6-9-2011

Solving nonrigid registration with sliding and bending boundary conditions

Troy Winkstern

Solving Nonrigid Registration with Sliding and Bending Boundary Conditions

Troy Winkstern

Rochester Institute of Technology

College of Science

Department of Mathematics Sciences

A thesis submitted for the degree of

Master of Mathematical Sciences

June 9, 2011

Approved by:

Nathan D. Cahill
Thesis Advisor

David S. Ross
Professor

John F. Hamilton
Assistant Professor

Tamas I. Wiandt
Associate Professor

Acknowledgements

First and foremost, I would like to thank my parents and my sister Tara for all their support and encouragement. I would also like to thank my advisor, Nathan Cahill, for his leadership and patience with me over the past year. A special thanks to my friend Nate Russ for all the help he has given me. I'd like to thank Alison Coladonato for editing my thesis. Lastly, I would like to thank the rest of family and my friends.

Abstract

In the past 20 years, there have been significant advances in the use of medical imaging in patient care. Today, image registration is being used by doctors all over the world to compare computed tomography (CT), Magnetic Resonance Imaging (MRI), X-rays, Ultrasound (US), Positron Emission Tomography (PET), or Single Photon Emission Computed Tomography (SPECT) images of their patients taken at different times in order to formulate an accurate diagnosis. Mathematically, image registration can be posed as an initial boundary value problem. However, when applied to medical imaging, the current boundary conditions being used are not physically meaningful. In order to improve upon these methods we will propose two new boundary conditions that are more physically meaningful and will show how to apply them to current image registration techniques.

This thesis is organized as follows: in Chapter 2, the mathematical development of the initial boundary value problem (IBVP) and its different components are discussed. Then, in Chapter 3, we introduce a simplified version of the initial boundary value problem, called a toy problem. We then use the toy problem to develop the mathematical techniques that will be applied, as well as illustrate how registration will work on scalar and vector-valued functions in two and three dimensions. Furthermore, we present the new boundary conditions for use in registration. Chapter 4 draws upon the concepts established in Chapter 3 to test our new

boundary conditions on real world examples of medical image registration. The boundary conditions are tested for both two and three dimensional registration problems and the results are displayed at the end of Chapter 4.

Contents

List of Figures

Chapter 1

Introduction

Image registration is the process of transforming data from multiple sources into a single coordinate system. The sources of data can be things such as photographs, pictures from multiple viewpoints, and/or images taken on different occasions. Image registration is used in many fields, including are computer vision [13], remote sensing [12], and most relevant for us, medical imaging. In medical imaging, the commonly used imaging modalities are Computed Tomography (CT), Magnetic Resonance Imaging (MRI), X-rays, Ultrasound (US), Positron Emission Tomography (PET), or Single Photon Emission Computed Tomography (SPECT). One common use of medical imagery is in imaging the brain. An example of this can be seen in figure (1.1) where axial slices of a patient's brain using CT, MRI, and PET scans

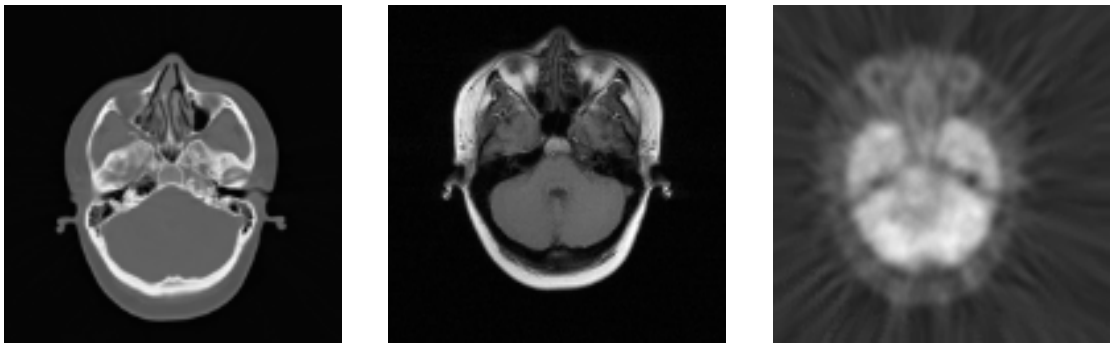


Figure 1.1: Scans of a patient's brain with CT, MR, and PET scans respectively.

respectively are shown [16].

Medical imaging is used to help doctors see exactly what has changed in the patient from month to month or year to year. This allows the doctors to accurately diagnose and treat their patients. Over the last few years, the world has seen a massive growth in image processing and computer graphics algorithms which have changed the way patients are diagnosed and the way they are treated. The image registration process [10] can be conducted on images of the same patient captured at different times. The process begins with a reference image R and a template T. In figure (1.2) two CT axial slices of a patient's lungs can be seen; the left image is the reference image and the right image is the template.

In figure (1.3) a reference image, a template, and another image called the temporal subtraction image (or difference image) are given. These images are chest X-rays of the same patient captured at different times. The temporal subtraction image D is created by subtracting the reference image R from the template T. Ideally, we hope that most of D will be visualized as a mid-level gray color, which means it has constant value of zero. In other words, R and T display the same information. White or black areas indicate they do not display the same information. On each image in figure (1.3), three areas have been highlighted in red, green, and blue zones. If you look closely at the blue zones of the reference image and the template, they appear to display the same thing; thus the area in the blue zone of the temporal subtraction

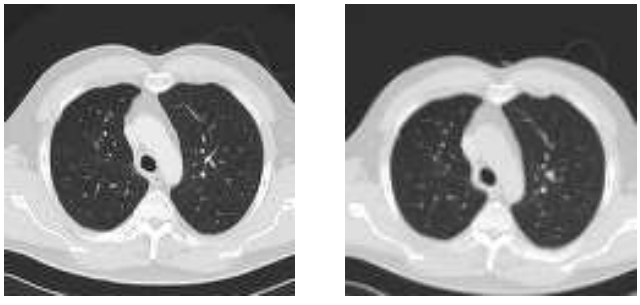


Figure 1.2: The reference image R (left) and the template T (right) are the basis for image registration.

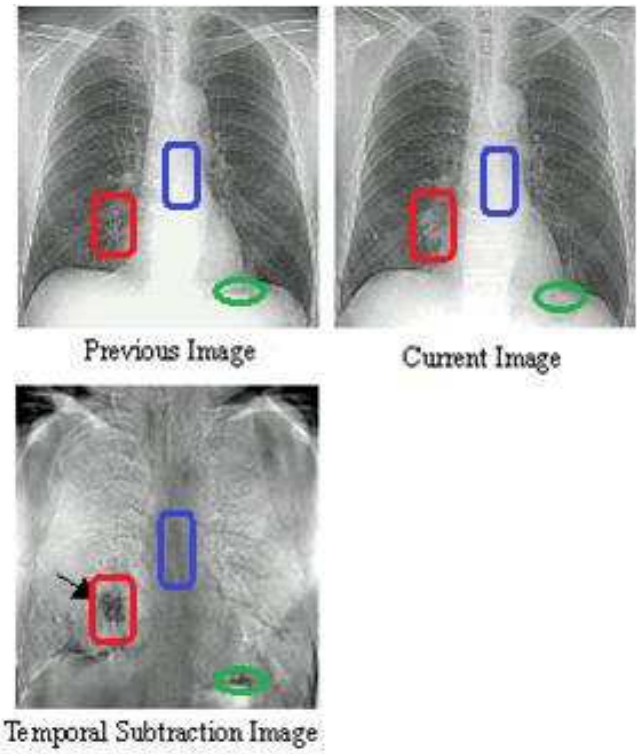


Figure 1.3: The reference image R (top left), the template T (top right), and the temporal subtraction image (bottom left) help verify the registration process was successful.

image is gray. However, in the red and green zones the reference image and the template do not display the same thing, explaining the black areas in the red and green zones of the temporal subtraction image.

Nonrigid image registration can be posed as an initial boundary value problem. Current boundary conditions being used have been chosen for ease of computation but don't translate into something that is physically meaningful. The goal of this thesis is to improve on the current formulations of the nonrigid registration problem. We will do this by presenting two new boundary conditions, explaining how they are physically meaningful, and showing that they will provide better results than the current boundary conditions in medical image registration problems. A portion of the work presented in this thesis has been published in the proceedings of the 2011 International Symposium on Biomedical Imaging [14].

Chapter 2

Origin of the Diffusion Equation

In this chapter, we discuss how the Diffusion Equation arises from a pre-existing image registration algorithm (2.1). We take two images, a reference image and a template, and use the Euler-Lagrange equations to minimize their similarity measure and regularizer. From there we embed the Euler-Lagrange equations into another system and create the initial boundary value problem (2.5). The diffusion equation is defined by a particular similarity measure and regularizer chosen from a set of similarity measures and regularizers defined below.

2.1 Preliminaries

Image registration algorithms are comprised of three components: the objective function, an optimization technique, and a class of geometric transformations. The objective function is used to quantify the degree of similarity between the images. An optimization technique is used for optimizing the objective function over the set of valid transformations. The class of transformationss is needed for geometrically relating the images. There are two different classes of transformations: rigid and non-rigid. Rigid transformations include only rotation and translations while non-rigid transformations can include scale and skew (or general affine transformations) as well as locally deformable transformations. Some locally deformable transformations are parametric, such as thin plate splines, radial basis functions, local affine

transforms, and many others. Other non-rigid transformations are nonparametric [2], being defined in terms of vector fields. For the purposes of this paper, we are only going to consider nonparametric transformations being that they are the most flexible. When dealing with nonparametric registration problems, the objective function almost always is made up of an image similarity measure and a term that regularizes the deformations. The most widely used similarity measures are: the sum of squared differences (SSD) [2, 3], cross correlation (CC) [3], correlation ratio (CR) [3], and mutual information (MI) [3]. Some popular classifications of regularizers [5] include the diffusion, curvature, and elastic regularizer. This thesis however will focus solely on using the diffusion regularizer due to its simplicity and established application in many image registration algorithms. Optimizing the objective function will be achieved by using calculus of variations to form the Euler-Lagrange (E-L) equations and then introducing an artificial time variable to form an IBVP whose stationary solution solves the E-L equations.

Given two images, a reference image R and a template T defined on $\Omega \subseteq \mathbb{R}^N$ we want to find a transformation Φ: $\mathbb{R}^N \longmapsto \mathbb{R}^N$ such that $T(\Phi(x))$ is similar to R. We will define $\Phi(x) = x - \mathbf{u}(x)$ where $\mathbf{u}(x)$ is the displacement. Thus we want to minimize:

$$\Psi(\mathbf{u}) = D[R, T, \Phi] + \gamma S(\mathbf{u}) \tag{2.1}$$

where $\Psi(\mathbf{u})$ is the objective function mentioned above, D is a similarity measure between R and $T(\Phi)$, γ is a regularizing parameter and S is a smoothing term or regularizer. We choose to characterize the minimizer by the E-L equations because there are a large number of parameters we have to deal with since we chose to use nonparameteric transformations. Other optimization techniques like Newton's method will become very computationally intensive and would be to much to handle.

The E-L equations arise from computing the first variation of $\Psi(\mathbf{u})$ and setting it

equal to 0. So the E-L equations for (2.1) are

$$A(\mathbf{u}(x)) - f(x, \mathbf{u}(x)) = 0 \quad \forall x \in \Omega \tag{2.2}$$

where A is the first variation of S and is called a partial differential operator, f is the first variation of D and is called the force vector. To numerically approximate the solution to (2.2) we could use a fixed-point iteration with an initial guess of $\mathbf{u}^{(0)} \equiv 0$; i.e.,

$$A(\mathbf{u}^{(k+1)}(x)) = f(x, \mathbf{u}^{(k)}(x)). \tag{2.3}$$

This can be problematic, however, since A could have a nontrivial kernel. We can avoid this problem by introducing a variable t for time such that $\mathbf{u}$ is now time dependent [i.e. $\mathbf{u} = \mathbf{u}(x, t)$]. If we embed the E-L equations into a semilinear parabolic system, we obtain the equation:

$$\partial_t \mathbf{u}(x, t) + A(\mathbf{u}(x, t)) = f(x, t, \mathbf{u}(x, t)). \tag{2.4}$$

Note that the stationary solution of (2.4) is the solution of the E-L equations (2.3). This equation is exactly the initial boundary value problem (IBVP) that we want to solve, namely:

$$\begin{aligned} \frac{\partial}{\partial t}\mathbf{u}(x, t) + A(\mathbf{u}(x, t)) &= f(x, t, \mathbf{u}(x, t)) \\ 0 < x < \ell, 0 < t& \\ \mathbf{u}(x, 0) &= \mathbf{u}_0 \\ B[\mathbf{u}(x, t)] = 0, \quad x \in \partial\Omega& \end{aligned} \tag{2.5}$$

In medical imaging, there are three common regularizers: diffusion [2, 5], elastic [2, 4, 5, 6], and curvature [2, 5, 8]. Table 2.1 and Table 2.2 give the definition of the

Regularizer	$R(\mathbf{u})$
Diffusion	$\frac{1}{2}\sum_{r=1}^{\rho}\left(\int_{\Omega}\left(\nabla u_r\right)^T\left(\nabla u_r\right)dx\right)$
Elastic	$\int_{\Omega}\frac{\mu}{4}\sum_{r,s=1}^{\rho}\left(\partial_{x_r}u_s+\partial_{x_s}u_r\right)^2+\frac{\lambda}{2}\left(\nabla\cdot\mathbf{u}\right)^2dx$
Curvature	$\frac{1}{2}\sum_{r=1}^{\rho}\left(\int_{\Omega}\left(\Delta u_r\right)^2dx\right)$

Table 2.1: Regularizers

Partial Differential Operator	$A(\mathbf{u})$
Diffusion	$-\Delta\mathbf{u}$
Elastic	$-\mu\Delta\mathbf{u}-(\lambda+\mu)\nabla(\nabla\cdot\mathbf{u})$
Curvature	$\Delta^2\mathbf{u}$

Table 2.2: Partial Differential Operators

regularizers and partial differential operators that correspond to the three common regularizers. In Tables 2.1 and 2.2, Δ refers to the Laplacian, ∇ is the gradient, $\nabla\cdot$ indicates the divergence, ρ is the image dimension, u_r is the r^{th} component of the displacement field and λ,μ are constants that arise from the Lamé parameters of selected materials. Another common regularizer is the fluid regularizer [2, 4, 5, 7] though it is essentially the elastic regularizer as applied to a velocity field. We have chosen to work with the diffusion regularizer because it is the simplest and serves as the basis for many image registration algorithms.

Since we will be working solely with the diffusion regularizer, we can substitute $A=-\Delta$ and (2.5) becomes:

$$\begin{aligned}\frac{\partial}{\partial t}\mathbf{u}(x,t)-\alpha^2\Delta(\mathbf{u}(x,t))&=f(x,t,\mathbf{u}(x,t)) \qquad (2.6)\\ 0<x<\ell,0<t&\\ \mathbf{u}(x,0)&=\mathbf{u}_0\\ B[\mathbf{u}(x,t)]=K,\quad x\in\partial\Omega&\end{aligned}$$

where K is a constant and α^2 is a constant used to trade off the impacts caused by the similarity measure and the regularizer.

When deriving the EL equations for image registration, it is important to define appropriate boundary conditions. The current boundary conditions being applied in image registration algorithms are homogenous Dirichlet, homogenous Neumann, and periodic, however each of these pose unique problems. Homogenous Dirichlet conditions force $\mathbf{u}(x,t) = 0$ on the boundaries; thus the image on the boundary is unable to move during registration. But, we cannot guarantee a person will be lying in the exact same position for CT or PET scans taken at different times. Therefore the boundary will never register correctly. Neumann boundary conditions force $\frac{\partial}{\partial t}\mathbf{u}(x,t) = 0$ on the boundaries. While Neumann boundary conditions are mathematically convenient, they require us to extrapolate data outside of the original image. Periodic boundary conditions assume the displacement field is the same on opposite boundaries. If we consider that same person and his/her same chest scans with periodic boundary conditions, it would mean the displacement field would exhibit the same behavior at the top and bottom of the patient's care. It is obvious that, an image of two different sections of your chest can never be the same, making it quite clear why the current boundary conditions are not physically meaningful. The definition of Dirichlet, Neumann and periodic boundary conditions are given in Table 2.3.

In this thesis, we introduce sliding and bending boundary conditions for use in registration. Sliding boundary conditions would fix the problems affecting Dirichlet boundary conditions because it would allow the image on the boundary to slide along that boundary. So if a person is not lying in the exact same position as in previous images, sliding boundary conditions will be able to shift the images around to register them correctly. While bending boundary conditions also suffers from the same constraints afflicting Neumann boundary conditions, they allow the flexibility of the boundary. If a person is lying on an angle or is tilted in one image, bending boundary

Boundary Condition	$B[\mathbf{u}(x,t)] = K$
Homogeneous Dirichlet	$\mathbf{u}(x,t)$, $K = 0$
Homogeneous Neumann	$\nabla(\mathbf{u}(x,t))$, $K = 0$
Periodic	$K = \mathbf{u}(0,t) = \mathbf{u}(\ell,t)$, $K = \nabla(\mathbf{u}(0,t)) = \nabla(\mathbf{u}(\ell,t))$

Table 2.3: Standard Boundary Conditions

conditions will be able to bend the boundary so that it aligns with the previous image taken.

In this paper, we will develop a formal definition of sliding and bending boundary conditions for use in image registration. Note that [9] mentions using a DFT-based approach with bending boundary conditions for a diffusion regularizer, but we will show how to use both sliding and bending boundary conditions in a way that can be generalized to all regularizers. Furthermore we will explore various numerical algorithms for solving the resulting E-L equations with sliding and bending boundary conditions, and we will show how to compute a numerical solution rapidly using techniques based on the Discrete Fourier Transform (DFT). We will illustrate the new algorithms on a toy problem and then show how they perform on a real-world image registration problem. Our expectations are that both sliding and bending boundary conditions will enhance the results of image registration used in medical imaging because they are more physically meaningful than Dirichlet, Neumann, or periodic boundary conditions in this setting.

The rest of this thesis is organized into two chapters. Chapter 3 is broken into five parts, the introduction and four sections. The beginning of chapter 3 introduces a toy problem which is an alteration of the diffusion equation and summarizes the remainder of chapter 3. Section 3.1 establishes the mathematical framework to solve a one-dimensional scalar diffusion equation with the Crank-Nicolson method and also enforces Dirichlet and Neumann boundary conditions. It then touches on how to solve higher dimensional problems with the same boundary conditions. Section 3.3 takes us from solving a scalar version of the diffusion equation to working on a vector field

and being able to solve vector-valued diffusion equations with Dirichlet and Neumann boundary conditions. Section 3.4 introduces sliding and bending boundary conditions, describing how they arise from the other boundary conditions, and explains how to incorporate them into the solution method of the existing problem. Section 3.5 provides more numerical solution methods and demonstrates their use. Chapter 4 shows how to apply what was discussed in Chapter 3 to image registration, and then provides examples of the image registration problem being solved with various solutions. Finally chapter 4 ends with a discussion of the pros and cons of using the various methods and sliding/bending boundary conditions vs. Dirichlet/Neumann boundary conditions.

Chapter 3

Solving the Diffusion Equation in Multiple Dimensions with Various Boundary Conditions

As stated in Chapter 1, we are working with the diffusion equation (2.6) and hope to invoke sliding or bending boundary conditions for use in non-parametric registration. However, before implementing techniques for solving non-rigid registration problems it is necessary to ensure that sliding and bending boundary conditions are feasible. A very basic toy problem (where $f(x,t,u) = 0$, $\ell = 1$) can be used to illustrate how to numerically approximate solutions to the diffusion equation under various boundary conditions:

$$\begin{aligned} \frac{\partial}{\partial t}u(x,t) - \alpha^2 \Delta u(x,t) &= 0 \\ 0 < x < 1, 0 < t& \\ u(x,0) &= u_0 \\ B[u(x,t)] = 0, \quad x \in \partial\Omega& \end{aligned} \tag{3.1}$$

In the remainder of this chapter, we will work through numerically solving (3.1). We begin by discretizing (3.1) and computing the eigenvalues for the Crank-Nicolson

Algorithm 1 Algorithm for Numerically Solving the Diffusion Equation.

Select input u_t.
for $t = 1$ to n **do**

$$\text{Compute } y = \begin{bmatrix} (1-\lambda) & \frac{\lambda}{2} & 0 & \cdots & 0 \\ \frac{\lambda}{2} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \frac{\lambda}{2} \\ 0 & \cdots & 0 & \frac{\lambda}{2} & (1-\lambda) \end{bmatrix} u$$

Take $DST(y) = x$.
Compute $\frac{x}{c} = z$ where c are the eigenvalues.
Take $DST^{-1}(z) = u_{t+1}$.
end for

method. Then we will numerically solve the Diffusion Equation with Dirichlet boundary conditions in one dimension. Finally we will discuss how to discretize (3.1), compute the eigenvalues, and solve the Diffusion Equation in two and three dimensions. Algorithm [1] shows how we will numerically solve (3.1) using the Crank-Nicolson method.

3.1 Discretizing the Toy Problem using the Crank-Nicolson Method

To numerically approximate solutions to (3.1), we must discretize in time (t) and space (x). Two standard ways of discretizing in time are the Forward-Difference and Backward-Difference methods, given by

$$\frac{u(x,t+k) - u(x,t))}{k} - \alpha^2 \Delta(u(x,t)) = 0 \tag{3.2}$$

$$\frac{u(x,t) - u(x,t-k))}{k} - \alpha^2 \Delta(u(x,t)) = 0 \tag{3.3}$$

respectively, where k is the chosen time step. In space we chose to discretize using a

centered difference. In one spatial dimension, we have:

$$\begin{aligned}
\frac{\partial^2}{\partial x^2}u_{i,j} &\approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \quad \text{on the interior} \\
\frac{\partial^2}{\partial x^2}u_{i,j} &\approx \frac{2u_{i+1,j} - 2u_{i,j}}{h^2} \quad \text{if} \quad i = 1 \\
\frac{\partial^2}{\partial x^2}u_{i,j} &\approx \frac{2u_{i-1,j} - 2u_{i,j}}{h^2} \quad \text{if} \quad i = m
\end{aligned} \tag{3.4}$$

where $u(x,t)$ is discretized in space and time to form $u_{i,j}$ where $i = 1 : m$ and $j = 0 : n$. As you can see, if $i = 1$ or $i = m$ then the the equation for the interior points will not be applicable because it would require us to extrapolate data outside of our range. We called the points outside of the range ghost points and assumed they were symmetric to the points in the range. This allowed us to substitute $u_{i+1,j}$ or $u_{i-1,j}$ for each other when necessary to form the equations needed to calculate the boundary points. In two or more spatial dimensions, Δ can be discretized to form $\ddot{\Delta}$, which is simply the sum of 2nd partials of the form (3.4).

One basic method for approximating the solution to (3.1) is the Crank-Nicolson method [1] for solving the diffusion equation. This method averages the Forward-Difference method at the j^{th} time step with the $j + 1^{th}$ time step of the Backward-Difference, transforming (3.1) into

$$\begin{aligned}
&\tfrac{u_{i,j+1}-u_{i,j}}{k} - \tfrac{\alpha^2}{2}\left[\tfrac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2} + \tfrac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}}{h^2}\right] = 0 \quad \text{on the interior} \\
&\tfrac{u_{i,j+1}-u_{i,j}}{k} - \tfrac{\alpha^2}{2}\left[\tfrac{2u_{i+1,j}-2u_{i,j}}{h^2} + \tfrac{2u_{i+1,j+1}-2u_{i,j+1}}{h^2}\right] = 0 \quad \text{if} \quad i = 1 \\
&\tfrac{u_{i,j+1}-u_{i,j}}{k} - \tfrac{\alpha^2}{2}\left[\tfrac{2u_{i-1,j}-2u_{i,j}}{h^2} + \tfrac{2u_{i-1,j+1}-2u_{i,j+1}}{h^2}\right] = 0 \quad \text{if} \quad i = m
\end{aligned} \tag{3.5}$$

If we rearrange this formula so that all the $u_{i,j+1}$s are on one side and all the $u_{i,j}$s on

the other then we get the matrix form of the Crank-Nicolson equation.

$$\begin{bmatrix} (1+\lambda) & \frac{-\lambda}{2} & 0 & \cdots & 0 \\ \frac{-\lambda}{2} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \frac{-\lambda}{2} \\ 0 & \cdots & 0 & \frac{-\lambda}{2} & (1+\lambda) \end{bmatrix} \mathbf{u}_{i,j+1} = \begin{bmatrix} (1-\lambda) & \frac{\lambda}{2} & 0 & \cdots & 0 \\ \frac{\lambda}{2} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \frac{\lambda}{2} \\ 0 & \cdots & 0 & \frac{\lambda}{2} & (1-\lambda) \end{bmatrix} \mathbf{u}_{i,j} \tag{3.6}$$

We can compute the right-hand side outright, so

$$\begin{aligned} \hat{u}_i &= \frac{\lambda}{2}u_{i+1,j} + (1-\lambda)u_{i,j} + \frac{\lambda}{2}u_{i-1,j}, \quad \text{as} \quad j = 1 : n-1 \\ \hat{u}_i &= \lambda * u_{i+1,j} + (1-\lambda)u_{i,j} \quad \text{if} \quad i = 1 \\ \hat{u}_i &= \lambda * u_{i-1,j} + (1-\lambda)u_{i,j} \quad \text{if} \quad i = m \end{aligned} \tag{3.7}$$

where $\lambda = \frac{\alpha^2 \cdot k}{h^2}$.

3.1.1 Computing the Eigenvalues of the Toy Problem using the Crank-Nicolson Method

To solve (3.6) rapidly, we can exploit the eignevalue/eigenvector decomposition of A (the left-hand side (LHS) matrix in (3.6)). The corresponding eigenvalues were found by simplifying

$$\begin{aligned} \text{LHS} &= -\frac{\lambda}{2}u_{i+1,j+1} + (1+\lambda)u_{i,j+1} - \frac{\lambda}{2}u_{i-1,j+1} \\ \text{LHS} &= -\lambda * u_{i+1,j+1} + (1+\lambda)u_{i,j+1} \quad \text{if} \quad i = 1 \\ \text{LHS} &= -\lambda * u_{i-1,j+1} + (1+\lambda)u_{i,j+1} \quad \text{if} \quad i = m \end{aligned} \tag{3.8}$$

which is the LHS of (3.6). MATLAB defines the Discrete Fourier Transform (DFT)

by:

$$\tilde{x_k} = \frac{1}{N} \sum_{k=0}^{N-1} x_j e^{\frac{2\pi(j)(k)}{N}} \tag{3.9}$$

The DST can be computed by the DFT as follows. We use Euler's formula for $e^{i\theta}$ to get:

$$\begin{aligned} \tfrac{1}{N} \sum_{k=0}^{N-1} x_j e^{\frac{2\pi(j)(k)}{N}} &= \tfrac{1}{N} \sum_{k=0}^{N-1} x_j \left[\cos\left(\tfrac{2\pi(j)(k)}{N}\right) - i \sin\left(\tfrac{2\pi(j)(k)}{N}\right) \right] \\ &= \tfrac{1}{N} \sum_{k=0}^{N-1} x_j \cos\left(\tfrac{2\pi(j)(k)}{N}\right) - i \tfrac{1}{N} \sum_{k=0}^{N-1} x_j \sin\left(\tfrac{2\pi(j)(k)}{N}\right) \end{aligned} \tag{3.10}$$

This equation is in fact the DCT minus i multiplied by the DST.

First we establish homogeneous Dirichlet conditions by providing an input array of $\vec{u} = [u_0, u_1 \ldots, u_N]$ where

$$u_k = \begin{cases} 0 & , \text{if} \quad k = 0, N \\ \sin\left(\pi(h)(k-1)\right) & , \text{otherwise} \end{cases} \tag{3.11}$$

where h is the spatial step size. The eigenvalues can be found by expanding the left-hand side in terms of the Discrete Sine Transform (DST):

$$
\begin{aligned}
-\tfrac{\lambda}{2}u_{i+1,j+1} + (1+\lambda)u_{i,j+1} - \tfrac{\lambda}{2}u_{i-1,j+1} = \\
&\tfrac{-\lambda}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,j+1}\sin\left(\tfrac{\pi\ell(i)}{N}\right)\cos\left(\tfrac{\pi\ell}{N}\right)\right] \\
&+\tfrac{-\lambda}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,j+1}\sin\left(\tfrac{\pi\ell}{N}\right)\cos\left(\tfrac{\pi\ell(i)}{N}\right)\right] \\
&+(1+\lambda)\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,j+1}\sin\left(\tfrac{\pi\ell(i)}{N}\right)\right] \\
&+\tfrac{-\lambda}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,j+1}\sin\left(\tfrac{\pi\ell(i)}{N}\right)\cos\left(\tfrac{\pi\ell}{N}\right)\right] \\
&-\tfrac{-\lambda}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,j+1}\sin\left(\tfrac{\pi\ell}{N}\right)\cos\left(\tfrac{\pi\ell(i)}{N}\right)\right] \\
= \tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,j+1}\sin\left(\tfrac{\pi\ell(i)}{N}\right)&\left[-\lambda\cos\left(\tfrac{\pi\ell}{N}\right)+(1+\lambda)\right] \qquad (3.12)
\end{aligned}
$$

Thus the eigenvalues are

$$\mu_\ell = -\lambda\cos\left(\tfrac{\pi\ell}{N}\right) + 1 + \lambda. \qquad (3.13)$$

Note that if we assume Neumann boundary conditions, expanding in terms of the Discrete Cosine Transform (DCT) instead of the DST yields the same eigenvalues.

3.1.2 Solving the Diffusion Equation

Now we can formulate the algorithms that will solve the diffusion equation using the Crank-Nicolson method. At each time step, (3.7) computes the i^{th} entry of the right-hand side (call this $\vec{y}$). Then we take the DST of $\vec{y}$, divide by the corresponding eigenvalue, and take the inverse DST of the result to obtain $u_{i,j+1}$. To switch from homogeneous Dirichlet boundary conditions to homogeneous Neumann boundary conditions, we need to change the entries of the input array to $u_k = \cos(\pi(h)(k))$ as k goes from 0 to N and use the DCT instead of the DST.

3.2 Solving the Diffusion Equation in Multiple Dimensions

Sliding and bending boundary conditions are only meaningful for vector functions in two or higher dimensions. Becuase they are made up of combinations of Dirichlet and Neumann conditions, sliding and bending boundary conditions default to simply Dirichlet or Neumann in one dimension. Thus it is necessary to create toy examples of the diffusion problem on 2-D domains. The main differences between solving the toy problem in 2-D versus 1-D are:

1. Computing the eigenvalues of (3.6) in two dimensions.

2. Figuring out how to compute the matrix $\hat{y}$ after computing (3.6) in two dimensions.

3. Determining appropriate initial conditions to form input matrix u.

4. Extending u to be vector-valued.

Recall that we used (3.8) to compute the eigenvalues before. If we expand this to get,

$$\text{LHS} = u_{i,j+1} + \left[\frac{-\lambda}{2}u_{i-1,j+1} + \lambda u_{i,j+1} + \frac{-\lambda}{2}u_{i+1,j+1}\right] \tag{3.14}$$

it is easy to see that the terms in the brackets arise from discretizing in space and the other term arises from discretizing time. In 2-D space, we now have $u(x,t) = u_{i,j,k}$, where $i = 1:m$, $\ell = 1:p$, and $j = 0:n$. Therefore if we discretize in both the x and y spatial directions, (3.14) becomes:

$$\begin{aligned}\text{LHS} =& u_{i,\ell,j+1} + \left[\frac{-\lambda_x\lambda_y}{2}w_{i-1,\ell,j+1} + \frac{\lambda_x\lambda_y}{2}u_{i+1,\ell,j+1}\right] \\ &+ \lambda_x\lambda_y u_{i,\ell,j+1} + \left[\frac{-\lambda_x\lambda_y}{2}u_{i,\ell-1,j+1} + \frac{\lambda_x\lambda_y}{2}u_{i,\ell+1,j+1}\right]\end{aligned}$$

Using the same technique from (3.12) to expand this in terms of the DST yields eigenvalues of:

$$\mu_{i,\ell} = 1 + \lambda_x\lambda_y - \lambda_x\lambda_y \cos(a_x)\cos(a_y) \tag{3.15}$$

where $a_x = \frac{\pi i}{N}$ and $a_y = \frac{\pi \ell}{N}$ are the angles in the x and y directions respectively. At each time step, the i, ℓ^{th} entry of $\hat{y}$ will be

$$\hat{y}_{i,\ell,j} = (1 - \lambda_x\lambda_y)u_{i,\ell,j} + \tfrac{\lambda_x}{2}u_{i+1,\ell,j} + \tfrac{\lambda_y}{2}u_{i,\ell+1,j} + \tfrac{\lambda_x}{2}w_{i-1,\ell,j} + \tfrac{\lambda_y}{2}u_{i,\ell-1,j}. \tag{3.16}$$

Now that we know what the eigenvalues are going to be and how to compute our $\hat{y}$ matrix from the given input matrix w we need to determine an appropriate inital condition consistent with the chosen boundary conditions for the toy problem. If we choose homogeneous Dirichlet boundary conditions, one option is:

$$u_{i,\ell,0} = \sin\left(\frac{\pi(h_x)(i-1)}{L_x}\right)\sin\left(\frac{\pi(h_y)(\ell-1)}{L_y}\right) \tag{3.17}$$

where h_x,h_y are the spatial steps in x and y resp. and L_x,L_y are the lengths of the spatial interval in x and y respectively. To solve the Crank-Nicolson system we compute our new matrix $\hat{y}$ and then take the DST in both the x and y direction, then divide by the corresponding eigenvalues in each direction and take the inverse DST in both directions. Again, we can switch from homogeneous Dirichlet to homogeneous Neumann boundary conditions by using the same techniques we used while switching in the 1-D domain and employing the DCT in place of the DST.

Note that in 3-D space we have $u(x,t) = u_{i,\ell,k,j}$ where $i = 1:m$, $\ell = 1:p$, $k = 1:q$,

and $j = 0 : n$. Thus when we discretize in the x, y, and z directions, (3.14) becomes:

$$\begin{aligned}\text{LHS} =& u_{i,\ell,k,j+1} + \left[\frac{-\lambda_x\lambda_y\lambda_z}{2}u_{i-1,\ell,k,j+1} + \frac{\lambda_x\lambda_y\lambda_z}{2}u_{i+1,\ell,k,j+1}\right] \\ &+ \lambda_x\lambda_y\lambda_z u_{i,\ell,k,j+1} + \left[\frac{-\lambda_x\lambda_y\lambda_z}{2}u_{i,\ell-1,k,j+1} + \frac{\lambda_x\lambda_y\lambda_z}{2}u_{i,\ell+1,k,j+1}\right] \\ &+ \left[\frac{-\lambda_x\lambda_y\lambda_z}{2}u_{i,\ell,k-1,j+1} + \frac{\lambda_x\lambda_y\lambda_z}{2}u_{i,\ell,k+1,j+1}\right]\end{aligned}$$

Using the same techniques we did while computing the eigenvalues in one and two dimensions, we obtain eigenvalues of:

$$\mu_{i,\ell,k} = 1 + \lambda_x\lambda_y\lambda_z - \lambda_x\lambda_y\lambda_z \cos(a_x)\cos(a_y)\cos(a_z)\,. \tag{3.18}$$

The conversion from 2-D to 3-D will require the same additions that the conversion from 1-D to 2-D did, meaning we have to provide the appropriate 3-D input cube w, discretize w in all three directions to compute $\hat{y}$, and solve the resulting system by taking the DST in each dimension (to account for x,y,z directions), dividing by the new eigenvalues (3.18), and taking the inverse DST along each dimension.

3.3 Vector Fields

Thus far Dirichlet and Neumann boundary conditions have only been applied to scalar functions. However, sliding and bending are inherent properties of a vector field, not a scalar function. To incorporate sliding and bending, we must generalize our previous toy problem and its solution to the case of vector-valued functions. This means essentially turning (3.1) into:

$$\begin{aligned} \frac{\partial}{\partial t}\vec{u}(\vec{x},t) = \alpha^2\Delta\vec{u}(\vec{x},t), \quad \text{where} \quad \vec{u}(\vec{x},t) = \begin{pmatrix} u_1(\vec{x},t) \\ u_2(\vec{x},t) \\ \vdots \\ u_n(\vec{x},t) \end{pmatrix} \\ 0 < x_i < 1 \quad \text{for i} = 1,2,...,n, 0 < t \\ \vec{u}(\vec{x},0) = \vec{u}_0 \\ B[\vec{u}(\vec{x},t)] = 0, \quad x \in \partial\Omega \end{aligned} \tag{3.19}$$

Transitioning into vector valued equations will allow us to impose different boundary conditions for each component. For example if we visualize our 3-D example again, we can enforce sliding boundary conditions on all faces by choosing the appropriate selection of Dirichlet and Neumann boundary conditions on different faces for each component of $\vec{u}$.

In image registration, vector fields describe the deformations between two or more images. Therefore, we will take the toy problem and generalize it for vector valued equations. To do this we will simply duplicate what we have already developed for each additional component in the vector field. For example, if we want to impose homogeneous Dirichlet boundary conditions on a two dimensional vector field we must create two input matrices (call them u_1, u_2) where each of them is represented by (3.17). Then we compute the new matrices $\hat{y}_1$ and $\hat{y}_2$, take the DST of both in

the x and y directions, divide by the corresponding eigenvalues and take the inverse DST in both directions. Note that the results of this process will be the same unless we enforce different time step or spatial step sizes. The process is analogous when imposing homogeneous Neumann boundary conditions. Figures (3.1)–(3.3) show basic 2-D surfaces, vector fields under Dirichlet and Neumann boundary conditions, and zoomed-in versions of the vector fields.

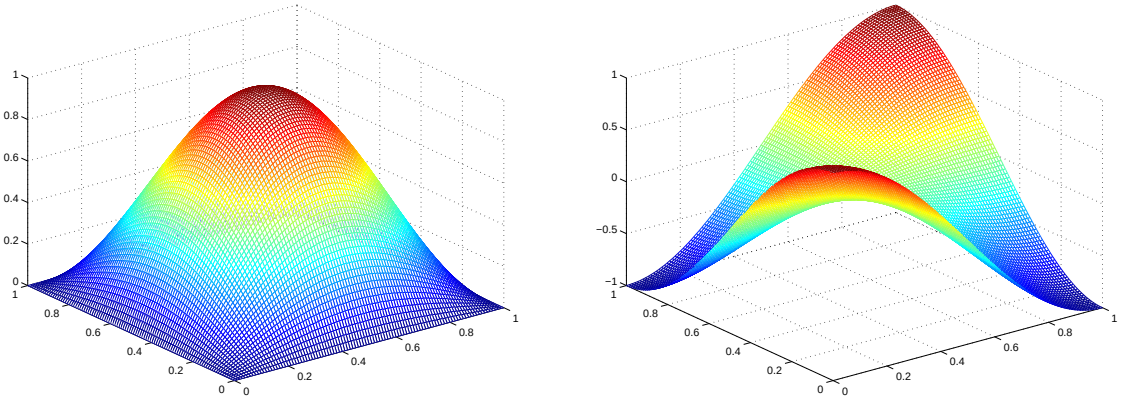


Figure 3.1: Surfaces of u under Dirichlet (left) and Neumann (right) boundary conditions

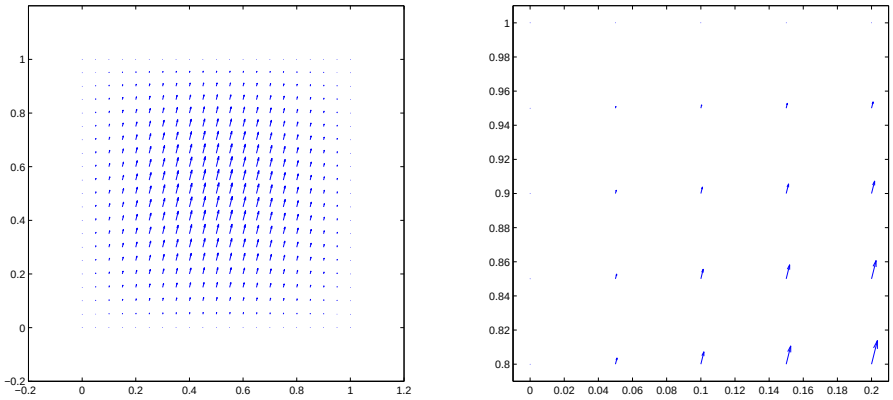


Figure 3.2: Vector field under Dirichlet Boundary Conditions

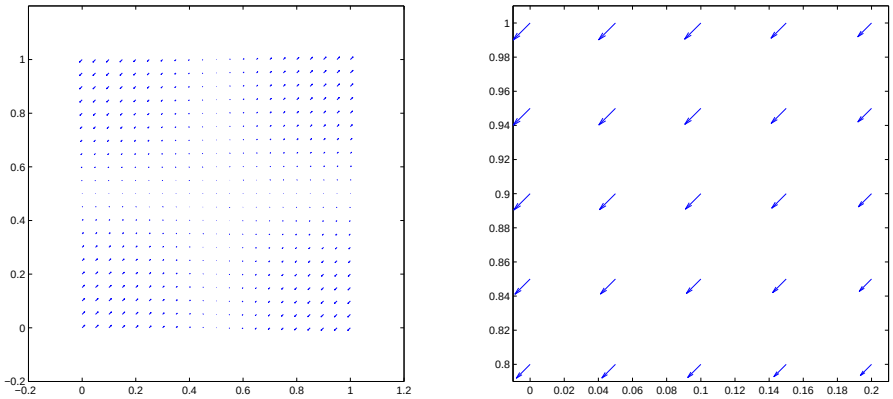


Figure 3.3: Vector field under Neumann Boundary Conditions

3.4 Sliding/Bending Boundary Conditions

Sliding and bending boundary conditions are essentially a mix of Dirichlet and Neumann boundary conditions applied to each component of the vector field. Sliding boundary conditions will allow a vector function to point along its boundaries and bending boundary conditions will allow a vector function to point orthogonally to its boundaries. To obtain sliding boundary conditions, the component of u orthogonal to each edge (or face of a cube) should be zero, however, we will allow the vector field to obey Neumann boundary conditions along its edges/faces. Bending boundary conditions come from forcing a point on the the edge (or face) to only move perpendicularly to the boundary but not along edge (or face). This is obtained by having a Neumann boundary condition in the direction of the corresponding component and then Dirichlet boundary conditions in the remaining directions. Figures (3.4) and (3.6) show example surfaces of u in the x-component and the y-component sliding and bending boundary conditions, respectively. Figure () shows the deformation grids of u after the first eight time steps. It can be seen in the figure that the deformation grids become more like a checkerboard. This is exactly what we would expect, because a checkerboard grid means u has completely diffused. Note that we could not achieve desirable figures to show the deformation grid for bending boundary conditions.

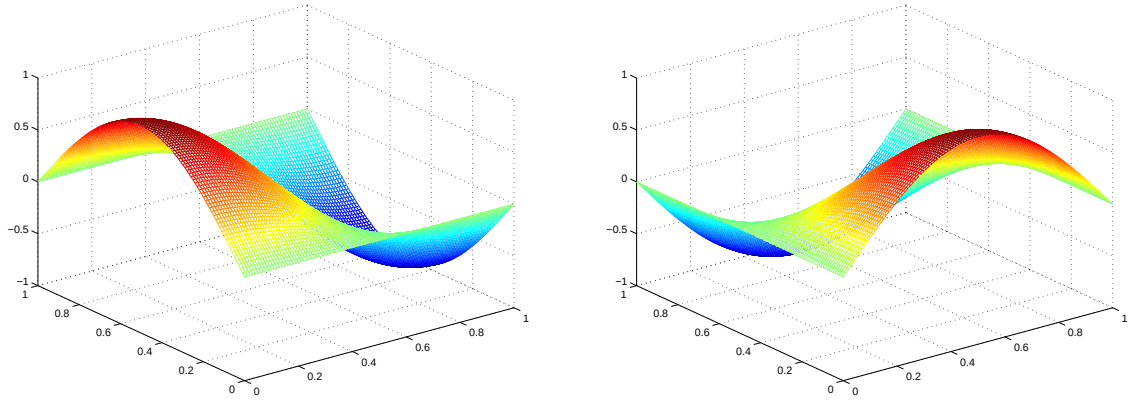
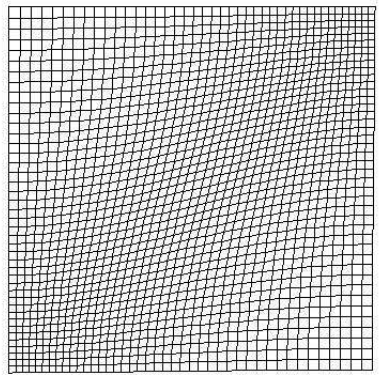
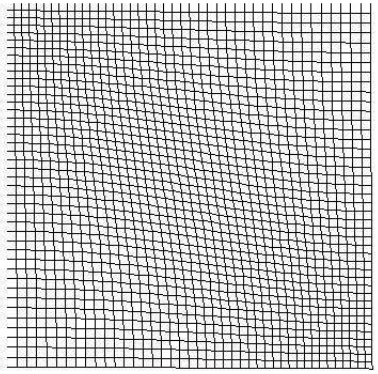


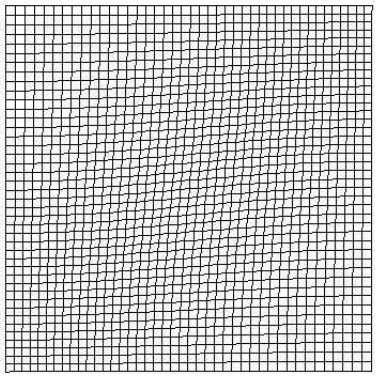
Figure 3.4: Surface of x-component (left) and y-component (right) with sliding boundary conditions.



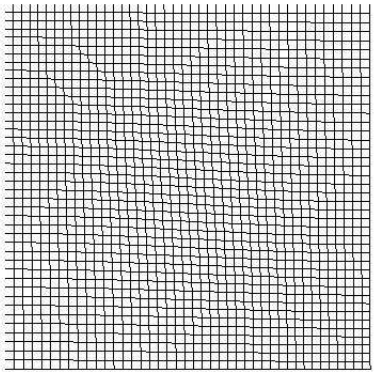
(a) $t = t_0$



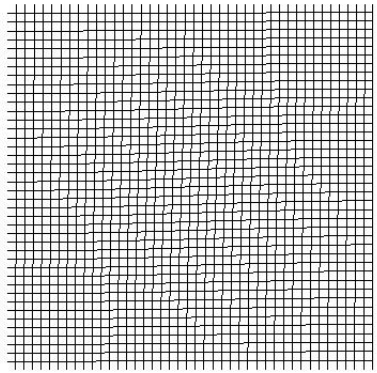
(b) $t = t_1$



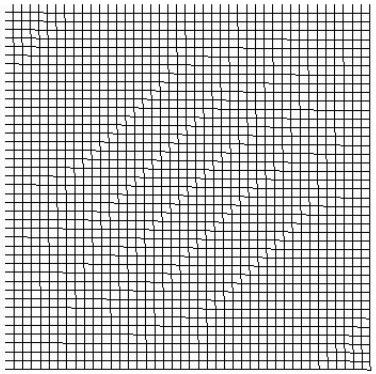
(c) $t = t_2$



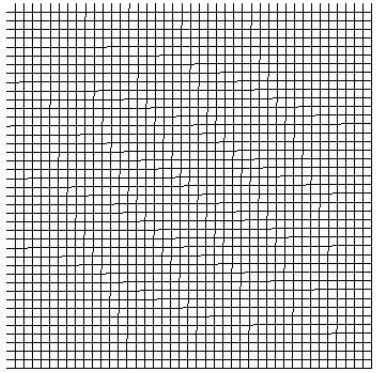
(d) $t = t_3$



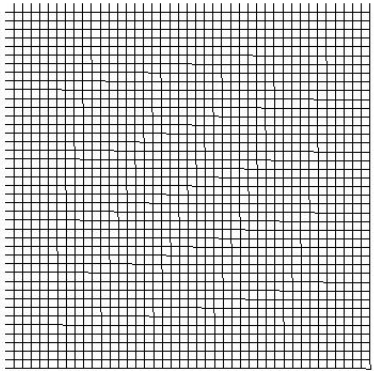
(e) $t = t_4$



(f) $t = t_5$



(g) $t = t_6$



(h) $t = t_7$

Figure 3.5: Deformation grids of u for the first eight time steps with sliding boundary conditions.

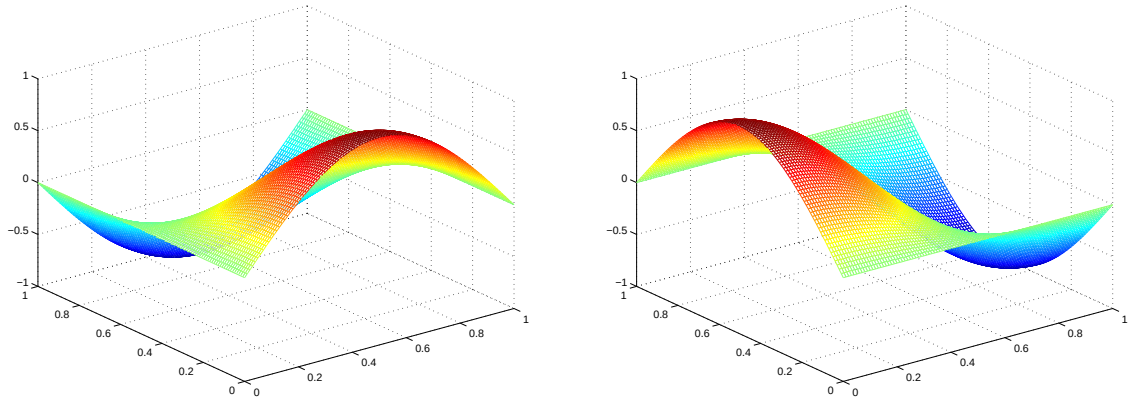


Figure 3.6: Surface of x-component (left) and y-component (right) with bending boundary conditions.

We will be able to incorporate sliding and bending boundary conditions by putting together the algorithms used for Dirichlet and Neumann conditions. When doing this it is important to make sure that the right conditions are imposed on the correct components. For example, to establish sliding boundary conditions on a 2-D domain, we will select an initial condition u with components u_1,u_2 as follows:

$$u_1 = \sin\left(\frac{\pi \cdot h_x(i-1)}{L_x}\right)\cos\left(\frac{\pi \cdot h_y(j-1)}{L_y}\right), i = 1 : m_x + 1, j = 1 : 2m_x + 1 \tag{3.20}$$

$$u_2 = \cos\left(\frac{\pi \cdot h_x(i-1)}{L_x}\right)\sin\left(\frac{\pi \cdot h_y(j-1)}{L_y}\right), i = 1 : 2m_x + 1, j = 1 : m_x + 1. \tag{3.21}$$

We will do our usual calculations to find $\hat{y}_1$ and $\hat{y}_2$. But now we have to take the DST in the x direction, then the DCT in the y direction of $\hat{y}_1$, divide by the eigenvalues and take the inverse DST and the inverse DCT in the corresponding direction to obtain the result. For $\hat{y}_2$, we need to take the DST in the y direction then the DCT in the x direction of $\hat{y}_2$, divide by the eigenvalues and take the inverse DST and the inverse DCT in the corresponding direction to obtain the result for the second component.

3.5 Various Methods for Solving the Diffusion Equation

Before we delve into the various solution methods that can also be applied, we would like to note that we have used MATLAB to assist us through these calculations. To solve the diffusion equation in MATLAB using Crank-Nicolson we need to provide an input array, matrix, or cube $\vec{u}$ depending on how many dimensions we are working with. Once we have $\vec{u}$ (or if we have a vector field, once we have $\vec{u}$ defined componentwise) we compute $\hat{y}$. Depending on the type of boundary conditions imposed, take the DST or DCT in the corresponding directions, divide by the eigenvalues, and take the inverse DST or inverse DCT in the corresponding directions. Note that in order to use MATLAB's definition of the DCT (which differs from (3.10)) we have to symmetrically replicate $\vec{y}$ prior to computing the DCT.

So far the only numerical method we have illustrated for solving the diffusion equation is the Crank-Nicolson method. However, there are many other numerical methods that can be used to solve a diffusion equation and we will explore some of the more common methods. We have chosen two explicit methods, the Forward Euler method and Heun's method, as well as three implicit methods, the Backward Euler method, the Modified Euler method and the fourth-order Runge-Kutta method. Note that (3.16) requires computing a scaled version of the discrete Laplacian at each element. However, all the previously mentioned methods require computing an unscaled version of the discrete Laplacian. Thus if we create a function to compute the discrete Laplacian of $\hat{y}$ it eliminates the process of computing each element explicitly. This makes switching between methods much simpler because we compute the new matrix $\hat{y}$ in the other methods by calling the discrete Laplacian function on u and insert that into the formula for the solution method of our choice.

3.5.1 Forward Euler

We can substitute the Forward Euler method [1] for the Crank-Nicolson method in the toy problem (3.1) so that (3.5) becomes

$$u_{t+1} = u_t + k \cdot g(x, u_t), \tag{3.22}$$

where α^2 has been absorbed into k. We now introduce $g(x, u_t) = \ddot{\Delta} u_t(x)$, so (3.22) becomes

$$u_{t+1} = u_t + k \cdot \ddot{\Delta} u_t \tag{3.23}$$

Notice that since this method is explicit in time, we do not need to use the DST or DCT when computing the solution; we simply compute u_{t+1} from our given input $\vec{u}$. Because there is no guarantee of convergence, the time step k must typically be chosen to be extremely small. For this reason we decided not to implement the Forward Euler Method for use in image registration.

3.5.2 Heun's Method

Heun's Method [1] is given by

$$u_{t+1} = u_t + \tfrac{k}{4}\left[g(x, u_t) + 3 \cdot g\left(x + \tfrac{2}{3}k, u_t + \tfrac{2}{3}g(x, u_t)\right)\right]. \tag{3.24}$$

Note that α^2 has been absorbed into k as in the Forward Euler method. Since this method is also explicit, all we have to do is expand the right-hand side so that it is completely in terms of u_t. So (3.24) becomes

$$u_{t+1} = u_t + \tfrac{k}{4}\ddot{\Delta} u_t + \tfrac{3k}{4}\ddot{\Delta}\left[u_t + \tfrac{2k}{3} g(x + \tfrac{2k}{3}, u_t)\right]$$
$$u_{t+1} = u_t + \tfrac{k}{4}\ddot{\Delta} u_t + \tfrac{3k}{4}\ddot{\Delta} u_t + \tfrac{k}{2}\ddot{\Delta}^2 u_t\left(x + \tfrac{2k}{3}\right)$$

To handle $u_t(x + \frac{2k}{3})$ we let it be equal to $u_t + \frac{2k}{3}\ddot{\Delta}u_t$.

$$u_{t+1} = u_t + k\ddot{\Delta}u_t + \tfrac{k^2}{2}\ddot{\Delta}^2\left[u_t + \tfrac{2k}{3}\ddot{\Delta}u_t\right]$$
$$u_{t+1} = u_t + k\ddot{\Delta}u_t + \tfrac{k^2}{2}\ddot{\Delta}^2 u_t + \tfrac{k^3}{3}\ddot{\Delta}^3 u_t \tag{3.25}$$

Thus, we will take our given input $\vec{u}$ and compute the right-hand side of (3.25) for each component of $\vec{u}$ to find the solution. Note that Heun's Method is explicit in time just like Forward Euler, so we run the risk of non-convergence with this method.

3.5.3 Backward Euler

Another option that avoids the problems of explicit iterations is to substitute the Backward Euler method [1] for the Crank-Nicolson method. Therefore (3.5) will become:

$$u_{t+1} = u_t + k \cdot g\left(x, u_{t+1}\right), \tag{3.26}$$

with g defined as in (3.22) so (3.26) becomes

$$u_{t+1} = u_t + k \cdot \ddot{\Delta}u_{t+1}, \quad \text{or}$$
$$u_{t+1} - k\ddot{\Delta}u_{t+1} = u_t. \tag{3.27}$$

Since we know from (3.14) and (3.13) that $\ddot{\Delta}$ has eigenvalues of $\lambda - \lambda\cos(\frac{\pi\ell}{N})$, then the eigenvalues of $u_{t+1} - k\ddot{\Delta}u_{t+1}$ are

$$\mu_\ell = 1 - k\left(\lambda - \lambda\cos\left(\tfrac{\pi\ell}{N}\right)\right) \tag{3.28}$$

Note that the Backward Euler method is implicit. To solve (3.27), take the DST and/or DCT in the x,y, and z directions on the correct components of $\vec{u}$, divide by

the eigenvalues (3.28) and take the inverse DST and/or inverse DCT to recover $\vec{u}$ at the next time step.

3.5.4 Modified Euler

Modified Euler [1] is

$$u_{t+1} = u_t + \tfrac{k}{2}\left[g(x, u_t) + g(x, u_{t+1} + k \cdot g(x, u_t))\right] \tag{3.29}$$

where t is time and k is our time step. Now let $g(x, u_t) = \ddot{\Delta} u_t(x)$ and we can consider $u_t = u_{i,j,t}$ so that it reflects our notation that was used in (3.16) except we are using t as the time variable and i,j as the spatial variables. Therefore the right-hand side becomes,

$$\begin{aligned} &= u_t + \tfrac{k}{2}\ddot{\Delta} u_t + \tfrac{k}{2}\ddot{\Delta}\left[u_{t+1} + k \cdot g(x, u_t)\right] \\ &= u_t + \tfrac{k}{2}\ddot{\Delta} u_t + \tfrac{k}{2}\ddot{\Delta} u_{t+1} + \tfrac{k^2}{2}\ddot{\Delta}^2 u_{t+1}. \end{aligned}$$

If we move all the terms with u_{t+1} to one side and the terms with u_t to the other we get:

$$u_{t+1} - \tfrac{k}{2}\ddot{\Delta} u_{t+1} - \tfrac{k^2}{2}\ddot{\Delta}^2 u_{t+1} = u_t + \tfrac{k}{2}\ddot{\Delta} u_t \tag{3.30}$$

Therefore the eigenvalues of the operator $I - \frac{k}{2}\ddot{\Delta} - \frac{k^2}{2}\ddot{\Delta}^2$ are

$$\mu_\ell = 1 + \lambda - \tfrac{\lambda}{2}\cos\left(\tfrac{\pi\ell}{N}\right) - \tfrac{\lambda^2}{2}\left(\cos\left(\tfrac{\pi\ell}{N}\right)^2\right). \tag{3.31}$$

(The process of computing the eigenvalues can be seen in Appendix A.) When writing our new MATLAB code for the Modified Euler method, we will take our give input $\vec{u}$ and compute $\hat{y} = \vec{u} + \frac{k}{2}\ddot{\Delta}\vec{u}$. Then we take the DST and/or DCT in the x, y, and z directions on the correct components of $\vec{u}$, divide by the eigenvalues (3.31) and take

the inverse DST and/or inverse DCT.

3.5.5 Runge-Kutta

A more accurate discretization is the fourth-order Runge-Kutta (RK4) [1] given by:

$$
\begin{aligned}
u_{t+1} &= u_t + \tfrac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4), \quad \text{where} \\
K_1 &= k \cdot g(x, u_t) \\
K_2 &= k \cdot g\left(x + \tfrac{k}{2}, u_t + \tfrac{1}{2}K_1\right) \\
K_3 &= k \cdot g\left(x + \tfrac{k}{2}, u_t + \tfrac{1}{2}K_2\right) \\
K_4 &= k \cdot g(x, u_{t+1} + K_3)
\end{aligned}
\tag{3.32}
$$

Now substitute $G_i = u_t + \frac{K_i}{2}$. For $i = 1, 2$. Making this subtitution helps determine that:

$$
\begin{aligned}
K_2 &= k \cdot \ddot{\Delta} G_1 \\
K_3 &= k \cdot \ddot{\Delta} G_2
\end{aligned}
$$

Finally, letting $G_3 = u_t + K_3$ makes $K_4 = k \cdot \ddot{\Delta} G_3$. We can put this all together to solve $u_{t+1} = u_t + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$. Thus

$$
u_{t+1} - \tfrac{1}{6} \cdot \ddot{\Delta} u_{t+1} = u_t + \tfrac{1}{6} k \cdot \ddot{\Delta} u_t + \tfrac{2}{6} k \cdot \ddot{\Delta} G_1 + \tfrac{2}{6} k \cdot \ddot{\Delta} G_2 + \tfrac{1}{6} k \cdot \ddot{\Delta} K_3. \tag{3.33}
$$

As shown in Appendix A, the eigenvalues of the LHS operator are $\mu_\ell = 1 - \frac{1}{6} k \cdot \left(\lambda - \lambda \cos\left(\frac{\pi \ell}{N}\right)\right)$.

3.5.6 Visual Examples

Figures (3.7)–(3.8), (3.9)–(3.10), and (3.11)–(3.12) show solutions of the diffusion equation subject to various initial and boundary conditions using Heun's, Modified Euler, and the Runge-Kutta methods, respectively. All of the figures show two layers, the first layer is the initial condition and the second layer is the solution after the first time step. To make the figures appear similar, we had to impose different parameters for each solution method. For the Modified Euler method we used parameters of: L = [1,1], $T = 10$, $\alpha = 5$, m = [100,100], $n = 100$, and finally $k = \frac{T}{7500n}$. For Heun's method we used: L = [1,1], $T = 1$, $\alpha = 1$, m = [100,100], $n = 100$, and $k = \frac{T}{n}$. For Runge-Kutta method we used: L = [1,1], $T = 10$, $\alpha = 5$, m = [100,100], $n = 100$, and $k = \frac{T}{200n}$. Note that for the cases with Dirichlet boundary conditions we showed only half of the solution surface for easier visualization.

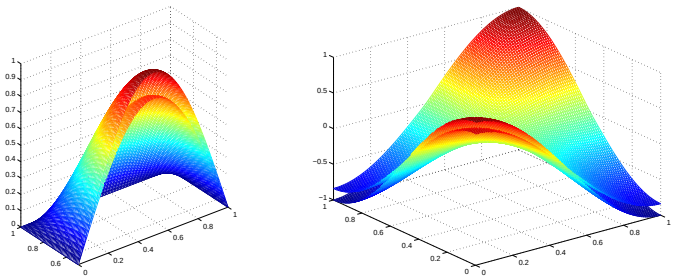


Figure 3.7: Shows the initial condition and the solution after one time step under Dirichlet (left) and Neumann (right) boundary conditions.

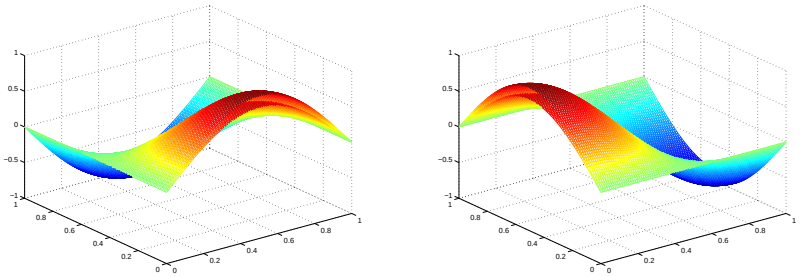


Figure 3.8: Shows the initial condition and the solution after one time step under sliding (left) and bending (right) boundary conditions.

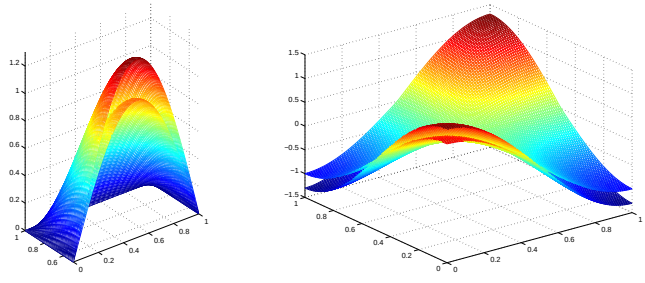


Figure 3.9: Shows the initial condition and the solution after one time step with Dirichlet (left) and Neumann (right) boundary conditions.

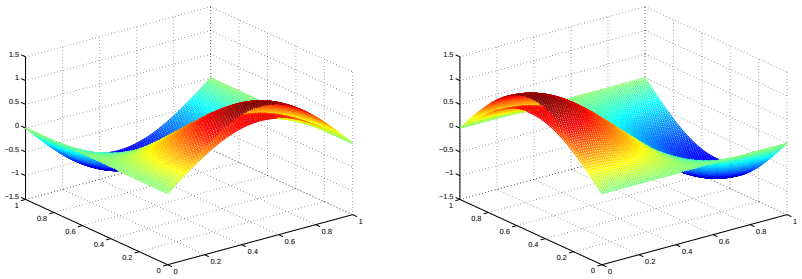


Figure 3.10: Shows the initial condition and the solution after one time step with sliding (left) and bending (right) boundary conditions.

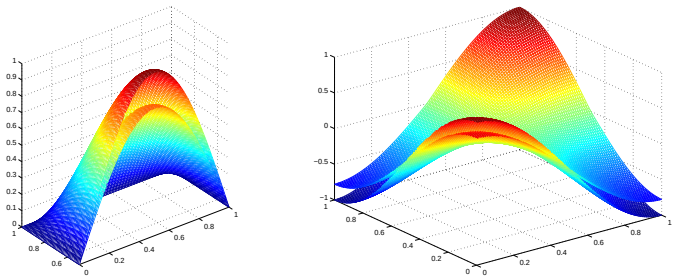


Figure 3.11: Shows the initial condition and the solution after one time step with Dirichlet (left) and Neumann (right) boundary conditions.

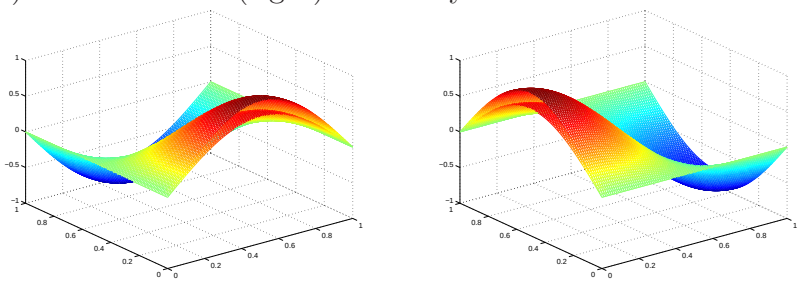


Figure 3.12: Shows the initial condition and the solution after one time step with sliding (left) and bending (right) boundary conditions.

Chapter 4

Application to Registration

The goal of non-rigid registration is to find a vector field that will transform a template image T into the frame of a reference image R. In this chapter, we will be considering images to be either two or three dimensional. (Although four dimensional CT and MRI images are sometimes used in cardiac imaging.) Non-rigid registration with a diffusion partial differential operator (PDO) can be achieved by finding the stationary solution to the following IBVP (2.6):

$$\begin{aligned} \frac{\partial}{\partial t}u(x,t) - \alpha^2\Delta(u(x,t)) &= f(x,t,u(x,t)) \\ 0 < x_i < 1 \quad \text{for i} = 1,2,...,n < \ell, 0 < t & \\ u(x,0) &= u_0 \\ B[u(x,t)] &= 0, \quad x \in \partial\Omega \end{aligned} \tag{4.1}$$

where α^2 is a constant and w.l.o.g. the right-hand side ($B[u(x,t)]$) is 0 (if not, insert a $-K$ into the left-hand side to form a new $B[u(x,t)]$). This differs from the toy problem (3.1) because no longer is $f(x,t,u(x,t)) \equiv 0$. $f(x,t,u(x,t))$ is non-trivial and defined by the variational derivative of the similarity measure. Non-rigid registration with either a curvature or elastic PDO is accomplished by changing A in (2.5) to the corresponding PDO found in Table (2.2).

Figure 4.1: The Prior slice (left) and the Current slice (middle) are shown above and the Current minus Prior slice (right)

4.1 Registration Examples

4.1.1 Two-dimensional Registration on Axial Slices of CT Lung Images

To illustrate the effect of both sliding and bending boundary conditions, we will consider registering 2-D lung CT images of a patient captured at different times. In figure (4.1) the prior and current slice (or R and T resp.) can be seen as well as an image that displays the difference D between them. Our main goal in registration is to take two given images and create a third image (or a registered image, call it S) from the original images which can then be compared to the reference image. When we subtract S from R, we get a difference image $\hat{D}$, and hope that $\hat{D}$ will appear more gray than D which was created by comparing T with R. When the image is entirely gray that means there is no difference between S and R; areas of whiter/darker regions of D indicate mis-registration. By utilizing these difference images, we can verify that the registration process is working correctly by visually comparing D and $\hat{D}$.

4.1.1.1 Backward Euler Method

We performed registration with the diffusion regularizer, under both sliding and bending boundary conditions. Registration was performed multiple times to try and achieve the best results through altering α^2 ($\frac{1}{(10)^2}$, $\frac{1}{(30)^2}$, $\frac{1}{(50)^2}$, $\frac{1}{(75)^2}$, $\frac{1}{(100)^2}$) and the

maximum number of iterations (50, 100, 150, or 300). Registration was terminated before it reached the maximum number of iterations whenever the similarity difference or the maximum of the magnitudes of the displacement vectors fell below a threshold of 0.1 percent. All of the tests were run with the Backward Euler method.

Figure (4.2) illustrates the registration results. The first row displays the reference image R, the template T, and the difference image D, respectively. The second and third row display the reference image again, then illustrate the best and worst result and the difference images for the best and worst results for sliding and bending boundary conditions, respectively. The fourth and fifth row display the same images across the columns but with Dirichlet and Neumann boundary conditions. After reviewing the twenty different tests in which sliding boundary conditions were imposed, we found that visually the best results were achieved when α^2 was $\frac{1}{(10)^2}$ and the number of iterations was 50. The worst visual results were obtained from an α^2 of $\frac{1}{(75)^2}$ and a maximum number of iterations of 150. When bending boundary conditions were imposed, we found the best visual results were also obtained from the same α^2 and maximum iterations as with sliding boundary conditions. However, visually the worst result came from $\alpha^2 = \frac{1}{(30)^2}$ and 300 iterations. In most cases with bending boundary conditions, we obtained artifacts near the boundaries because although the bending boundary conditions made sense phyiscally it is impossible to interpolate values outside of the boundary. The artifacts occur because the registration process can move the pixels/voxels along the boundary toward the center of the image, but cannot fill in the area where the pixels were moved from since there is nothing beyond the boundary to pull into the image. To justify that sliding and bending boundary conditions are improving on the registration process, we tested homogeneous Dirichlet and homogeneous Neumann boundary conditions against our best result ($\alpha^2 = \frac{1}{(10)^2}$, 50 iterations) and our worst result ($\alpha^2 = \frac{1}{(30)^2}$, 300 iterations). You can see from the difference images in figure (4.2) that sliding and bending boundary conditions

make significant improvements when compared to Dirichlet or Neumann boundary conditions.

4.1.1.2 Modified Euler Method

We have also run the same twenty tests using the Modified Euler solution methods. Figure (4.3) displays the registration results with the Modified Euler method. The images in figure (4.3) follow the same setup as figure (4.2). For sliding boundary conditions, we found that the best results were achieved when α^2 was $\frac{1}{(10)^2}$ and the number of iterations was 50. The worst results were obtained from an α^2 of $\frac{1}{(100)^2}$ and a maximum number of iterations of 150. The best results for bending boundary conditions are the same α^2 and maximum iterations as with sliding boundary conditions. The worst result came from $\alpha^2 = \frac{1}{(100)^2}$ and 300 iterations.

4.1.1.3 Runge-Kutta Method

Before we could run the tests using the Runge-Kutta method we had to address a new parameter that was being introduced. The new parameter is named k and arose while calculating the left-hand side of the problem at each time step, however the value of k in (3.32) was not limited to a certain value. We tested various values for k and concluded $k = 55$ was the best value because it reduced the similarity measure while going through a smaller number of iterations than other values of k. After we established $k = 55$, ran the same twenty tests that were run for the Backward Euler and Modified Euler methods. Figure (4.4) shows the results with the Runge-Kutta method. The images in figure (4.4) follow the same setup as figures (4.2) and (4.3). When sliding boundary conditions were imposed, we found that the best visual results were achieved when α^2 was $\frac{1}{(10)^2}$ and the number of iterations was 50 while worst results were obtained from an α^2 of $\frac{1}{(30)^2}$ and a maximum number of iterations of 150. Visually the best results for bending boundary conditions have α^2 equal to $\frac{1}{(10)^2}$ and

50 iterations as well. The worst visual result came from $\alpha^2 = \frac{1}{(100)^2}$ and 300 iterations.

4.1.1.4 Results

Below are two tables that display the number of iterations performed and the value of the similarity measure at the final iteration for solution methods that correspond to each of the twenty tests run. Tables (4.1)–(4.2) provide the data when sliding and bending boundary conditions were employed, respectively. After the two tables is a plot that displays the decreasing values of the similarity measure at each time step when sliding boundary conditions were enforced. The parameters were α^2 equal to $\frac{1}{(10)^2}$ and 50 iterations; these were chosen because they gave us the best visual results.

If you look at the results in both tables, you will notice the largest similarity measure occurs when our best parameters are enforced. This is from the trade off between the similarity measure and the regularizer. Without a regularizer, we could have a similarity measure of 0. However the resulting deformation would be physically meaningless since assign all pixels (voxels) in the template with a value of c would be mapped to one pixel (voxel) in the reference image with the same value.

4.1.2 Three-dimensional Registration on Truncated Chest CT Volumes

Now that we have tested and confirmed that sliding and bending boundary conditions improve on 2-D registration, we will make the transition into 3-D registration and consider the case of registering 3-D CT chest volumes of a patient captured at different times. We performed registration using the diffusion regularizer with both sliding and bending boundary conditions. Like with 2-D registration, 3-D registration was terminated before it reached the maximum number of iterations if the similarity difference or the maximum magnitude of the displacement vectors were below a threshold of 0.1 percent.

		Backward Euler		Modified Euler		Runge-Kutta	
$\frac{1}{\alpha^2}$	Maximum Iterations	Similarity Measure	Iterations Performed	Similarity Measure	Iterations Performed	Similarity Measure	Iterations Performed
50	150	512.270	87	521.051	105	508.865	112
30	150	520.317	118	538.200	135	533.469	150
10	150	712.928	150	818.428	150	874.739	150
75	150	455.322	83	517.834	85	430.433	133
100	150	436.000	64	471.161	75	407.146	111
50	50	559.772	50	623.757	50	675.167	50
30	50	722.566	50	825.021	50	879.990	50
10	50	1240.820	50	1397.57	50	1476.31	50
75	50	502.217	50	536.599	50	555.705	50
100	50	445.246	50	509.049	50	537.611	50
50	300	512.270	87	521.051	105	508.865	112
30	300	520.317	118	538.200	135	533.372	152
10	300	542.408	270	575.145	105	609.717	300
75	300	455.322	83	517.834	85	430.433	133
100	300	436.000	64	471.161	75	407.146	111
50	100	512.270	87	$521,369$	100	518.370	100
30	100	527.132	100	576.458	100	611.203	100
10	100	882.292	100	1007.69	100	1072.310	100
75	100	455.322	83	517.834	85	465.853	100
100	100	436.000	64	471.161	75	409.129	100

Table 4.1: Results from Sliding Boundary Conditions

		Backward Euler		Modified Euler		Runge-Kutta	
$\frac{1}{\alpha^2}$	Maximum Iterations	Similarity Measure	Iterations Performed	Similarity Measure	Iterations Performed	Similarity Measure	Iterations Performed
50	150	380.804	144	401.949	150	419.623	150
30	150	464.875	150	505.164	150	528.587	150
10	150	899.817	150	1030.840	150	1098.700	150
75	150	441.047	150	392.899	146	394.696	146
100	150	461.446	63	360.679	150	364.391	150
50	50	560.282	50	630.090	50	669.937	50
30	50	716.618	50	815.978	50	868.724	50
10	50	1513.950	50	1684.140	50	1768.21	50
75	50	496.646	50	536.009	50	558.585	50
100	50	457.204	50	493.384	50	534.259	50
50	300	380.804	144	390.795	213	397.000	195
30	300	396.593	244	417.821	270	510.658	215
10	300	656.806	292	825.487	251	948.044	217
75	300	433.648	218	392.899	146	394.696	146
100	300	461.446	63	328.830	300	325.604	300
50	100	433.331	100	484.258	100	510.633	100
30	100	525.644	100	575.044	100	608.508	100
10	100	1105.200	100	1251.980	100	1362.180	100
75	100	449.834	100	405.546	100	425.068	100
100	100	461.446	63	384.258	100	402.083	100

Table 4.2: Results from Bending Boundary Conditions

After registration we looked at axial slices of the volumes for comparison. We decided to crop the volumes at two intermediate portions as well as took the top and bottom slices. However, when we registered the images with bending boundary conditions we noticed that we were getting large undesired artifacts on the top and bottom slices. These artifacts often distorted the images so much that we could not achieve a desirable result. Thus we recommend using only sliding boundary conditions for 3-D registration since the difference in the intermediate slices is minimal when choosing sliding or bending boundary conditions.

Figure (4.5) illustrates the registration results: in this figure, the corresponding α^2 values is $\frac{1}{(10)^2}$ and we set the maximum iterations at 300. Slices from the prior and current volumes of the patient are shown in the first two columns. The third column displays the difference between the prior and current volumes. The fourth column shows the current volume after registration under homogeneous Dirichlet boundary conditions, and the fifth column shows the difference between the prior and registered volumes with Dirichlet boundary conditions. The sixth and seventh columns show the registered current volume and difference between the prior and registered current volume, respectively, under sliding boundary conditions. The top row corresponds to the top axial slices of the CT volumes and the bottom row corresponds to the bottom axial slices. The intermediate rows correspond to intermediate slices in the volumes.

4.1.3 Summary of Results

As you can see from the difference images in both the 2-D and 3-D image registration examples, every boundary condition provides adequate registration. However, it is clear in 2-D registration that sliding and bending boundary conditions provide superior alignment when compared to Dirichlet or Neumann boundary conditions. In 3-D registration, the top and bottom axial difference images show significant improvement when sliding boundary conditions are implemented. As a result of our

experiments, we recommend using sliding and/or bending boundary conditions for image registration.

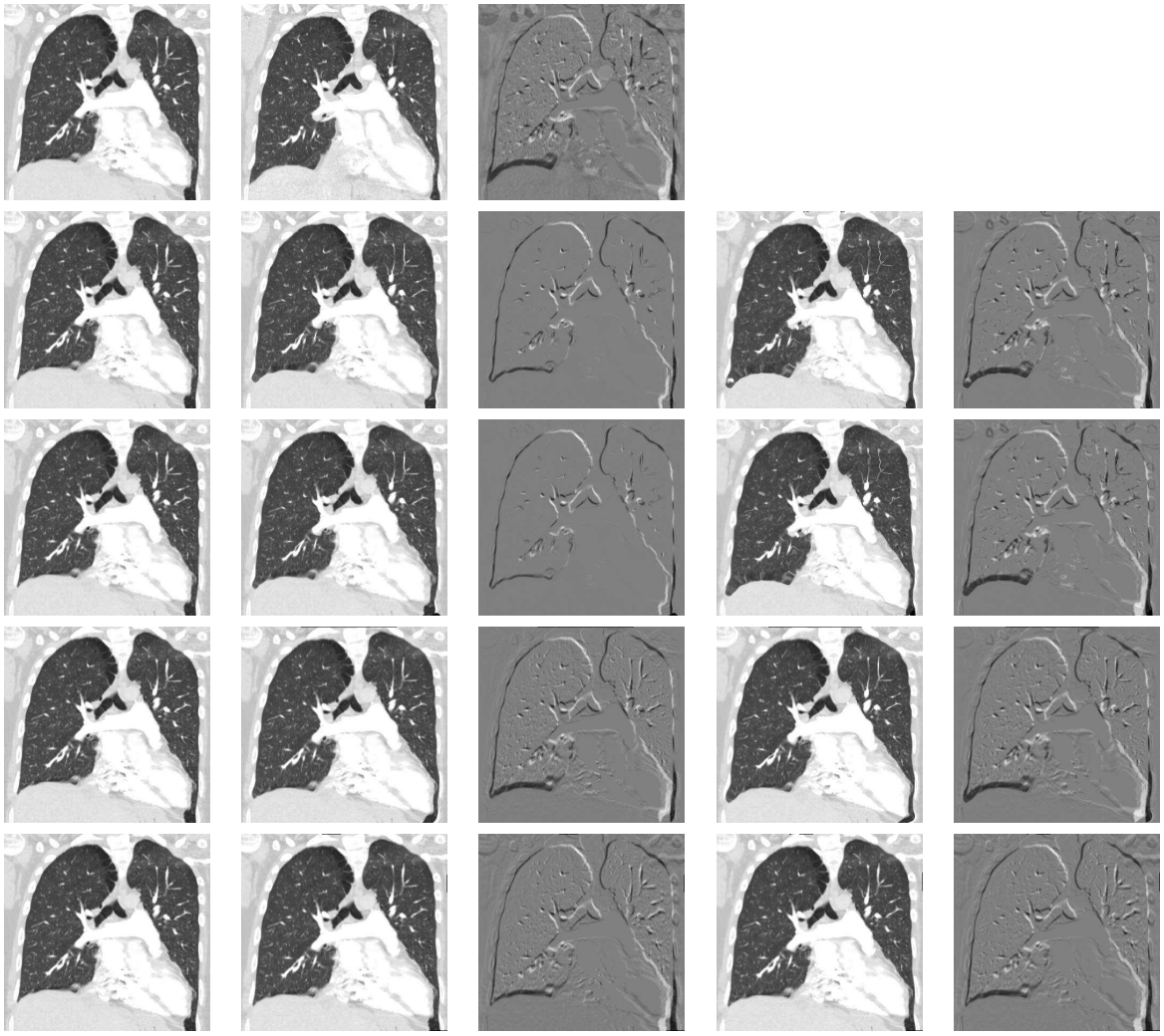


Figure 4.2: Rows: R, T, and D Images (top); Sliding Boundary Conditions (second); Bending B.C. (middle); Dirichlet B.C. (fourth); Neumann B.C. (bottom). Columns: Prior Slice (one); Current Slice/Best Results (two); Best Results minus Prior Image (three); Worst Results (four); Worst Results minus Prior Image (five).

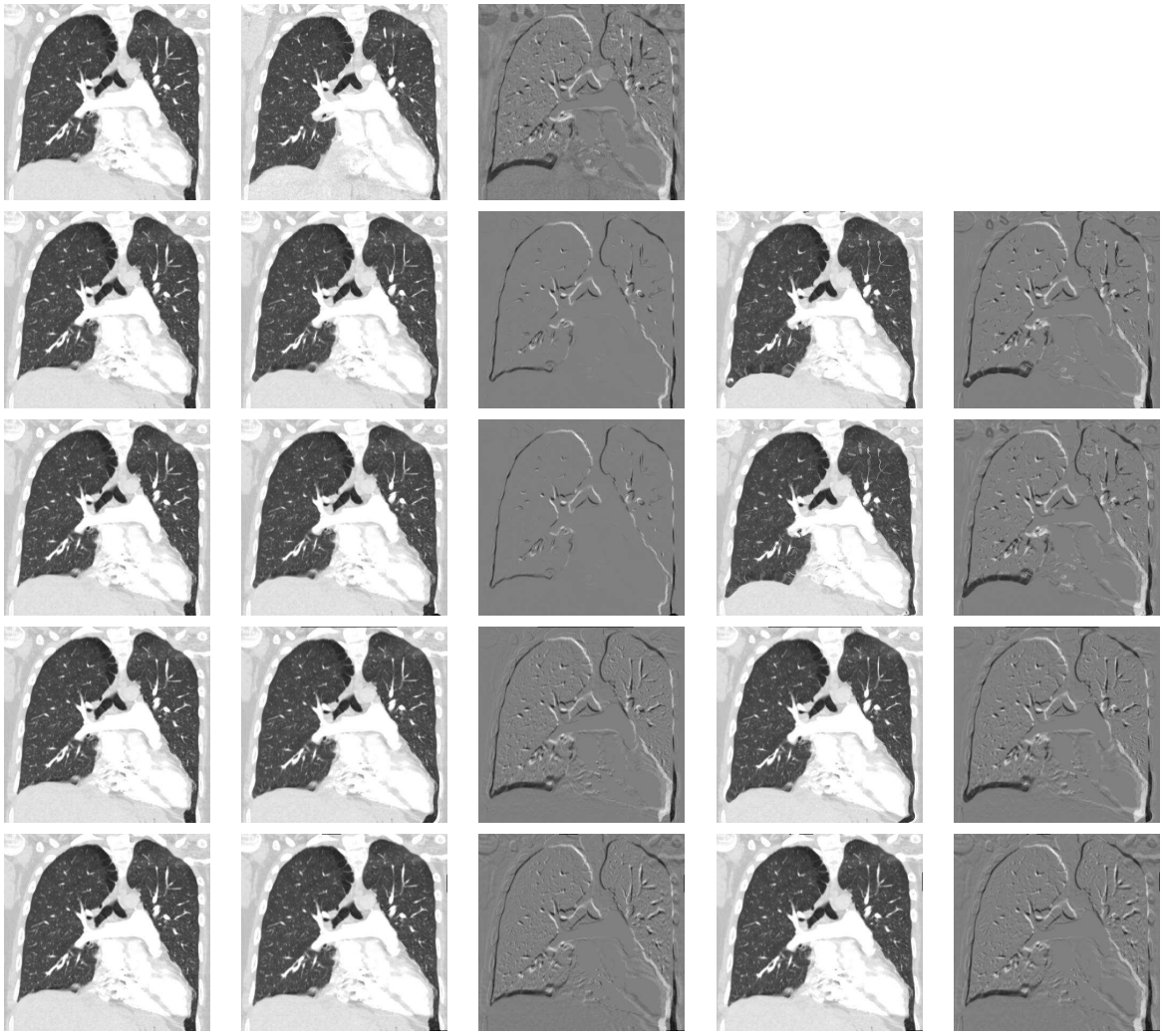


Figure 4.3: Rows: R, T, and D Images (top); Sliding Boundary Conditions (second); Bending B.C. (middle); Dirichlet B.C. (fourth); Neumann B.C. (bottom). Columns: Prior Slice (one); Current Slice/Best Results (two); Best Results minus Prior Image (three); Worst Results (four); Worst Results minus Prior Image (five).

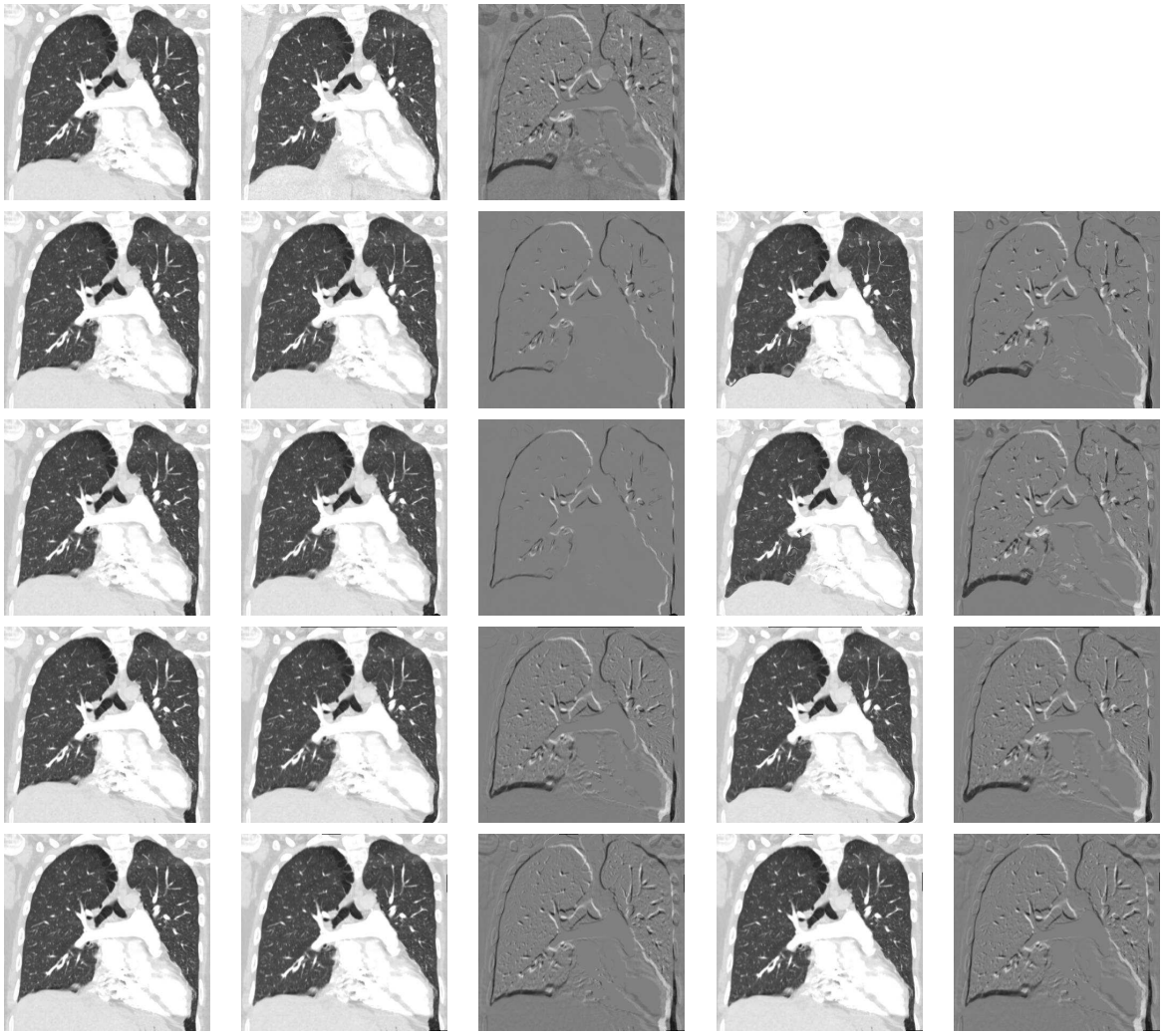
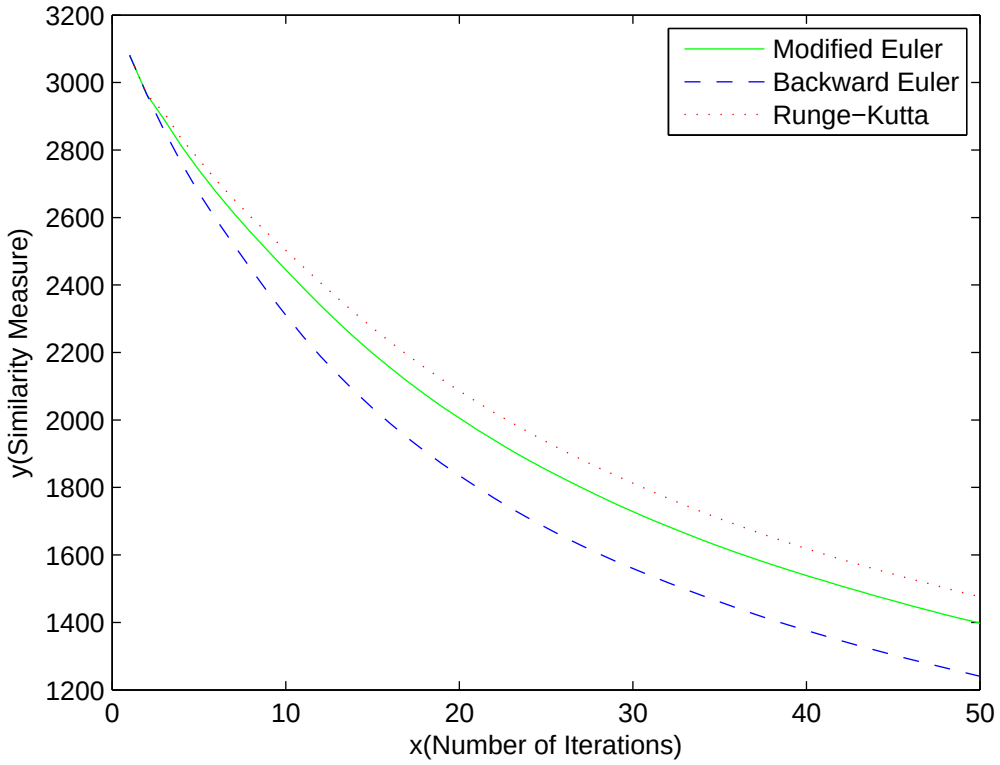


Figure 4.4: Rows: R, T, and D Images (top); Sliding Boundary Conditions (second); Bending B.C. (middle); Dirichlet B.C. (fourth); Neumann B.C. (bottom). Columns: Prior Slice (one); Current Slice/Best Results (two); Best Results minus Prior Image (three); Worst Results (four); Worst Results minus Prior Image (five).



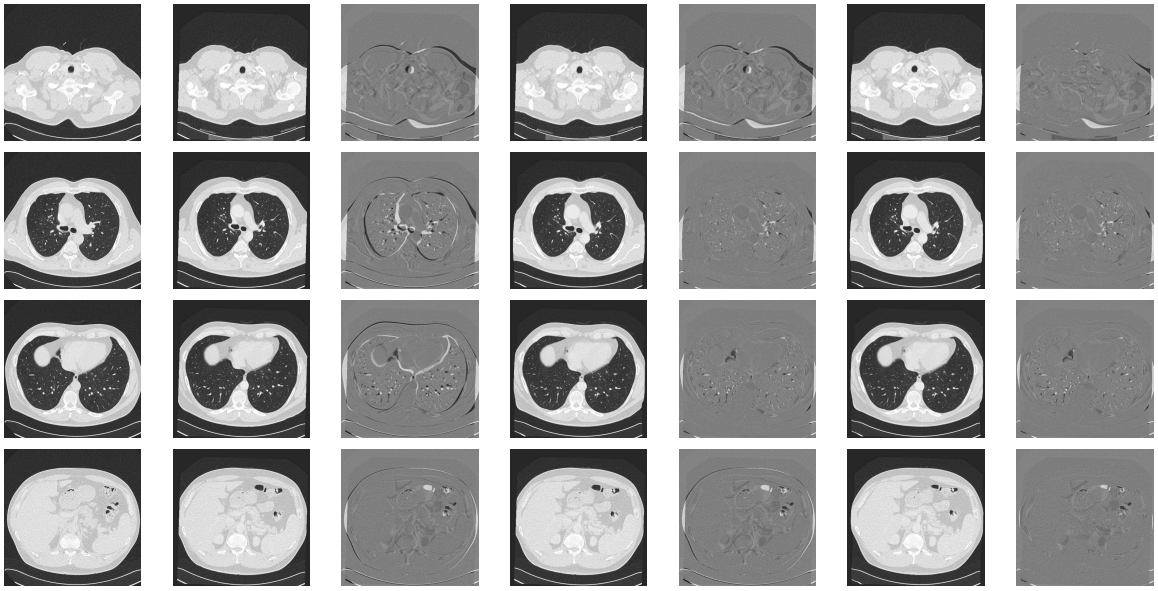


Figure 4.5: Rows: top axial slice (top); intermediate axial slices (middle); bottom axial slice (bottom). Columns: current volume (one); prior volume (two); prior minus current (three); Dirichlet-registered current volume (four); prior minus Dirichlet-registered current (five); sliding-registered current volume (six); prior minus sliding-registered current (seven).

Chapter 5

Conclusion

The goal of this thesis was to improve upon the registration techniques being used today. We conjectured that sliding and bending boundary conditions would improve the flexibility of current nonrigid registration algorithms. In chapter 1, we formulated an IBVP that can be used to describe image registration algorithms used in medical imaging and established that we would be using the diffusion regularizer. Then in chapter 2 we went through numerical solutions of a toy problem on scalar functions in one, two, or three dimensions. We showed that sliding/bending boundary conditions can be applied to vector-valued IBVP's, and how numerical solutions to these problems can be found using DST's and DCT's.

Finally in chapter 3, we used the methods established in chapter 2 to test sliding and bending boundary conditions on real-world image registration problems. We used a few different solution methods to test sliding and bending boundary conditions. Each test revealed that sliding and bending boundary conditions provided visually superior results to Dirichlet or Neumann boundary conditions (in two dimensions; only sliding was superior in three dimensions). It is easy to see from our results and the figures provided in chapter 3 that sliding or bending boundary conditions can provide better and more physically realistic registration results than either Dirichlet or Neumann boundary conditions.

From our experiments, it appears sliding and bending boundary conditions are

significantly better than the boundary conditions used today and thus it may be beneficial to apply them in other situations. For instance, sliding and bending boundary conditions can be applied to elastic and curvature registration [2] to see if they improve those techniques as well. In [5] the process of extending Dirichlet, Neumann, and periodic boundary conditions have already been extended to elastic, curvature, and even fluid registration. We have begun exploring this idea in [14], but need to do some further work on the bending boundary conditions in order to fully realize the potential of these conditions. It would be constructive to implement sliding and bending boundary conditions in multimodal registration settings that use other similarity measures as well. It would also be beneficial to find a way to eliminate the artifacts created while using bending boundary conditions in three dimensions. Even though 2-D registration with bending boundary conditions creates artifacts, we did not find these as problematic as in the 3-D case. It would be helpful to develop a way to minimize the size of the artifacts so that they are no longer a problem.

Appendix A

Computation of Eigenvalues

A.1 Computation of Modified Euler Eigenvalues (in 1-D)

From(3.30), we are interested in solving:

$$u_{i,t+1} - \tfrac{k}{2}\ddot{\Delta}u_{i,t+1} - \tfrac{k^2}{2}\ddot{\Delta}^2 u_{i,t+1} = u_{i,t} + \tfrac{k}{2}\ddot{\Delta}u_{i,t}. \tag{A.1}$$

Note that we have expressed u and $u_{i,t+1}$ where i is the discrete spatial index and t is the time step to make it as similar as possible to when we computed the eigenvalues for the Crank-Nicolson method in equation (refu). If we let $k = \lambda$ and then expand $\ddot{\Delta}$ and $\ddot{\Delta}^2$, the left-hand side of (A.1) becomes:

$$\begin{aligned} u_{i,t+1} &- \tfrac{\lambda}{2}\left[u_{i+1,t+1} - 2u_{i,t+1} + u_{i-1,t+1}\right] \\ &- \tfrac{\lambda^2}{2}\left[u_{i+2,t+1} - 2u_{i+1,t+1} + u_{i,t+1}\right] \\ &- \tfrac{\lambda^2}{2}\left[-2\left(u_{i+1,t+1} - 2u_{i,t+1} + u_{i-1,t+1}\right)\right] \\ &- \tfrac{\lambda^2}{2}\left[u_{i,t+1} - 2u_{i-1,t+1} + u_{i-2,t+1}\right] \end{aligned} \tag{A.2}$$

By combining like terms, (A.2) becomes:

$$\begin{aligned}(1+\lambda-3\lambda^2)u_{i,t+1} + \left(2\lambda^2-\tfrac{\lambda}{2}\right)\left(u_{i+1,t+1}+u_{i-1,t+1}\right)\\ -\tfrac{\lambda^2}{2}\left(u_{i+2,t+1}+u_{i-2,t+1}\right)\end{aligned} \tag{A.3}$$

Now using Matlab's definition of the DST ($\frac{1}{N}\sum_{k=0}^{N-1} x_j \sin\left(\frac{2\pi(j)(k)}{N}\right)$ with the sum/difference of sine angles formula, (A.3) becomes:

$$\begin{aligned}
&\tfrac{-\lambda^2}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell i}{N}\right)\cos\left(\tfrac{2\pi\ell}{N}\right)\right]\\
+&\tfrac{-\lambda^2}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{2\pi\ell}{N}\right)\cos\left(\tfrac{\pi\ell i}{N}\right)\right]\\
+&\left(2\lambda^2-\tfrac{\lambda}{2}\right)\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell i}{N}\right)\cos\left(\tfrac{\pi\ell}{N}\right)\right]\\
+&\left(2\lambda^2-\tfrac{\lambda}{2}\right)\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell}{N}\right)\cos\left(\tfrac{\pi\ell i}{N}\right)\right]\\
+&(1+\lambda-3\lambda^2)\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell i}{N}\right)\right]\\
+&\left(2\lambda^2-\tfrac{\lambda}{2}\right)\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell i}{N}\right)\cos\left(\tfrac{\pi\ell}{N}\right)\right]\\
-&\left(2\lambda^2-\tfrac{\lambda}{2}\right)\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell}{N}\right)\cos\left(\tfrac{\pi\ell i}{N}\right)\right]\\
+&\tfrac{-\lambda^2}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell i}{N}\right)\cos\left(\tfrac{2\pi\ell}{N}\right)\right]\\
-&\tfrac{-\lambda^2}{2}\left[\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{2\pi\ell}{N}\right)\cos\left(\tfrac{\pi\ell i}{N}\right)\right]
\end{aligned} \tag{A.4}$$

Finally, (A.4) simplifies to:

$$\tfrac{-1}{N}\sum_{\ell=1}^{N-1}\tilde{u}_{\ell,t+1}\sin\left(\tfrac{\pi\ell i}{N}\right)\left[\left(1+\lambda-3\lambda^2\right)+2\left(2\lambda^2-\tfrac{\lambda}{2}\right)\cos\left(\tfrac{\pi\ell}{N}\right)-\lambda^2\cos\left(\tfrac{2\pi\ell}{N}\right)\right]$$

Using the trig. identity $\cos\left(\frac{2\pi\ell}{N}\right) = 2\cos^2\left(\frac{\pi\ell}{N}\right) - 1$, we get the eigenvalues,

$$\mu_\ell = \left(1 + \lambda - 2\lambda^2\right) + \left(4\lambda^2 - \lambda\right)\cos\left(\tfrac{\pi\ell}{N}\right) - 2\lambda^2\cos^2\left(\tfrac{\pi\ell}{N}\right) \tag{A.5}$$

A.2 Computation of the Left-Hand Side of the Runge-Kutta Method and its Eigenvalues

We know,

$$\begin{aligned}
u_{t+1} &= u_t + \tfrac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) \\
K_1 &= k \cdot \ddot{\Delta} u_t \\
K_2 &= k \cdot \ddot{\Delta} G_1, \text{where} \quad G_1 = u_t + \tfrac{K_1}{2} \\
K_3 &= k \cdot \ddot{\Delta} G_2, \text{where} \quad G_2 = u_t + \tfrac{K_2}{2} \\
K_4 &= k \cdot \ddot{\Delta} G_3, \text{where} \quad G_3 = u_t + K_3
\end{aligned} \tag{A.6}$$

So, (3.33) becomes:

$$\begin{aligned}
u_{t+1} &= u_t + \tfrac{1}{6}\left[k \cdot \ddot{\Delta} u_t + 2k \cdot \ddot{\Delta} g_1 + 2k \cdot \ddot{\Delta} g_2 + k \cdot \ddot{\Delta} g_3\right] \\
&= u_t + \tfrac{1}{6}\left[k \cdot \ddot{\Delta} u_t + 2k \cdot \ddot{\Delta} g_1 + 2k \cdot \ddot{\Delta} g_2\right] \\
&\quad + \tfrac{1}{6}\left[k \cdot \ddot{\Delta}\left(u_{t+1} + K_3\right)\right] \\
&= u_t + \tfrac{1}{6}\left[k \cdot \ddot{\Delta} u_t + 2k \cdot \ddot{\Delta} g_1 + 2k \cdot \ddot{\Delta} g_2\right] \\
&\quad + \tfrac{1}{6} k \cdot \ddot{\Delta} u_{t+1} + \tfrac{1}{6} k \cdot \ddot{\Delta} K_3 \\
u_{t+1} - \tfrac{1}{6} k \cdot \ddot{\Delta} u_{t+1} &= u_t + \tfrac{1}{6}\left[k \cdot \ddot{\Delta} u_t + 2k \cdot \ddot{\Delta} g_1 + 2k \cdot \ddot{\Delta} g_2\right] \\
&\quad + \tfrac{1}{6}\left[k \cdot \ddot{\Delta} K_3\right]
\end{aligned} \tag{A.7}$$

Therefore the Left-Hand Side of the Runge-Kutta method is

$$u_{t+1} - \tfrac{1}{6}k \cdot \ddot{\Delta} u_{t+1}. \tag{A.8}$$

Since we know $\ddot{\Delta}$ provides eigenvalues of $\lambda - \cos(\frac{\pi\ell}{N})$, then the eigenvalues are

$$\mu_\ell = 1 - \tfrac{1}{6}k \cdot \left(\lambda - \lambda \cos\left(\tfrac{\pi\ell}{N}\right)\right) \tag{A.9}$$

Appendix B

MATLAB code

In this part we have provided one example of the MATLAB code we have developed to explore numerical solutions to the various versions of the toy problem. The example uses the Crank-Nicolson solution method on a one-dimensional scalar function with Dirichlet boundary conditions. The function provided is called cnfft.m and served as the building block for the rest of our code. The entire set of code developed for solving the various toy problems using a variety of techniques has been uploaded to the MATLAB Central File Exchange (http://www.mathworks.com/matlabcentral/fileexchange/30252-matlab-code-for-my-graduate-thesis). The other files included are examples using Crank-Nicolson, Modified Euler, Heun, and Runge-Kutta solution methods in not only one but two and three dimensions using all of the boundary conditions discussed in this thesis.

```
function w = cnfft(varargin)
% crankNicolson: uses Crank-Nicolson algorithm to approximate the solution
%   to the parabolic PDE:
%       u_{t}(x,t) - alpha^2 u_{xx}(x,t) = 0, 0<x<L, 0<t<T
%   subject to the boundary conditions
%       u(0,t) = u(L,t) = 0, 0<t<T
%   and the initial conditions
%       u(x,0) = f(x), 0<=x<=L
%
% arguments:
%   L (scalar) - upper bound of spatial (x) variable
%       (Default L = 1)
%   T (scalar) - upper bound of time (t) variable
%       (Default T = 1)
%   alpha (scalar) - square root of coefficient of u_{xx} term
%       (Default alpha = 1)
%   m (scalar) - number of discrete spatial intervals
%       (Default m = 100)
%   n (scalar) - number of discrete time intervals
%       (Default n = 100)
%
%   w (m+1 x n) - approximation to u(x,t) at discrete space/time positions
%

% author: Troy J. Winkstern
% email: tjw8191@rit.edu
% date: 22 Dec 2010

% parse input arguments
[L,T,alpha,m,n] = parseInputs(varargin{:});

% initialize h, k, lambda, and w
h = L./m;
k = T./n;
lambda = (alpha.^2).*k./(h.^2);
w = zeros(m+1,n);

% initialize first column of w
w(2:m,1) = sin(pi.*h.*(1:(m-1)).');

for i=1:(n-1)
   y = zeros(m+1,1);
   y(2) = (1-lambda)*w(2,i) + (lambda/2)*w(3,i);
   for j = 3:(m-1)
      y(j) = (lambda/2)*w(j-1,i) + (1-lambda)*w(j,i) + (lambda/2)*w(j+1,i);
```

```
   end
   y(m) = (lambda/2)*w(m-1,i) + (1-lambda)*w(m,i);

   % went out to length 2m
   X = sqrt(2./m).*imag(fft(y, 2*m));
   X = X(1:m+1);

   % angles divided by m
   a = (pi./m).*(0:m).';

   % changed lambda to negative
   c = -lambda*(cos(a)) + 1 + lambda;

   % went out to length 2m
   Z = sqrt(2./m).*imag(fft(X./c, 2*m));
   w(2:m,i+1) = Z(2:m);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% subfunction parseInputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [L,T,alpha,m,n] = parseInputs(varargin)

% check number of input arguments
nargs = length(varargin);
error(nargchk(0,5,nargs));

% get/set n
if nargs<5
    n = [];
else
    n = varargin{5};
end
if isempty(n)
    n = 100;
end

% check n has valid value
if (numel(n)>1) || (n<1) || ~isequal(round(n),n)
    error('n must be a positive integer.');
end

% get/set m
if nargs<4
```

```
    m = [];
else
    m = varargin{4};
end
if isempty(m)
    m = 100;
end

% check m has valid value
if (numel(m)>1) || (m<1) || ~isequal(round(m),m)
    error('m must be a positive integer.');
end

% get/set alpha
if nargs<3
    alpha = [];
else
    alpha = varargin{3};
end
if isempty(alpha)
    alpha = 1;
end

% check alpha has valid value
if numel(alpha)>1
    error('alpha must be a real scalar.');
end

% get/set T
if nargs<2
    T = [];
else
    T = varargin{2};
end
if isempty(T)
    T = 1;
end

% check T has valid value
if (numel(T)>1) || (T<=0)
    error('T must be a positive real scalar.');
end

% get/set L
if nargs<1
```

```
    L = [];
else
    L = varargin{1};
end
if isempty(L)
    L = 1;
end

% check L has valid value
if (numel(L)>1) || (L<=0)
    error('L must be a positive real scalar.');
end

% warning if L not integer
if ~isequal(round(L),L)
    warning(['L should be an integer so that the inital condition',...
        'takes on the value of 0 for t = 0 and t = T.']);
end
```

References

[1] R. Burden and J. Faires. Numerical Analysis. 8^{th} Edition. 2005.

[2] J. Modersitzki. Numerical Methods for Image Registration. Oxford University Press, December 2003.

[3] G. Hermosillo Valadez. Variational Methods for Multimodal Image Matching. D.Sc. Thesis, Universite de Nice - Sophia Antipolis. May 2002.

[4] G. E. Christensen, R. Rabbit, and M. Miller. Deformable Templates using Large Deformation Kinematics. IEEE Transactions on Image Processing. 5(10):1435-1447, October 1996.

[5] N. D. Cahill, J. A. Noble, and D. J. Hawkes, Fourier Methods for Nonparametric Image Registration, Proceedings of the CVPR Workshop on Image Registration and Fusion, June 2007.

[6] C. Broit. Optimal Registration of Deformed Images. Ph.D. Thesis, Computer and Information Sciences, University of Pennsylvania, 1981.

[7] M. Bro-Nielsen and C. Gramkow. Fast Fluid Registration of Medical Images.

Proceedings of the 4^{th} International Conference on Visualization in Biomedical Computing, LNCS 1131, 267-276, 1996.

[8] B. Fischer and J. Modersitzki. A Unified Approach to Fast Image Registration and a New Curvatuve Based Registration Technique. Linear Algebra and its Applications 380:107-124, 2004.

[9] M. I. Miller, S. C. Joshi, and G. E. Christensen, Rapid Convolution Based Large Deformation Image Matching via Landmark and Volume Imagery, US Patent No. 6,226,418, 2001.

[10] J. Hajnal, D. L. G. Hill, and D. Hawkes, Medical Image Registration, CRC Press, 1st Ed., 2001.

[11] M. Gockenbach, Partial Differential Equations, Society for Industrial and Applied Mathematics, 1st Ed., 2002.

[12] Satellite Image Overview, GDA Corporation. Available from http://www.gdacorp.com/satellite-image-processing/. Internet; accessed 3 February 2011.

[13] Computer Vision Online, Available from http://www.computervisiononline.com/. Internet; accessed 3 February 2011.

[14] T. Winkstern and N. D. Cahill, Rapid DFT-based Variational Image Registration with Sliding Boundary Conditions, Proceedings of the International Symposium on Biomedical Imaging, 2011.

[15] R. Richtmeyer and K. Morton, Difference Methods for Initial-Value Problems, Krieger Publishing Company, 2nd Ed., 1994.

[16] J. West, et. al, Comparision and Evaluation of Retrospective Intermodality Brain Image Registration Techniques, Journal of Computer Assisted Tomography, 21(4):554-568, July/August 1997.