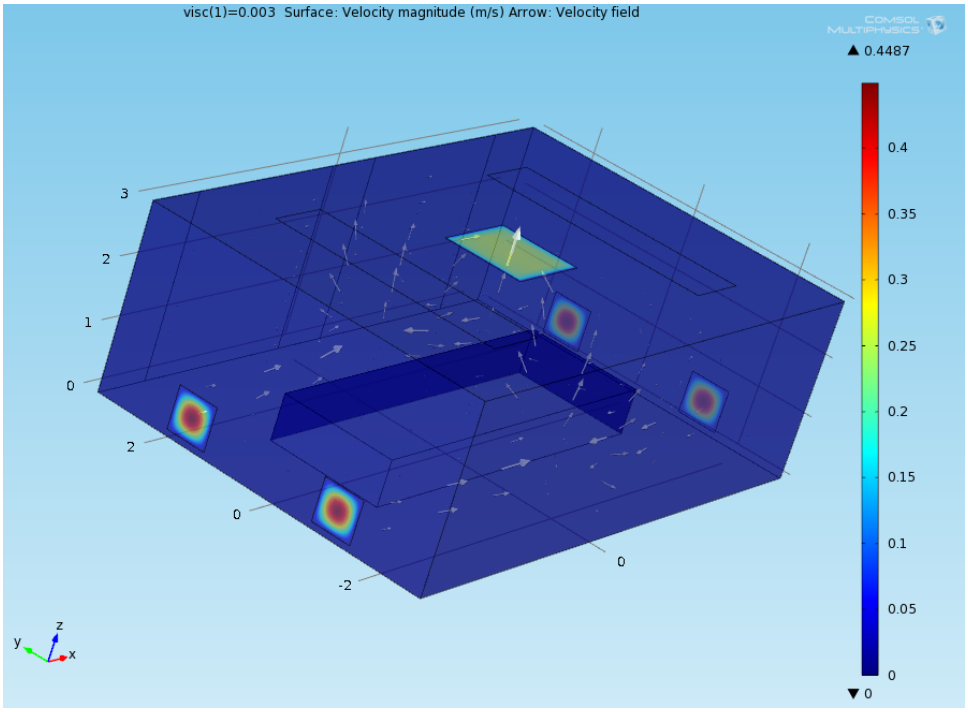

Figure 4.1: Stationary Velocity in Simple Room

Name	Description	Value
ρ_0	Density	$1.18\frac{\text{kg}}{\text{m}^3}$
g	|Gravitational Acceleration|	$9.81\frac{m}{s^2}$
μ	Viscosity	$0.003\ \frac{\text{kg}}{\text{ms}}$
α_T	Coef. of Thermal Expansion	$3.38 \times 10^{-3}\ \frac{1}{\text{K}}$
T_0	Room Temperature	293.15°K
κ	Thermal Diffusivity	$2.08 \times 10^{-5}\frac{\text{m}^2}{\text{s}}$
C_p	Specific Heat	$1006\ \frac{\text{J}}{\text{kg K}}$
k	Thermal Conductivity	$0.0257\frac{\text{W}}{\text{m K}}$

Table 4.1: 3D Example Coefficient Values

In this example, we focus on the room shown in Figure 4.1. To begin, we compute an "ideal" (steady-state) air flow, with uniform temperature, beginning with 0-velocity initial conditions, and parabolic-type inflow. If one initially specifies that $\mu = 0.003$, the nonlinear solver fails to converge to a steady-state solution. However, by beginning with a time dependent model, we can first run the model with $\mu = 0.1$ until it appears to be near a steady-state solution. We can then use this solution as the initial condition for a stationary problem at the same viscosity. COMSOL® can then be set to successively find a new stationary solution at slightly lower viscosity using the previous solution for initial conditions. Figure 4.1 is a depiction of the steady state velocity profile. Here we readily observe the symmetric behavior of the flow field as well as the air's tendency to channel away from boundaries (which we would expect).

Next, we investigate the impact of including heat source disturbances. We use the steady-state solution as an initial condition and apply heat flux on the inner boundary (representing a conference table with people around it). We also apply heat flux to other portions of the boundary representing ceiling lights and windows. We then use COMSOL® to produce a time dependent solution for $0 \leqslant t \leqslant 40$ seconds (See Figure 4.2).

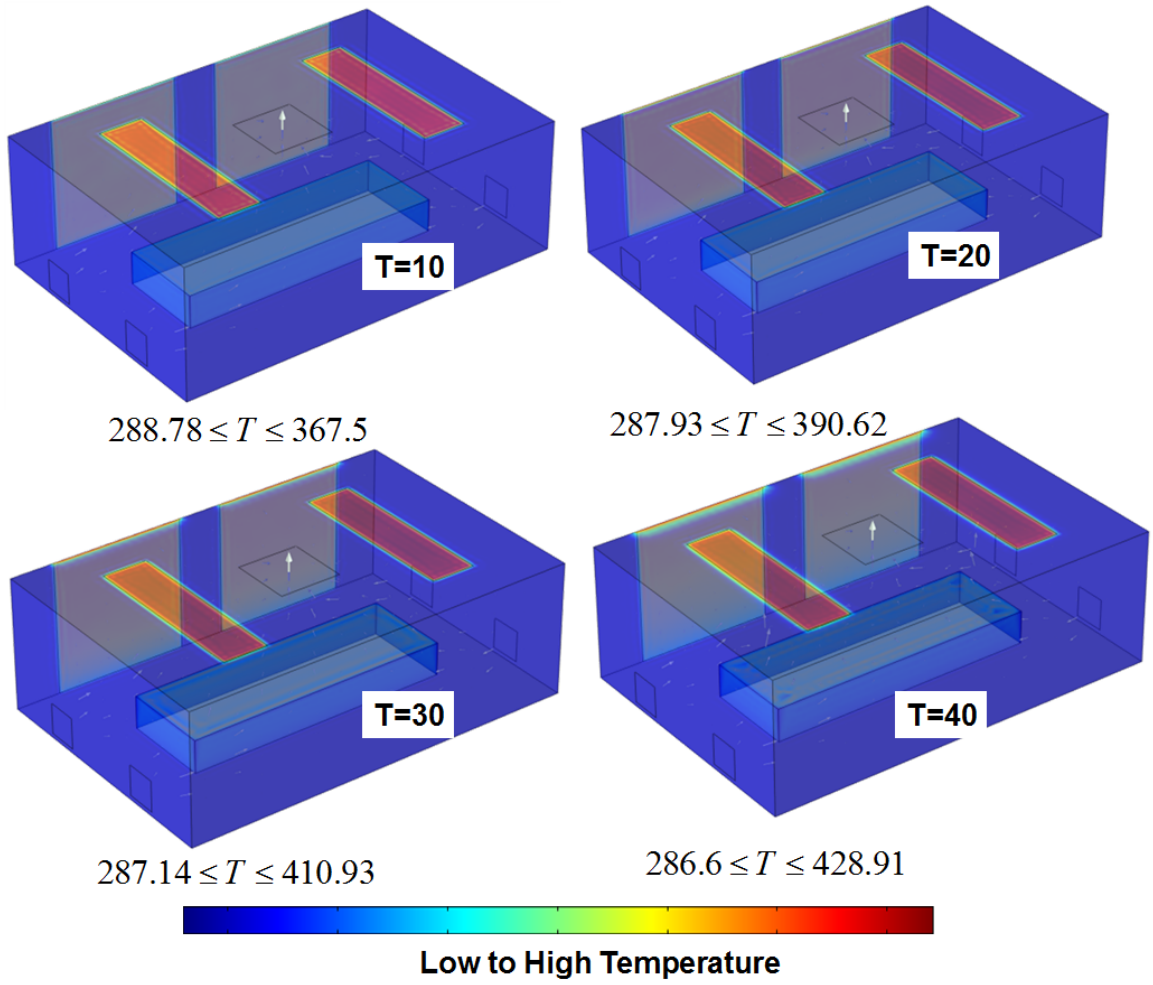

Figure 4.2: Time-Dependent Solutions for $0 \leqslant t \leqslant 40$ seconds

Notice that the boundary disturbances seem to be generating unrealistically high temperatures compared to what we would normally detect on windows, lights, and people. A more realistic way to model these sources would be to specify the temperatures (either constant or time-dependent in the case of the windows) instead of heat flux. Secondly, the nonlinear solver for the time-dependent model under these conditions and meshing (See Figure 4.3) failed to converge to a solution at t=55 seconds.

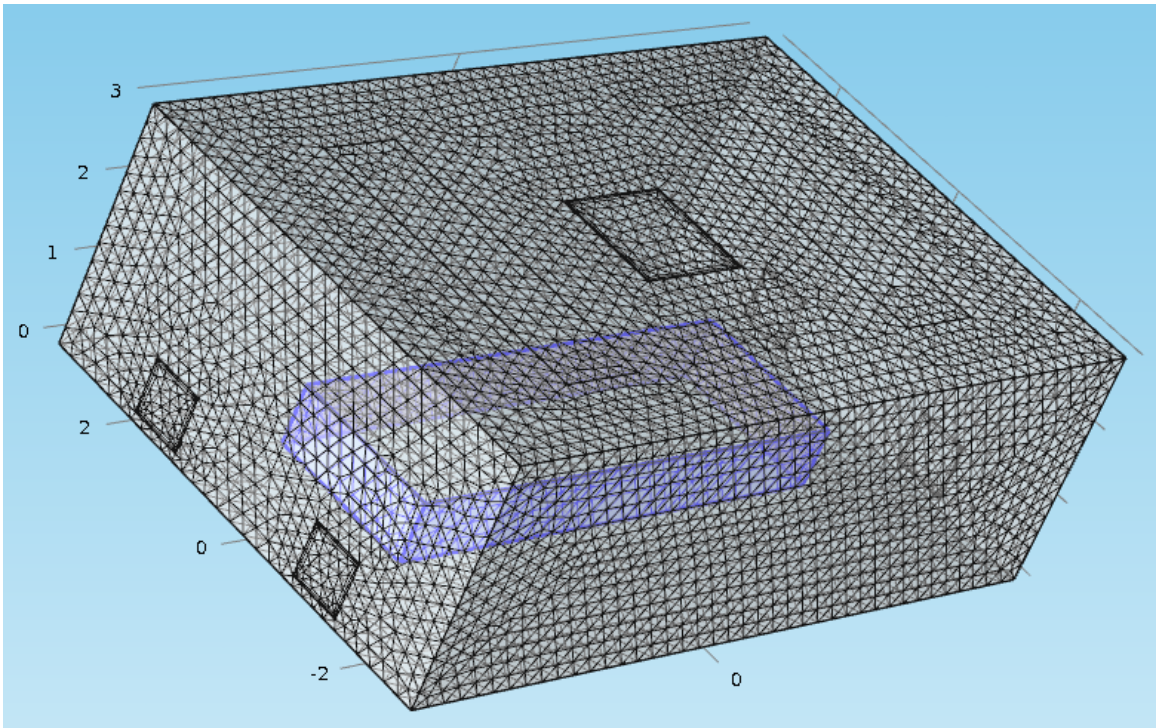

Figure 4.3: 3D Mesh for Example 3D Problem

4.2 Control of the 2D Heat Equation

Here we first focus on control of the 2D heat equation to illustrate the approach as outlined below.

Step 1

Set up the problem in COMSOL® and build the mass, stiffness, and control input matrices so that the linearized control system

$$\dot{\mathbf{x}}^h(t) = \mathbf{A}^h\mathbf{x}^h(t) + \mathbf{B}^h\mathbf{u}(t) \tag{4.2.1}$$

can be constructed. We use the COMSOL® interface "mphstate" to import $\mathbf{A}^h$ and $\mathbf{B}^h$ into MATLAB® .

Step 2

Use the MATLAB® control toolbox to produce a feedback law. Here we focus on LQR control, but once Step 1 is complete, we can use any controller design available in MATLAB.® In the LQR design case, we obtain

$$\mathbf{u}^{h*}(t) = -\mathbf{K}^h\mathbf{x}^h(t), \tag{4.2.2}$$

where $\mathbf{K}^h = \left(\mathbf{B}^h\right)^T \mathbf{P}^h$ is the LQR feedback gain matrix and

$$\left(\mathbf{A}^h\right)^T \mathbf{P}^h + \mathbf{P}^h \mathbf{A}^h - \mathbf{P}^h \mathbf{B}^h \mathbf{R}^{-1} \left(\mathbf{B}^h\right)^T \mathbf{P}^h + \mathbf{Q}^h = 0. \tag{4.2.3}$$

Step 3

Import the gain matrix, $\mathbf{K}^h$ back into COMSOL® and use COMSOL® to approximate the functional gains by interpolation of the data contained in $\mathbf{K}^h$ to produce an approximate functional gain $k^h(\mathbf{x})$.

Step 4

Use the COMSOL® integration ability to simulate the closed-loop system with non-local boundary condition

$$T(t, \mathbf{x})|_{\Gamma_{TE}} = -\int_{\Omega} k^h(\mathbf{x})\, T(t, \mathbf{x})\, dx, \tag{4.2.4}$$

to validate the control law.

This basic procedure will also be followed for the full Boussinesq system in the next section.

4.2.1 A Heat Equation Example

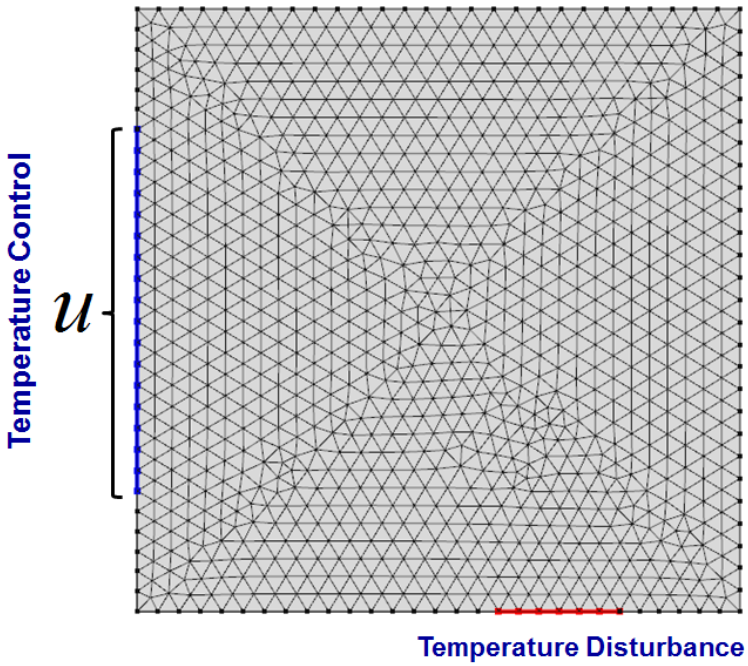

Figure 4.4: Boundary and Mesh

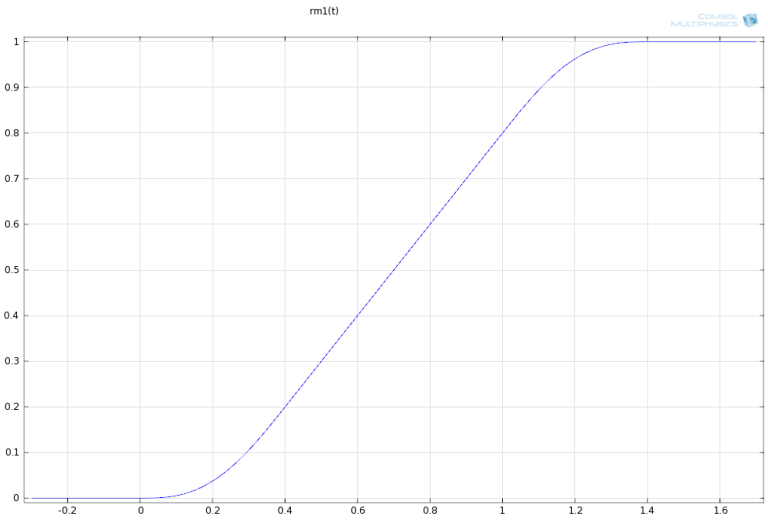

Figure 4.5: Ramp Function

As an initial proof of concept, we begin with a simple example of a diffusion equation model. For simplicity, we assume that the ideal temperature in the unit-square domain Ω is $0°K$. We also assume $\kappa = 1$.

$$\dot{T}(t, \mathbf{x}) = \Delta T(t, \mathbf{x}) \tag{4.2.5}$$

Figure 4.4 provides the associated mesh and boundary conditions.

We run the time-dependent model for 300 seconds and use COMSOL's built-in ramp function to smoothly build up a temperature disturbance from $0°K$ to a constant $2°K$ in approximately 1.4 seconds (see Figure 4.5). This ramp function emulates the idea that one would generally expect temperature on the boundary to increase smoothly and is a numerical artifact that improves convergence. It also helps to avoid numerical pitfalls associated with inconsistent boundary and initial conditions.

In order to test the ability (within COMSOL®) to include non-local feedback, we also use a feedback law of the form $u = -\gamma \int_{\Omega} T(t, \mathbf{x})\, d\mathbf{x}$. To simulate the closed-loop response in COMSOL® , we make use of the built-in "Integration" operator (See Figure 4.6). We create this operator by navigating through the "Model Builder" in COMSOL® to right-click on "Model 1"→ "Definitions" → "Model Couplings" → "Integration" and then specifying the integration "Domain", "Integration order", and "Operator Name" in the corresponding window (See Figure 4.6). This operator is then applied on the controlled portion of the boundary (See Figure 4.7).

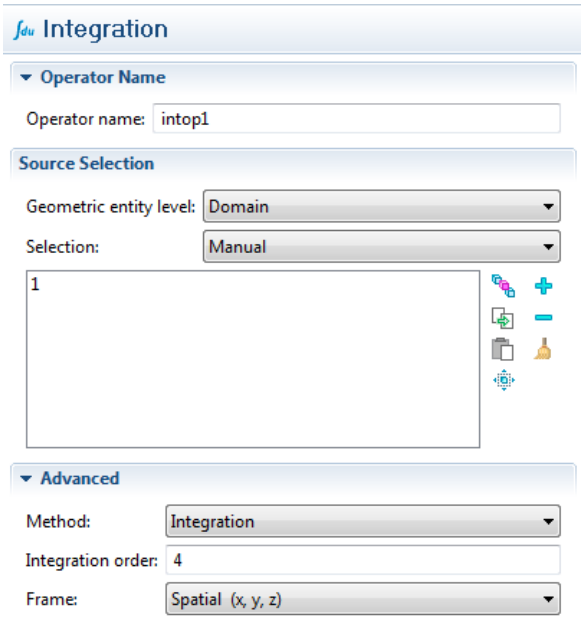

Figure 4.6: COMSOL® Integration Function Setup

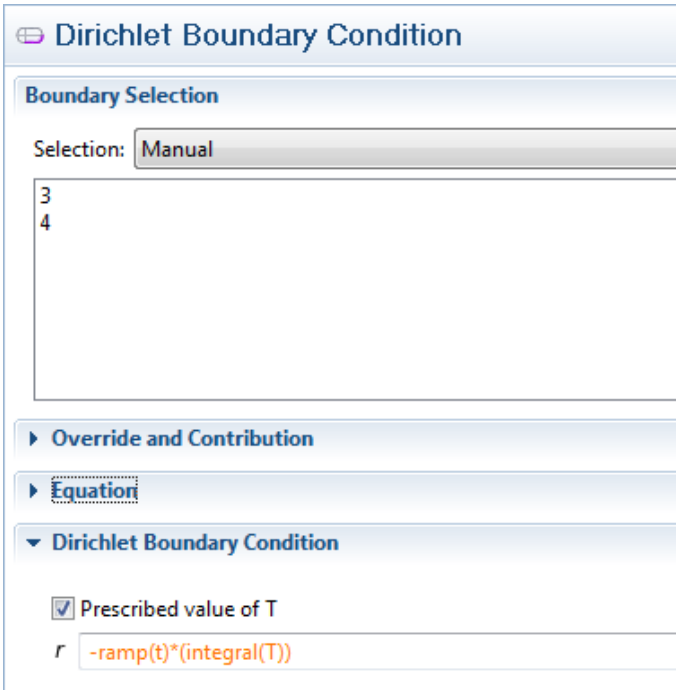

Figure 4.7: $u = -\int_{\Omega} T(t, \mathbf{x})\, d\mathbf{x}$ as Specified in COMSOL®

Figure 4.8 displays a side-by-side of the final-time temperature state with average tem-

perature (integral) feedback control versus no control. Here, we set $\gamma = 1$ and see that this control enhances stability. This example demonstrates how COMSOL® has the capability to integrate, at each time step, over space to simulate a closed-loop system. This same method allow us to specify any functional gain that can be imported into COMSOL® as part of a closed-loop control.

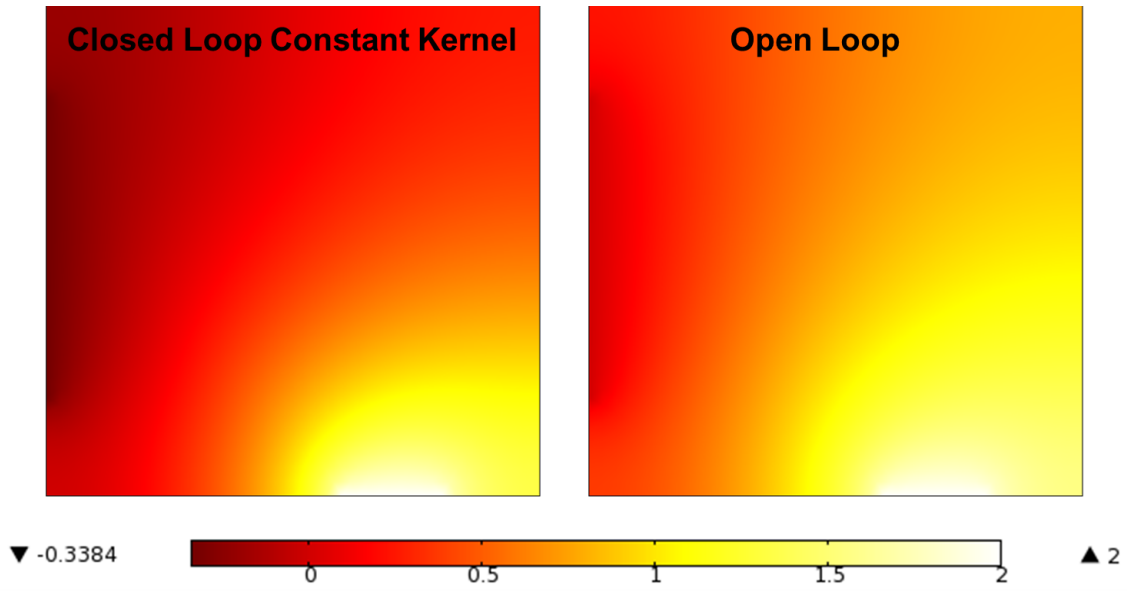

Figure 4.8: Comparison: Closed Loop with Constant Kernel versus Open Loop $u = 0$ after 300 Seconds

4.2.2 LQR Boundary Control for the 2D Heat Equation (Including MATLAB®)

To illustrate the potential benefits of a COMSOL® -MATLAB® method, we first apply the idea to the 2D heat equation (4.2.5). First we set up the FEM model in COMSOL® with appropriate settings to extract information we need later in MATLAB®. We then import the system generated by COMSOL® (i.e. mass, stiffness, B matrix, etc.) into MATLAB® for controller design. This process produces a finite-dimensional, linearized state-space system. Next, we use these matrices to solve an ARE (see (3.4.6)) and determine an optimal LQR feedback law. The feedback gain matrix provides information to approximate the feedback functional gains. We translate this gain matrix into the nodal values of a kernel function that is interpolated in COMSOL® to define the feedback for $\mathbf{u}^{h*}(t) = -\int_{\Omega} k^h(\mathbf{x}) T(t, \mathbf{x}) dx$.

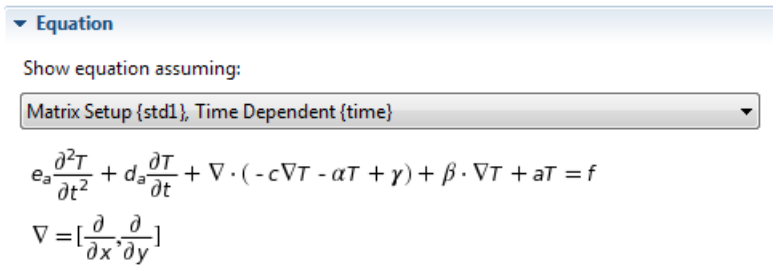

Figure 4.9: COMSOL® Coefficient-Form PDE

As a first step we ignore the disturbance (see Figure 4.10). Also, rather than use COMSOL's physics interface, we set up the model as a dimensionless coefficient-form PDE (see Figure 4.9) where $d_a = c = 1$ and $e_a = \alpha = \gamma = \beta = a = f = 0$. Thus, the system we solve is

$$\dot{T}(t, \mathbf{x}) = \Delta T(t, \mathbf{x}), \tag{4.2.6}$$

$$T(t, 0) = 5, \tag{4.2.7}$$

$$T(t, 0)|_{\Gamma_{TE}} = u, \tag{4.2.8}$$

$$\mathbf{n} \cdot \nabla T(t, 0)|_{\Gamma/\Gamma_{TE}} = 0. \tag{4.2.9}$$

with the intent that the temperature dissipates to zero.[1]

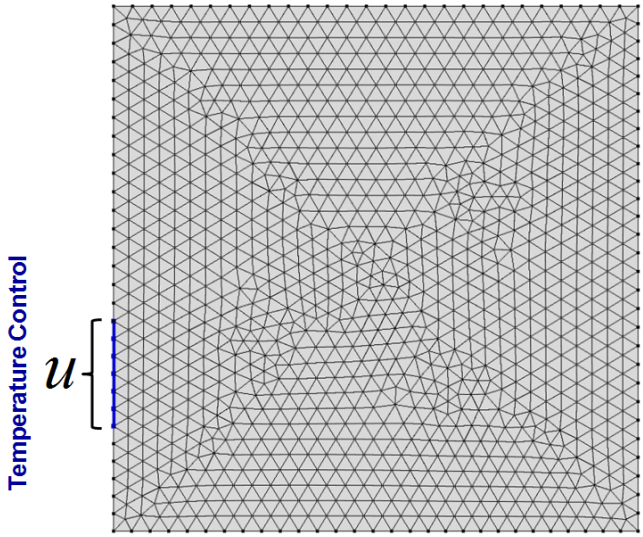

Figure 4.10: Dimensionless 2D Domain

[1]Note that we slightly change the extent of boundary control for this scenario to provide the framework for adding velocity field dynamics later on.

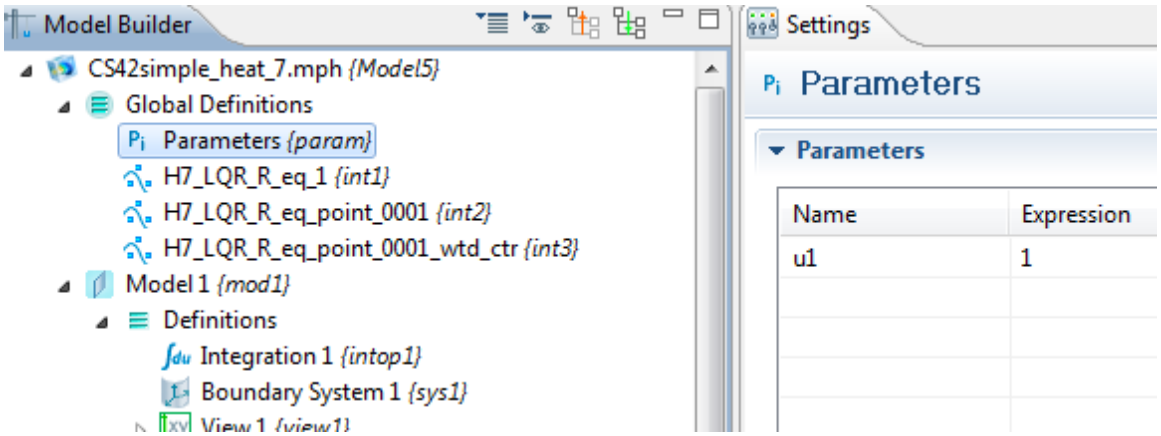

Figure 4.11: Comsol® Parameter Setup Snapshot

In order to use Comsol® to generate the system matrices $\mathbf{A}^h$ and $\mathbf{B}^h$ needed in Matlab®, one must prescribe certain parameters in Comsol.® First, one creates a control parameter under 'Global Definitions' (see Figure 4.11). The control parameter should be designated as the value it would take on at the ideal steady state. This parameter is specified so that Comsol® will create the B^h matrix for the state space system. If the discrete system solved by Comsol® is in the form (see (3.4.29)-(3.4.38))

$$\begin{aligned} &\mathbf{F}\left(\mathbf{U},\dot{\mathbf{U}},t,\mathbf{u}\right)-\mathbf{N}(\mathbf{U},t,\mathbf{u})^T\Lambda=0,\\ &\underbrace{\mathbf{C}(\mathbf{U},t,\mathbf{u})=0}_{\text{Constraints}},\ \ \mathbf{N}\left(\mathbf{U},t,\mathbf{u}\right)=\frac{\partial\mathbf{C}(\mathbf{U},t,\mathbf{u})}{\partial\mathbf{U}}, \end{aligned} \tag{4.2.10}$$

then Comsol® calculates the operator on the control input (designated as $\mathbf{MB}$) as

$$\mathbf{MB}=\frac{\partial\mathbf{F}\left(U,\dot{U},t,\mathbf{u}\right)}{\partial\mathbf{u}}=\mathbf{G}^h. \tag{4.2.11}$$

If $\mathbf{M}$ is nonsingular, Comsol® can calculate $\mathbf{B}^h=\mathbf{M}^{-1}\mathbf{G}^h$ directly. If $\mathbf{M}$ is singular, as is the case for the incompressible Navier-Stokes equations, then the state space system must be manually modified to eliminate the divergence-free constraints.

Because Comsol® performs automatic rescaling to better condition linear systems, this capability must be set to "None" to obtain the proper (physically meaningful) state space matrices for manipulation in Matlab®. The actual solver must also be disabled so that the state space system remains the original linearized discrete system before Comsol® begins its iterative solver.

After saving the model in Comsol®, one can import and manipulate the corresponding state matrices in Matlab.® [2] The appendix at the end of this work provides detailed

[2]This requires launching "Comsol® 4.2 with Matlab®", which brings up a Matlab® interface which is linked to Comsol®.

MATLAB® code for how to extract and manipulate the COMSOL-generated discrete system matrices. However, we briefly describe the process.

The information one must extract from COMSOL® is the eliminated mass matrix[3], the negative eliminated stiffness matrix, the matrix that will represents the $\mathbf{B}^h$ matrix after constraint elimination and pre-multiplication by the inverse mass matrix, and the boundary constraint null space which provides insights about the unknown dependent variables ("Dc", "MA", "MB", and "Null" respectively in COMSOL®). This information is obtained by first loading the COMSOL® model into MATLAB® using the "mphload" function, then using the "mphmatrix" and "mphstate" functions to import these matrices. If the mass matrix is nonsingular, one can actually use the "mphstate" function to extract the A and B matrices directly.

Each dependent variable (called a "degree of freedom" or "dof") of the discrete system solved by COMSOL® represents a state value (or increment) at a specific node with specific spatial coordinates. We obtain this data by first extracting all the mesh information from the model (".xmeshInfo"), then from the ".dof" portion of this data structure, we extract the dependent variable information. This includes node number, x coordinate, y coordinate, and state type.[4] The nodal information for the unknown dependent variables (after eliminating constraints) can be determined by observing the matrix (Null), whose columns form the standard orthogonal basis of the constraint null space. For example, in the system

$$\underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{Null} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}, \tag{4.2.12}$$

if there are a total of five "dofs" obtained from the .xmeshInfo, one knows that X_1, X_2, X_3 correspond to the fourth, first, and third "dofs" respectively.

For this particular example, the eliminated mass matrix is non-singular, so we can obtain the $\mathbf{A}^h$ and $\mathbf{B}^h$ matrices using the mphstate function directly. Referring to Section 3.4 and (3.4.2), we can now build and export the system matrices to MATLAB.® In particular, the following MATLAB® commands will build the system:

```
>> matrices =mphmatrix(model,'sol1','Out',{'Dc','ud','Null'});
>> state = mphstate(model,'sol1','out',{'A' 'B'},'input','u1')
   ;
```

[3]By eliminated, we mean Dirichlet boundary constraint elimination.

[4]In this case, COMSOL® provides an index indicating whether a dependent variable represents a velocity component value, a pressure value, or a temperature value. The indexing scheme can be determined by extracting "dofs.dofNames".

```
>> M=full(matrices.Dc);
>> A=full(state.A);
>> B=full(state.B);
```

To address the approximate LQR problem, the $\mathbf{R}$ matrix remains unchanged and $\mathcal{Q}$ is constructed as in (3.4.2)-(3.4.3). For the first test, we simply set $\mathcal{Q} = \mathbf{M}$ (to approximate the identity) and $\mathbf{R} = 1$ using the MATLAB® "care" function:

```
>> [XX,LL,FF]=care(A,B,M,1,'nobalance');
```

Here, FF is a gain matrix that pre-multiplies the state $\mathbf{x}^h(t)$, which is nodal temperature in this particular case. If we let

$$\mathbf{K} = FF\mathbf{M}^{-1}, \tag{4.2.13}$$

this implies

$$FF\mathbf{x}^h(t) = \mathbf{K}\mathbf{M}\mathbf{x}^h(t). \tag{4.2.14}$$

For the case of temperature (similarly for the linearized Boussinesq system),

$$u^{h*}(t) = -\int_\Omega \mathbf{K}\left(\Phi^h(\mathbf{x})\right)\left(\Phi^h(\mathbf{x})\right)^T\mathbf{T}(t)\,d\mathbf{x} \cong -\int_\Omega k(\mathbf{x})\,T(\mathbf{x},t)\,d\mathbf{x} = u^*(t), \tag{4.2.15}$$

where we define $k(\mathbf{x})$ as a simple linear interpolation of $\mathbf{K}$. This function can be easily defined in COMSOL®. Thus, the approximate boundary control has the form

$$u^h(t) = \int_\Omega k^h(\mathbf{x})\,T^h(\mathbf{x},t)\,d\mathbf{x}, \tag{4.2.16}$$

where the nodal values values for $k^h(\mathbf{x})$ are computed by $\mathbf{K}\mathbf{M}^{-1}$ in MATLAB®.

After we have used the "care" function to determine "FF", we input the following into MATLAB®:

```
>> K=FF*inv(M);
>> K=[null_dofs_full_info(:,3:4) K'];
```

The second MATLAB® command line is simply a concatenation of the appropriate node coordinate information with **K**.

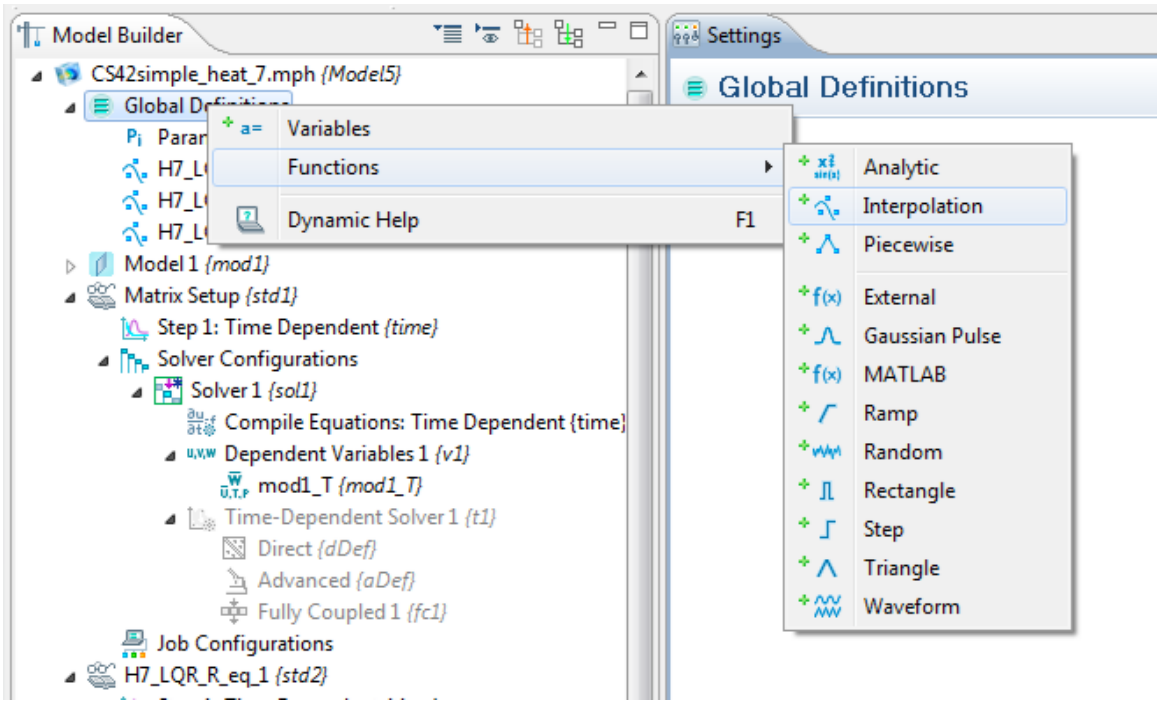

Figure 4.12: COMSOL® Menu Call for an Interpolation Function

Now we must import the information from **K** into COMSOL®. This is accomplished by first copying **K** (with its coordinate information) into a spreadsheet, then saving the spreadsheet as a tab-delimited text file. Another technique one can use is to save this data array directly from MATLAB® by using the "dlmwrite" function with the tab (\t) option. Then under "Global Definitions" in the COMSOL® "Model Builder", one can then import this file into a linearly-interpolated function (see Figures 4.12 and 4.14).

Figure 4.13 is a COMSOL® plot that depicts this interpolated kernel function, or "functional gain". Note that much more weight is given near the controlled portion of the boundary than the rest of the domain. Figure 4.15 is a screen shot of how we implement (4.2.16) in COMSOL®. In the "Model Builder", we set up the Dirichlet boundary condition corresponding to the controlled portion of the boundary and prescribe T.[5]

[5]Here H7_LQR_R_eq_1 is the label chosen for this particular kernel function.

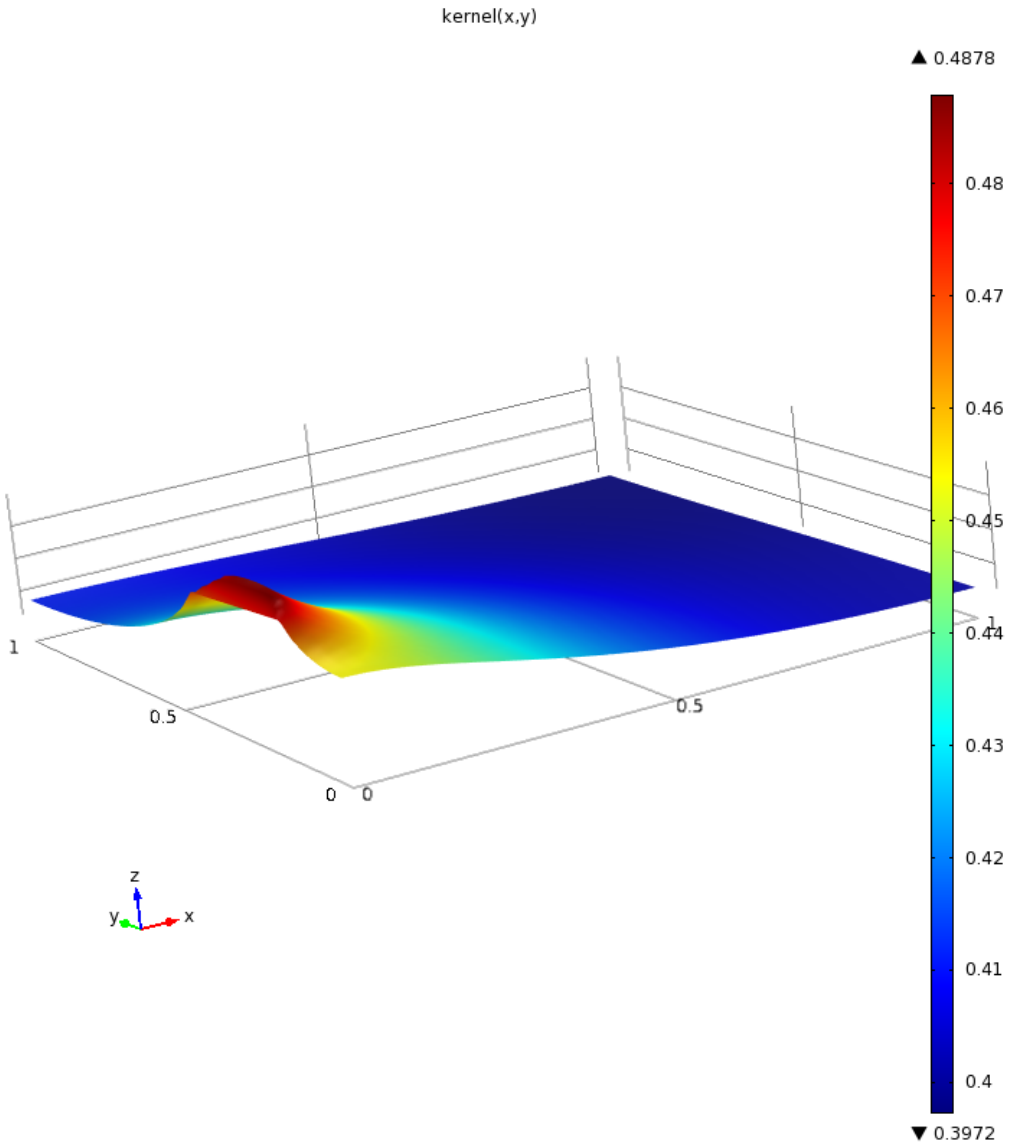

Figure 4.13: Functional Gain $k^h(\cdot)$ for $R = 1$

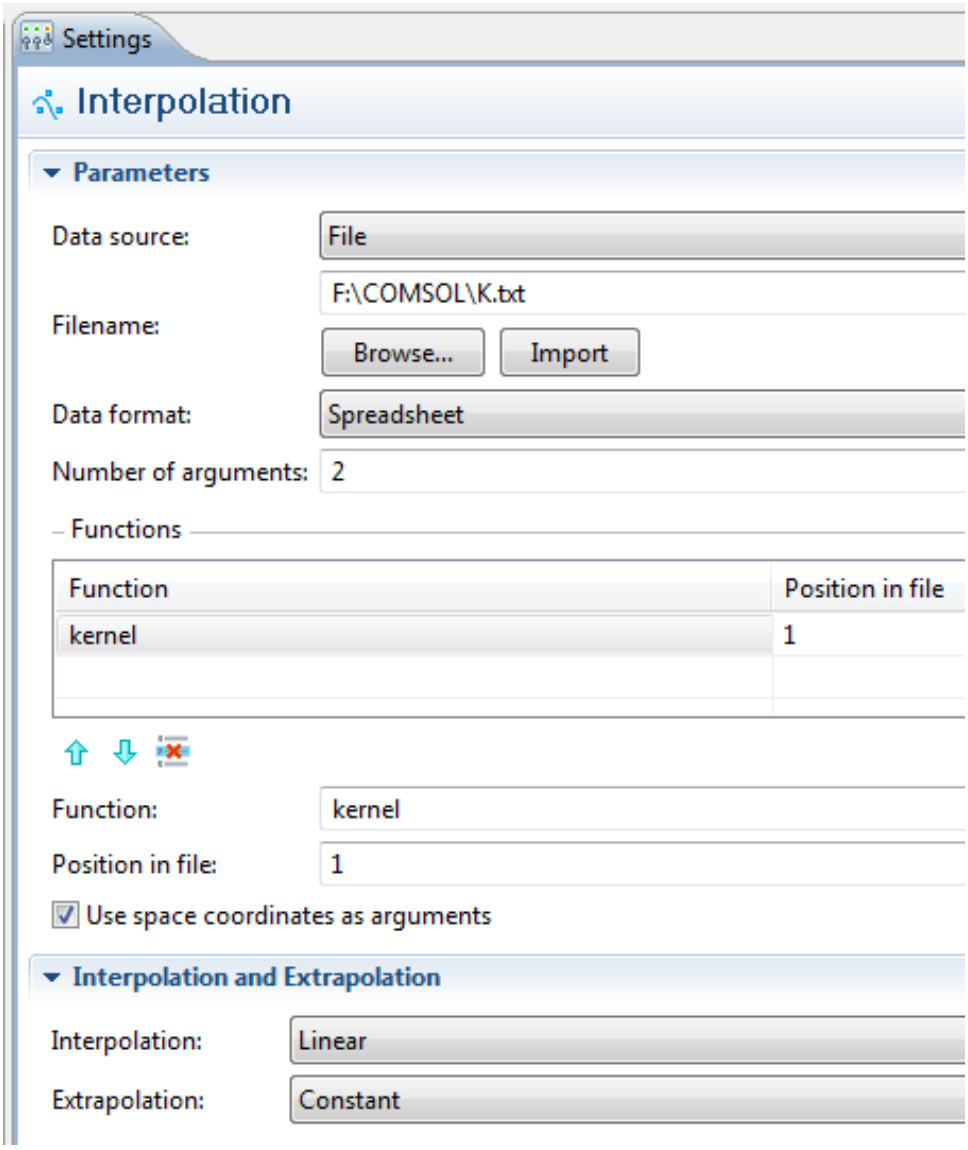

Figure 4.14: Importing and Interpolating Kernel Function in COMSOL®

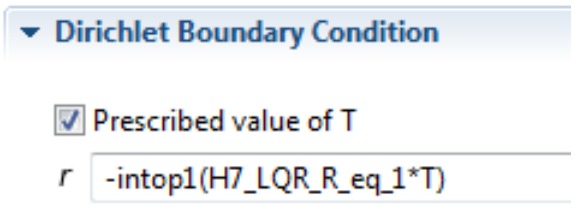

Figure 4.15: Implementation of (4.2.16) in COMSOL®

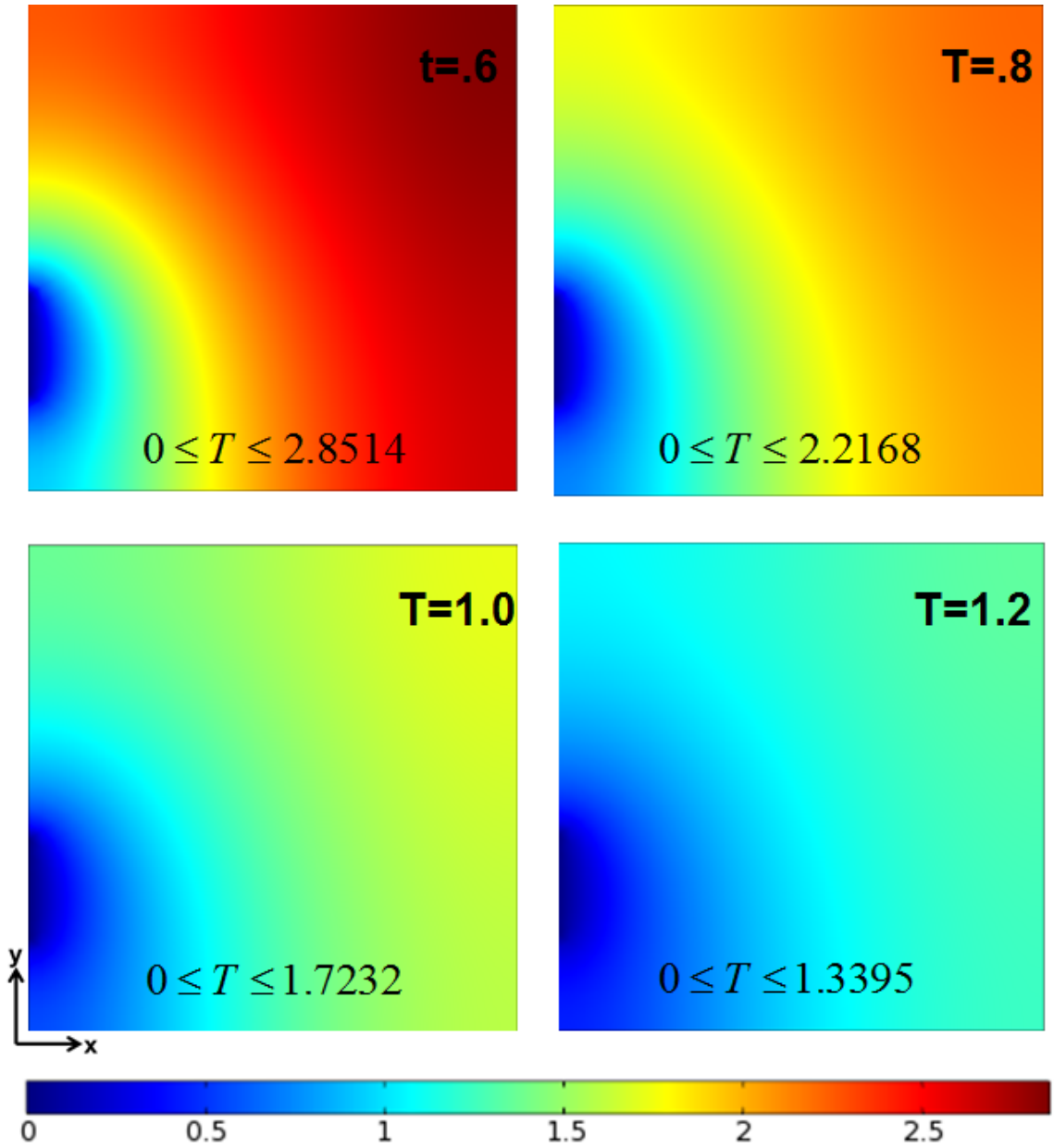

Figure 4.16: Time Evolution of Temperature with No Control

Figure 4.17 depicts snapshots of the COMSOL®-generated temperature evolution under these conditions. As a reminder, we set the initial temperature at five units above ideal with the goal of bringing the temperature uniformly near zero. Within ten seconds, we have $-5.7878 \times 10^{-8} \leqslant T \leqslant 2.0376 \times 10^{-7}$. This can be compared with the system with zero control (see Figure 4.16), where after ten seconds we have $0 \leqslant T \leqslant 2.0609 \times 10^{-5}$.

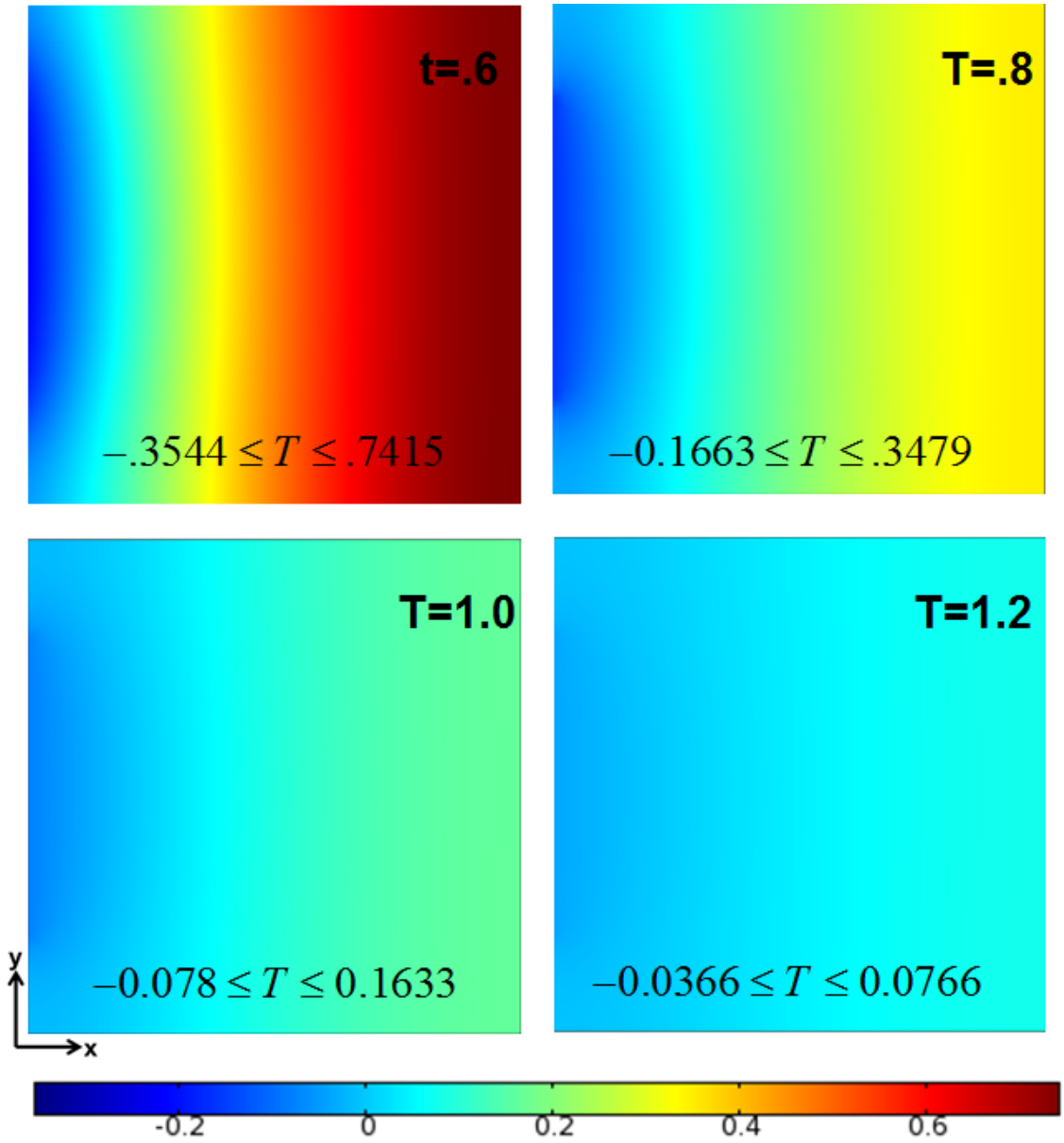

Figure 4.17: Time Evolution of Temperature with LQR-derived Kernel Function where $\mathbf{Q} = \mathbf{M}$ and $\mathbf{R} = \mathbf{I}$

Example 2

In this example we modify the control weight operator and reduce the penalty on the control. First, we reduce the cost penalty on the control function $\mathbf{R} = 0.0001$. Next, keeping the

same $\mathbf{R}$ value, we specify $\mathbf{Q} = \text{diag}\left(g\left(\mathbf{x}_i^h\right)\right)^{\frac{1}{2}}\mathbf{M}\,\text{diag}\left(g\left(\mathbf{x}_i^h\right)\right)^{\frac{1}{2}}$ where

$$g\left(\mathbf{x}_i\right) = \begin{cases} 1 \text{ if } \ 0.1 \leqslant \mathrm{x}_i, y_i \leqslant 0.9 \\ 0.1 \ \ \text{otherwise} \end{cases} \tag{4.2.17}$$

and $\mathbf{x}_i$ represents the coordinates of all the independent variables.

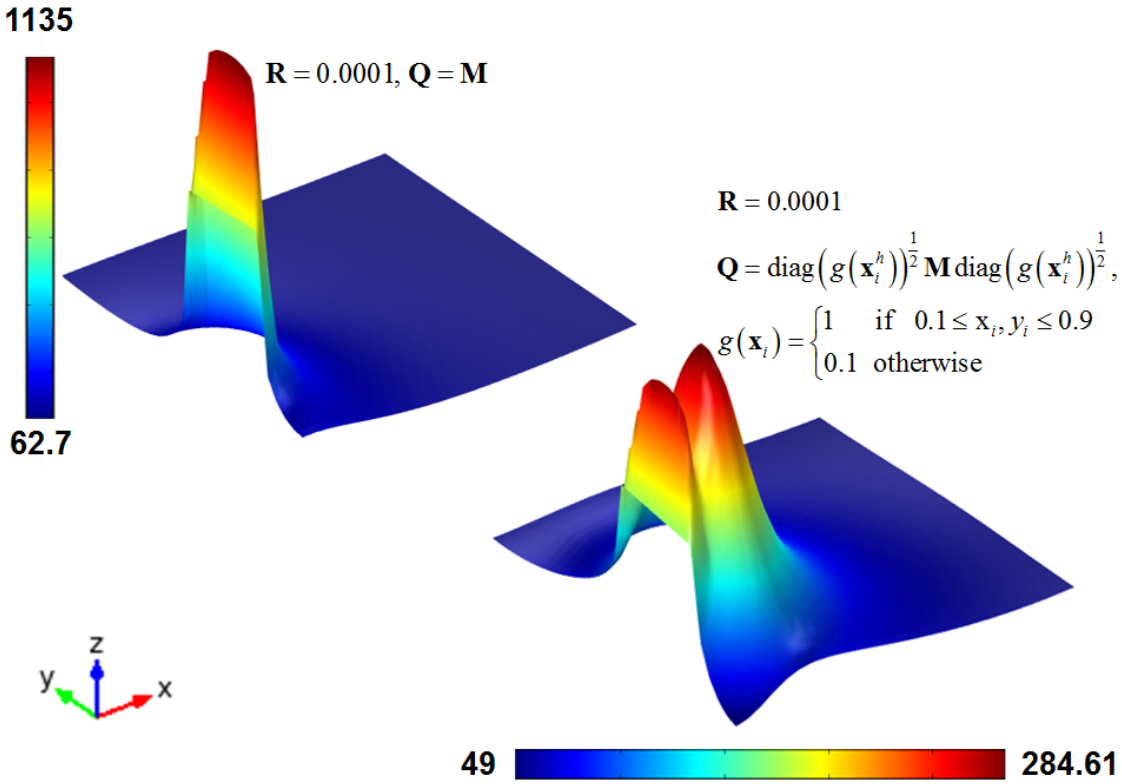

Figure 4.18: Functional Gains for Very Small Penalty (Both) and Spatially-Weighted Q Operator

Observe that the weighting operator defined by (4.2.17) places more penalty on temperature away from the wall. The associated functional gains are plotted in Figure 4.18. Also included is the control $u\left(t\right) = -\gamma \int\limits_{\Omega} T\left(t, \mathbf{x}\right) d\mathbf{x}$, $\gamma = 1$. Figures 4.19 and 4.20 depict comparative snapshots of the time evolutions corresponding to each kernel at times 0.5 and 1.0, respectively. Namely, we consider four cases where (i) $k\left(\mathbf{x}\right) = 1$; (ii) $k\left(\mathbf{x}\right)$ is obtained from the LQR design with $\mathbf{Q} = \mathbf{M}, \mathbf{R} = 1$; (iii) $k\left(\mathbf{x}\right)$ is obtained from the LQR design with $\mathbf{Q} = \mathbf{M}, \mathbf{R} = 0.0001$; and (iv) $k\left(\mathbf{x}\right)$ is obtained from the LQR design with $\mathbf{Q} = \text{diag}\left(g\left(\mathbf{x}_i^h\right)\right)^{\frac{1}{2}}\mathbf{M}\,\text{diag}\left(g\left(\mathbf{x}_i^h\right)\right)^{\frac{1}{2}}, \mathbf{R} = 0.0001$ (see (4.2.17)).

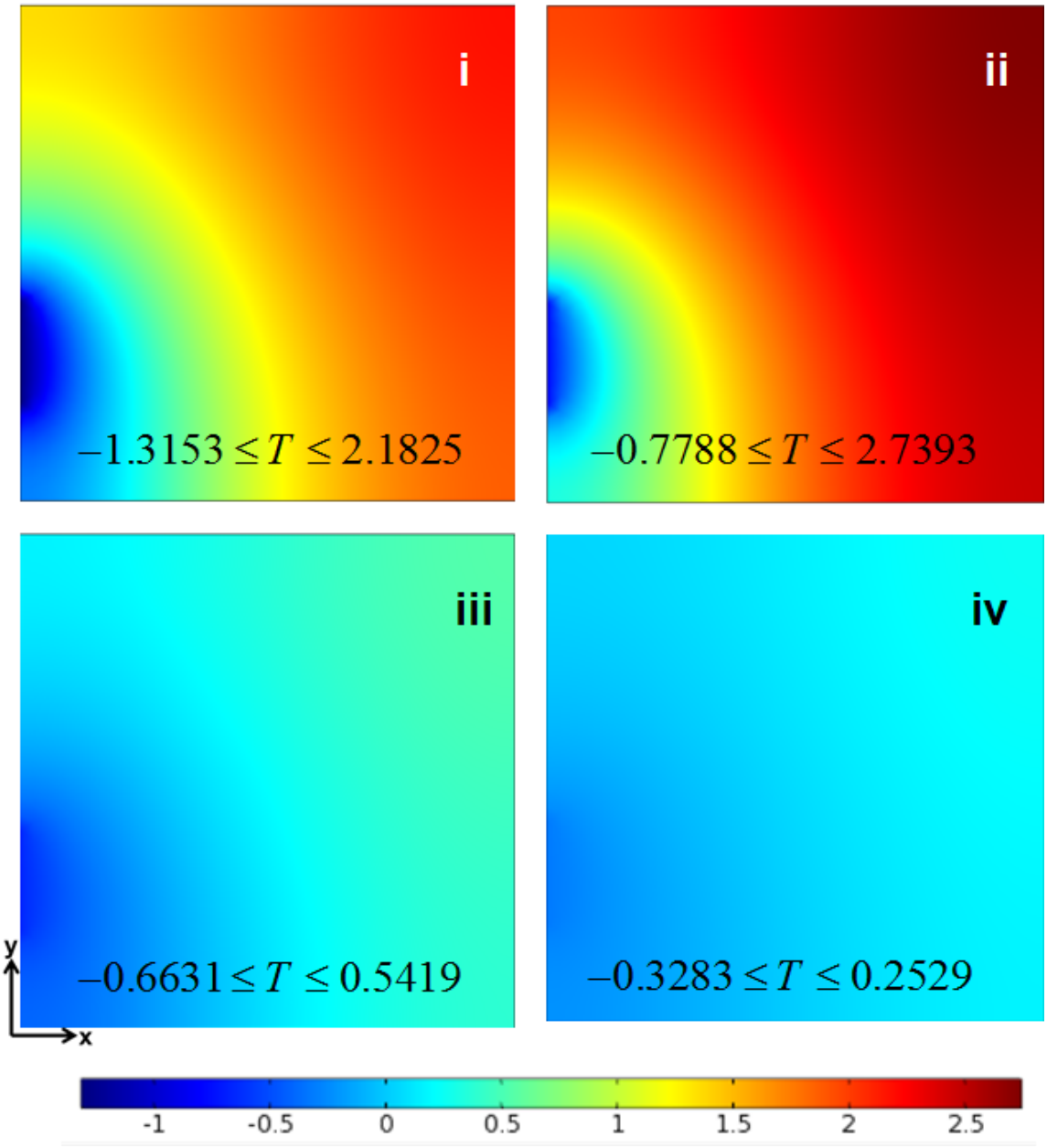

Figure 4.19: Controlled Temperature at $t = 0.5$

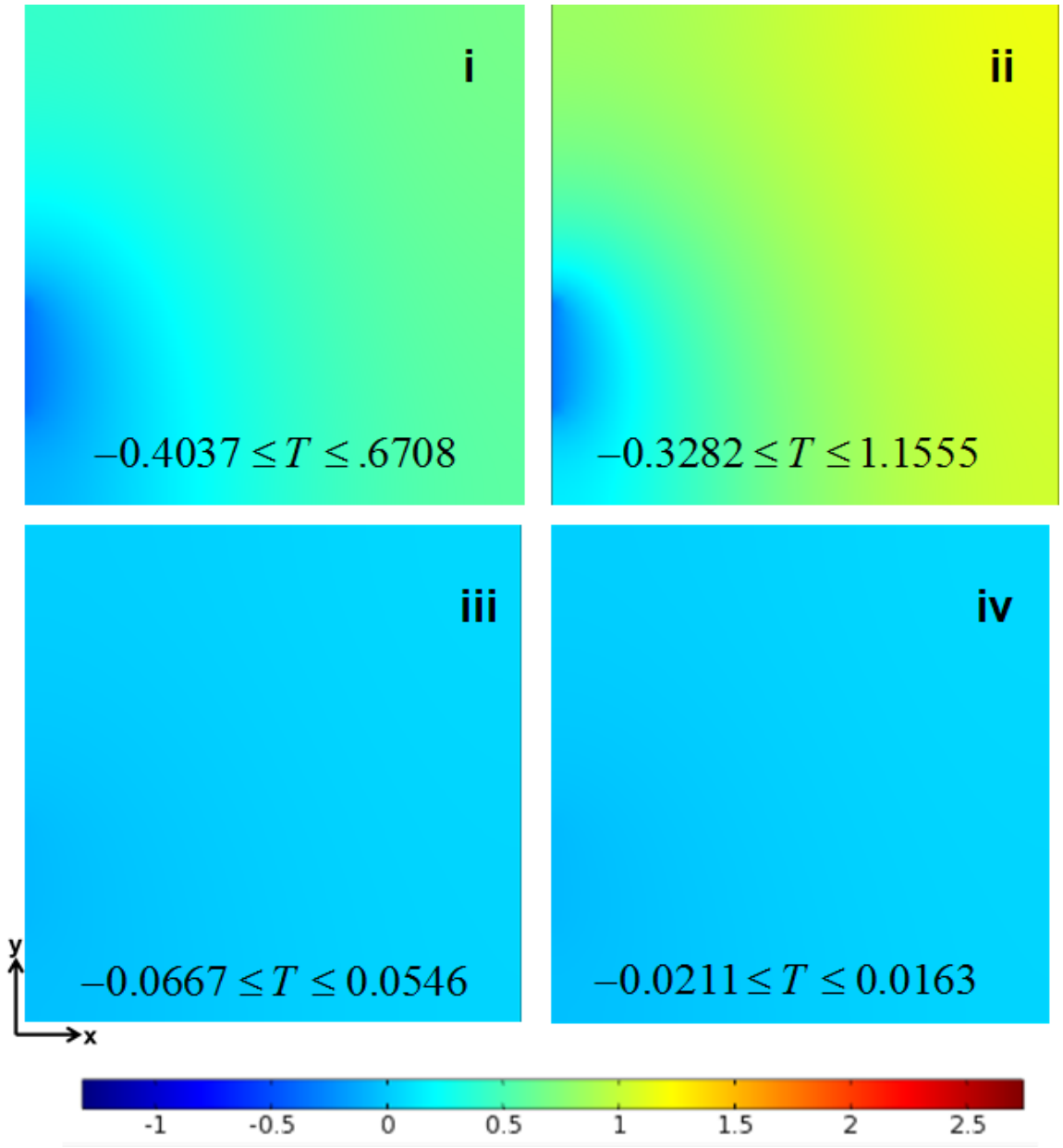

Figure 4.20: Controlled Temperature at $t = 1.0$

As expected, a larger penalty on the control input,$\mathbf{R} = 1$ for example, can have an impact on how quickly the temperature state is steered close to zero. Figure 4.21 is a line graph of the L^2 norm temperature evolution for each type of control. It is clear that reducing the penalty decreases the overall temperature energy over time. Of course, in a practical system, such a reduction must be weighed against the energy cost of introducing more control. Figure 4.22 is a graph of the control temperature over time for each of the kernels we created. We see that there is very little difference after two units of time. Therefore, it's possible

that more control may have little impact on the overall energy consumed by the system. This simply highlights the fact that some type of sensitivity analysis should be conducted to determine the cost function parameters that truly minimize the energy cost (electrical energy, for example) of control.

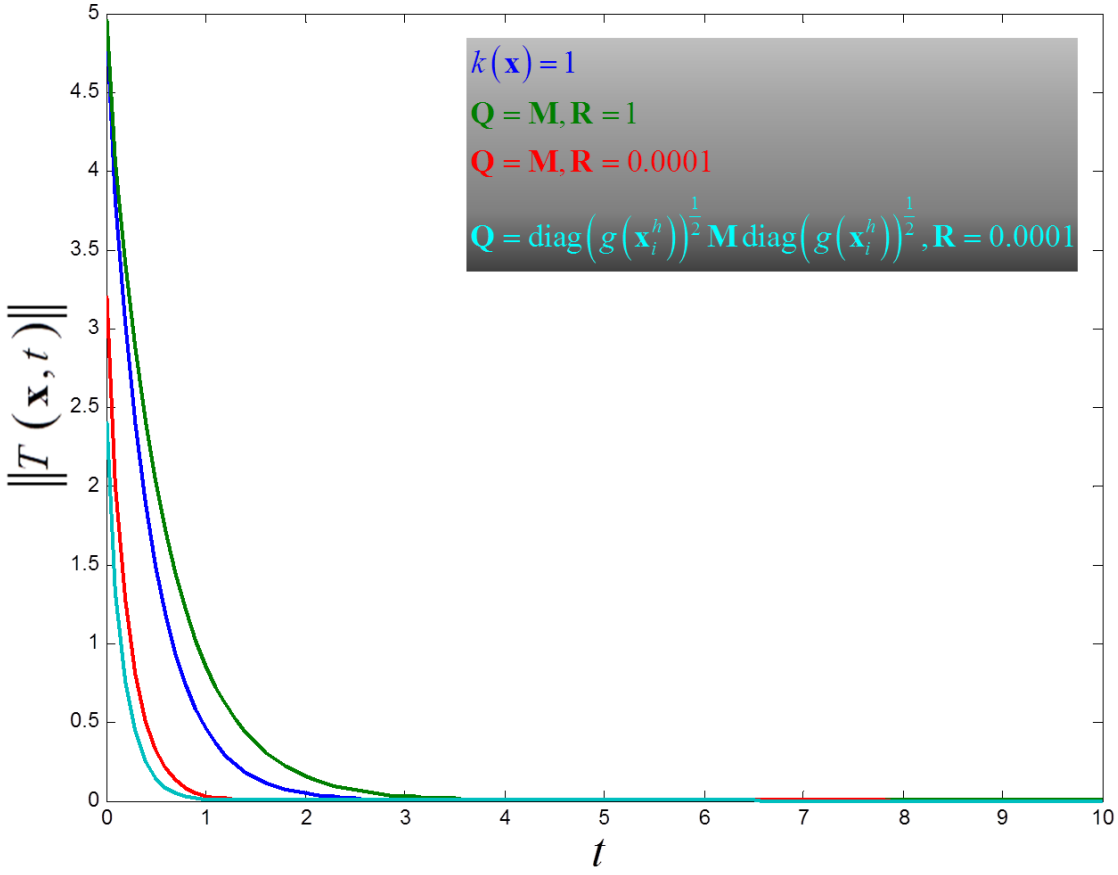

Figure 4.21: L^2 Norm of Temperature over Time for Each Type of Kernel

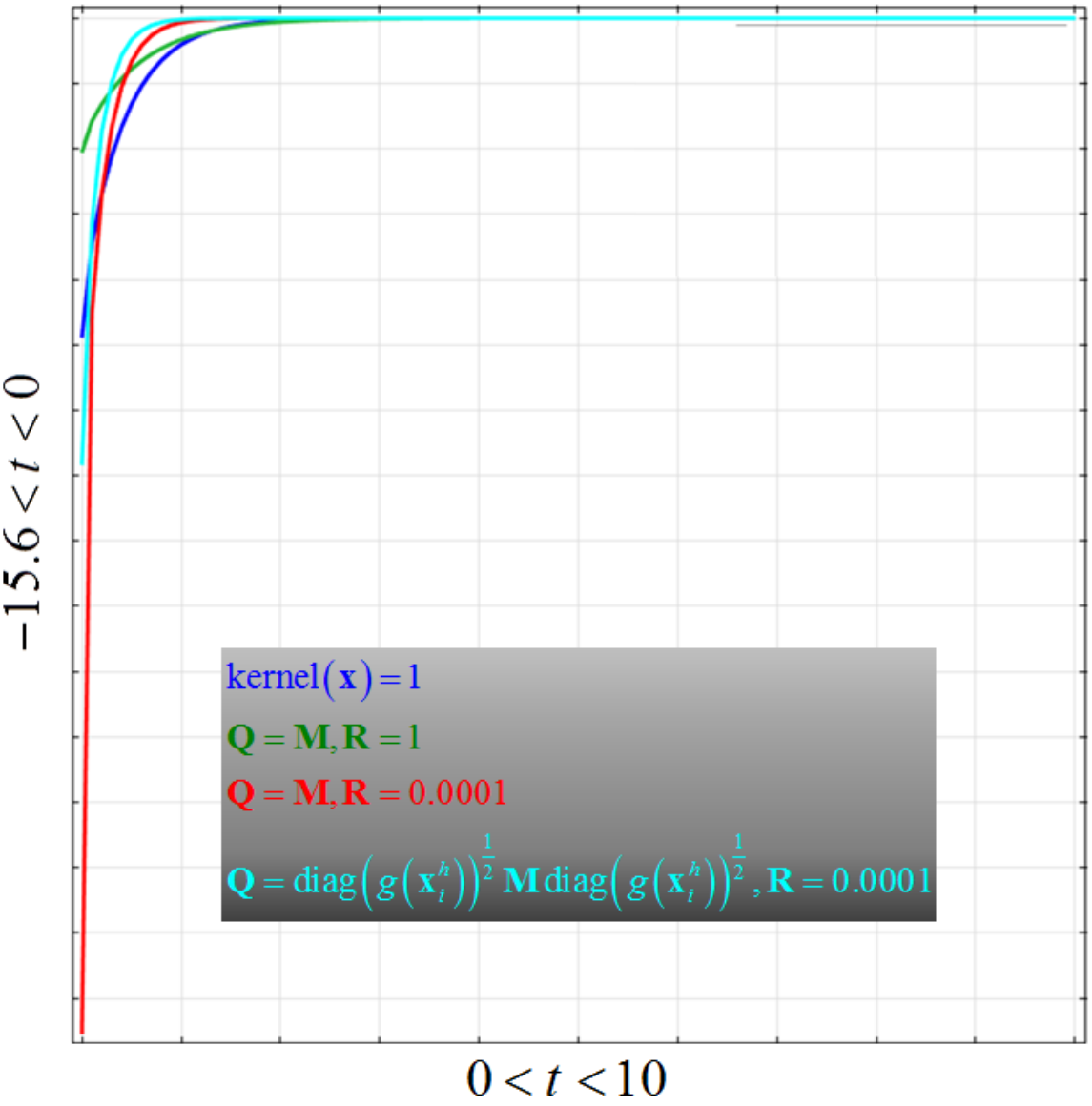

Figure 4.22: Control Temperature over Time

4.3 Control of the Boussinesq System

We now proceed with feedback controls on a Boussinesq System using MATLAB® and COMSOL®. Because it is difficult to determine precisely the equations solved in COMSOL's® physics models for laminar flow and heat transfer, we again use the COMSOL® PDE interface to model the system, this time in the dimensionless form (see for example [24]),

$$\frac{\partial \mathbf{v}}{dt} + \mathbf{v} \cdot \nabla \mathbf{v} = \nabla \cdot \left(-pI + \frac{1}{\mathrm{Re}} \left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T \right) \right) + \frac{\mathrm{Gr}}{\mathrm{Re}^2} T \mathbf{e}_3, \tag{4.3.1}$$

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \frac{1}{\mathrm{Re}\,\mathrm{Pr}} \Delta T, \tag{4.3.2}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{4.3.3}$$

where, given a length scale l_s, a velocity scale V_s, and temperature scale T_s,

$$\mathrm{Re} \triangleq \frac{\rho_0 V_s l}{\mu}, \ \mathrm{Pr} \triangleq \frac{\mu C_p}{\kappa}, \ \mathrm{Gr} \triangleq \frac{\alpha_T l^3 \rho_0^2 g T_s}{\mu^2}. \tag{4.3.4}$$

Name	Description	Value
Re	Reynolds Number	10
Gr	Grashof Number	30
Pr	Prandtl Number	10

Table 4.2: Boussinesq Dimensionless Parameters for Computation

For the remainder of this section, these parameters are as given in Table 4.2. Though these values are unrealistic for air in a room, they provide solutions that demonstrate the interplay between between various physical states and demonstrate proof of concept for designing control between MATLAB® and COMSOL.®

4.3.1 Output Feedback Control in COMSOL® Without MATLAB®

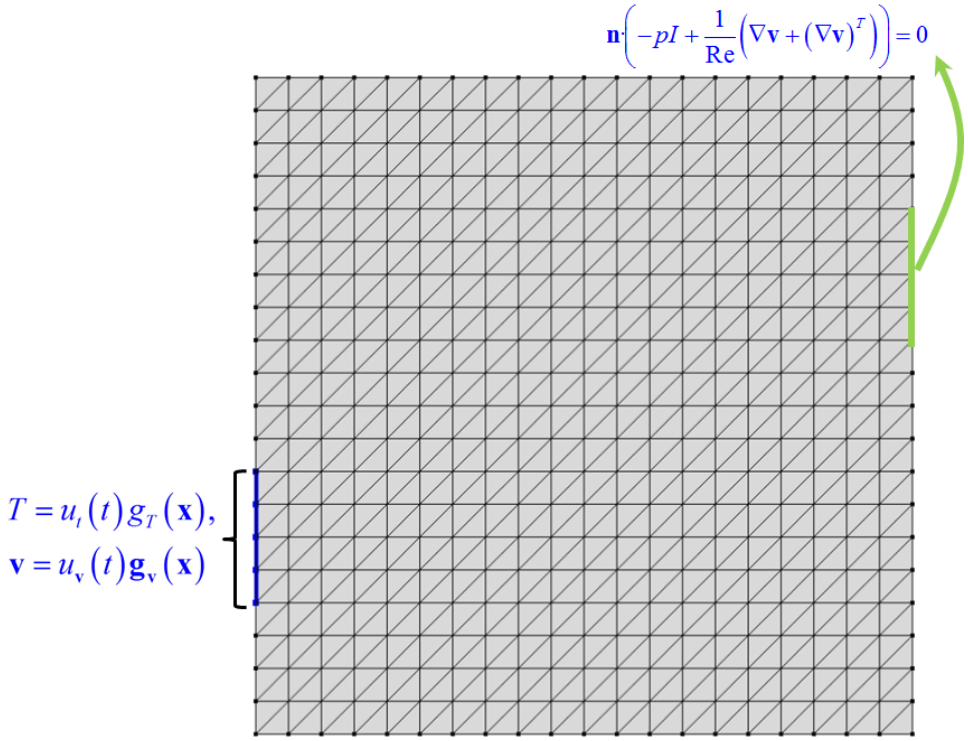

Figure 4.23: Re $= 10$, $g_T(\mathbf{x}) = 1$,, $\mathbf{g}_{\mathbf{v}}(\mathbf{x}) = ((0.4-y)(y-0.2), 0)^T$

To begin with, we assume that $T(t,0) = 5$ with the intent to steer $\|T\|$ to zero by controlling the inflow temperature and velocity on a portion of the boundary. Namely, at the inlet the air velocity field has a specified parabolic profile in the $\mathbf{x}$ direction that is controlled in amplitude, $\mathbf{v}|_{\Gamma_{E\mathbf{v}}}(\mathbf{x},t) = u_{\mathbf{v}}(t)\,\mathbf{g}(\mathbf{x})$. Similarly, the temperature control is specified as $T = u_t(t)\,g_T(\mathbf{x})$, where $g_T(\mathbf{x}) = 1$ Figure 4.23 provides the associated mesh and boundary information.

The simulation is run for 100 seconds with three inflow conditions. For a baseline comparison, first we consider the open-loop system with (zero control on the inlet)

$$u_{\mathbf{v}} = u_T = 0. \tag{4.3.5}$$

Next, we simulate the closed-loop system with a feedback controller dependent only on the average temperature,

$$u_T(t) = -\gamma \int_\Omega T(t,\mathbf{x})\,d\mathbf{x},\ \gamma = 1. \tag{4.3.6}$$

Finally a closed-loop system with control input dependent on weighted sums of averages for

each of the three states, T, v_1, and v_2, is specified so that

$$u_T = -k_{T1}\int_\Omega T d\mathbf{x} - k_{T2}\int_\Omega v_1 d\mathbf{x} - k_{T3}\int_\Omega v_2 d\mathbf{x}, \tag{4.3.7}$$

$$u_{\mathbf{v}} = -k_{\mathbf{v}1}\int_\Omega T d\mathbf{x} - k_{\mathbf{v}2}\int_\Omega v_1 d\mathbf{x} - k_{T3}\int_\Omega v_2 d\mathbf{x}. \tag{4.3.8}$$

Figures 4.24-4.26 provide snapshots of each of the three runs. For Figure 4.26 we have $k_{T1} = k_{\mathbf{v}2} = 1$ and $k_{T2} = k_{T3} = k_{\mathbf{v}1} = k_{T3} = 0$.[6]

[6]An attempt was made to designate all six of these constants as nonzero, however the simulation failed to converge using the mesh size depicted in Figure 4.23.

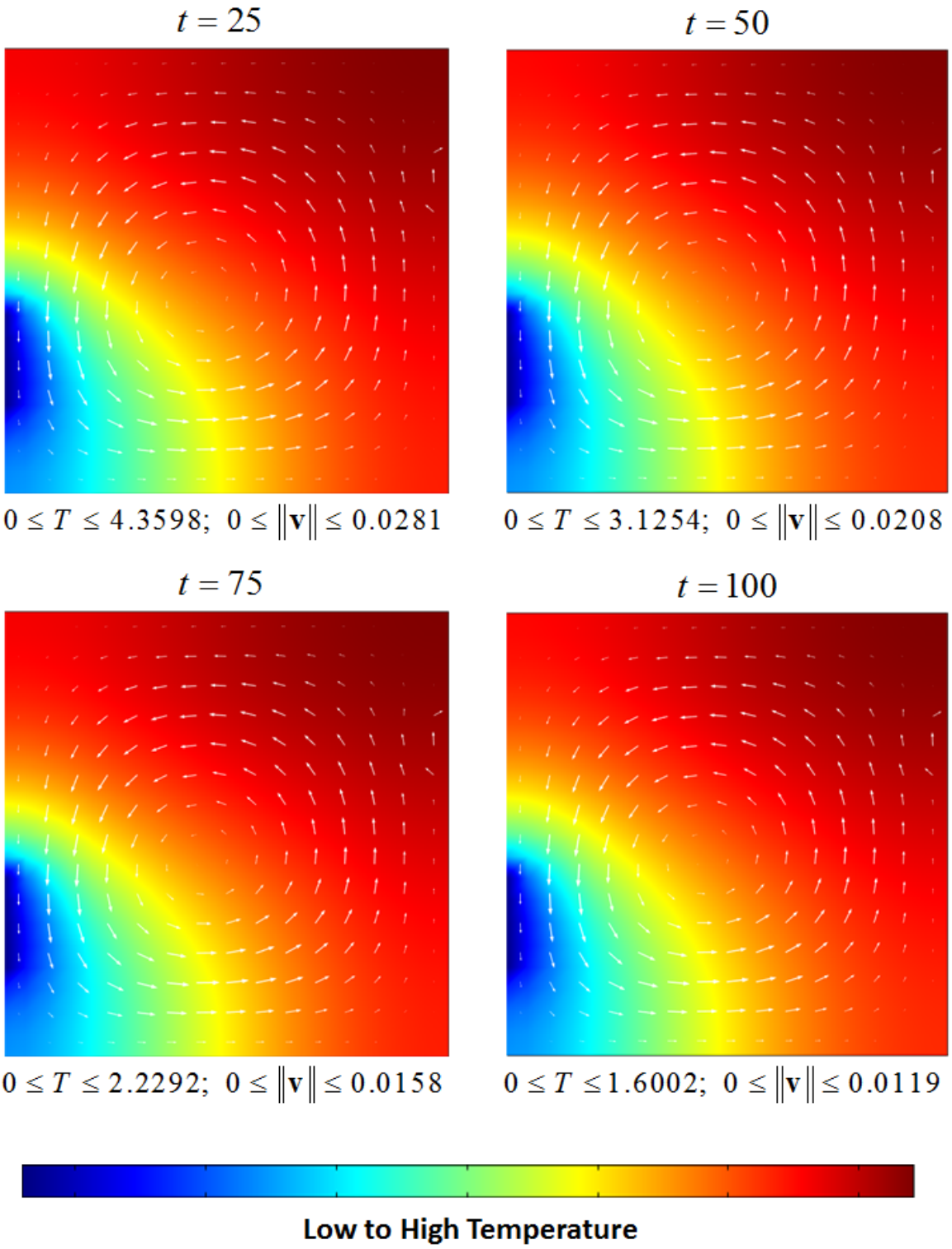

Figure 4.24: System with Zero Control

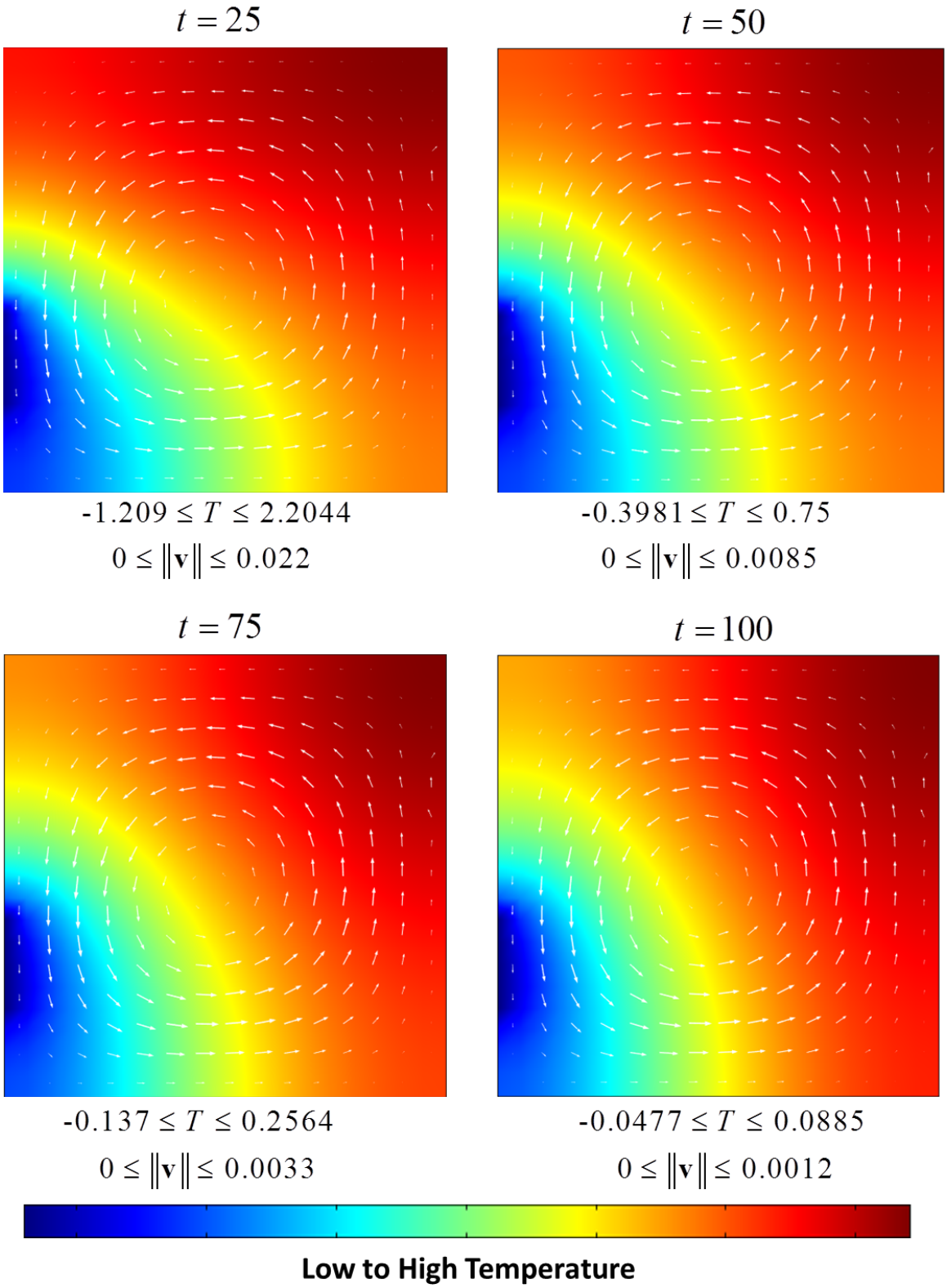

Figure 4.25: System with $u_T(t) = -\gamma \int_\Omega T(t, \mathbf{x})\, d\mathbf{x}$, $\gamma = 1$

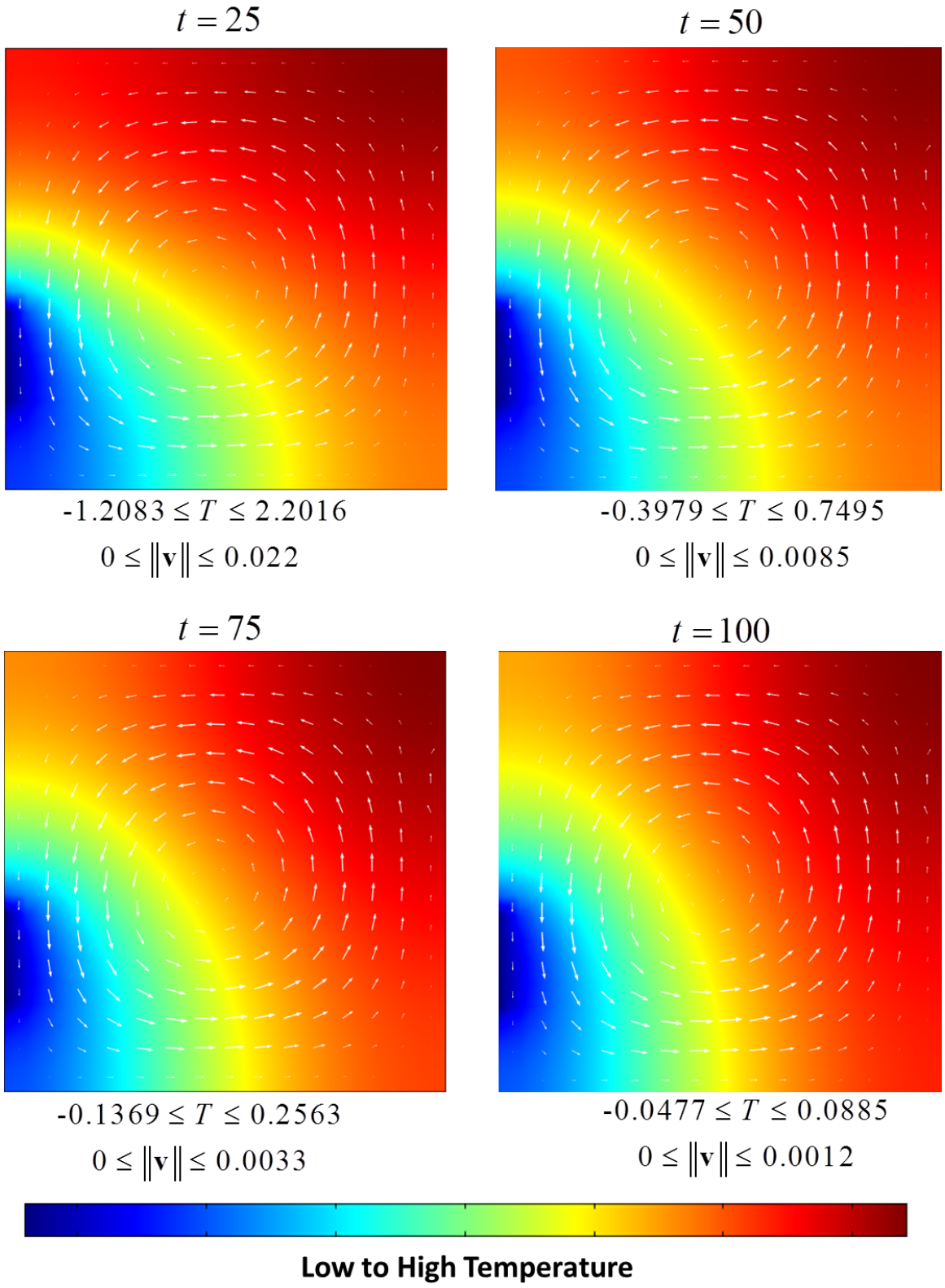

Figure 4.26: System with $u_T = -\int_\Omega T d\mathbf{x},\ u_\mathbf{v} = -\int_\Omega v_1 d\mathbf{x}$

By inspection, it is evident that even a simple average state closed loop feedback control tends to produce improved results compared with no control at all. Since the velocity magnitudes are negligible for this particular experiment, we compare $\|T(t)\|$ for each system in Figure 4.27. Note that velocity control had little bearing on the results for this problem.

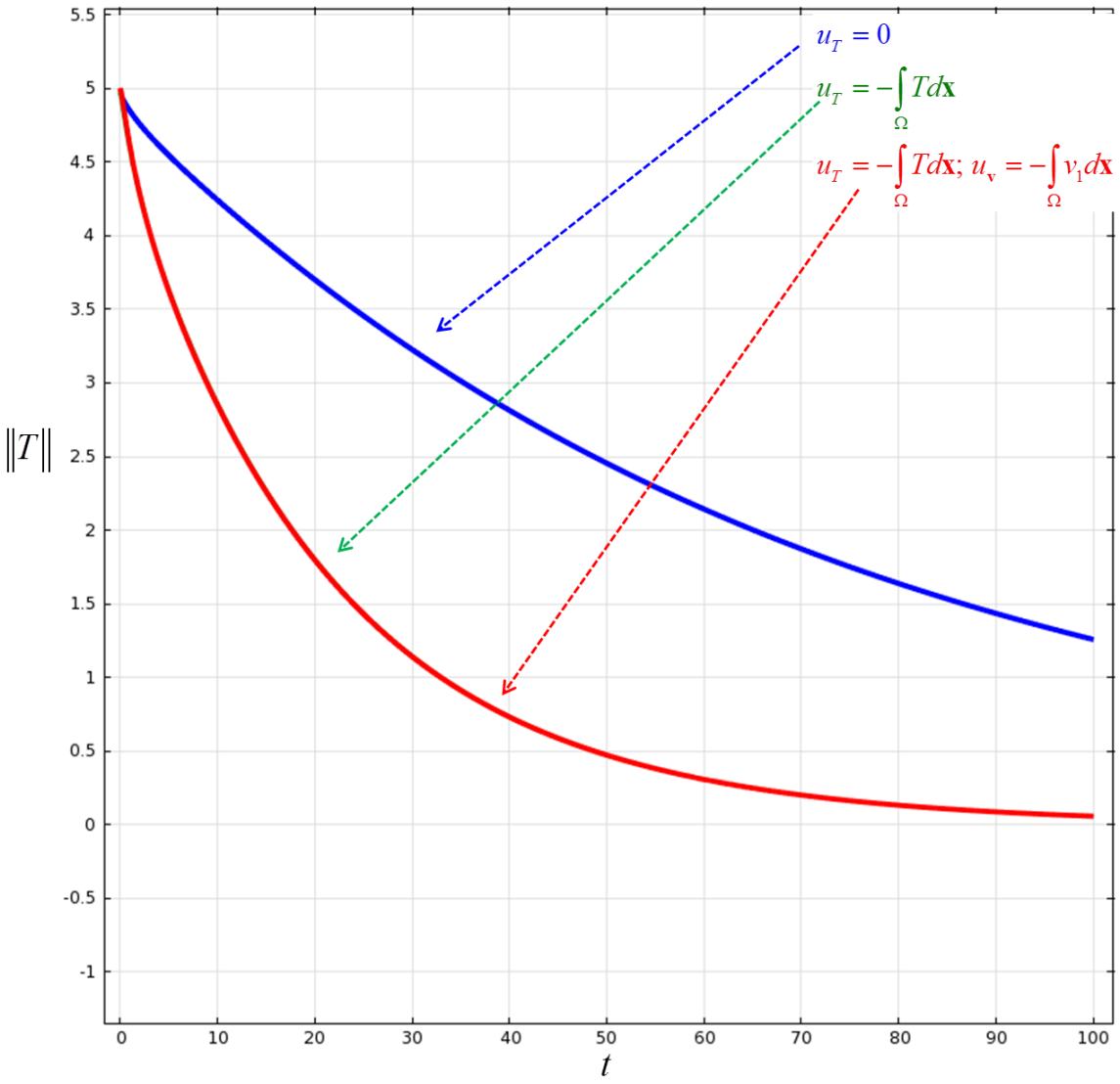

Figure 4.27: $\|T(t)\|$ for Zero Control, Feedback Control of T, and Feedback Control of Both T and v_1

As we begin to introduce boundary feedback control based on weighted integration of the state, we remark that direct integration over the domain in COMSOL® at every time step, even with linear quadratures, is quite costly for computing time. In particular, a closed-loop problem takes up to twenty times longer to solve than the corresponding open-loop problem.

However, with an existing capability in the MATLAB® - COMSOL® interface to access the mass matrix, M, one can define a kernel function in MATLAB® which can be converted to a Gain Matrix which, via some innovation, can be multiplied by the state vectors in COMSOL.® For example, assume that we wish to approximate

$$u_T = \int_{\Omega} g(\mathbf{x})\, T d\mathbf{x}, \tag{4.3.9}$$

for some integrable function $g(\mathbf{x})$ that we've defined. With the same basis functions used to approximate the weak form, one can approximate u_T as

$$\cong \sum_{i,j=1}^{N_{\mathcal{P}\mathcal{T}^h}} \int_{\Omega} g(\mathbf{x}_i)\, \phi_i(\mathbf{x})\, \phi_j(\mathbf{x})\, T_j(\mathbf{x})\, d\mathbf{x} = \left\langle g\left(\mathbf{X}^h_{vtx}\right), \mathbf{M}_T \mathbf{T} \right\rangle \triangleq \mathbf{KT}. \tag{4.3.10}$$

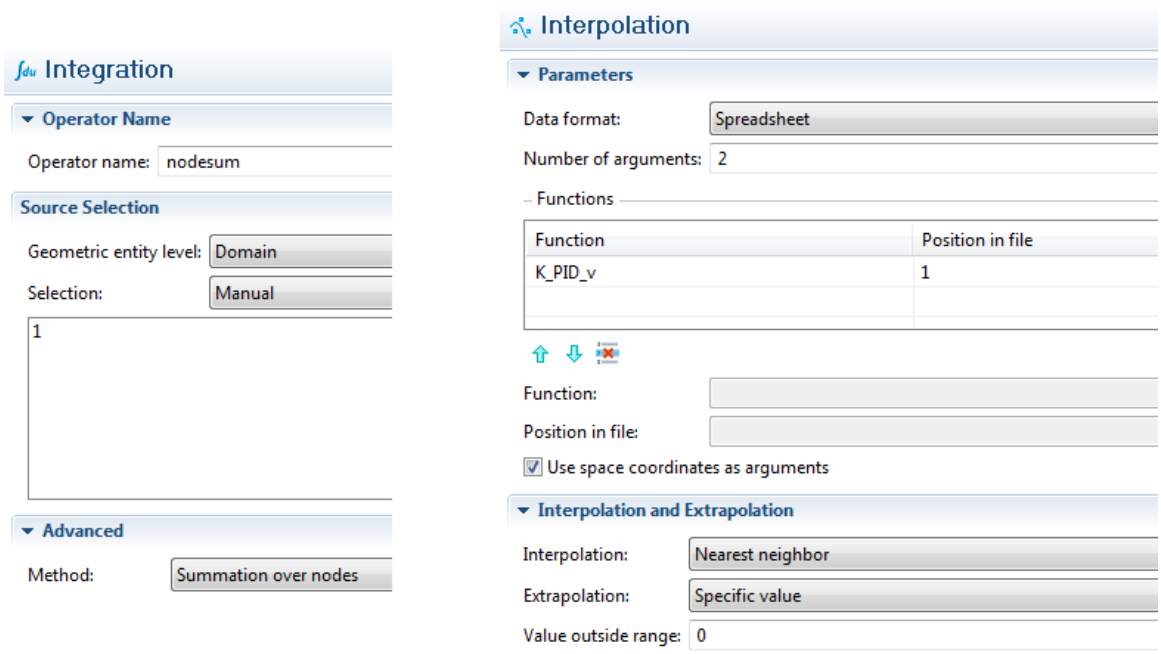

Figure 4.28: Creating Nearest Neighbor Interpolation and Nodal Sum in COMSOL®

The approximate integral is evaluated in COMSOL by creating a linear interpolation function from $\mathbf{K}$ (see Figures 4.12 and 4.14), and then finding the sum of $K_i T_i$ on all the nodes. This is accomplished via a "nodal sum", where the interpolation function is designated to use a "nearest neighbor" approach for value assignment (see Figure 4.28). Note that this same technique can be applied to LQR feedback, where the mass matrix is already built into the gain matrix, $\mathbf{K} = \mathbf{kM}$ (see (4.2.15)). Unfortunately, this technique is limited to those independent variables whose discretizations include a value at *every* node point implemented

by COMSOL® , including internal nodes associated with higher-degree shape functions (here quadratic). This means that any temperature integrations (piecewise linear basis functions) can not be represented this way. However, having the ability to at least represent velocity-component functional gains in this manner creates a huge savings in computation time.

4.3.2 LQR Control in COMSOL® Using MATLAB®

We are now ready to address LQR control using linkages between COMSOL® and MATLAB.® Much of the technique described in Section 4.2.2 still applies, with some key additional issues. First, because of the increase in dependent variables for the Boussinesq system due to the requirement for quadratic velocity basis functions and now four coupled states (v_1, v_2, T, p) to be resolved, it is necessary to implement the technique described around equations (4.3.9)-(4.3.10). In other words, at least for velocity, we use the COMSOL® "nodal sum" function as well as the integral function to compute

$$u_{\mathbf{v}}(t) \cong \mathbf{K}_{1,j_{\mathbf{V_1}}} \mathbf{V_1}(t) + \mathbf{K}_{1,j_{\mathbf{V}_2}} \mathbf{V}_2(t) + \int_{\Omega} k_{1,T}(\mathbf{x})\, T(\mathbf{x},t)\, d\mathbf{x}, \tag{4.3.11}$$

$$u_T(t) \cong \mathbf{K}_{2,j_{\mathbf{V_1}}} \mathbf{V_1}(t) + \mathbf{K}_{2,j_{\mathbf{V}_2}} \mathbf{V}_2(t) + \int_{\Omega} k_{2,T}(\mathbf{x})\, T(\mathbf{x},t)\, d\mathbf{x}, \tag{4.3.12}$$

where $\mathbf{K}_{1,j_{\mathbf{V_1}}}$ is the portion of the gain matrix, derived from the CARE function in MATLAB®, that pre-multiplies the discrete nodal $\mathbf{e_1}$-direction velocity state, $\mathbf{V_1}(t)$.[7] $k_{1,T}(\mathbf{x}), k_{2,T}(\mathbf{x})$ are found using the same technique discussed in (4.2.15)-(4.2.16).

We also must address the pressure term, which we eliminate using the "penalty method" described in Section 3.4.4. Implementation of this method between COMSOL® and MATLAB® requires additional effort. Essentially, after applying similar procedures to those described in Section 4.2.2, one obtains the boundary-constraint-eliminated mass and stiffness matrices, along with an operator on the control inputs. We designate these in MATLAB® as "M","MA", and "MB". Since in COMSOL® , "M" is found by differentiating the residual system as described in (3.4.30), it is singular because the system contains no pressure derivatives. However, because we have access to all of the independent variable information, we can rearrange the linear system in MATLAB to the form,

$$\begin{pmatrix} \mathbf{M} & & & \\ & \mathbf{M} & & \\ & & \mathbf{M_T} & \\ & & & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{V}}_1 \\ \dot{\mathbf{V}}_1 \\ \dot{\mathbf{T}} \\ \dot{\mathbf{P}} \end{pmatrix} = \begin{pmatrix} \mathbf{K} & \mathbf{N^T} \\ \mathbf{N} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{V_1} \\ \mathbf{V_1} \\ \mathbf{T} \\ \mathbf{P} \end{pmatrix} + \mathbf{Bu}. \tag{4.3.13}$$

[7] Similarly for the other discrete terms.

This is accomplished in MATLAB® by constructing a rearranging matrix based on the independent variable data we retrieve in MATLAB® via the ".xmeshinfo" data structure and the "Null" matrix which provides information about the ordering of dependent variables after boundary constraint elimination. Figure 4.29 provides a simplified example of the array of dependent variable information one can construct with COMSOL® data in MATLAB® . Designate this array as **H**.

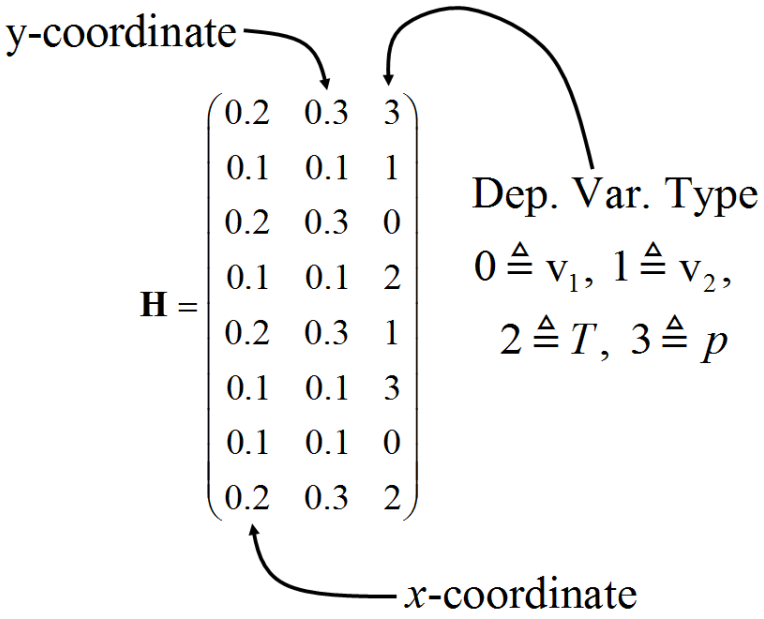

Figure 4.29: A Simple Dependent Variable Information Array

To create the proper rearranging matrix, simply concatenate an identity matrix to the right side of **H**,

$$\mathbf{H} = [\mathbf{H}, \mathbf{I}] = \left(\begin{array}{ccc|cccccccc} 0.2 & 0.3 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0.3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0.2 & 0.3 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0.1 & 0.1 & 3 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0.1 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0.2 & 0.3 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right), \tag{4.3.14}$$

and use the MATLAB® "sortrows" command to sort **H** on column three. After removing the first three columns of **H**, one is left with a permuted identity matrix, which can be used

to reorder the entire system.[8]. So if we have

$$\mathbf{M}\dot{\mathbf{x}} = \mathbf{K}\mathbf{x} + \hat{\mathbf{B}}\mathbf{u}, \tag{4.3.15}$$

and let $\mathbf{z} = \mathbf{H}\mathbf{x}$, which implies $\mathbf{x} = \mathbf{H}^T\mathbf{z}$, then (4.3.15) becomes

$$\mathbf{M}\mathbf{H}^T\dot{\mathbf{z}} = \mathbf{K}\mathbf{H}^T\mathbf{z} + \hat{\mathbf{B}}\mathbf{u}, \tag{4.3.16}$$

so that

$$\mathbf{H}\mathbf{M}\mathbf{H}^T\dot{\mathbf{z}} = \mathbf{H}\mathbf{K}\mathbf{H}^T\mathbf{z} + \mathbf{H}\hat{\mathbf{B}}\mathbf{u}. \tag{4.3.17}$$

Therefore, $\mathbf{H}\mathbf{M}\mathbf{H}^T$, $\mathbf{H}\mathbf{K}\mathbf{H}^T$, and $\mathbf{H}\hat{\mathbf{B}}$ constitute the new "mass", "stiffness", and input operator matrices that correspond to sorting the independent variables according to their type.

To obtain the $\mathbf{M}_p$ matrix for the penalty method, we must return to the COMSOL® application and designate the coefficient for $\dot{p}$ as 1. This creates a system of equations that has nothing to do with the Boussinesq equations, but it creates an identical mass matrix to the previous system, only with pressure terms now included.

We then re-import the model into MATLAB® , careful to leave all the variables from the first import intact, so that we obtain a new differently-named mass matrix, that has terms associated with $\dot{p}$ all in the right place to match the spatial coordinates and nodal information of the original system. Next, we use the COMSOL® *Null* matrix to convert this new mass matrix into one without the rows and columns associated with the boundary constraints. Afterwards, we rearrange the rows and columns of this new "eliminated" mass matrix so that it matches the original rearranged "eliminated" mass matrix. Subtracting the two matrices then provides the needed $\mathbf{M}_p$ matrix,

$$\begin{pmatrix} \mathbf{M} & & & \\ & \mathbf{M} & & \\ & & \mathbf{M_T} & \\ & & & \mathbf{M}_p \end{pmatrix} - \begin{pmatrix} \mathbf{M} & & & \\ & \mathbf{M} & & \\ & & \mathbf{M_T} & \\ & & & 0 \end{pmatrix} = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \mathbf{M}_p \end{pmatrix}. \tag{4.3.18}$$

Now a nonsingular system, without the pressure term (recall (3.4.59)) can be formed to solve the ARE.

4.3.3 Numerical Results

We now apply the above techniques to derive discrete LQR functional gains. First, we approximate functional gains on three progressively finer meshes (mesh size $h = 0.1,\ 0.05,\ 0.025$) as shown in Figure 4.30.

[8]Note that one could also do secondary and tertiary sorts on x and y coordinates, however this is computationally expensive in MATLAB® .

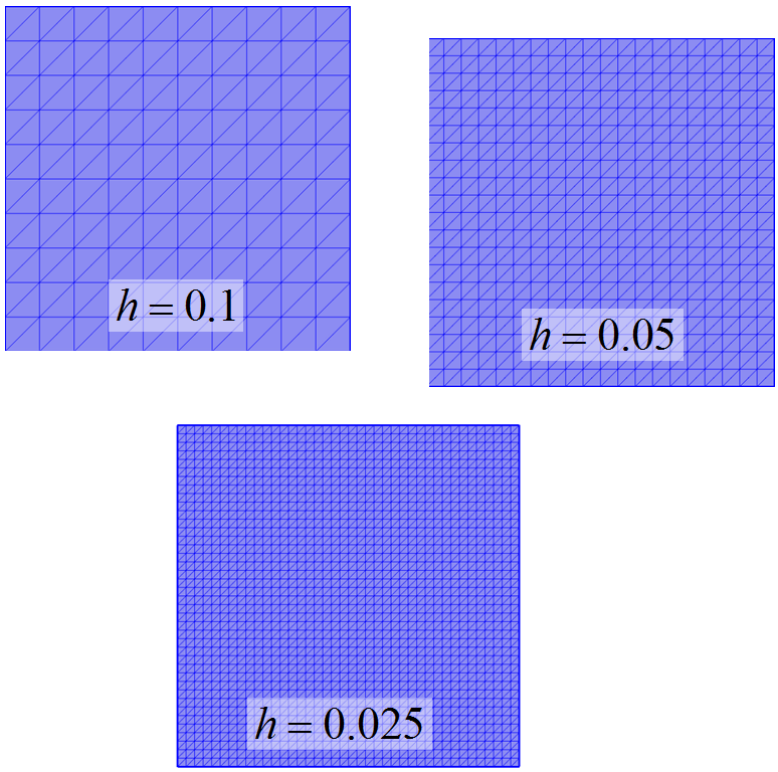

Figure 4.30: Three Meshes to Compare Approximate Functional Gains for the Same Problem

For a given mesh size, h, we define

$$u_{\mathbf{v}}^h(t) \triangleq -\int_\Omega k_{1v_1}^h(\mathbf{x}) v_1(\mathbf{x},t) - \int_\Omega k_{1v_2}^h(\mathbf{x}) v_2(\mathbf{x},t) - \int_\Omega k_{1T}^h(\mathbf{x}) T(\mathbf{x},t), \tag{4.3.19}$$

$$u_T^h(t) \triangleq -\int_\Omega k_{2v_1}^h(\mathbf{x}) v_1(\mathbf{x},t) - \int_\Omega k_{2v_2}^h(\mathbf{x}) v_2(\mathbf{x},t) - \int_\Omega k_{2T}^h(\mathbf{x}) T(\mathbf{x},t), \tag{4.3.20}$$

so that the functional gains associated with each state variable and control input are clear. We now depict side by side comparisons of the functional gain associated with each state variable and control input at varied mesh sizes (Figures 4.31-4.36). For this case, we set $\mathbf{R}$ as the identity multiplied by 0.1 and $\mathbf{Q} = \mathbf{M}$, the mass matrix.

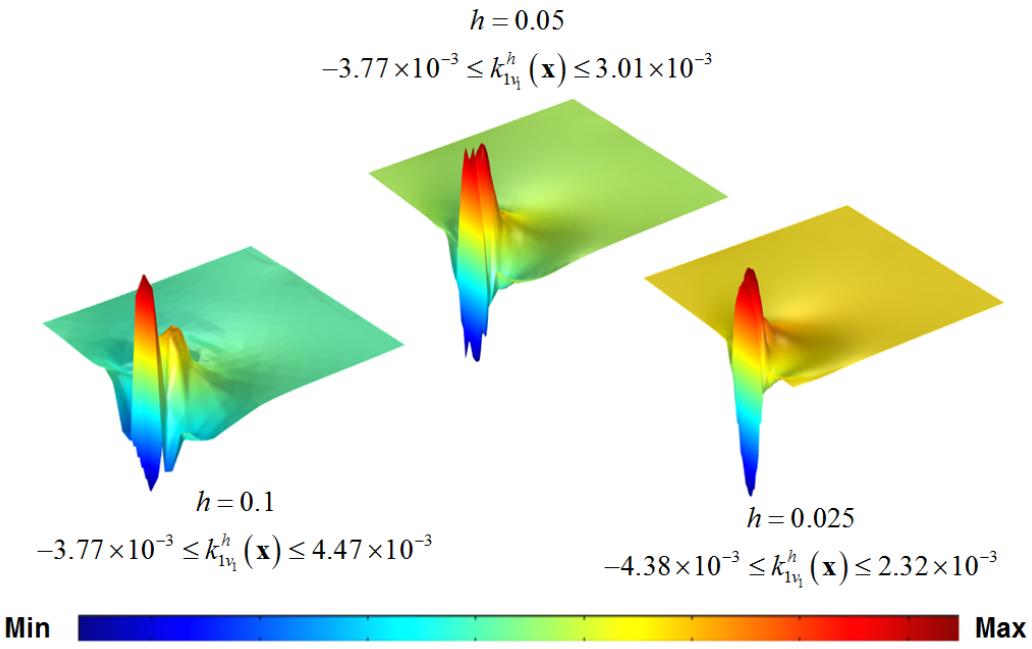

Figure 4.31: Functional Gain Convergence, $k^h_{1v_1}(\mathbf{x})$

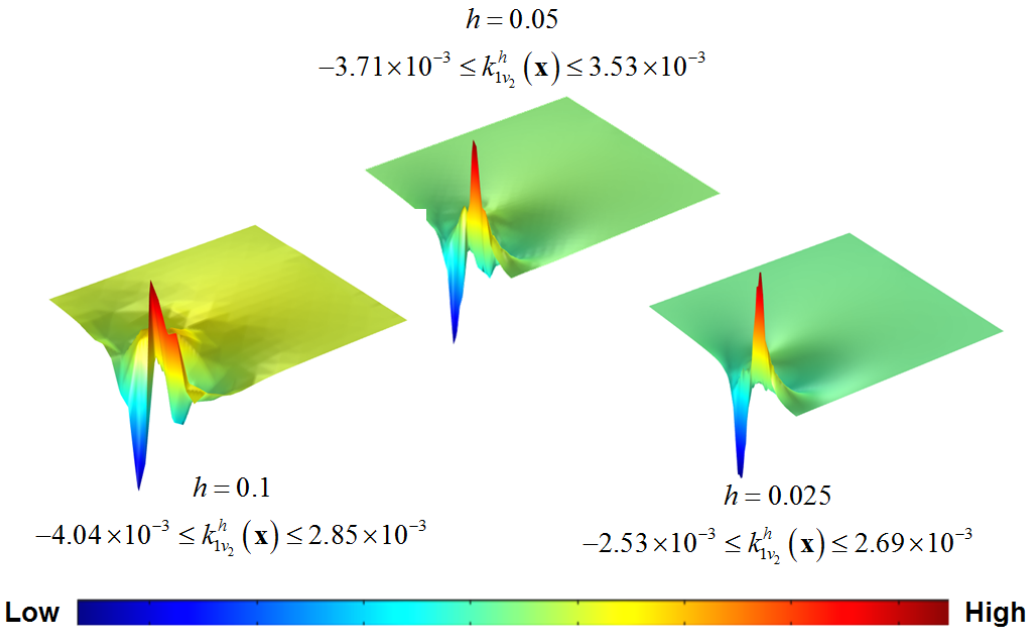

Figure 4.32: Functional Gain Convergence, $k^h_{1v_2}(\mathbf{x})$ Control Input, Kernel Function with v_2

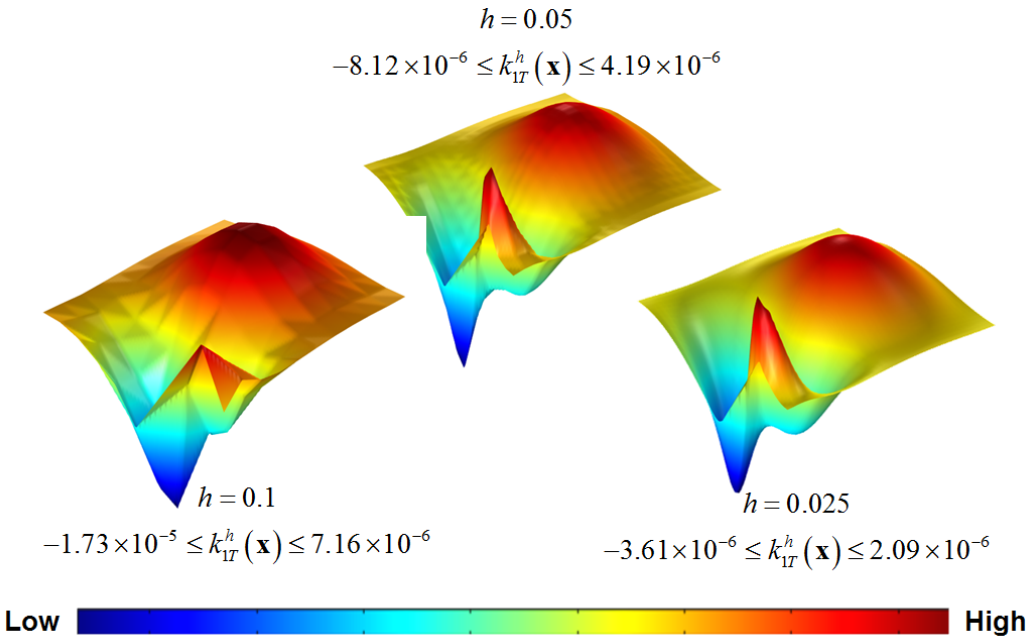

Figure 4.33: Functional Gain Convergence, $k_{1T}^{h}(\mathbf{x})$

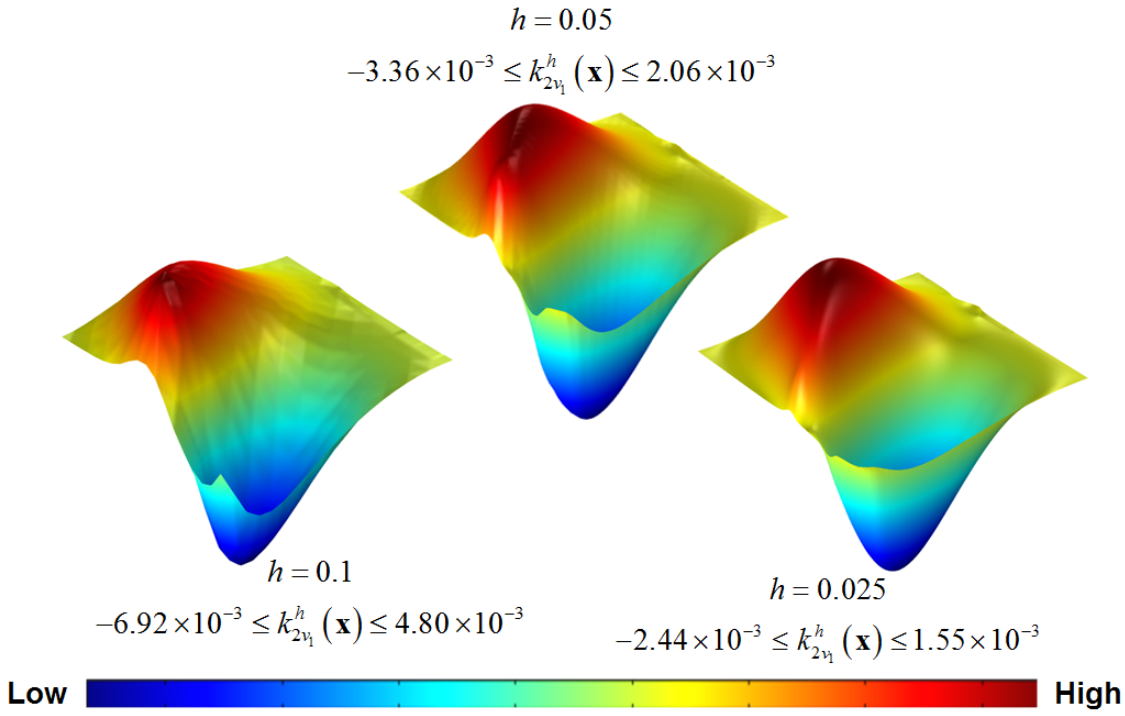

Figure 4.34: Functional Gain Convergence, $k_{2\mathbf{v}_1}^{h}(\mathbf{x})$

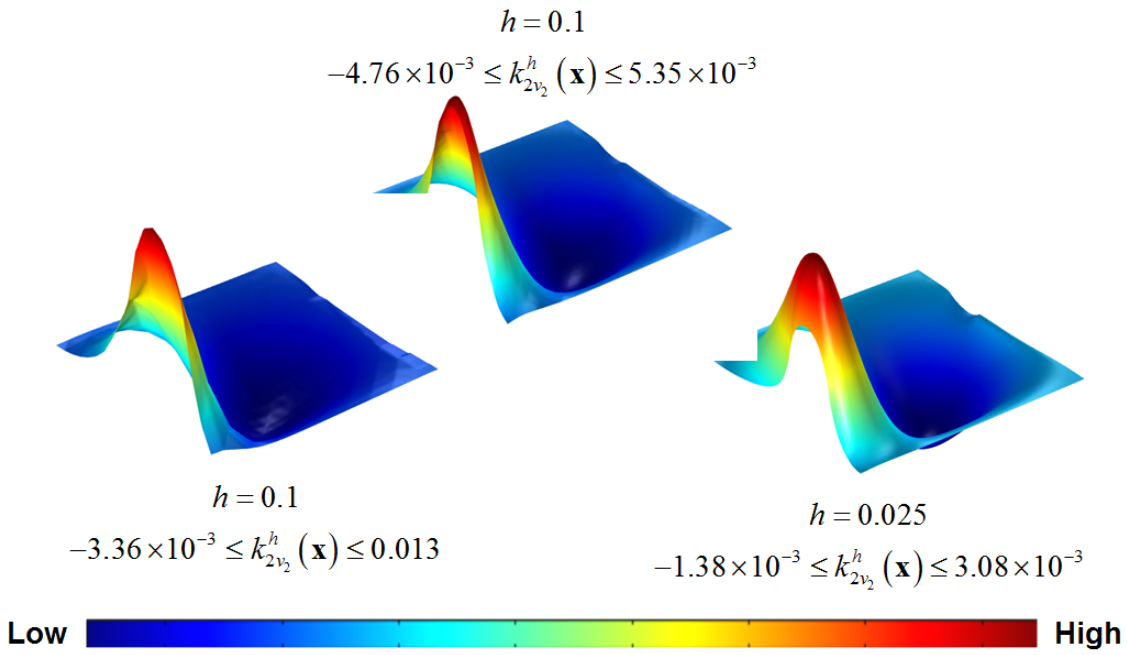

Figure 4.35: Functional Gain Convergence, $k^h_{2\mathbf{v}_2}(\mathbf{x})$

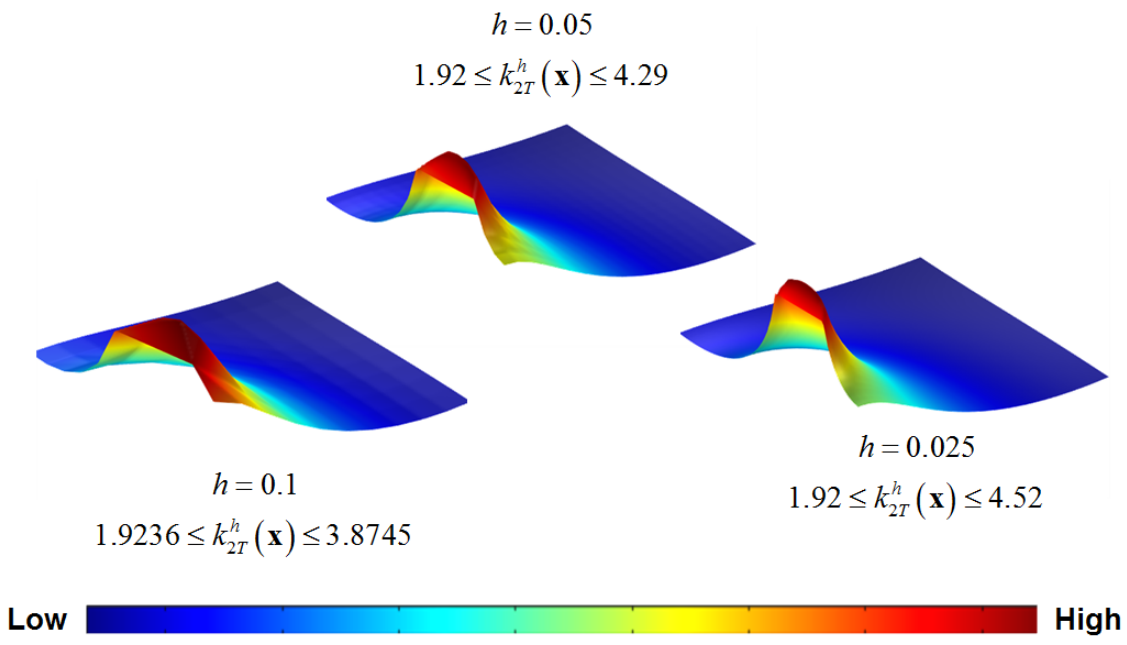

Figure 4.36: Functional Gain Convergence, $k^h_{2T}(\mathbf{x})$

The computational resources available for this thesis make a thorough convergence anal-

ysis of these functional gains impractical, but one can still make some basic observations. First, despite the mesh size, each of these kernel functions appear to take on the same basic shape, where maximum and minimum values tend to still be on the same order of magnitude. In some cases, Figure 4.36 for example, it is quite clear that the coarser mesh size fails to capture the steepness of the peak near the control portion of the boundary. Another observation is that the kernels associated with the velocity input are insignificant. This is likely due to the fact that the temperature gradient in the room generates such low velocities that little needs to be accomplished to control them. An obvious future study, given adequate computing resources, would be to determine if the same types of gains are observed using the true physical properties of air in a typical room at standard temperatures. This is likely not the case because of the fact that the thermal diffusivity of air is extremely small compared with our numerical experiments.

Uniform Initial Temperature

We now begin actual CFD simulations of the closed-loop system in COMSOL® using functional gains derived from the solutions of ARE's in MATLAB® . We use the same mesh $h = 0.05$ and conditions as those depicted in Section 4.3.1 and Figure 4.23, with one exception. In the present case we set the initial temperature at two units above the ideal "zero" rather than five units. We also designate the final time t_f as 40 vice 100. Figure 4.39 provides snapshots of the state evolution, with the kernel functions depicted in Figures 4.31-4.36. For reference, Figures 4.37-4.38 are snapshots of the system with zero control and constant kernal integral control, respectively.

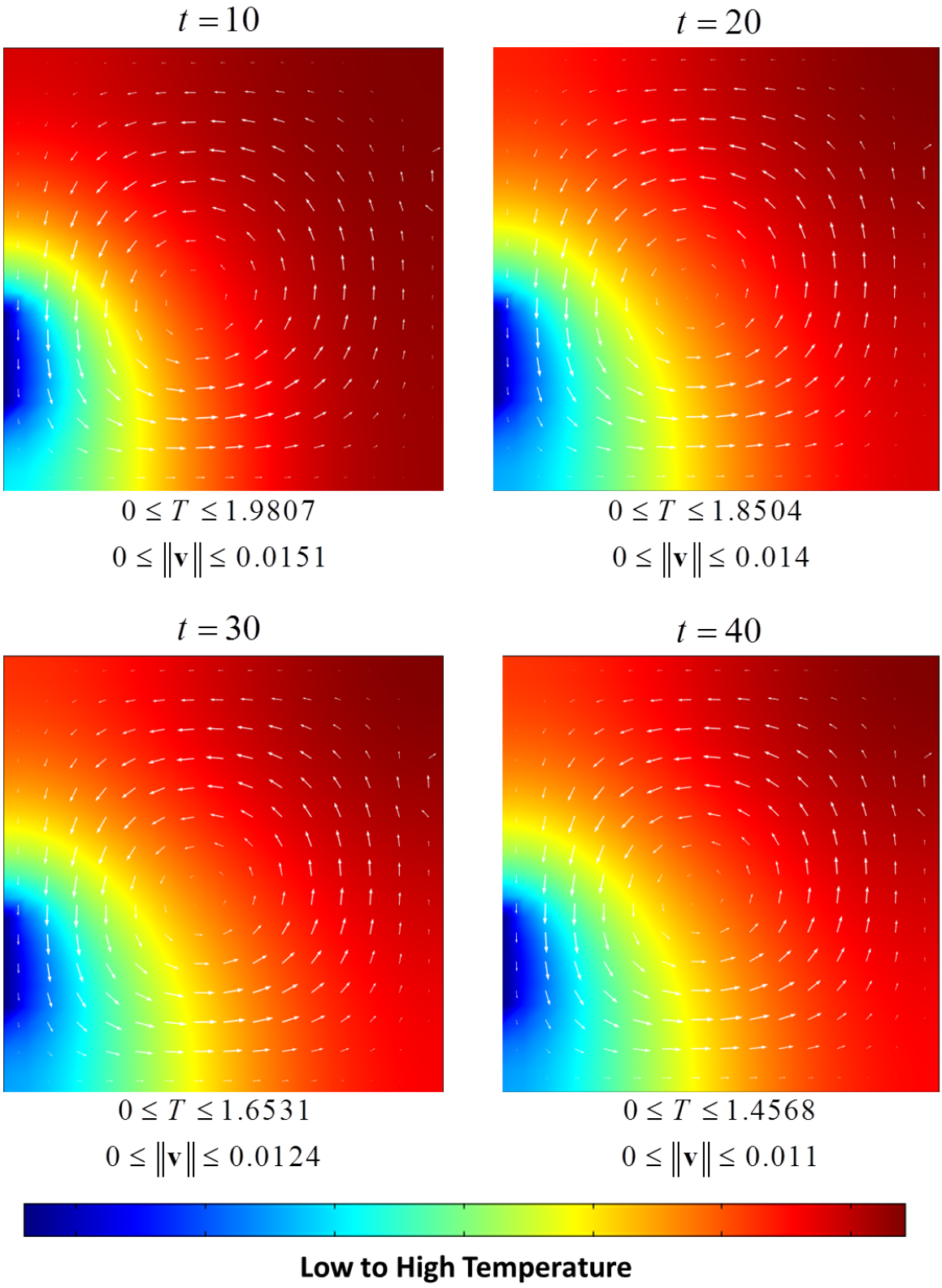

Figure 4.37: Temperature and Velocity Field with Zero Control

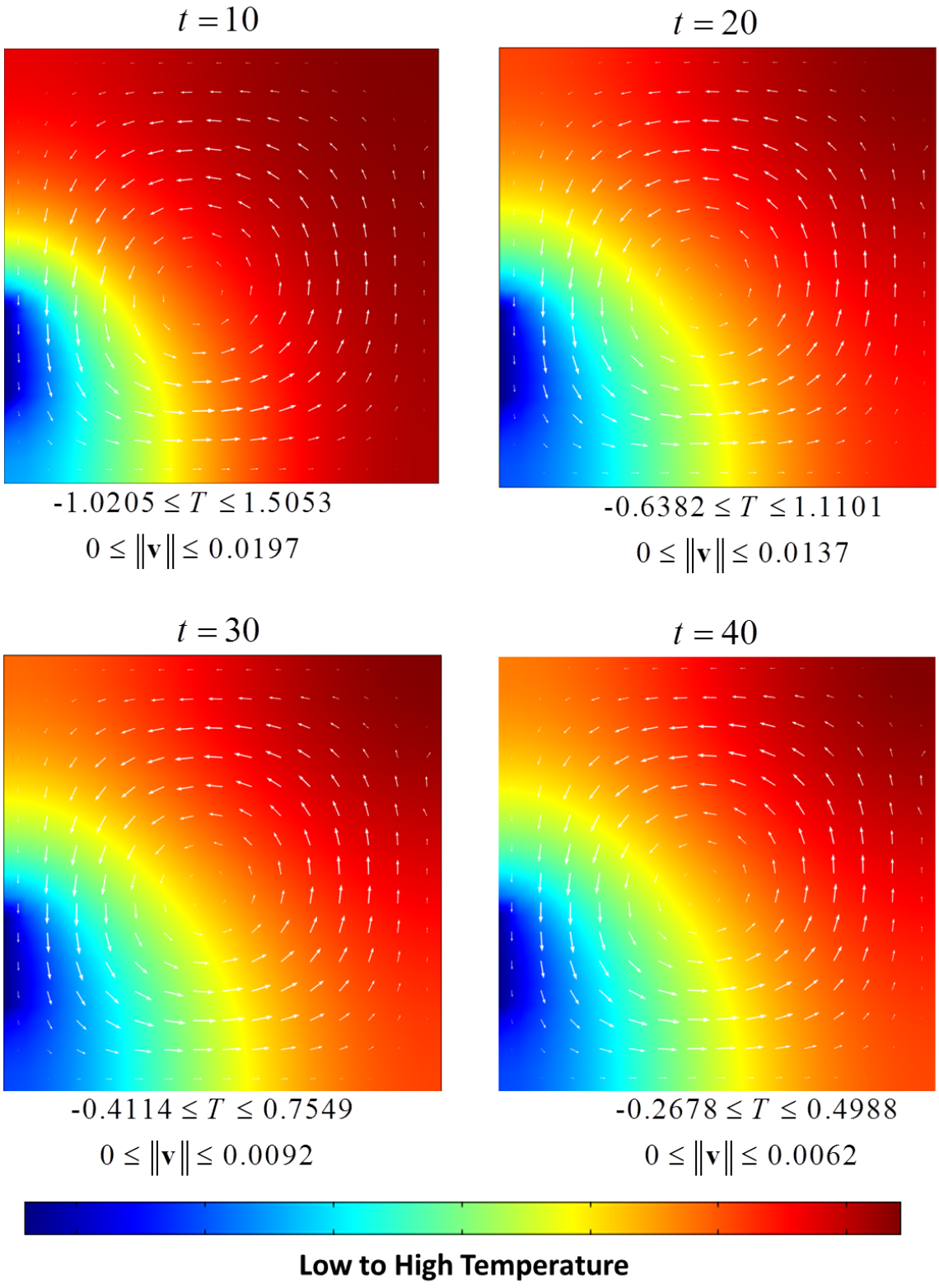

Figure 4.38: Temperature and Velocity Field with $u_T = -\int_{\Omega} T(\mathbf{x}, t)\, d\mathbf{x}$

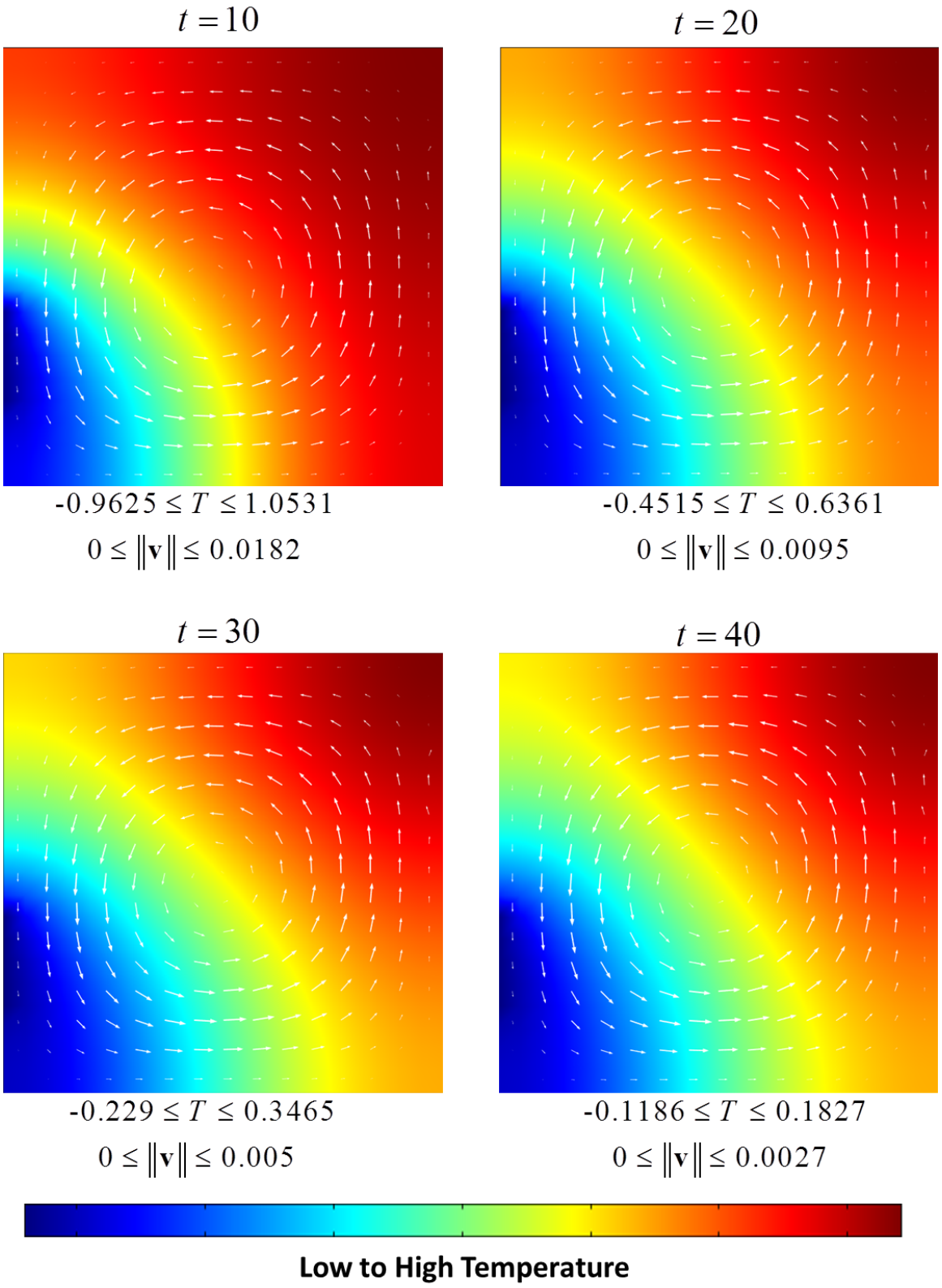

Figure 4.39: Temperature and Velocity Field with LQR control, Q = M, R = 0.1 I

As expected, the LQR controller, in this case, appears to outperform the average temperature (i.e. $k(\mathbf{x}) = 1$) feedback controller. In any discussion about energy efficiency for a real-world design scenario, the true energy costs of control would need to be balanced with how quickly a state is steered to "ideal" state.

We now experiment with varying the weights in the cost functional. We first reduce the penalty on control input input,

$$\mathbf{R} = \begin{pmatrix} 0.00001 & 0 \\ 0 & 0.00001 \end{pmatrix}. \tag{4.3.21}$$

Figure 4.40 depicts the corresponding approximate functional gains (again with mesh size $h = 0.05$). Figure 4.41 provides snapshots of the state evolution with this type of control. Here we see the results driving to steady state much faster than the other controllers surveyed thus far. This example also suggests that the system is mostly temperature-driven, probably due to the relatively low velocities induced by the buoyancy force.

The final experiment we carry out for weights in the cost functional is to continue with reduced penalty on the control input *and* and prescribe much greater weights for temperature than velocity, i.e.

$$\mathbf{R} = \begin{pmatrix} 0.00001 & 0 \\ 0 & 0.00001 \end{pmatrix}, \ \mathbf{Q} = \begin{pmatrix} 1 & 0 \\ 0 & 100 \end{pmatrix}, \tag{4.3.22}$$

See (3.4.2).

Figure 4.42 shows the functional gains that come from the discrete system where, again, $h = 0.5$. We note that by increasing cost of temperature disturbance, the temperature input kernel scales up. However, the velocity kernels associated with temperature actually scale down. Figure 4.42 provides time dependent snapshots of the solved discrete system.

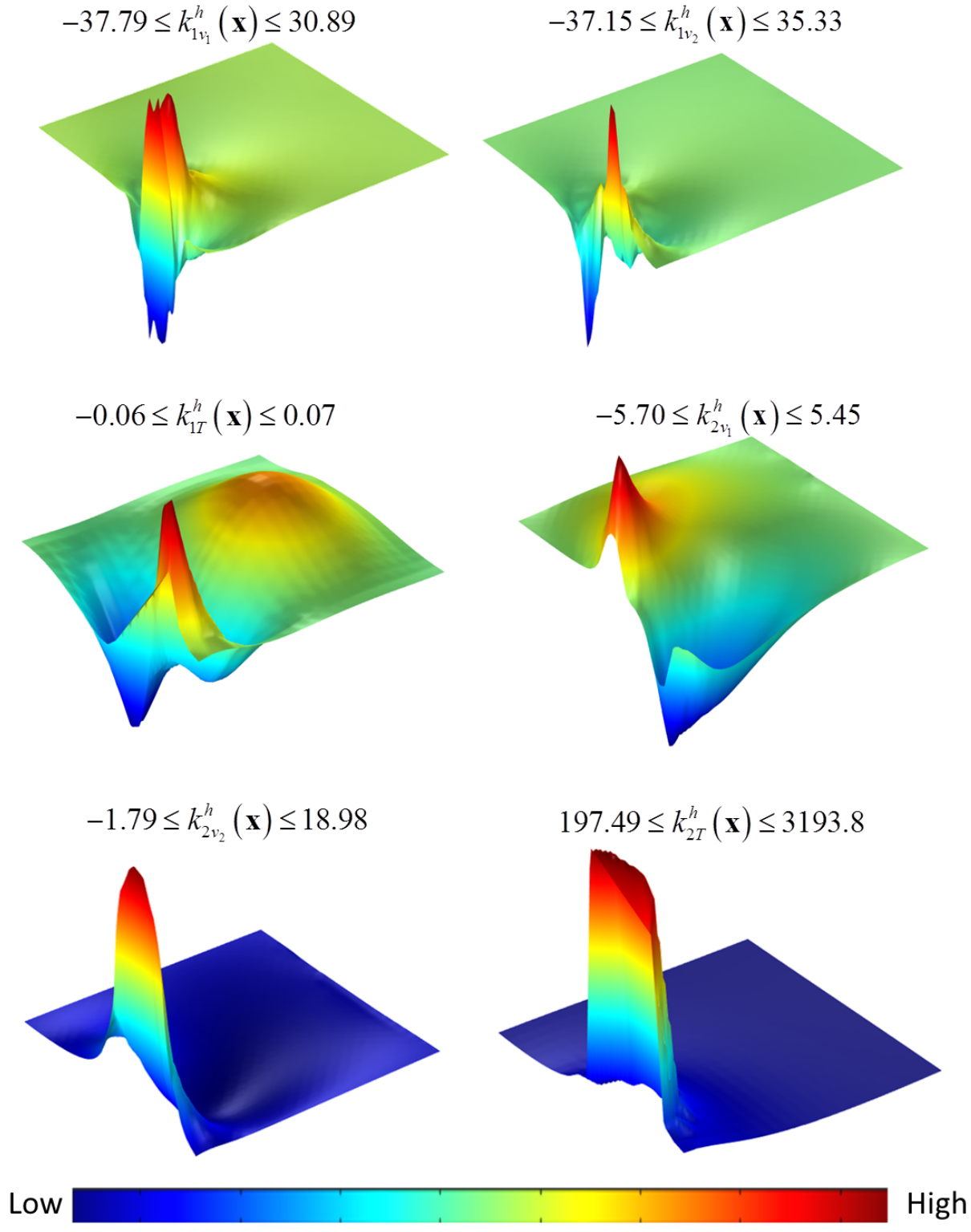

Figure 4.40: Functional Gains, Q = M, R = (0.00001)I

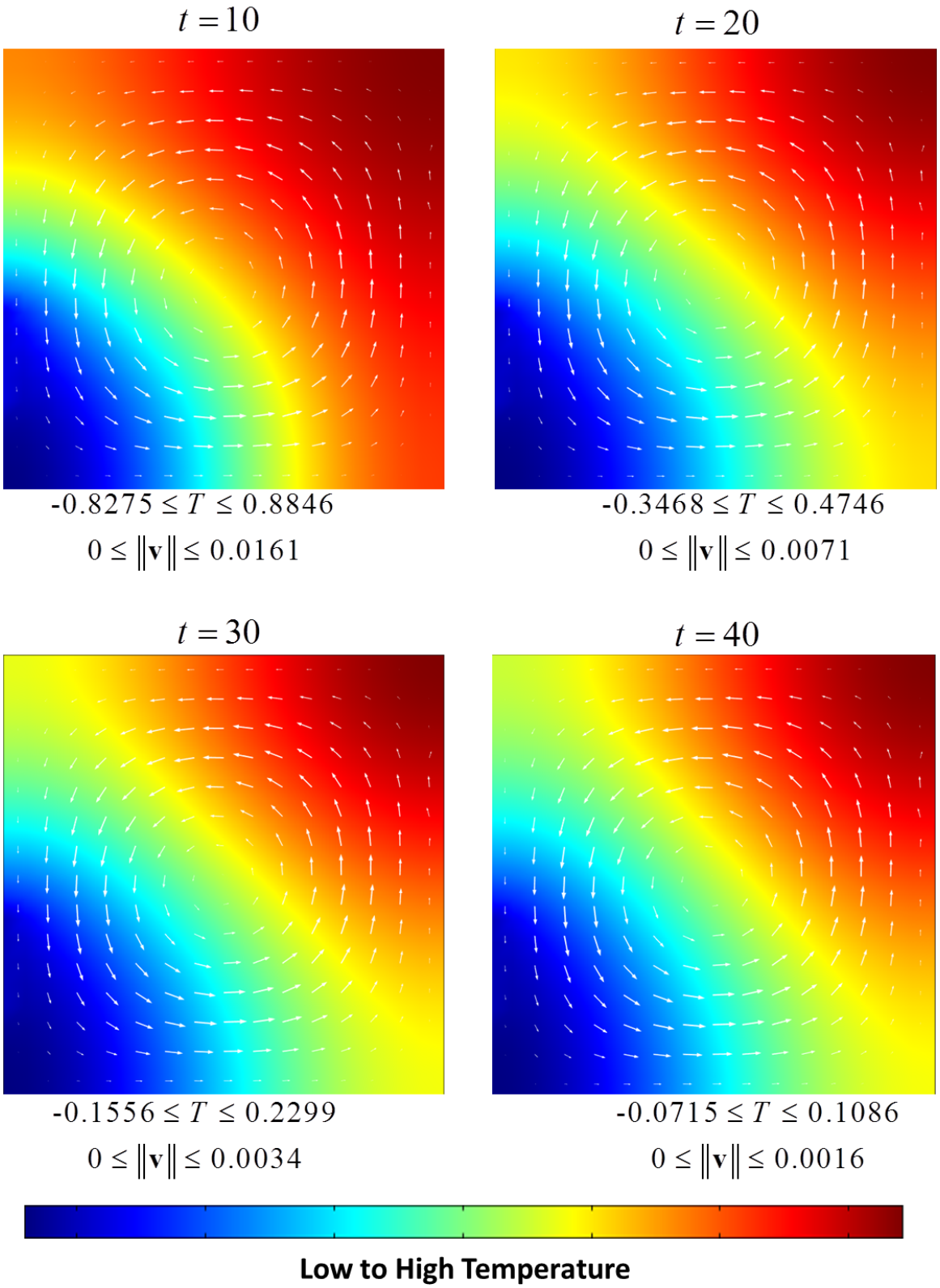

Figure 4.41: Time Evolution with LQR Control, Q = M, R = (0.00001)I

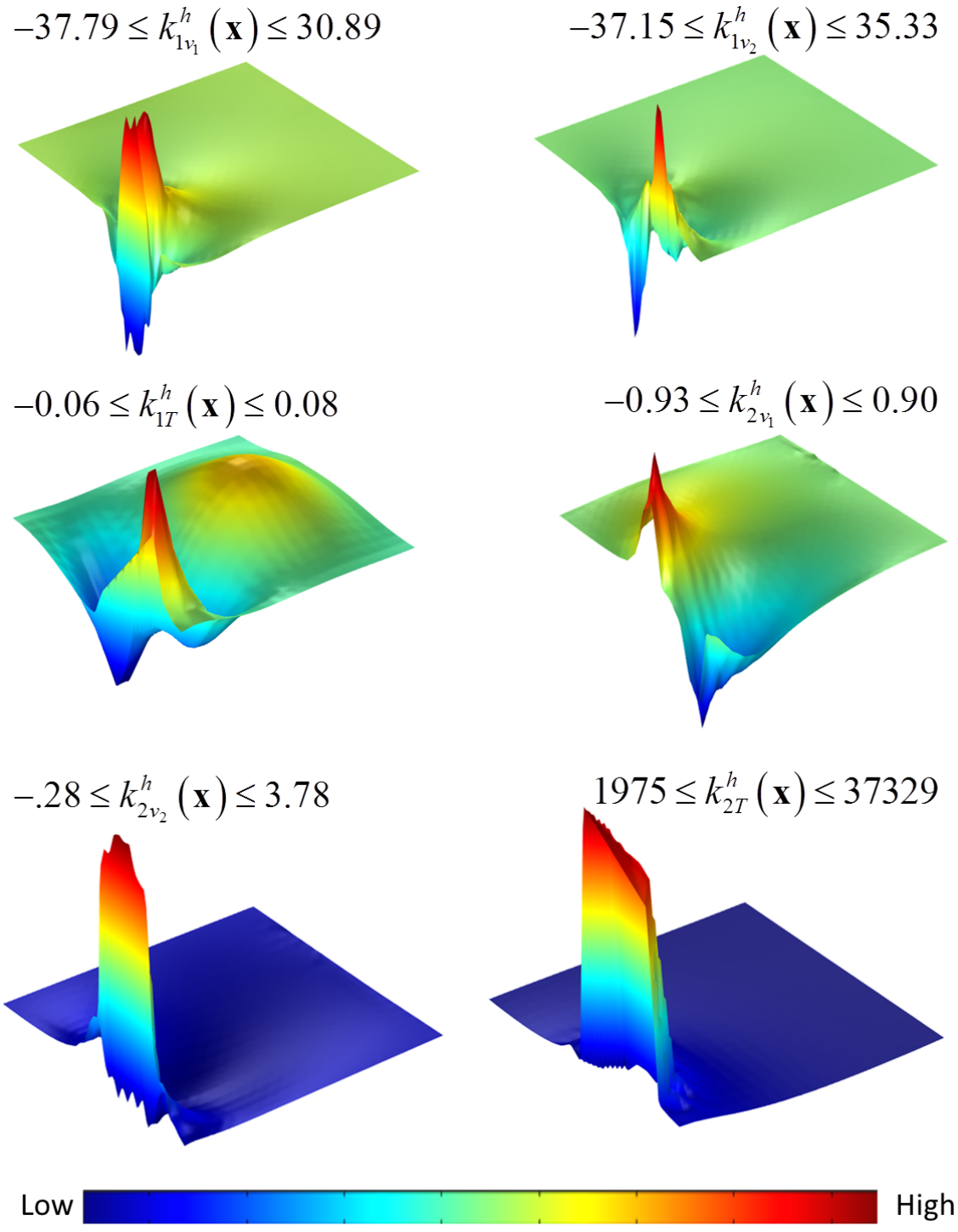

Figure 4.42: Functional Gains, See (4.3.22) for Q and R

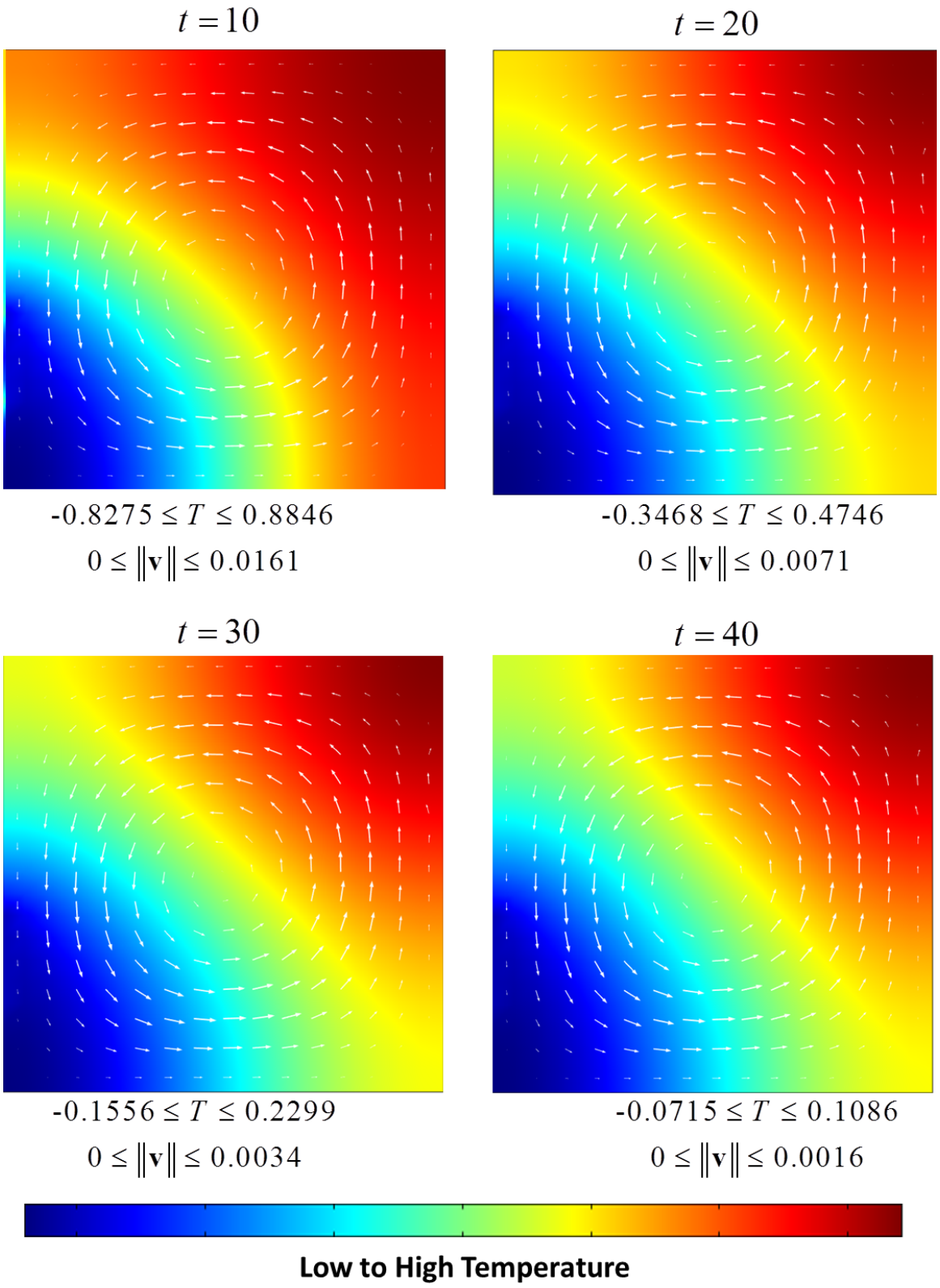

Figure 4.43: Time Evolution with LQR Control, See (4.3.22) for Q and R

Figure 4.44 is a graph depicting the L^2 norm of the open loop system, as well as all the closed-loop discussed above. As expected, the LQR controllers that were weighted according to (4.3.21) and (4.3.22) performed best when measuring average square temperature. However, the controller from (4.3.22) begins with a temperature over ten units lower than the "steady state". This is too costly for practical purposes where the goal is efficiency and not performance. Therefore, it is important that proper weights be assigned to the cost functions that realistically reflect, in this case, the entire design of the HVAC system (an area for future research).

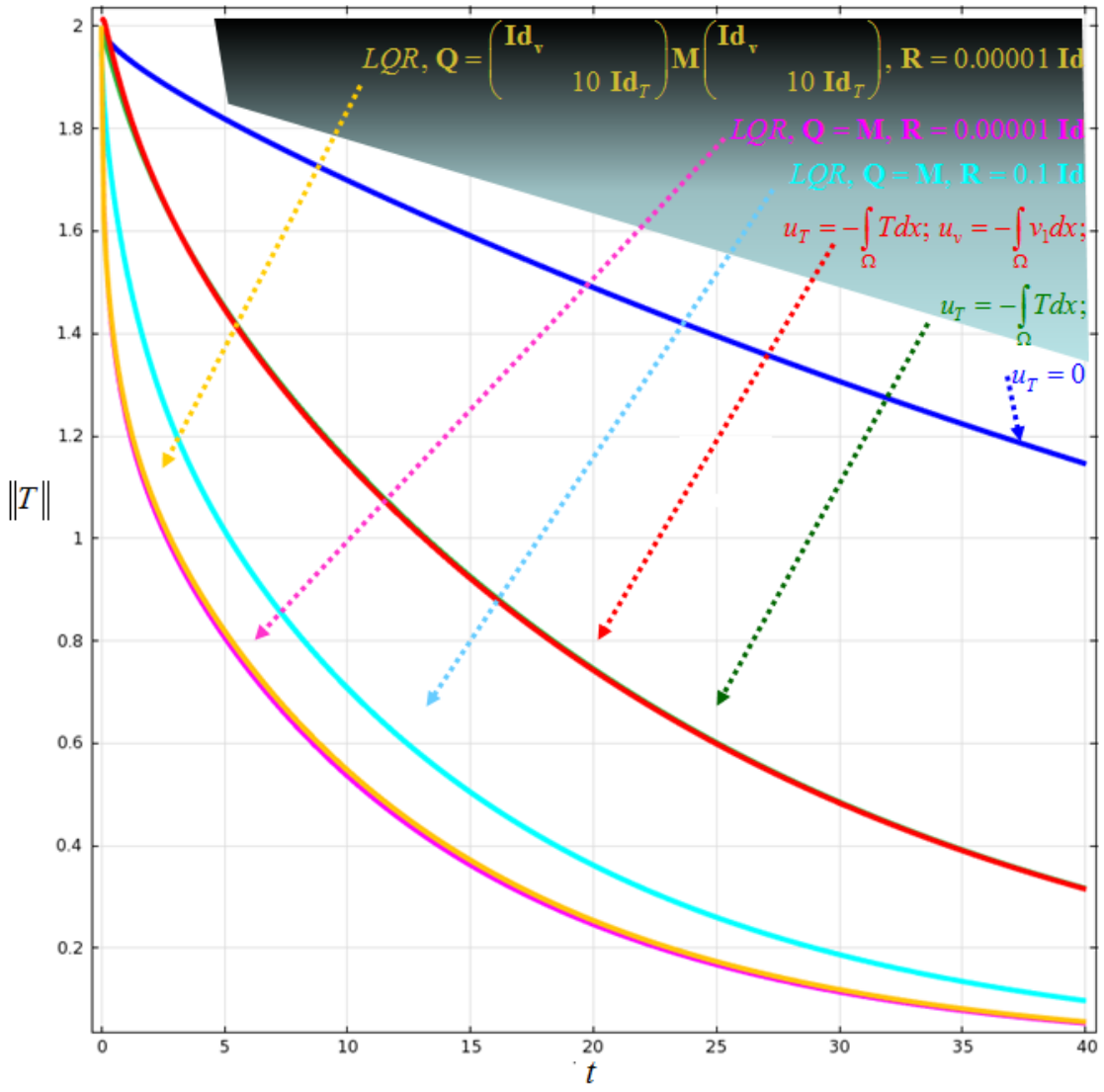

Figure 4.44: L^2 norm of Temperature over Time - Open Loop, Naive Feedback Control, LQR Control

Response to a Disturbance

Now using the feedback laws above, we experiment with control of a room that begins at ideal steady state, but undergoes a temporary boundary disturbance. For a proof-of-concept, we use the same portion of the boundary for this disturbance as in Figure 4.4 where we prescribe a heat flux that smoothly raises and lowers over time using the function (see Figure 4.45)

$$e^{-\frac{1}{4}(t-6)^2}. \tag{4.3.23}$$

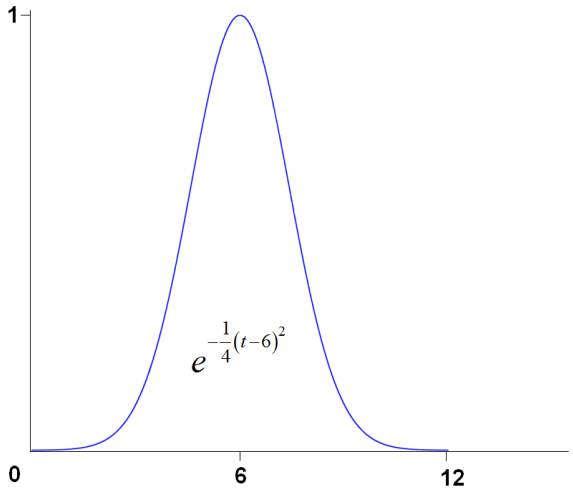

Figure 4.45: Time-only-dependent Disturbance Function

Here, we compare the zero-control system with the LQR control system from (4.3.22) (see Figures 4.46 and 4.47). Figure 4.48 provides an L^2-norm comparison. The ability to quickly add disturbances (not just boundary, but point or area disturbances as well) to a COMSOL® simulation and test an existing control law demonstrates a possible advantage of this methodology compared to self-generated code.

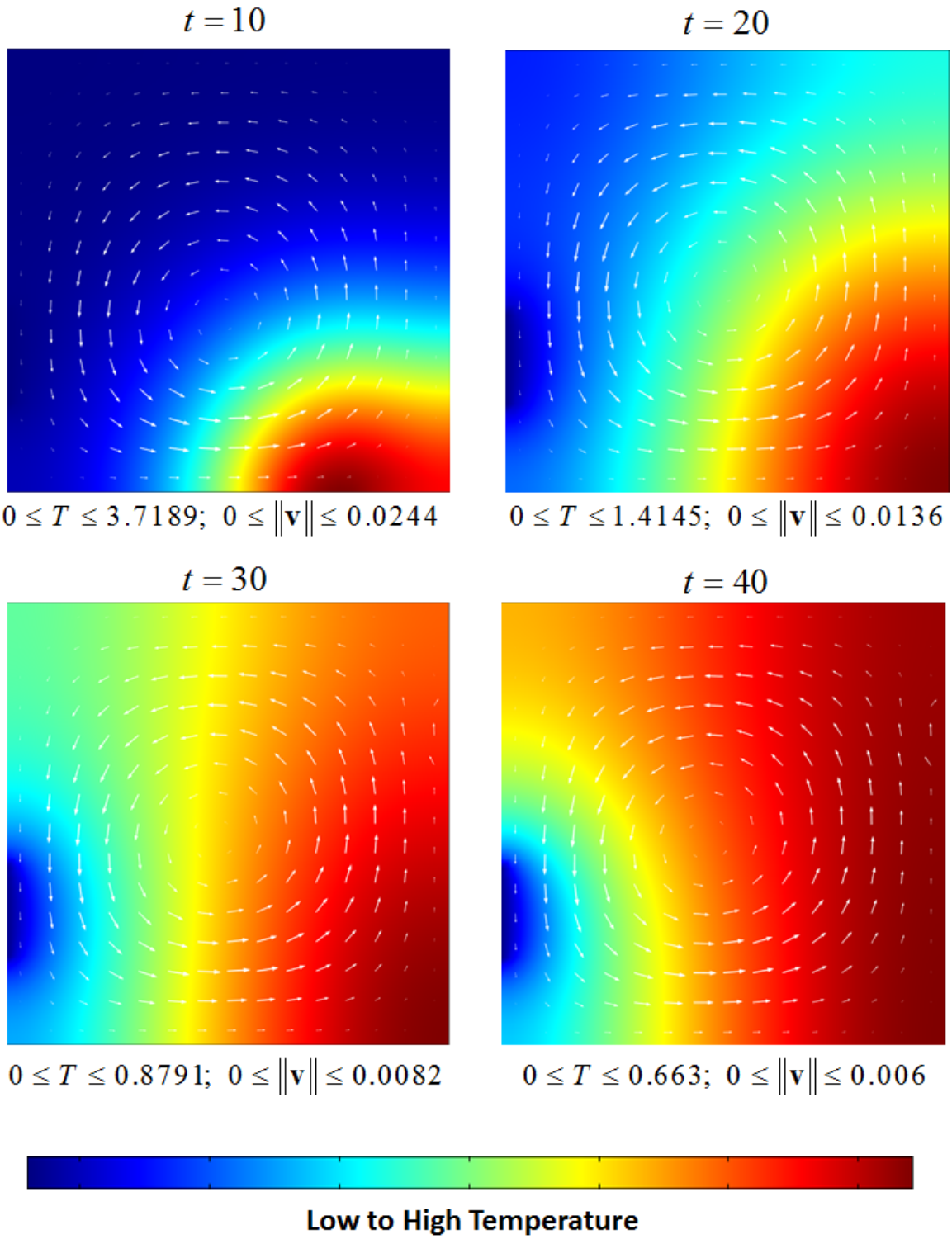

Figure 4.46: Open-Loop System with Disturbance

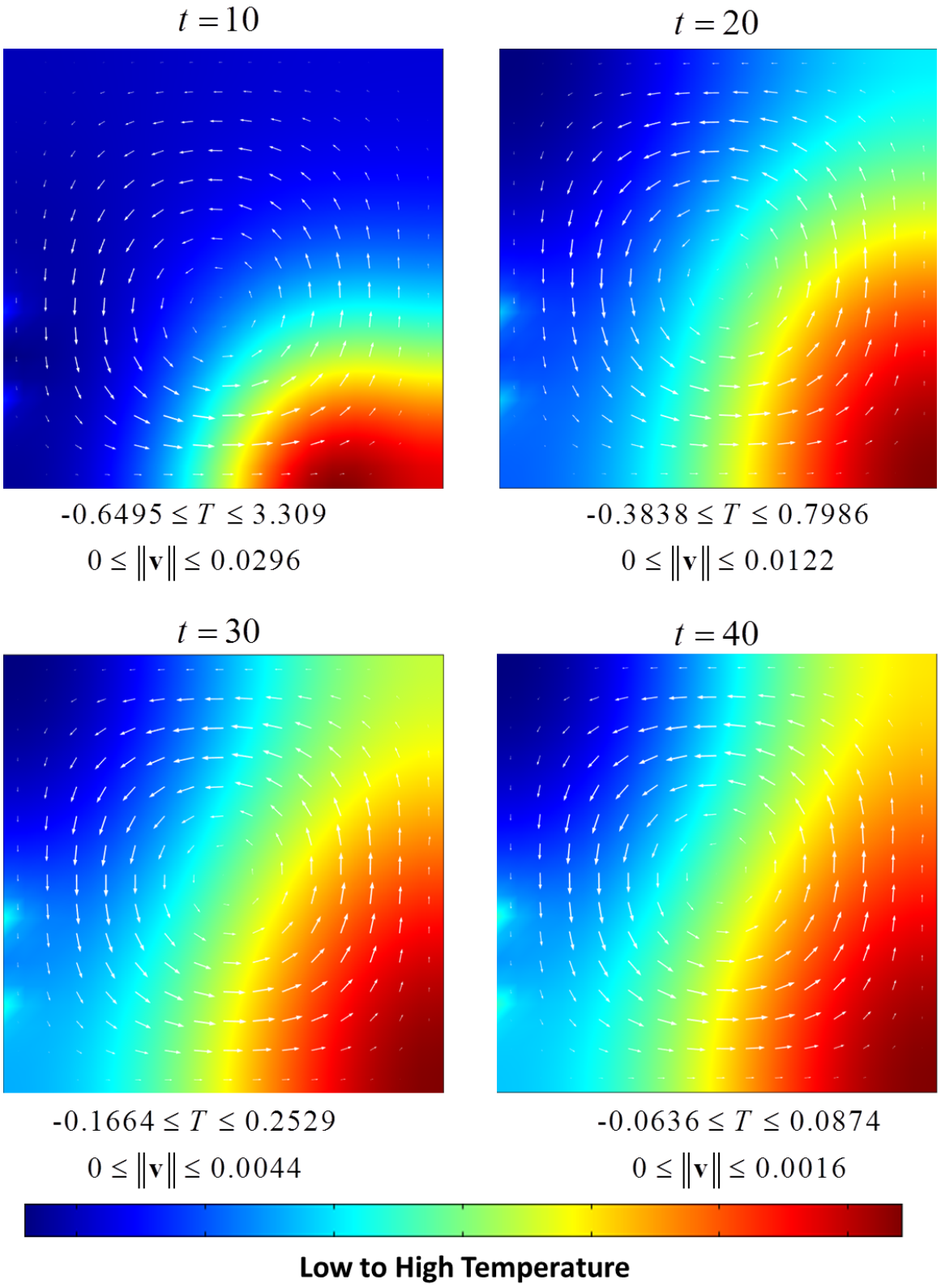

Figure 4.47: Closed-Loop System with Disturbance

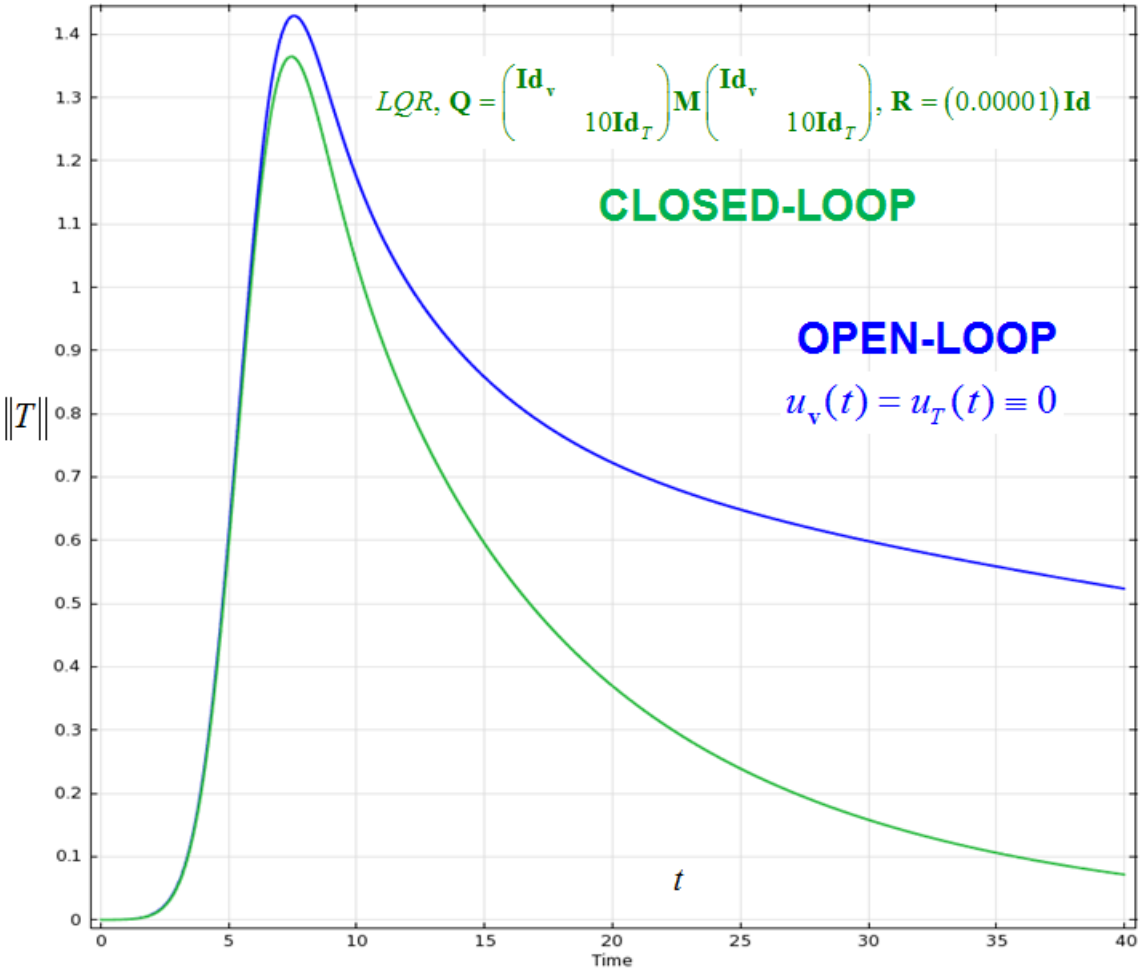

Figure 4.48: L^2 Comparison: Open-Loop versus LQR Closed-Loop

Summary

In summary, we have demonstrated how it is possible to use COMSOL® and MATLAB® to build finite-dimensional systems for controller design and then use COMSOL® to validate the design. As expected, the control obtained by reducing the penalty on input control creates large improvement in steering the temperature state to zero, with the caveat that executing the resulting control may be more expensive overall if not impossible. We've also seen the effectiveness of applying one of the same LQR controllers to removing the effects of a disturbance. All of these test cases were conducted using unrealistic parameters for the physical circumstances of a real-world "One-Room Model Problem," but they do provide proof that at least some level of control design and testing is possible between COMSOL® and MATLAB® .

Chapter 5

Conclusions and Future Work

This thesis was specifically written to support current research efforts in reducing the energy costs associated with building HVAC systems. Regardless of how complicated the inner workings of a specific HVAC system in a building might be, understanding how to reduce energy consumption should begin with understanding how air is heated, cooled and circulated at the room level, which is really an applied study of fluid dynamics, an extremely mature field of research. Therefore, not only should existing science and engineering typically applied to *other* problems in fluid dynamics be gleaned, but any mature commercial software designed to perform simulations in fluid dynamics should be assessed and leveraged to the maximum extent possible to facilitate understanding complex building systems. In this thesis, we began the process of such an assessment by investigating the applicability of combining commercial CFD sofware, COMSOL® , with more general computational software (MATLAB®) to produce a control design for a one room model problem.

We first specified appropriate physical and mathematical continuum models associated with the dynamics of room heating, cooling and air flow. In this vein, we reviewed the physical and mathematical framework that leads to the derivation of the Boussinesq Equations, coupling heat energy with fluid momentum. We introduced a simplified notion of boundary control on this system of equations as well as an energy cost to be minimized or at least reduced via this boundary control. Then, we demonstrated how this physical model can be approximated using the Finite Element Method (FEM). In this vein, we made a careful study of the COMSOL® implementation of this discretization method, to ensure that any external computations to design controls would be in harmony with COMSOL® methodologies. We explained the COMSOL® methodology for handling system boundary conditions as constraints and described the COMSOL® linearization scheme. We then developed step-by-step procedures between COMSOL® and MATLAB® to extract adequate information to form the classical "state system" with its associated state vector data to solve the ARE as a solution to the LQR problem. This included a novel approach to implement the "penalty method" in order to obtain a reduced system with a nonsingular mass matrix.

Finally we conducted several numerical studies using COMSOL® for open-loop and closed-loop simulation. Also, we used COMSOL® for closed-loop simulations using functional gains

obtained from MATLAB.® We considered two different approaches to input these functional gains into COMSOL® as part of boundary feedback control.

In the process of developing computational techniques to implement LQR control between COMSOL® and MATLAB® , some limiting factors were discovered.

First, the only way to obtain consistent reliable data in MATLAB® with which to design controllers is to manually input the relevant equations using the COMSOL® "Mathematics" interface to enter coupled "Coefficient Form PDE's". One therefore loses much of the advantage of using the COMSOL® physics modules with their own built-in advanced capabilities to automatically couple multiple physics models, and set the problems up to achieve maximum solvability (e.g. for CFD; use automatic boundary layer creation, streamline diffusion, cross-wind diffusion, etc.). When these "black-box" features were used, the matrices extracted in MATLAB® were not readily useful for control design.

Second, using MATLAB® to solve an ARE creates a computational bottleneck. To solve an ARE with matrices based on 16000 independent variables completely taxes a scientific workstation with 24 processors and 48 GB RAM for hours. A possible effort for future work would be to find a more efficient ARE solver that can import data from MATLAB® and export the solution back to MATLAB® . In particular, one should take advantage of the research by Benner, Mehrmann et al. (see, for example, [7], [8], [9], [10]), [11], [27]) to develop ARE solvers for large-scale systems.

Despite these limiting factors, the COMSOL® and MATLAB® interface remains worthy of continued exploration. An area ripe for future research is the capability to entirely run, and even build, a COMSOL® model in MATLAB.® If this capability includes the ability to access and modify all of the COMSOL® - assembled state space matrices with MATLAB® functions, one might begin to explore finite-horizon LQR as well as other optimization techniques. One can also purchase a separate COMSOL® "CFD" module, which includes multiple ways to deal with turbulent flow. If these techniques can be completely understood at the equation and data level, they might be exploited to help provide good results for truly realistic physical parameters in three-dimensional domains. Finally, John and Wachsmuth [26] developed a technique for including an entire optimality system (including adjoint equations) into COMSOL® for the Navier-Stokes equations. Similar procedures for the Boussinesq equations are probably within reach.

Other control problems are worth investigating. For example, one could add an extra boundary control parameter for the vertical component of inlet velocity allowing for experimentation with control of airflow strength *and* direction. This extra parameter would allow for direct control of the vents. Another immediate future project would be to change the steady state to incorporate a positive velocity at the inlet. This would represent a typical ventilation requirement for continuous air flow into and out of a room and would provide a means to better analyze the functional gains associated with velocity control. Next, these control techniques still need to be applied with realistic physical parameters and in three-dimensional domains. This will require computational resources on a large scale (supercomputer), or application of some type of reduced order modeling that captures the essence of most of the near-term state components. An observer could be implemented as

well. For this purpose, COMSOL® has the capability to include state probes (observers) in a given model that will provide for the generation of output system matrices as well as the classical state system matrices. Finally, especially in three dimensions, much more work can be done in modeling the dynamics of lights, electronic equipment, windows, people, doors opening and shutting, non-perfect insulation, etc.

Bibliography

[1] Air Force instruction 90-1701, energy management, 2009.

[2] B.D.O. Anderson and J.B. Moore. *Optimal control: linear quadratic methods*. Prentice-Hall Englewood Cliffs (NJ):, 1989.

[3] M. Badra. Abstract settings for stabilization of nonlinear parabolic system with a Riccati-based strategy. Application to Navier-Stokes and Boussinesq equations with Neumann or Dirichlet control. 2010.

[4] V. Barbu, I. Lasiecka, and R. Triggiani. Local exponential stabilization strategies of the Navier-Stokes equations, d= 2, 3, via feedback stabilization of its linearization. *Control of coupled partial differential equations*, pages 13–46, 2007.

[5] Viorel Barbu, Irena Lasiecka, and Roberto Triggiani. Abstract settings for tangential boundary stabilization of Navier-Stokes equations by high- and low-gain feedback controllers. *Nonlinear Anal.*, 64(12):2704–2746, 2006.

[6] Viorel Barbu, Irena Lasiecka, and Roberto Triggiani. Tangential boundary stabilization of Navier-Stokes equations. *Mem. Amer. Math. Soc.*, 181(852):x+128, 2006.

[7] S. Barrachina, P. Benner, E.S. Quintana-Orti, and G. Quintana-Ortí. Parallel algorithms for balanced truncation of large-scale unstable systems. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 2248–2253. IEEE, 2005.

[8] P. Benner, R. Byers, V. Mehrmann, and H. Xu. Robust numerical methods for robust control. *Preprint*, 6, 2004.

[9] P. Benner, S. Gorner, and J. Saak. Numerical solution of optimal control problems for parabolic systems. *Parallel Algorithms and Cluster Computing*, pages 151–169, 2006.

[10] P. Benner, V. Hernández, and A. Pastor. The Kleinman iteration for nonstabilizable systems. *Mathematics of Control, Signals, and Systems (MCSS)*, 16(1):76–93, 2003.

[11] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving linear-quadratic optimal control problems on parallel computers. *Optimisation Methods & Software*, 23(6):879–909, 2008.

[12] Alain Bensoussan, Giuseppe Da Prato, Michel C. Delfour, and Sanjoy K. Mitter. *Representation and control of infinite dimensional systems*. Systems & Control: Foundations & Applications. Birkhäuser Boston Inc., Boston, MA, second edition, 2007.

[13] Dietrich Braess. *Finite elements*. Cambridge University Press, Cambridge, third edition, 2007. Theory, fast solvers, and applications in elasticity theory, Translated from the German by Larry L. Schumaker.

[14] J. A. Burns, B. B. King, and D. Rubio. Feedback control of a thermal fluid using state estimation. *Int. J. Comput. Fluid Dyn.*, 11(1-2):93–112, 1998. Flow control and optimization.

[15] Ruth F. Curtain and Hans Zwart. *An introduction to infinite-dimensional linear systems theory*, volume 21 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 1995.

[16] Charles R. Doering and J. D. Gibbon. *Applied analysis of the Navier-Stokes equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 1995.

[17] Howard C. Elman, David J. Silvester, and Andrew J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2005.

[18] R.Z. Freire, G.H.C. Oliveira, and N. Mendes. Predictive controllers for thermal comfort optimization and energy savings. *Energy and buildings*, 40(7):1353–1365, 2008.

[19] M. D. Gunzburger and S. Manservisi. The velocity tracking problem for Navier-Stokes flows with boundary control. *SIAM J. Control Optim.*, 39(2):594–634 (electronic), 2000.

[20] Max D. Gunzburger. *Finite element methods for viscous incompressible flows*. Computer Science and Scientific Computing. Academic Press Inc., Boston, MA, 1989. A guide to theory, practice, and algorithms.

[21] Michael Hinze and Ulrich Matthes. Optimal and model predictive control of the Boussinesq approximation. In *Control of coupled partial differential equations*, volume 155 of *Internat. Ser. Numer. Math.*, pages 149–174. Birkhäuser, Basel, 2007.

[22] http://zeb.buildinggreen.com/. Zero energy buildings: Zero energy buildings database.

[23] Weiwei Hu, John A. Burns, and Xiaoming He. Feedback stabilization of Boussinesq equations and approximations. *Preprint, Interdisciplenary Center for Applied Mathematics, Virginia Tech*, 2011.

[24] K. Ito and S. S. Ravindran. Optimal control of thermally convected fluid flows. *SIAM J. Sci. Comput.*, 19(6):1847–1869 (electronic), 1998.

[25] K. Ito and J. D. Schroeter. Reduced order feedback synthesis for viscous incompressible flows. *Math. Comput. Modelling*, 33(1-3):173–192, 2001. Computation and control, VI (Bozeman, MT, 1998).

[26] C. John and D. Wachsmuth. Optimal dirichlet boundary control of Navier-Stokes equations with state constraint. *Numerical Functional Analysis and Optimization*, 2009.

[27] P. Kunkel, V. Mehrmann, and W. Rath. Analysis and numerical solution of control problems in descriptor form. *Mathematics of Control, Signals, and Systems (MCSS)*, 14(1):29–61, 2001.

[28] H. Kwakernaak and R. Sivan. *Linear optimal control systems*, volume 188. Wiley-Interscience New York, 1972.

[29] Irena Lasiecka and Roberto Triggiani. *Control theory for partial differential equations: continuous and approximation theories. I*, volume 74 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2000. Abstract parabolic systems.

[30] William Layton. *Introduction to the numerical analysis of incompressible viscous flows*, volume 6 of *Computational Science & Engineering*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008. With a foreword by Max Gunzburger.

[31] EB Lee and L. Markus. Foundations of optimal control. *SIAM, Philadelphia*, 1967.

[32] Hyung-Chun Lee. Analysis and computational methods of Dirichlet boundary optimal control problems for 2D Boussinesq equations. *Adv. Comput. Math.*, 19(1-3):255–275, 2003. Challenges in computational mathematics (Pohang, 2001).

[33] Hyung-Chun Lee and Youngmi Choi. Analysis and approximation of linear feedback control problems for the Boussinesq equations. *Comput. Math. Appl.*, 51(5):829–848, 2006.

[34] Hyung-Chun Lee and Soohyun Kim. Finite element approximation and computations of optimal Dirichlet boundary control problems for the Boussinesq equations. *J. Korean Math. Soc.*, 41(4):681–715, 2004.

[35] Hyung-Chun Lee and Byeong Chun Shin. Piecewise optimal distributed controls for 2D Boussinesq equations. *Math. Methods Appl. Sci.*, 23(3):227–254, 2000.

[36] J.-L. Lions. *Optimal control of systems governed by partial differential equations.* Translated from the French by S. K. Mitter. Die Grundlehren der mathematischen Wissenschaften, Band 170. Springer-Verlag, New York, 1971.

[37] J.L. Lions. *Control of distributed singular systems.* Gauthier-Villars, 1985.

[38] W. Lu, A.T. Howarth, and A.P. Jeary. Prediction of airflow and temperature field in a room with convective heat source. *Building and environment*, 32(6):541–550, 1997.

[39] J.-P. Raymond. Feedback boundary stabilization of the three-dimensional incompressible Navier-Stokes equations. *J. Math. Pures Appl. (9)*, 87(6):627–669, 2007.

[40] Jean-Pierre Raymond. Feedback boundary stabilization of the two-dimensional Navier-Stokes equations. *SIAM J. Control Optim.*, 45(3):790–828 (electronic), 2006.

[41] Jean-Pierre Raymond and Laetitia Thevenet. Boundary feedback stabilization of the two dimensional Navier-Stokes equations with finite dimensional controllers. *Discrete Contin. Dyn. Syst.*, 27(3):1159–1187, 2010.

[42] J. N. Reddy and D. K. Gartling. *The finite element method in heat transfer and fluid dynamics*. CRC Press, Boca Raton, FL, second edition, 2001.

[43] Michael Renardy and Robert C. Rogers. *An introduction to partial differential equations*, volume 13 of *Texts in Applied Mathematics*. Springer-Verlag, New York, second edition, 2004.

[44] D. Rubio. *Distributed parameter control of thermal fluids*. PhD thesis, Virginia Polytechnic Institute and State University, 1997.

[45] S.L. Sinha, R.C. Arora, and S. Roy. Numerical simulation of two-dimensional room air flow with and without buoyancy. *Energy and buildings*, 32(1):121–129, 2000.

[46] S.S. Sritharan. *Invariant manifold theory for hydrodynamic transition*. Longman Scientific & Technical (Harlow, Essex, England and New York), 1990.

[47] S.S. Sritharan. *Optimal control of viscous flow*, volume 59. Society for Industrial Mathematics, 1998.

[48] Robert F. Stengel. *Optimal control and estimation*. Dover Publications Inc., New York, 1994. Corrected reprint of the 1986 original.

[49] Gilbert Strang and George Fix. *An analysis of the finite element method*. Wellesley-Cambridge Press, Wellesley, MA, second edition, 2008.

[50] Shu-Ming Sun. Partial differential equations lecture notes. 2010.

[51] B. Tashtoush, M. Molhim, and M. Al-Rousan. Dynamic model of an HVAC system for control analysis. *Energy*, 30(10):1729–1745, 2005.

[52] Roger Temam. *Navier-Stokes equations*. AMS Chelsea Publishing, Providence, RI, 2001. Theory and numerical analysis, Reprint of the 1984 edition.

[53] AWM van Schijndel. Integrated modeling of dynamic heat, air and moisture processes in buildings and systems using SimuLink and COMSOL. In *Building Simulation*, volume 2, pages 143–155. Springer, 2009.

[54] Gengsheng Wang. Stabilization of the Boussinesq equation via internal feedback controls. *Nonlinear Anal.*, 52(2):485–506, 2003.

[55] Wikipedia. R-value (insulation) — wikipedia, the free encyclopedia, 2011. [Online; accessed 2-August-2011].

[56] C. Yang, X. Yang, T. Xu, L. Sun, and W. Gong. Optimization of bathroom ventilation design for an ISO Class 5 clean ward. In *Building Simulation*, volume 2, pages 133–142. Springer, 2009.

Appendix A

Obtaining LQR Functional Gains Between COMSOL® and MATLAB® in 2D

A.1 COMSOL® Study Setup

1. Begin a "New Model" in COMSOL® , and select space dimension "2D".
2. Add physics: "Mathematics"→"PDE Interfaces"→"Coefficient Form PDE".
3. Select "Field Name" "u".
4. Select "Number of dependent variables" "2".
5. Name dependent variables "u" and "v".
6. Add physics: "Mathematics"→"PDE Interfaces"→"Coefficient Form PDE".
7. Select "Field Name" "T".
8. Select "Number of dependent variables" "1".
9. Name dependent variables "T".
10. Add physics: "Mathematics"→"PDE Interfaces"→"Coefficient Form PDE".
11. Select "Field Name" "p".
12. Select "Number of dependent variables" "1".
13. Name dependent variables "p".
14. Select "Preset Studies for Selected Physics"→"Time Dependent".

15. Build "Geometry" with boundary segments as desired for controlled inflow, outflow, controlled heat source/sink, heat disturbance source/sink.

16. Create Mesh as desired.

    Note: Be sure to select the full domain or intended boundaries when establishing settings

17. In "Model Builder"→"Model 1"→"PDE" select "Discretization" "Shape Function Type" "Lagrange", "Element order" "Quadratic".

18. In "Model Builder"→"Model 1"→"PDE"→"Coefficient Form PDE 1", make the following settings:

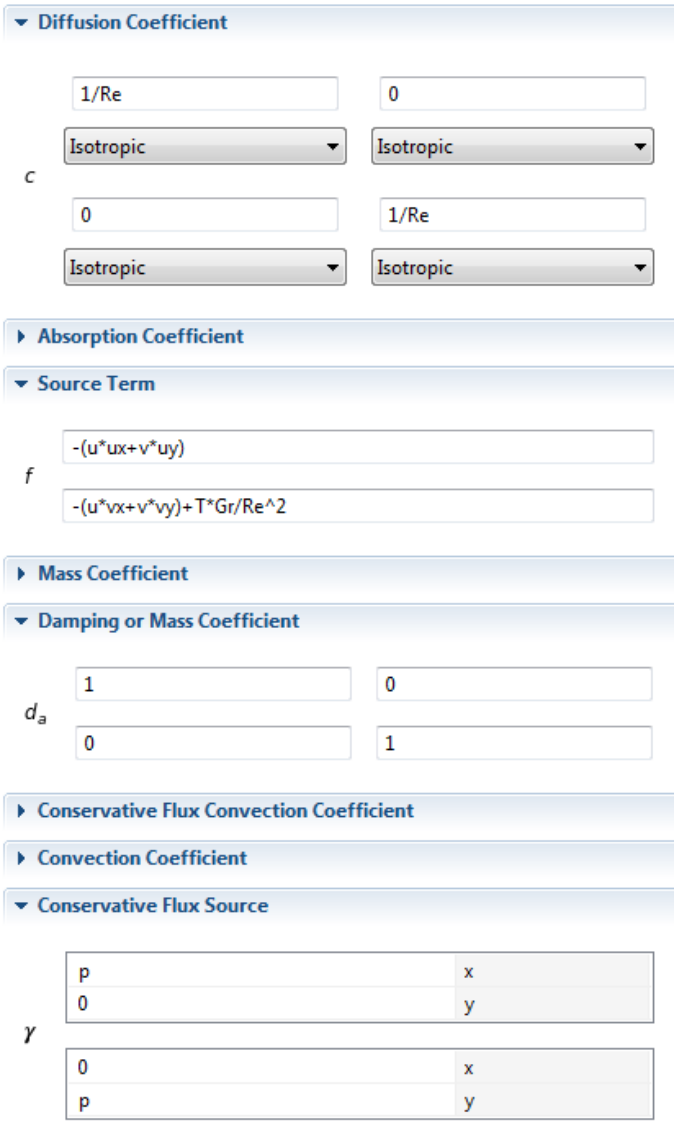

Figure A.1: COMSOL® Momentum Equation Setup

19. In "Model Builder"→"Model 1"→"PDE 2" select "Discretization" "Shape Function Type" "Lagrange", "Element order" "Linear".

20. In "Model Builder"→"Model 1"→"PDE 2"→"Coefficient Form PDE 2", make the following settings:

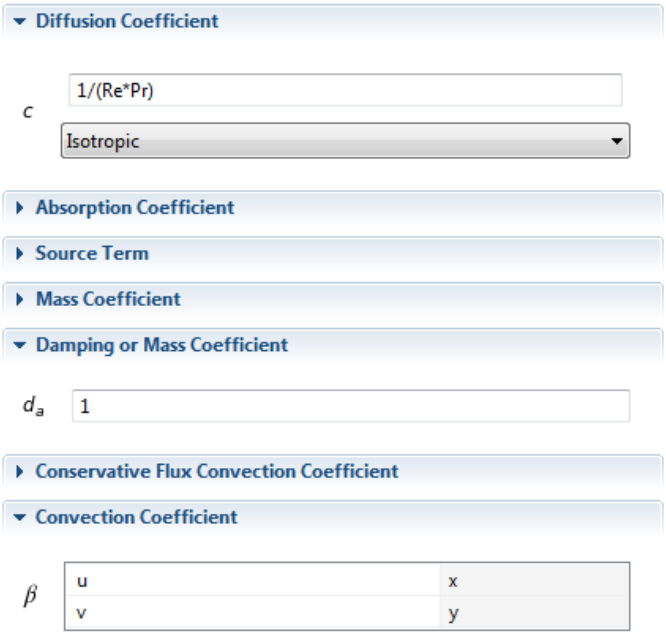

Figure A.2: COMSOL® Heat Equation Setup

21. In "Model Builder"→"Model 1"→"PDE 3" select "Discretization" "Shape Function Type" "Lagrange", "Element order" "Linear"

22. In "Model Builder"→"Model 1"→"PDE 3"→"Coefficient Form PDE 3", make the following settings:

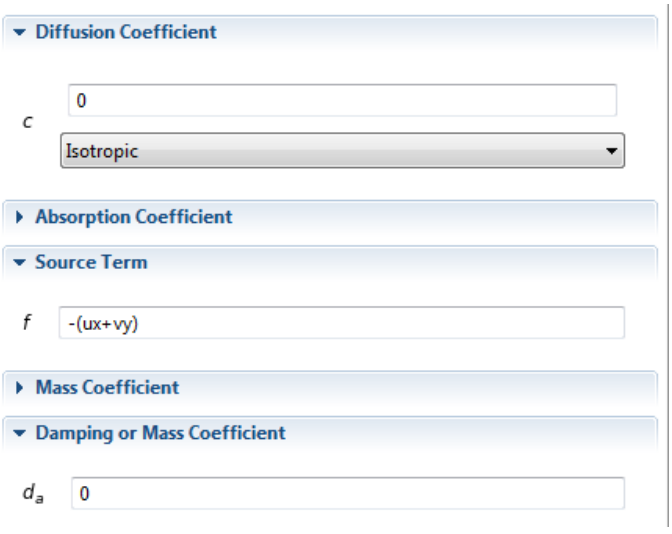

Figure A.3: COMSOL® Incompressibility Constraint

23. Under Global Definitions→"Parameters", set the following parameters:[1]

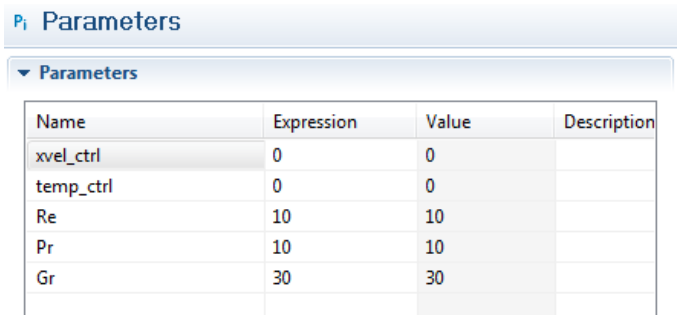

Figure A.4: COMSOL® Parameter Setup

24. Right Click on "Model Builder"→"Model 1"→"PDE" to select "Dirichlet Boundary Condition" and establish the velocity "no-slip" boundaries:

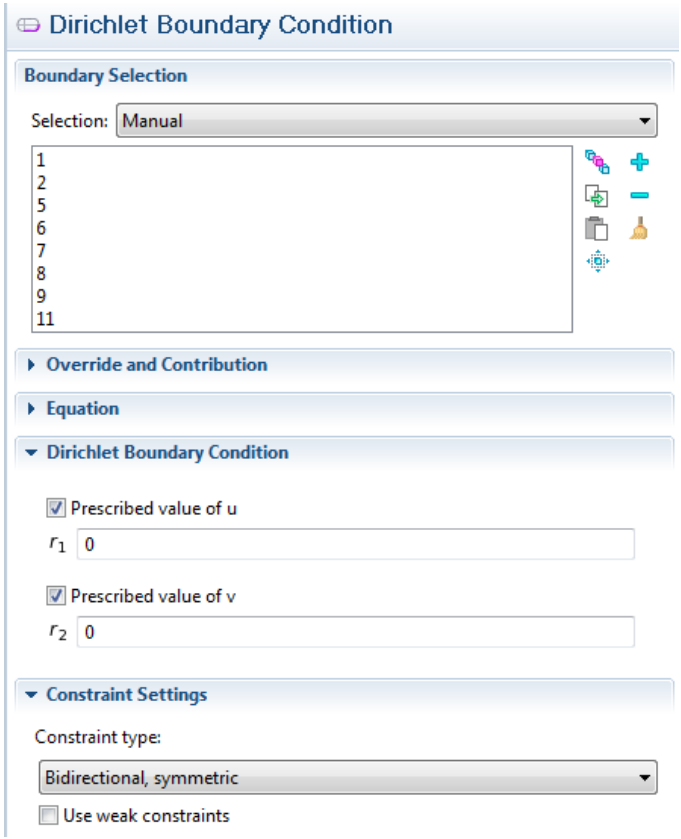

Figure A.5: COMSOL® No-slip Boundary Setup

[1]Note that the horizontal velocity control "xvel_ctrl" and the temperature control "temp_ctrl" should be set at the value they would be at ideal steady state.

25. Right Click on "Model Builder"→"Model 1"→"PDE" to select "Dirichlet Boundary Condition" and establish the velocity "controlled" boundaries:[2]

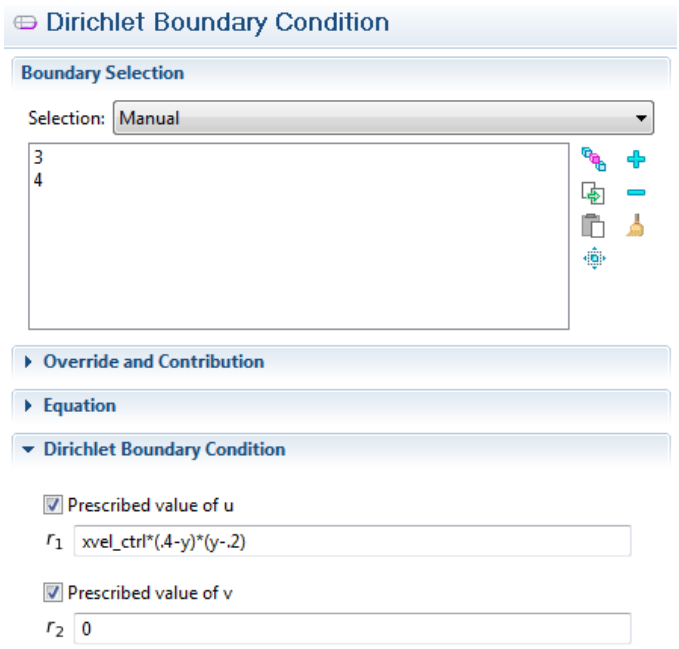

Figure A.6: COMSOL® Velocity Control Setup

26. Right Click on "Model Builder"→"Model 1"→"PDE" to select "Zero Flux" and prescribe the velocity "Do-Nothing" boundaries (outflow).

27. Right Click on "Model Builder"→"Model 1"→"PDE 2" to select "Dirichlet Boundary Condition" and establish the heat "controlled" boundaries:

[2]Note that it is important that the name of the control parameter be included in the prescribed value.

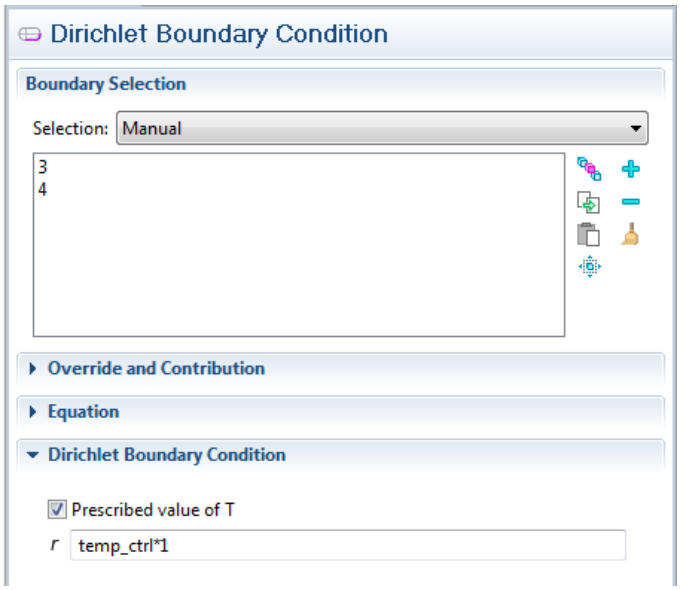

Figure A.7: COMSOL® Heat Boundary Control Setup

28. In all three PDE interfaces (velocity, temperature, divergence-free), set consistent initial values. For this thesis, all initial values were set to "0", with consistent boundary conditions.

29. Right Click "Study 1" and select "Show Default Solver". Then Right Click "Solver Configurations"→"Solver 1"→"Time-Dependent Solver" and set to "Disable"

30. Select "Study 1"→"Solver Configurations"→"Solver 1"→"Dependent Variables" and change "Scaling Method" to "None".[3]

31. Save the file (remembering the path you used).[4]

32. Launch "COMSOL 4.2 with MATLAB" and set the path in MATLAB® to match the location of your newly-created COMSOL® file.[5]

33. Run the following MATLAB® code, substituting the file name you created in COMSOL® :[6]

[3]Also, under "Initial Values of Variables Solved For", if you have generated an alternate "Steady State" in a separate study for the same model, you can change the "Method" to "Solution" and specify the solver associated with this "Steady State".

[4]For the purposes of these instructions, we call the file "State_Matrix_Setup".

[5]In this case, the location of "State_Matrix_Setup.mph"

[6]Note that this code should be modified if control parameter nomenclature has changed, or if additional boundary controls have been implemented.

```
model = mphload('State_Matrix_Setup.mph');
matrices =mphmatrix(model,'sol1','Out',{'D','Dc','Null','N'});
state = mphstate(model,'sol1','out',{'MA' 'MB'},'input',...
    {'xvel_ctrl' 'temp_ctrl'});
Mass= full(matrices.D);
M= full(matrices.Dc);
Null= full(matrices.Null);
MA= full(state.MA);
MB= full(state.MB);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%get general mesh information
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xmi=model.sol('sol1').xmeshInfo;
%get general dof mesh info
dofs=xmi.dofs;
%get coords for each dependent variable (called dofs)
coords=dofs.coords;
coords=coords';
%get node number for each dof
nodes=dofs.nodes;
%get the total number of degrees of freedom
nDofs=xmi.nDofs;
%see link of index number with dependent variable
dofNames=dofs.dofNames
%specify dependent variable associated with dof
nameInds=dofs.nameInds;
%create array for all Dofs listing nodes/xcoord/ycoord/dep var
   index
Dof_coords_nodes_names=zeros(nDofs,5);
Dof_coords_nodes_names(1:nDofs,2)=nodes;
Dof_coords_nodes_names(1:nDofs,3:4)=coords;
Dof_coords_nodes_names(1:nDofs,5)=nameInds;
for i=1:nDofs
  Dof_coords_nodes_names(i,1)=i;
end;
%give number of DoFs for each field
fieldNDofs=xmi.fieldNDofs;
%Give List of Nodes with DOF Positions
nodes=xmi.nodes;
nodes_dofs=nodes.dofs';
%find null space node positions in DOFs
sizeDOFs=size(M,1);
multiplier=ones(sizeDOFs,1);
null_nodes=Null*multiplier;
```

```
null_node_positions=find(null_nodes);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%get full information about null space dofs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nbr_of_null_dofs=size(null_node_positions);
nbr_of_null_dofs=nbr_of_null_dofs(1);
null_dofs_full_info=zeros(nbr_of_null_dofs,5);
for i=1:nbr_of_null_dofs
   null_dofs_full_info(i,:)=...
       Dof_coords_nodes_names(null_node_positions(i),:);
end;
%%%%%%%%%%%%%%%%%%
%change numbering scheme for state types to 0->u, 1->v,2->T,3->p
%IMPORTANT: You may need to modify this code if COMSOL
%generated a different numbering scheme, see dofs.dofNames
null_dofs_full_info(:,5)= null_dofs_full_info(:,5)-2;
for i = 1:nbr_of_null_dofs
    if null_dofs_full_info(i,5)==-2
    null_dofs_full_info(i,5)=2;
    elseif  null_dofs_full_info(i,5)==-1
    null_dofs_full_info(i,5)=3;
    end
end;
%rearrange null dofs  (matrix called rearrange) that separates
%by u (0) then v (1) then T (2) then P (3)
rearrange=sparse([null_dofs_full_info, eye(nbr_of_null_dofs)]);
%rearrange=sortrows(rearrange, [5 4 3]);
rearrange=sortrows(rearrange, 5);
rearrange=rearrange(:,6: nbr_of_null_dofs +5);
null_dofs_full_info=rearrange*null_dofs_full_info;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%rearrange key state matrices
M=sparse(M);
MA=sparse(MA);
MB=sparse(MB);
M= rearrange*M*rearrange';
MA= rearrange*MA*rearrange';
MB= rearrange*MB;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%get count of pressure
nbr_presure_dofs=0;
for i=1:nbr_of_null_dofs
    if null_dofs_full_info(i,5)==3
    nbr_presure_dofs=nbr_presure_dofs+1;
```

```
    end
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%get count of temp dofs
nbr_temp_dofs=0;
for i=1:nbr_of_null_dofs
    if null_dofs_full_info(i,5)==2
    nbr_temp_dofs=nbr_temp_dofs+1;
    end
end;
%%%%%%%%%%%%%%%%%%%%%%%%%
nbr_nonpressure_dofs=nbr_of_null_dofs-nbr_presure_dofs;
%%%%%%%%%Get count of vel dofs%%%%%%%%%%%%%%
nbr_vel_dofs=0;
for i=1:nbr_of_null_dofs
    if null_dofs_full_info(i,5)<2
    nbr_vel_dofs= nbr_vel_dofs+1;
    end
end;
%%%%%%%%%%%%%%%%%%%%%%
```

34. Return to Comsol® , and in "Model Builder"→"Model 1"→"PDE 3"→"Coefficient Form PDE 3", change the "Damping or Mass Coefficient", d_a, to 1. Then re-save the file.

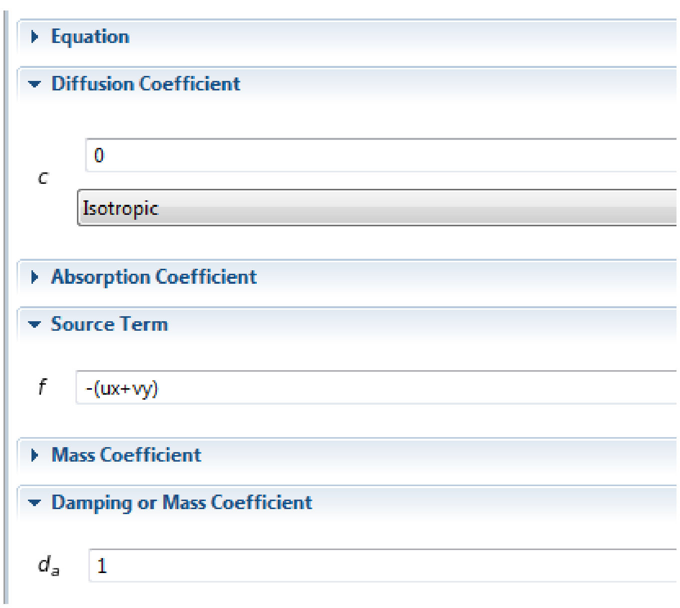

Figure A.8: Comsol® Coefficient to Create Pressure Mass Matrix

35. Run the following MATLAB® code:[7]

```
%%%%%%%%%%%%%%%%%%%%Get P Mass Matrix
%%%%%%%%%%Note: Be sure to create a p_dot term in the
%%%%%%%%%%COMSOL gui before proceeding
model2 = mphload('State_Matrix_Setup.mph');
matrix =mphmatrix(model2,'sol1','Out',{'D'});
MassWp= full(matrix.D);
MassWp=sparse(MassWp);
Null=sparse(Null);
MassWpElim=Null'*MassWp*Null;
M=sparse(M);
rearrange=sparse(rearrange);
MassWpElim=rearrange*MassWpElim*rearrange';
M=sparse(M);
Mp=MassWpElim-M;
Mp=Mp(size(Mp,1)-nbr_presure_dofs+1:size(Mp,1),...
    size(Mp,1)-nbr_presure_dofs+1:size(Mp,1));
%%%%%%%Note: Now be sure to get rid of the p_dot coefficient
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Specify mesh size h
h=.05;
%specify epsilon
epsil=h^2;
%%%%%%%%%%%%%%%%%%%%%%%%%%
%penalty method setup
%%%%%%%%%%%%%%%%%%%%%%%%%%
%Determine divergence free constraint matrix
div_free_const=MA(nbr_nonpressure_dofs+1:nbr_of_null_dofs,...
    1:nbr_vel_dofs);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Determine term to add to MA matrix (pressure substitution)
Mp_inv=Mp\eye(size(Mp,1));
MAadd=zeros(size(MA,1));
MAadd(1:nbr_vel_dofs,1:nbr_vel_dofs)=(1/epsil)...
    *div_free_const'*Mp_inv*div_free_const;
%%%%%%%%%%%%%%%%%%%%%%%%%%
%Modify MA Matrix Subtracting the Penalty Term
%(Same as adding to the stiffness matrix)
MA=MA-MAadd;
%%%%%%%%%%%%%%%%%%%%%%%%%%
%Create matrix to eliminate pressure
```

[7]This should be modified for different Q or R cost functional matrices.

```
pres_elim=[eye(size(MA,1)-nbr_presure_dofs);...
    zeros(nbr_presure_dofs,size(MA,1)-nbr_presure_dofs)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Convert State Matrices to remove pressure term
MM=pres_elim'*M*pres_elim;
MAA=pres_elim'*MA*pres_elim;
MBB=pres_elim'*MB;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get State Matrices
A=MM\MAA;
B=MM\MBB;
%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%Creating Weighted Q Matrix
% Q=eye(size(MM,1)- nbr_temp_dofs);
% Q=1*Q;
% Q=[Q,zeros(size(Q,1), nbr_temp_dofs);
% zeros(nbr_temp_dofs,size(Q,1)),100*eye(nbr_temp_dofs)];
% sqrootQ=Q.^(1/2);
%%%%%%%%%%%%%%%%%%%%%%%%
%%%Algebraic Ricatti Equations
R=eye(2);
R(1,1)=.00001;
R(2,2)=.00001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Cutoff Before Running Care
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [XX,LL,FF]=care(A,B,sqrootQ*MM*sqrootQ,R,'nobalance');
[XX,LL,FF]=care(A,B,MM,R);
%%%%%%%%%%%%%%
%%%%%For Integral Kernel
Kern=FF*(MM\eye(size(MM,1)))*pres_elim';
%%%%%%Feedback Thing
K=FF*pres_elim';
%%%%%%%%%%%%%%%%%%
K=[null_dofs_full_info(:,3:4) K' null_dofs_full_info(:,5)];
Kern=[null_dofs_full_info(:,3:4) Kern' null_dofs_full_info(:,5)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%Break out kernels for each dependent
%%%%%variable type and save as text files
    u_kernel_u=zeros(1,3);
    T_kernel_u=zeros(1,3);
```

```
    u_kernel_v=zeros(1,3);
    T_kernel_v=zeros(1,3);
    u_kernel_T=zeros(1,3);
    T_kernel_T=zeros(1,3);
for i=1:nbr_of_null_dofs
  if Kern(i,5)==0
    u_kernel_u=[u_kernel_u; Kern(i,1:3)];
    T_kernel_u=[T_kernel_u; Kern(i,1:2) Kern(i,4)];
  end;
  if Kern(i,5)==1
    u_kernel_v=[u_kernel_v; Kern(i,1:3)];
    T_kernel_v=[T_kernel_v; Kern(i,1:2) Kern(i,4)];
  end;
  if Kern(i,5)==2
    u_kernel_T=[u_kernel_T; Kern(i,1:3)];
    T_kernel_T=[T_kernel_T; Kern(i,1:2) Kern(i,4)];
  end;
end;
    u_kernel_u=u_kernel_u(2:size(u_kernel_u,1),:);
    T_kernel_u=T_kernel_u(2:size(T_kernel_u,1),:);
    u_kernel_v= u_kernel_v(2:size(u_kernel_v,1),:);
    T_kernel_v= T_kernel_v(2:size(T_kernel_v,1),:);
    u_kernel_T= u_kernel_T(2:size(u_kernel_T,1),:);
    T_kernel_T= T_kernel_T(2:size(T_kernel_T,1),:);
dlmwrite('u_kernel_u.txt', u_kernel_u, '\t');
dlmwrite('u_kernel_v.txt', u_kernel_v, '\t');
dlmwrite('u_kernel_T.txt', u_kernel_T, '\t');
dlmwrite('T_kernel_u.txt', T_kernel_u, '\t');
dlmwrite('T_kernel_v.txt', T_kernel_v, '\t');
dlmwrite('T_kernel_T.txt', T_kernel_T, '\t');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%Break out gain matrices K for each dependent variable
%%%%%type and save as text files
    u_K_u=zeros(1,3);
    T_K_u=zeros(1,3);
    u_K_v=zeros(1,3);
    T_K_v=zeros(1,3);
    u_K_T=zeros(1,3);
    T_K_T=zeros(1,3);
for i=1:nbr_of_null_dofs
  if K(i,5)==0
    u_K_u=[u_K_u; K(i,1:3)];
    T_K_u=[T_K_u; K(i,1:2) K(i,4)];
  end;
```

```
      if K(i,5)==1
       u_K_v=[u_K_v; K(i,1:3)];
       T_K_v=[T_K_v; K(i,1:2) K(i,4)];
      end;
      if K(i,5)==2
       u_K_T=[u_K_T; K(i,1:3)];
       T_K_T=[T_K_T; K(i,1:2) K(i,4)];
      end;
    end;
       u_K_u=u_K_u(2:size(u_K_u,1),:);
       T_K_u=T_K_u(2:size(T_K_u,1),:);
       u_K_v= u_K_v(2:size(u_K_v,1),:);
       T_K_v= T_K_v(2:size(T_K_v,1),:);
       u_K_T= u_K_T(2:size(u_K_T,1),:);
       T_K_T= T_K_T(2:size(T_K_T,1),:);
    dlmwrite('u_K_u.txt', u_K_u, '\t');
    dlmwrite('u_K_v.txt', u_K_v, '\t');
    dlmwrite('u_K_T.txt', u_K_T, '\t');
    dlmwrite('T_K_u.txt', T_K_u, '\t');
    dlmwrite('T_K_v.txt', T_K_v, '\t');
    dlmwrite('T_K_T.txt', T_K_T, '\t');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

36. Return to COMSOL® , and in "Model Builder"→"Model 1"→"PDE 3"→"Coefficient Form PDE 3", change the "Damping or Mass Coefficient", d_a, back to 0. Then re-save the file.

    At this stage, the functional gains are now available to be imported into COMSOL.®

37. Right-click on "Model 1"→"Definitions"→"Model Couplings"→"Integration".

38. Select the entire domain and designate an integration order. Also, provide an "Operator name" (for the example, we use "integral").

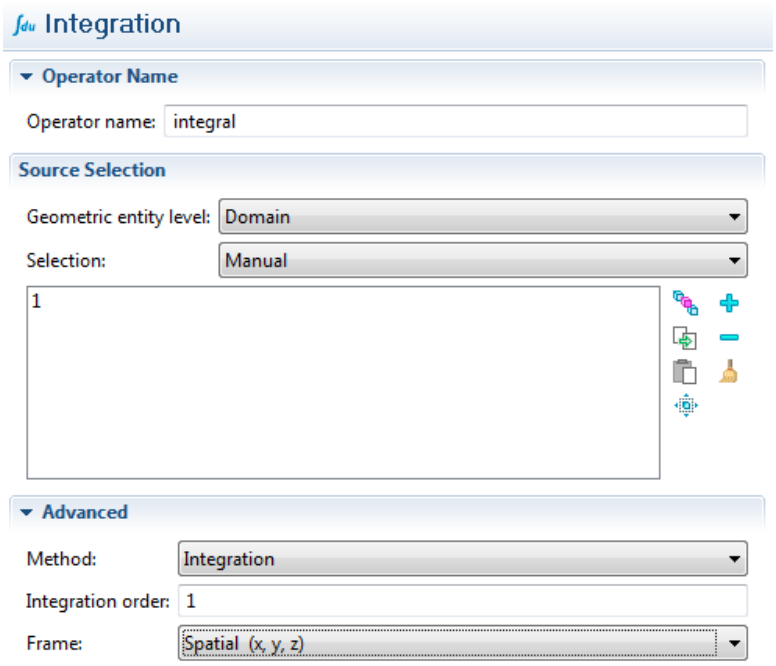

Figure A.9: COMSOL® Integration Setup

39. Right-click on "Model 1" → "Definitions" → "Model Couplings" → "Integration".

40. Select the entire domain and change the "Method" to "Summation over nodes". Also, provide an "Operator name" (for the example, we use "nodesum").

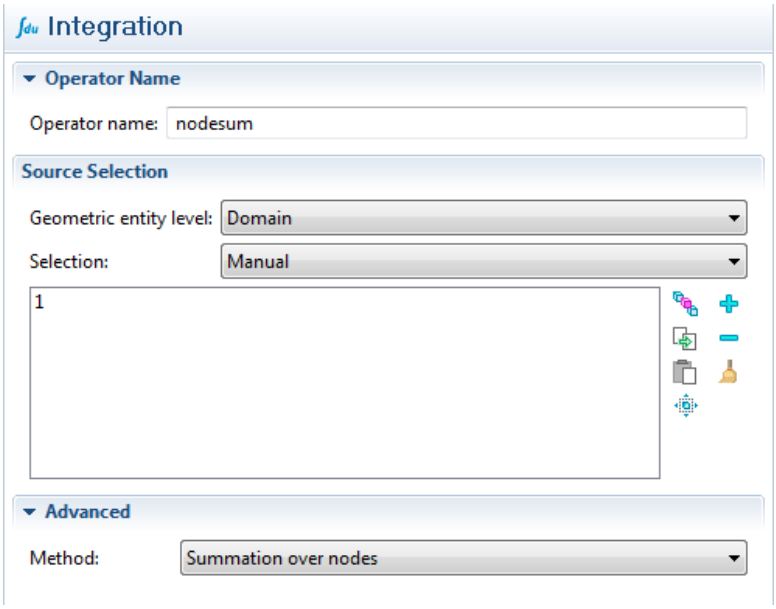

Figure A.10: COMSOL® Nodal Sum Setup

41. Right-click on "Global Definitions"→"Functions"→"Interpolation"
42. Change "Data source" to "File" and browse for the "T_kernel_T" file ($k_{2T}(\mathbf{x})$) that was created in MATLAB.®
43. Change the "Function Name" as well as the "Interpolation Name" to "T_kernel_T".
44. Select "Use space coordinates as arguments" and ensure that "Interpolation" is set to "Linear" and "Extrapolation" is set to "Constant", then left-click on "Import".

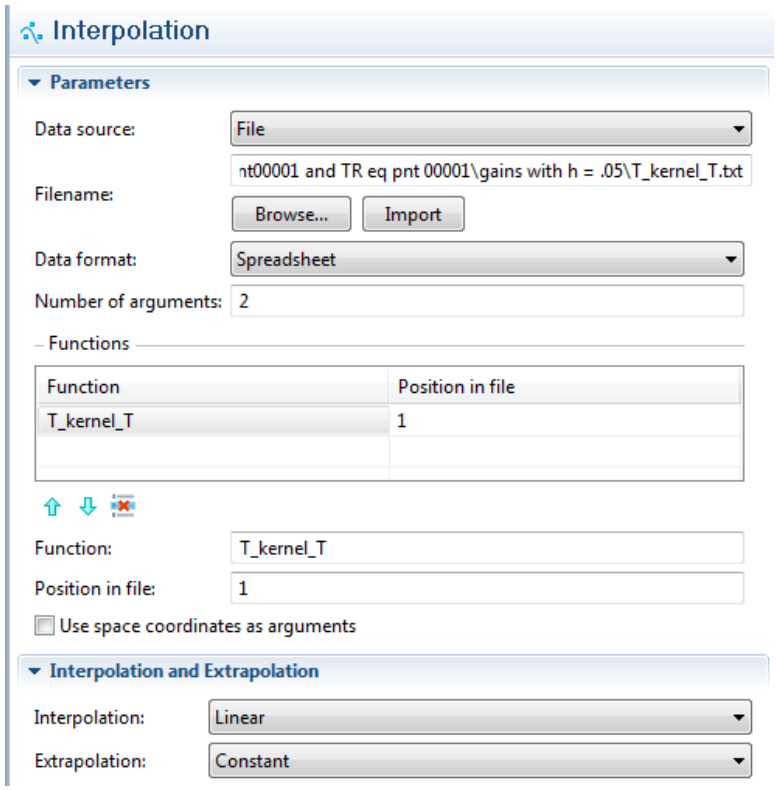

Figure A.11: COMSOL® Continued Nodal Sum Setup

45. To view the imported functional gain, click

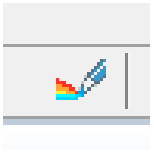

Figure A.12: COMSOL® Functional Gain View

in the "Interpolation" window.

46. Repeat instructions 41-45 for the "u_kernel_T" file ($k_{1T}(\mathbf{x})$) that was created in MATLAB.®
47. Right-click on "Global Definitions" → "Functions" → "Interpolation"
48. Change "Data source" to "File" and browse for the "T_K_u" file ($k_{2v_1}(\mathbf{x})$) that was created in MATLAB.®
49. Change the "Function Name" as well as the "Interpolation Name" to "T_K_u".
50. Select "Use space coordinates as arguments" and ensure that "Interpolation" is set to "Nearest neighbor", "Extrapolation" is set to "Specific value", and "Value outside range" is set to 0, then left-click on "Import".

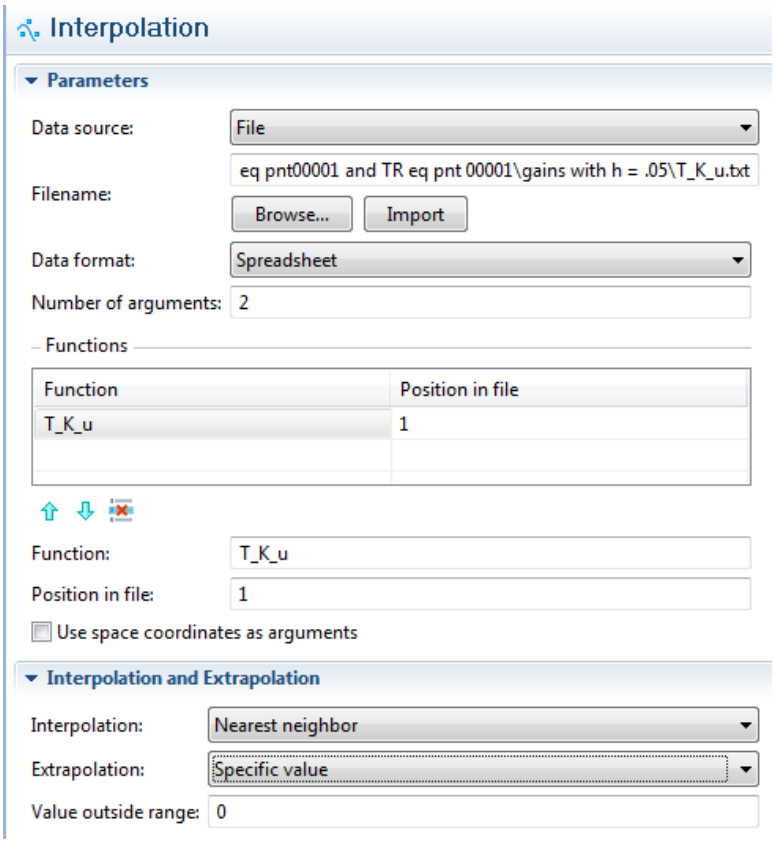

Figure A.13: COMSOL® Nodal Sum Extrapolation Setup

51. To view the imported functional gain, click

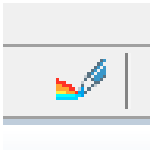

Figure A.14: COMSOL® Functional Gain View

in the "Interpolation" window.

52. Repeat instructions 47-51 for the "T_K_v" ($k_{2v_2}(\mathbf{x})$), "u_K_u" ($k_{1v_1}(\mathbf{x})$), and "u_K_v" ($k_{1v_2}(\mathbf{x})$) files that were created in MATLAB.®

53. Right-click on "Global Definitions" → "Functions" → "Analytic" and create the following "ramp" function:

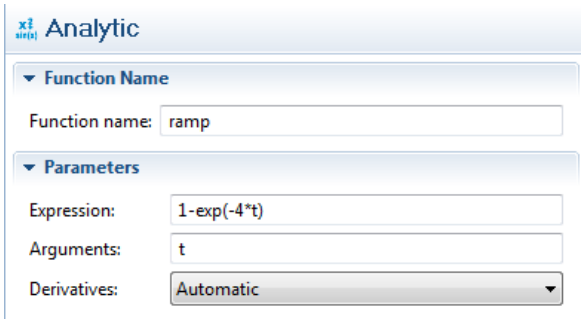

Figure A.15: COMSOL® Smooth Ramp Function

54. Use the imported functional gains to set up LQR boundary control, by selecting the velocity-controlled portion of the boundary and setting the following Dirichlet boundary conditions where the "Prescribed value of u" is "-ramp(t) *(integral(u_kernel_T*T) +nodesum(u_K_u*u)+ nodesum(u_K_v*v))*(.4-y)*(y-.2)":

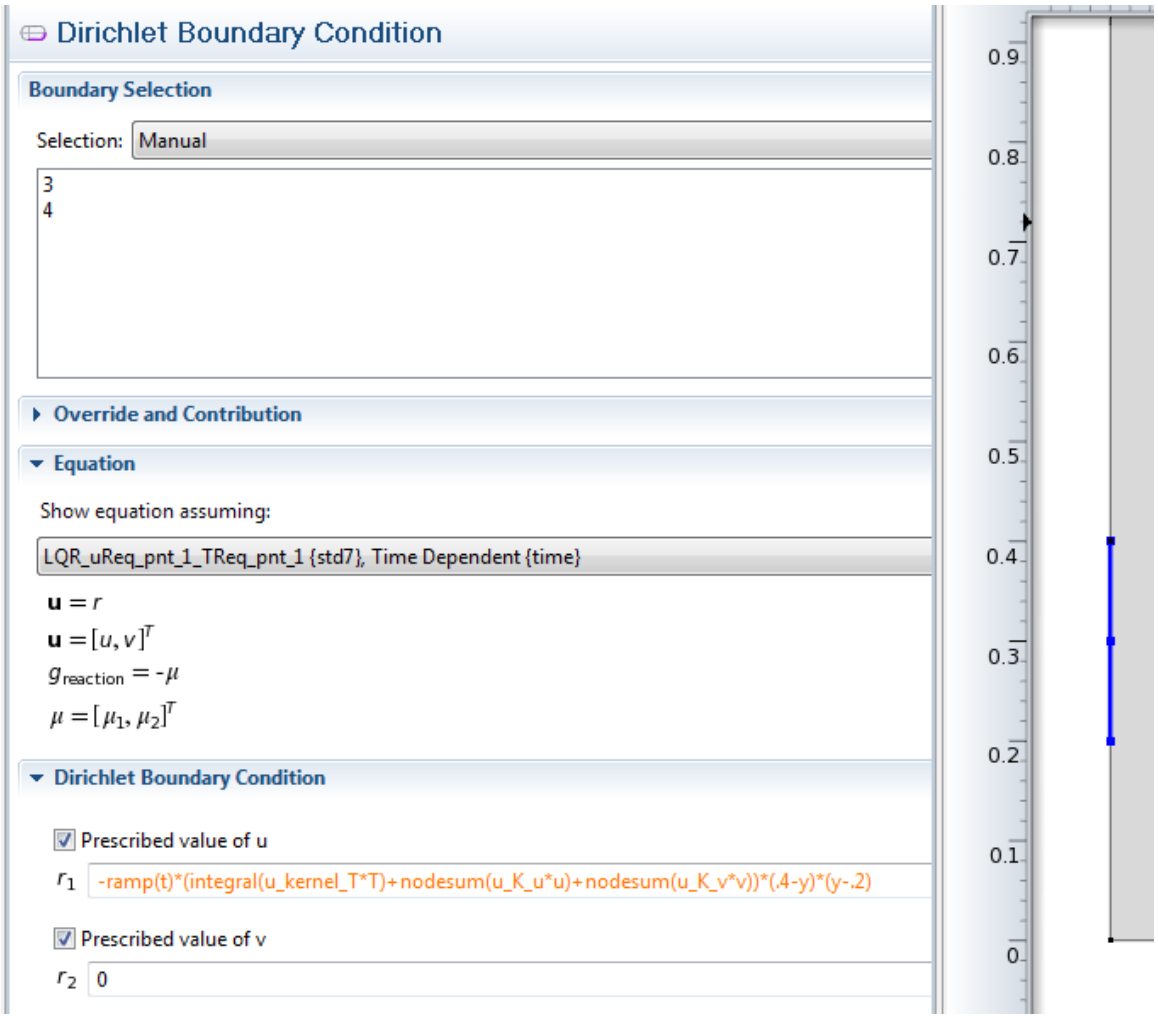

Figure A.16: COMSOL® LQR Velocity Boundary Control Setup for Simulation

55. Use the imported functional gains to set up LQR boundary control, by selecting the temperature-controlled portion of the boundary and setting the following Dirichlet boundary conditions where the "Prescribed value of T" is "2*(1-ramp(t))-ramp(t) *(integral (T_kernel_T*T) +nodesum (T_K_u*u) +nodesum (T_K_v*v))":[8]

[8]In this case, 2 is multiplied by the ramp function term to accommodate for the fact that the initial temperature condition was 2.

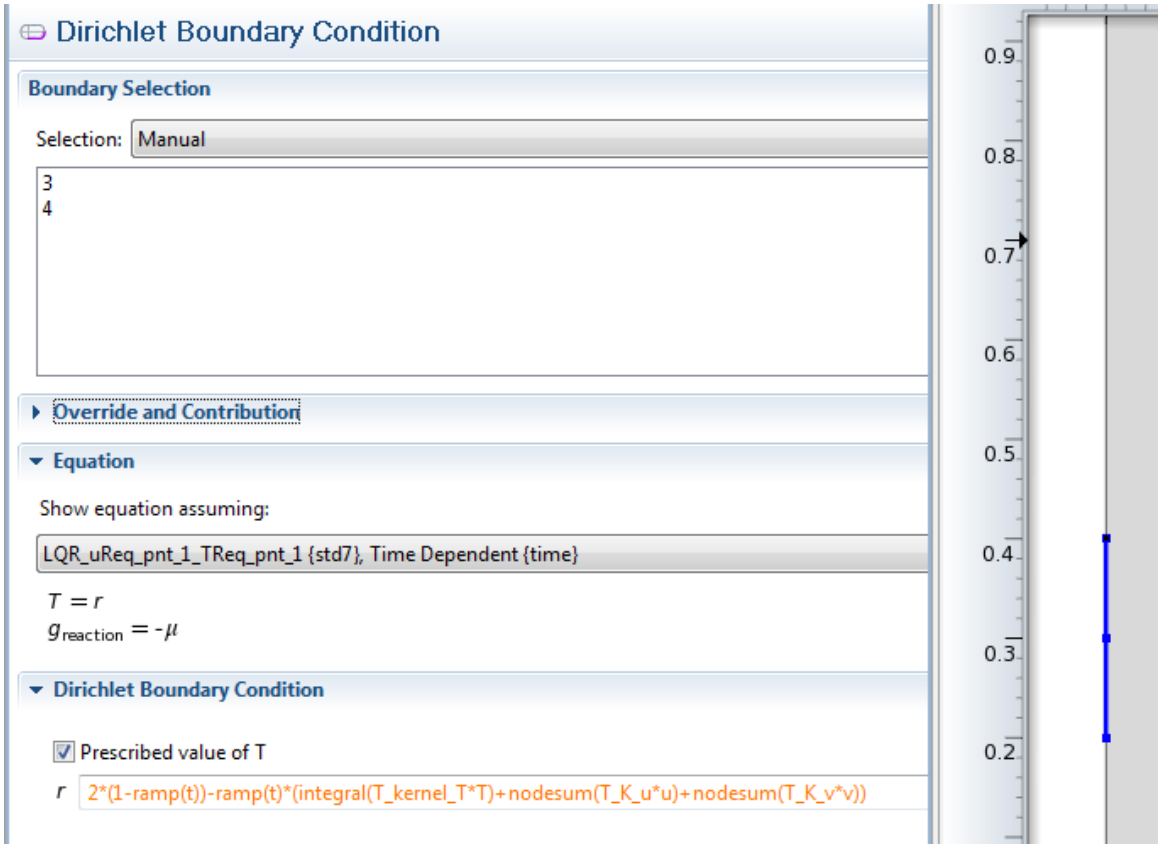

Figure A.17: COMSOL® LQR Temperature Boundary Control Setup for Simulation

56. Right Click "Study 1" → "Solver Configurations" → "Solver 1" → "Time-Dependent Solver" and set to "Enable".

57. In "Solver Configurations" → "Solver 1" → "Dependent Variables" change "Scaling" to "Automatic".

58. Run the study as desired.