MODELING FIBROBLAST GROWTH FACTOR 10 EXPRESSION IN EMBRYONIC MOUSE LUNGS

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Applied Mathematics

with a Concentration in

Biomathematics

by

Geneva RoseEmma Porter

Summer 2020

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Geneva RoseEmma Porter:

Genetic Patterning in Lung Development

Uduak Z. George, Chair
Department of Mathematics and Statistics

Joseph Mahaffy
Department of Mathematics and Statistics

Hakan Töreyin
Department of Electrical and Computer Engineering

Approval Date

DEDICATION

To Bill

All models are wrong, but some are useful.

– George Box

ABSTRACT OF THE THESIS

Genetic Patterning in Lung Development
by
Geneva RoseEmma Porter
Master of Science in Applied Mathematics with a Concentration in Biomathematics
San Diego State University, 2020

Mammalian lungs develop through a process of repetitive branching of epithelial structures. This process begins during the embryonic stage of development and is vital for the formation of the tree-like lung airways. Findings from wet-lab experiments performed using murine lungs have identified fibroblast growth factor 10 (FGF10) as a key regulator of lung formation. Spatiotemporal expression of FGF10 is highly stereotyped. FGF10 is expressed at the lung surface, distal to the branching epithelial structures, and plays a key role in regulating the branching of lung epithelial structures. In the absence of FGF10, murine lungs do not undergo branching. Despite enormous progress in understanding the mechanisms that control epithelial lung branching, the factors that determine the spatiotemporal expressions of FGF10 are not well understood. Here we propose and implement a new method to study FGF10 expression at the lung surface using a system of reaction-diffusion equations. The numerical approximation of the model equations is carried out using the surface finite element method. Using linear stability theory, we validate numerical simulation results on a unit sphere. Simulations of FGF10 expression are performed on a lung geometry segmented from three-dimensional confocal microscopy images. Simulated patterns for FGF10 expression are consistent with findings from wet-lab experiments. In particular, our model identifies the lung surface area, a previously unknown factor, as one of the main regulators of FGF10 expression in embryonic lungs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Thank you to my advisor and teacher, Dr. Uduak George, for providing feedback, guidance, and encouragement through this process. Thanks also to John Morgan, for being a source of kinship and humor for many long hours in the lab. Finally, I would like to thank my family, who gave me the gifts of creativity, academic integrity, and unconditional love. Without such support, I would surely have lost my way.

CHAPTER 1
Background and Motivation

Before exploring a mathematical model for Fibroblast Growth Factor 10 expression in embryonic mouse lungs, we examine several important ideas in physiology and theoretical biology. Specifically, we discuss morphogenesis in developmental biology, lung growth physiology in both mice and humans, and gene proteins that are believed to regulate such processes.

1.1 Introduction

In 1952, Alan Turing published his highly influential work *The Chemical Basis of Morphogenesis* [36], which changed our understanding of theoretical biology. Turing discussed how patterns can arise from chemical disturbances in biologically homogeneous systems. He states that phenomena such as skin pigmentation patterns can be explained by chemical reaction-diffusion systems occurring during the embryonic stage of development.

Turing suggests that deviations from homogeneity during embryonic development are needed to spark morphogenesis and form organs, limbs, and skin patterns. He also noted that these disturbances can be described as unstable equilibria; a small deviation from the homogeneous state will interrupt the system from developing uniformly. He noted that such equilibria are not observed to exist *per se* in nature, but rather occur when the disturbance in a system causes a stable equilibrium to become unstable. This phenomenon is known as a *Turing Instability*, and is observed in mathematical systems with the addition of a diffusion term.

Several notable works have since adapted and expanded Turing's model [14, 19, 17]. This thesis will use the proposed by Schnakenberg in his 1979 work *Simple Chemical Reaction Systems With Limit Cycle Behavior* [32] to study molecular patterning during embryonic lung development. The Schnakenberg model, also known as the activator-depletion model, is characterized by the kinetics of two reacting morphogens exhibiting auto-catalytic behavior.

We will apply Turing's theory and a variation of Schnakenberg's model to branch patterning in the developing lung. Lung branching morphogenesis is a complex process that begins at the embryonic stage of development. In murine lungs, three different branching types have been identified: domain branching, planar bifurcation and orthogonal bifurcation [24]. How the lung regulates branching morphogenesis is not fully understood. In particular, how the location and orientation of different branching types are specified remains unknown [13]. However, empirical evidence suggests that two gene proteins, Fibroblast Growth Factor-10 (FGF10) and Sonic Hedgehog (SHH), are crucial to lung development [31]. FGF10 is associated with stimulating tissue growth, and SHH is associated with inhibiting tissue growth. Experimental research suggests that FGF10 is expressed near the lung surface, then diffuses through the fluid-filled region called the mesenchyme. It then binds to the Fibroblast Growth Factor receptor 2b (FGFR2b), which in turn allows it to bind to SHH. This binding happens most primarily near the tips of airway branches where we see bifurcation during lung development. The branch tips, or epithelial buds, are the site of high concentrations of SHH [28, 25, 39].

FGF10 and SHH are only two of the many morphogens that influence lung morphogenesis. We will only focus on these two gene proteins, and specifically how they might be concentrated near the surface of the lung. In particular, we will frame our analysis around the presence of FGF10, since it is more abundant near the lung surface. We are interested in understanding how FGF10 expression patterns might form at the lung surface, before it diffuses inward to drive branching morphogenesis. Mathematical modeling of the molecular interactions at the lung surface may give insight into how FGF10-SHH interactions regulate lung branching morphogenesis. This may identify potential avenues to explore for treating hypoplastic lungs in a human fetus.

1.2 Lung Branching Morphogenesis

This section addresses the anatomy of a mammalian lung, specifically during the pseudoglandular stage, in which primary branching structures are formed. This stage of development involves lung buds branching into the surrounding mesenchyme, an unspecialized tissue that a developing embryo utilizes to form many different organ structures. Lung buds branch into the mesenchyme because of the accelerated growth of cells along the epithelial buds or stalks. Research has shown that this process includes significant communication between mesenchyme tissue cells and epithelial cells [23]. Figure 1.1 shows a general overview of a mammalian lung branch structure during the embryonic, pseudoglandular, and canalicular stages of development. Notice that for the pseudoglandular stage, the branching structure is not fully formed; only the first few generations of branching have occurred. We are interested in the branching generations after the bronchial split, but before the formation of alveoli.

During development, the mesenchyme plays an important role in both cellular growth during branching morphogenesis and in the orientation and positioning of individual branches. There are three types of branching modes that have been identified during this stage of development: domain branching, planar bifurcation, and orthogonal bifurcation. Figure 1.2 illustrates the differences in these branching types. All three are needed to form the complex structure of the lung, however the “genetic clock” that determines the manner and execution of branching has yet to be fully understood [24].

There are several more interesting and complex mechanisms needed for a healthy lung to form, however, we will only focus on where our genetic cycling reaction-diffusion equation may be relevant. The theory we are investigating proposes that the process driving branching patterns occurs on the surface of the fluid-filled sac that contains the mesenchyme. It is already well documented that SHH is produced in the cells of the epithelial buds. We propose that as SHH diffuses through the mesenchyme, it forms distribution patterns with the FGF10 near the surface of the lung. It has been posited that FGF10 then diffuses through the mesenchyme, as illustrated in Figure 1.3 [6, 27, 37].

The distribution pattern formed by this diffusion tells us where FGF10 is theoretically concentrated. Regions with a higher concentration of FGF10 will allow more rapid cellular growth, and the pattern formation may inform which type of branching occurs (domain, planar, or orthogonal). Where FGF10 is less dense, SHH will dominate, hindering the growth of cells. The proposed consequences of this theoretical process can be seen in the imaging slides shown in Figure 1.4. These images are taken from slices of murine lungs, and show how an epithelial bud can morph during the branching process.

It is important to note that while we are discussing the lung development of humans, the mathematical analysis will use a murine lung in the pseudoglandular stage as the domain. There are many developmental differences, but since we focus primarily on theory rather than medical applications, such differences will not be scrutinized. Figure 1.5 shows the major sections of a typical murine lung, which represents a structure consistent with our model. We will reference the geometry of this structure when discussing our results in Chapter 6. Ultimately, we hope to gain insight into mammalian lung development in general.

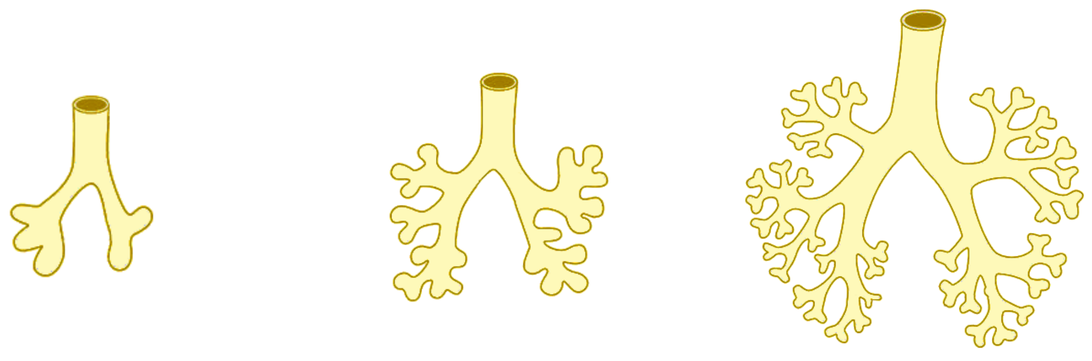


Figure 1.1. Human lung development stages during weeks 4 to 24. (a) Embryonic stage: major airway are formed (weeks 4-8). (b) Pseudoglandular stage: bronchial tree is formed (weeks 6-18). (c) Canalicular stage: epithelial differentiation phase (weeks 16-24). Figure adapted with permission from the British Lung Foundation, [*How children's lungs grow*, British Lung Foundation, https://www.blf.org.uk, Accessed 2020-02-13 (2019)].

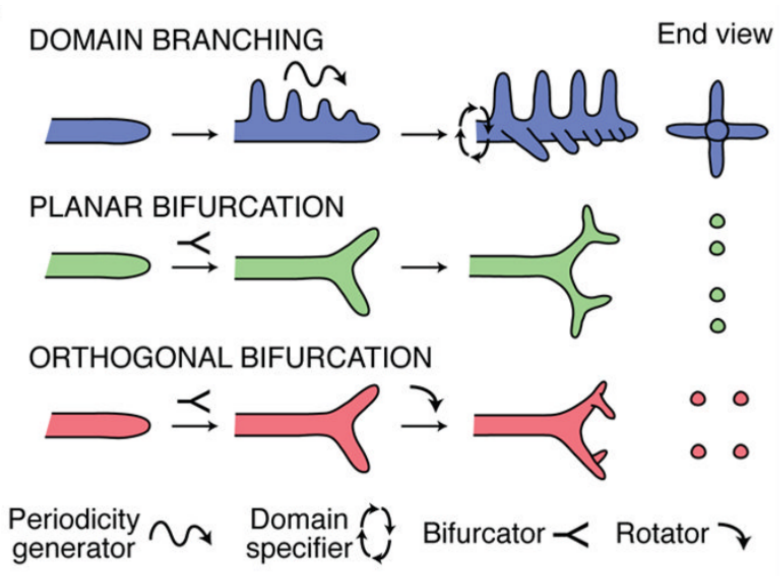


Figure 1.2. An illustration of three branching types that occur during lung development. Domain branching occurs when cellular growth is rapid, quickly forming the foundational structure of the lung airways. Planar bifurcation is on a smaller scale, and occurs at the very tip of the epithelial bud. We focus our examination here. Orthogonal bifurcation determines when the lung structure rotates the branching orientation, forming 3D structures. Figure adapted with permission from Nature, [R. J. Metzger, O. D. Klien, G. R. Martin, and M. A. Krasnow, *The branching programme of mouse lung development*, Nature, 453 (2008), pp 745–750].

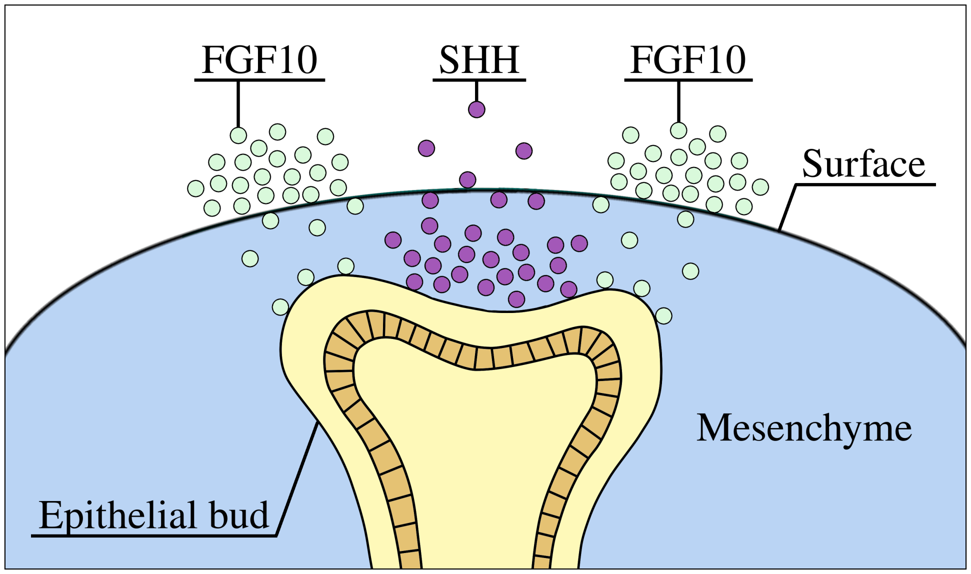


Figure 1.3. The proposed mechanism by which pattern formation on the surface of the lung may influence branching via distribution of FHF10 at key growth points.

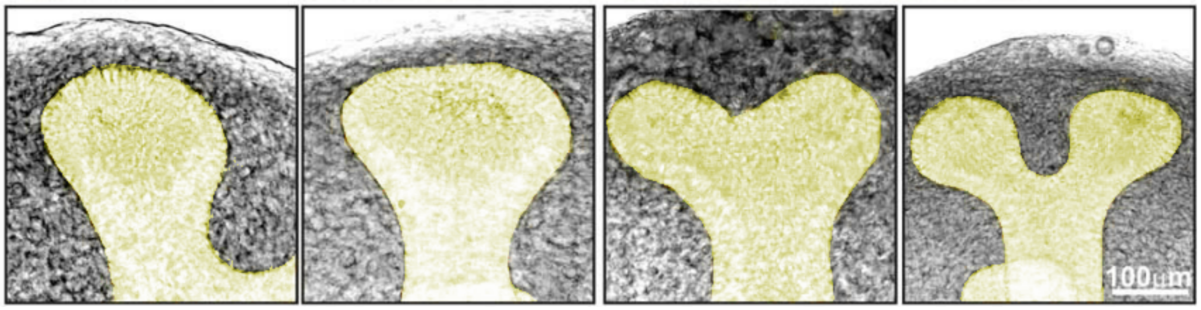


Figure 1.4. Scans of a murine lung epithelial bud during the pseudoglandular stage. Notice the flattening and migration of cells to form new branches at the top edges of the bud (left to right). Figure adapted with permission from Developmental Dynamics, [C. Schnatwinkel and L. Niswander, *Multiparametric image analysis of lung-branching morphogenesis*, Developmental Dynamics, 242 (2013), pp. 622–637].

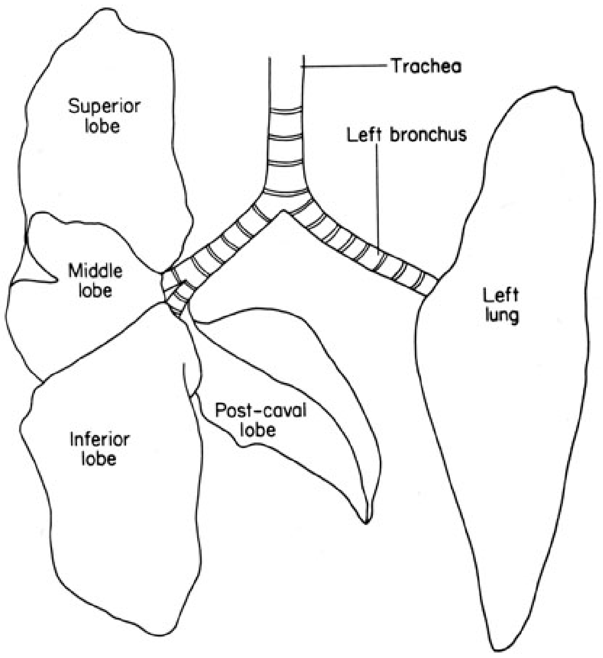


Figure 1.5. Diagram of the major parts of the murine lung. Figure adapted with permission from Academic Press, [M. J. Cook, *The Anatomy of the Laboratory Mouse*, Academic Press, 1965].

1.3 Genetic Cycling

Before exploring the intricacies of genetic modeling, it is necessary to establish a general foundation of the biological mechanisms behind genetic expression. Genetic information is determined by genotype and hard-coded into our DNA. During fetal development, DNA provides the building schematics needed to create each component of the body. It is though these schematics that the correct genes are identified and expressed via protein synthesis. The expression of genes occurs in two general stages: transcription and translation. During transcription, the enzyme RNA polymerase attaches to a group of genes within a DNA strand and creates a template for RNA replication. This template is used to synthesize proteins via translation and form a gene product [15].

For this investigation, we are particularly interested in genetic regulation at the transcription stage. During the formation of branching structures in the lung, expression and repression signaling is largely mediated by peptide growth factors, which can stimulate or inhibit mitosis and regulate cellular differentiation [15]. As described in the previous section, we will be focusing on lung growth during the pseudoglandular stage of fetal lung development, when epithelial cells differentiate to form a tubular structure and elongate into the surrounding mesenchyme tissue. Branching morphogenesis relies on signaling pathways between receptors on the epithelial buds and peptide growth factors in the mesenchyme [28].

The foundation of our analysis relies on the theory of reaction-diffusion equations describing feedback loop gene interactions. It is theorized that FGF10 encourages SHH formation, and in turn SHH inhibits FGF10 [16]. As more SHH is created, more FGF10 is inhibited. Without high FGF10 levels, SHH depletes and therefore becomes less effective in inhibiting FGF10, which can then continue to propagate, starting the cycle over again. Figure 1.6 shows a simple visualization for this system.

Such feedback loops have been successfully described by reaction-diffusion equations, such as with murine hair growth and crocodile teeth patterns [26]. It has been challenging for the scientific community to find real-world biological applications of reaction-diffusion systems. Nonetheless, they serve as useful models to gain insight into possible mechanisms at work in the incredibly complex system of biological development.

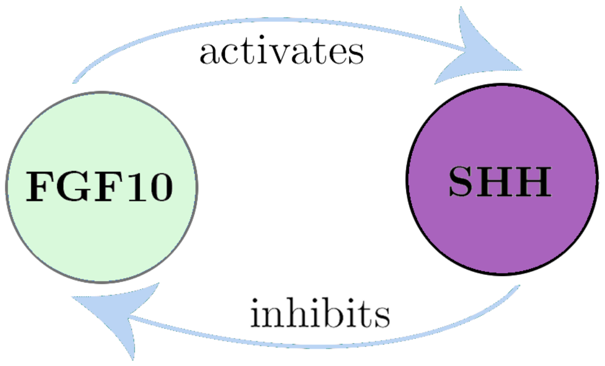


Figure 1.6. Simple diagram of the auto-catalytic interaction between FGF10 and SHH.

1.4 Research Goals

This thesis offers one of many possible models of FGF10 and SHH signaling pathways in epithelial bud branching. Its goal is to propose a new theoretical pattern formation map of FGF10 at the lung surface-mesenchyme border by modeling the interaction between FGF10 and SHH. Hopefully, such a model will shed light on how cell proliferation is regulated and which physical mechanisms contribute to lung branching in fetal development. Experimental verification of the model will be needed before deciding if the proposed mechanism is valid.

Before solving the model equations on the lung surface, some precedence must be set that ensures the validity of the solution. First, we will conduct a thorough stability analysis of the model before offering an analytic solution on a spherical surface. It is necessary to perform a stability analysis when there is no diffusion present in the system, then include the diffusion terms along with a small perturbation from the fixed points. From here we can determine which parameters give rise to Turing instabilities in this context.

A visualization of the analytic solution will then be compared to one using a numeric solution, in order to show consistency and display the validity of using the finite element method code. Further error analysis tests for consistency and convergence of the system will provide more evidence for the soundness of the model. We will then examine patterns that emerge on a growing domain. This involves comparing patterns on spheres of varying sizes, then manipulating parameters to mimic domain growth.

Finally, the methods used for these studies will be applied to 3D lung geometries representing the pseudoglandular stage of murine fetal lung development. The results from these simulations will inform the discussion on applicability, which appears in Chapter 6. We will end with a discussion of possible applications in the biomedical field.

CHAPTER 2
Modeling Fibroblast Growth Factor 10 Expression

Mathematical models of pattern formation have been exhaustively studied [26]. Here we give a basic overview of the process of Turing analysis on a 3D domain. We will describe the reaction-diffusion model for FGF10 expression in embryonic lungs. We will nondimensionalize the model and perform a linear stability analysis. Before solving the model on a lung geometry, we will validate the methods by solving both analytically and numerically on a sphere. Here we will assume a closed domain representing the surface of the unit sphere, and the methods used will be extended to the geometry of a lung in later chapters. This chapter details the analytical basis for numerical approximations to follow.

2.1 Reaction-Diffusion Models

The original basis for Turing's model is the interaction of two substances that undergo chemical reactions and spatial diffusion. This classic reaction-diffusion system has this general form:

$$\begin{aligned} \overbrace{\frac{\partial U}{\partial t}}^{\text{change in time}} &- \overbrace{D_U \Delta U}^{\text{diffusion}} &= \overbrace{f(U,V)}^{\text{reaction}} \\ \underbrace{\frac{\partial V}{\partial t}}_{\text{change in time}} &- \underbrace{D_V \Delta V}_{\text{diffusion}} &= \underbrace{g(U,V)}_{\text{reaction}} \end{aligned} \tag{2.1}$$

Both U and V are functions of position and time. The reaction functions $f(U,V)$ and $g(U,V)$ describe the reaction dynamics, including the rates of production and degradation of U and V. The diffusion terms represent the diffusion of U and V, with diffusion rates D_U and D_V.

Let us examine Schnakenberg's approach. He considered the kinetics of a tri-molecular reaction, given by:

$$X \underset{k_2}{\overset{k_1}{\rightleftharpoons}} \mathrm{F} \qquad 2\mathrm{F} + \mathrm{S} \xrightarrow{k_3} 3\,\mathrm{F} \qquad Y \xrightarrow{k_4} \mathrm{S}$$

Here, F represents the activator FGF10, and S represents the inhibitor SHH. X and Y are their respective precursor substrates. Using the law of mass action, this series of reactions is represented by Equation (2.2) below. Let the function F designate the behavior of FGF10, while S applies to SHH. This model is then given by:

$$\begin{aligned} \frac{\partial F}{\partial t} - D_F \Delta F &= k_1 - k_2 F + k_3 F^2 S \\ \frac{\partial S}{\partial t} - D_S \Delta S &= k_4 - k_3 F^2 S \end{aligned} \tag{2.2}$$

All parameters are positive and real. Substrate precursors of FGF10 and SHH are given by k_1 and k_4, respectively. The activator FGF10 is modeled as an auto-catalytic function, represented by the terms $k_3 F^2 S$ (positive auto-catalysis with SHH) and $k_2 F$ (negative auto-catalysis with the precursor substrate). The $k_3 F^2 S$ term subtracted from the inhibitor equation for SHH, since the SHH decreases in concentration as FGF10 is produced during the k_3 reaction. The two morphogens form a feedback loop that ensures a cyclic production and degradation of each substance when in a steady state.

The diffusion rates D_F and D_S account for delays in the auto-catalysis process. Normally, diffusion terms largely depend on the size of the molecular components that are interacting with each other, and have a square root relational dependence on such elements [36]. However, FGF10 and SHH do not directly bind to each other, but to intermediary receptor genes that slow down the diffusion process. Later we will assign values to the diffusion factors so that FGF10 diffuses about 10 times more slowly than SHH, which accounts for these delays.

2.2 Nondimensionalization

Here we nondimensionalize the model equations. Referencing equation (2.2), we can use the following substitutions:

$$F = u u_a \qquad S = v v_a \qquad \vec{x} = \hat{x} x_a \qquad t = \tau t_a \qquad \Delta \to \Delta_\Gamma$$

This results in the following equation, where the derivative is taken with respect to τ:

$$\begin{aligned} \frac{\partial u}{\partial \tau} - \frac{D_F t_a}{x_a^2} \Delta u &= \frac{k_1 t_a}{u_a} - k_2 t_a u + k_3 u_a v_a t_a u^2 v \\ \frac{\partial v}{\partial \tau} - \frac{D_S t_a}{x_a^2} \Delta v &= \frac{k_4 t_a}{v_a} - k_3 u_a^2 t_a u^2 v \end{aligned} \tag{2.3}$$

We can then make substitutions for each term, and define some new ones:

$$u_a = \sqrt{\frac{k_2}{k_3}} \qquad v_a = \sqrt{\frac{k_2}{k_3}} \qquad x_a^2 = \frac{D_F}{k_2} \qquad t_a = \frac{1}{k_2}$$

$$\text{and} \quad \alpha = \frac{k_1}{k_2}\sqrt{\frac{k_3}{k_2}} \qquad \beta = \frac{k_4}{k_2}\sqrt{\frac{k_3}{k_2}} \qquad \delta = \frac{D_S}{D_F} \qquad \gamma = \frac{1}{k_2}$$

This gives us the simplified system:

$$\begin{aligned}
\frac{\partial u}{\partial \tau} - \Delta u = f(u,v) \qquad &\text{with} \qquad f(u,v) = \gamma\left(\alpha - u + u^2 v\right) \\
\frac{\partial v}{\partial \tau} - \delta \Delta v = g(u,v) \qquad &\text{with} \qquad g(u,v) = \gamma\left(\beta - u^2 v\right)
\end{aligned} \tag{2.4}$$

We now have scaled equations, with γ acting as a scaling measure for the relative strength of the reactions [26]. The rates of production and degradation of the morphogens rely on the precursor substrate values of α and β.

2.3 Surface Reaction-Diffusion Model

In this section, we will model the spatial expression of FGF10 at the surface of a unit sphere. We will accomplish this by using a surface reaction-diffusion model. The surface reaction-diffusion model is obtained by replacing the Laplace operator in the reaction-diffusion model (2.4) with the Laplace-Beltrami operator. Analysis and numerical simulations of partial differential equations have been thoroughly studied on different geometric models [9, 38, 5]. The following definitions are adapted from these studies:

Let Ω be an open subset in $\mathbb{R}^3$ and Γ be a 2D hypersurface contained in Ω. Let $w : \Gamma \to \mathbb{R}$ be differentiable at $x \in \Gamma$. We define the tangential gradient of w at $x \in \Gamma$ by:

$$\nabla_\Gamma w(x) = \nabla \bar{w}(x) - \nabla \bar{w}(x) \cdot \hat{n}(x) \hat{n}(x), \tag{2.5}$$

where $\bar{w}$ is a smooth extension of $w : \Gamma \to \mathbb{R}$ to an $(n+1)-$dimensional neighborhood of the surface Γ, so that $\bar{w}|_\Gamma = w$, and $\hat{n}(x)$ is a unit normal at x. The Laplace-Beltrami operator applied to a twice differentiable function $w \in (\Gamma)$ is:

$$\Delta_\Gamma w = \nabla_\Gamma \cdot \nabla_\Gamma w \tag{2.6}$$

Substituting the Laplace-Beltrami operator into the nondimensionalized reaction-diffusion model (2.4) gives us:

$$\frac{\partial u}{\partial \tau} - \Delta_\Gamma u = \gamma f(u,v) \qquad \text{and} \qquad \frac{\partial v}{\partial \tau} - \delta \Delta_\Gamma v = \gamma g(u,v) \tag{2.7}$$

which we will use going forward.

2.4 Steady-State Analysis

Here we carry out a steady state analysis of the model, Equation (2.4). We observe from Equation (2.4) that the model relies on parameters α and β to determine stability regions. We can find these regions by first removing the diffusion terms, then solving for the fixed points of the system, which are the solutions to the equations:

$$\alpha - u + u^2 v = 0 \qquad \text{and} \qquad \beta - u^2 v = 0.$$

This system has a fixed point at $(u^*, v^*) = (\alpha + \beta, \frac{\beta}{(\alpha+\beta)^2})$. To determine the region of stability about these fixed points, we first perturb the system by some small $|\varepsilon| << 1$.

$$u = u^* + \varepsilon\ u \qquad \longrightarrow \qquad \frac{\partial u}{\partial \tau} = \varepsilon\ u_t = \gamma\ f(u^* + \varepsilon\ u)$$
$$v = v^* + \varepsilon\ v \qquad \longrightarrow \qquad \frac{\partial v}{\partial \tau} = \varepsilon\ v_t = \gamma\ g(v^* + \varepsilon\ v)$$

Next, we perform a Taylor expansion about the fixed point. The notation $\dot{u}$ and $\dot{v}$ will be used to denote the partial derivatives for u and v with respect to τ, while f_u, f_v, g_u, and g_v denote the partial derivatives for f and g with respect to u and v:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \gamma \begin{pmatrix} f_u(u^*, v^*) & f_v(u^*, v^*) \\ g_u(u^*, v^*) & g_v(u^*, v^*) \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} + \mathcal{O}(\varepsilon^2) \tag{2.8}$$

Or more simply, without higher order terms:

$$\dot{W} = \gamma J W \tag{2.9}$$

where the $(\dot{\ })$ operator indicates the time derivative. Before plugging in the partial derivatives and parameters into J, there are some useful generalizations to highlight. Solving the linear equation $\det(J - \lambda I)$ will reveal the necessary constraints for stability:

$$\det \begin{pmatrix} f_u - \lambda & f_v \\ g_u & g_v - \lambda \end{pmatrix} = 0 \qquad \longrightarrow \qquad \lambda^2 - \lambda \operatorname{tr}(J) + \det(J) = 0$$

It is assumed that the partial derivatives of f and g are evaluated at the fixed point. The the solution is:

$$\lambda = \frac{\operatorname{tr}(J) \pm \sqrt{\operatorname{tr}(J)^2 - 4\det(J)}}{2} \tag{2.10}$$

Since stability is desired, we have constraints on the values for $\operatorname{tr}(J)$ and $\det(J)$. This requires $\operatorname{re}(\lambda) < 0$, which implies $\operatorname{tr}(J) < 0$ and $\det(J) > 0$. These constraints are satisfied when:

$$f_u + g_v < 0 \qquad \text{and} \qquad f_u g_v - f_v g_u > 0 \tag{2.11}$$

Now it is useful to fill in the partial derivative and parameter values. The Jacobian is therefore:

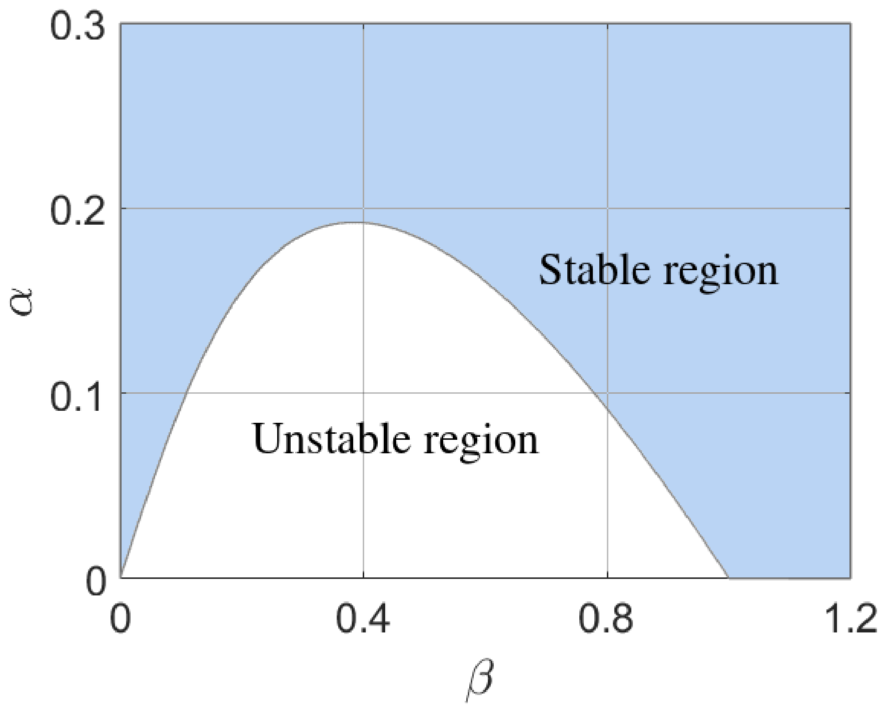
$$J = \begin{pmatrix} -1 + 2uv & u^2 \\ -2uv & -u^2 \end{pmatrix}\Bigg|_{(u^*,\ v^*)} = \begin{pmatrix} -1 + \dfrac{2\beta}{\alpha+\beta} & (\alpha+\beta)^2 \\ -\dfrac{2\beta}{\alpha+\beta} & -(\alpha+\beta)^2 \end{pmatrix} \tag{2.12}$$

This defines the constraints as:

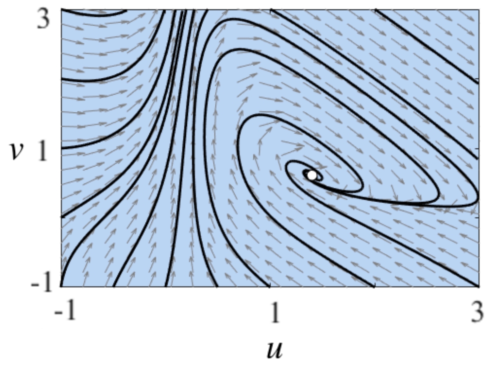
$$\beta - \alpha < (\alpha+\beta)^3 \qquad \text{and} \qquad (\alpha+\beta)^2 > 0 \tag{2.13}$$

The first constraint is illustrated in Figure 2.1(a), which highlights the stable region of the model system when no diffusion is present. Note that its border is defined by a Hopf bifurcation at $\text{tr}(J) = 0$. The second constraint is already fulfilled for all real values of α and β.

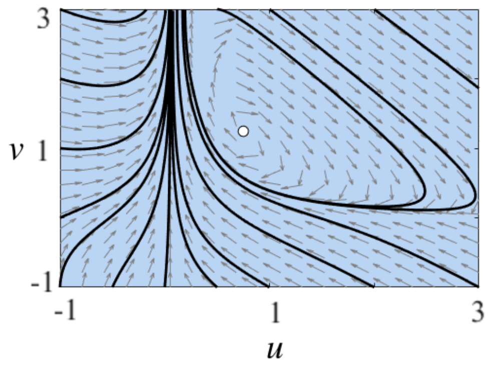
The phase planes in Figure 2.1(b) and 2.1(c) verify that the fixed point is stable for the parameter values $(\alpha, \beta) = (0.1, 0.9)$ in the desired region, and unstable for the pair (0.2, 1.2) outside that region. Now we have a starting point for the Turing Instability region.



(a)



(b)



(c)

Figure 2.1. Stability dynamics for the model system without diffusion. (a) The stability regions for parameters α and β. (b) Phase plane with $\alpha = 0.2$ and $\beta = 1.2$. Note that the parameters are in the stable region, thus the phase lines converge to the equilibrium. (c) Phase plane with $\alpha = 0.1$ and $\beta = 0.9$. Note that the parameters are outside the stable region, thus the phase lines diverge from the equilibrium.

2.5 Turing Instability Regions

The system is stable for $\beta - \alpha < (\alpha+\beta)^3$, so it is necessary find the region fitting this constraint where the activator-depletion system *with* its diffusion terms is unstable. To examine the system with the diffusion terms, Equation (2.9) is revisited. Without diffusion, there was $\dot{W} = \gamma\ JW$. With diffusion, there is:

$$\dot{W} - D\Delta_\Gamma W = \gamma\ JW \qquad \text{with} \qquad D = \begin{pmatrix} 1 & 0 \\ 0 & \delta \end{pmatrix} \tag{2.14}$$

To turn this into a linear system, we can define the following eigenvalue problems to use as substitutes into Equation (2.14):

$$\dot{W} = \lambda W \qquad \text{and} \qquad \Delta_\Gamma W = -k^2 W \tag{2.15}$$

Here, we need $k \neq 0$, for diffusion to affect the model. We are concerned with finding constraints on the eigenvalues in λ. In Section 2.6, we will solve these definitions analytically. For now, it is necessary to examine the linear system:

$$\lambda W + Dk^2 W = \gamma JW \tag{2.16}$$

Once the characteristic equation for λ is found, it will provide constraints on the parameters $\alpha,\ \beta,\ \gamma$, and δ that will ensure instability, and thus reveal the Turing region. Stability without diffusion required the eigenvalues produced by the Jacobian matrix be negative. With diffusion, the goal is to find at least 1 positive eigenvalue. The characteristic equation is found in the same way as seen in Section 2.4:

$$\det(-Dk^2 + \gamma\ J - \lambda I) = 0 \qquad \longrightarrow \qquad \det(X - \lambda I) = 0 \qquad \longrightarrow$$

$$\det\left[\begin{pmatrix} -k^2 + \gamma f_u & \gamma f_v \\ \gamma g_u & -\delta k^2 + \gamma g_v \end{pmatrix} - \lambda I\right] = 0 \tag{2.17}$$

Here, $f_u,\ f_v,\ g_u$, and g_v are assumed to be evaluated at the fixed point (u^*, v^*). Now we can utilize the equation $\lambda^2 - \lambda \text{tr}(X) + \det(X) = 0$, yielding:

$$\lambda^2 - \lambda[\gamma\ (f_u + g_v) - k^2(1+\delta)] + \det(X) = 0 \tag{2.18}$$

$$\det(X) = \delta k^4 - \gamma(\delta f_u + g_v)k^2 + \gamma^2(f_u g_v - f_v g_u) \tag{2.19}$$

A value satisfying $\det(X) < 0$ is desired, because it will ensure that the fixed point becomes an unstable saddle node. Alternatively, we could constrain $\text{tr}(X) > 0$ to ensure instability, however we have already established that $f_u + g_v < 0$ in Equation (2.11). Thus $\text{tr}(X) < 0$, and we must focus on the parameter values that satisfy $\det(X) < 0$.

We can examine each term in $\det(X)$ to determine the deciding factors for instability. We know that δk^4 must always be positive, since $\delta > 0$. Also, since $\gamma^2(f_u g_v - f_u g_v) = \gamma^2 \det(J)$, it must be positive, as this was a constraint established on the system without diffusion. The coefficient of interest is $\gamma(\delta f_u + g_v)k^2$, which must be positive for $\det(X) < 0$. We already know that $f_u + g_v < 0$ from Equation (2.11) and $g_v < 0$ from Equation (2.12). Constraining f_u to be positive, we can change the sign of this term. This requires $\alpha < \beta$ and $\delta(\beta - \alpha) > (\alpha + \beta)^3$, although this will not be sufficient to fulfill all the needed constraints. In addition, we must make this term large enough to shadow its positive neighbors, so we need $\delta > \delta_c$ for some critical value $\delta_c > 1$. This critical value is found by solving $\det(X) = 0$.

We now know that for instability to occur, we need to find the threshold for δ that makes the γ coefficient term in $\det(X)$ larger that the other two terms, as well as restrict the parameters α and β so that f_u is positive. This constraint is seen when examining the form of Equation (2.10), as a negative value for the determinant ensures one eigenvalue will be positive and one will be negative. The threshold for δ_c can be found by taking the derivative of $\det(X)$ with respect to k^2, and examining the region where the minimum is negative.

$$2\delta_c k^2 - \gamma(\delta_c f_u + g_v) = 0 \qquad \longrightarrow \qquad k^2 = \gamma \frac{\delta_c f_u + g_v}{2\delta_c} \tag{2.20}$$

Next, we plug this value into $\det(X)$=0 and solve for δ_c:

$$\begin{aligned} 4\delta_c(f_u g_v - f_v g_u) - (\delta_c f_u + g_v)^2 = 0 \qquad \longrightarrow \\ \Big(\delta_c(\beta - \alpha) - (\alpha + \beta)^3\Big)^2 = 4\delta_c(a + b)^4 \end{aligned} \tag{2.21}$$

These parameter constraints describe Turing regions in terms of α, β, and δ. They also provide a minimum delta value, specifically:

$$\delta_c = \frac{(\alpha + \beta)^2}{\beta - \alpha}\left(\alpha + \beta - \sqrt{(\alpha + \beta)^2 - 4(\beta - \alpha)}\right) \tag{2.22}$$

The wave number k^2 is also a critical value. Variations can effect the instability that depends on $\det(X) < 0$. The critical values of k are found when $\det(X) = 0$, as in

Equation (2.19). Since $\det(X)$ is degree 2 polynomial with respect to k^2, we can apply the quadratic formula to find the critical wave numbers. This yields:

$$\begin{aligned} k^2 &= \frac{\gamma}{2\delta}\left[\delta f_u + g_v \pm \sqrt{(\delta f_u + g_v)^2 - 4\delta(f_u g_v - f_v g_u)}\right] \\ &= \frac{\gamma}{2\delta}\left[\delta\left(\frac{\beta-\alpha}{\alpha+\beta}\right) - (\alpha+\beta)^2 \pm \sqrt{\left(\delta\frac{\beta-\alpha}{\alpha+\beta} - (\alpha+\beta)^2\right)^2 - 4\delta(\alpha+\beta)^2}\right] \end{aligned} \tag{2.23}$$

We then have $k_- < k < k_+$. More about the wavenumbers and their numerical approximations will be discussed in Chapter 3.

There are now multiple constraints on α, β, and δ that define the Turing instability region. We can visualize these regions for various values of δ, as shown in Figure (2.2). The shaded areas are Turing instability regions, which are stable when there is no diffusion present, but unstable when diffusion is added. Notice that the region grows for increasing δ, and for $\delta >> 1$, the slope at the origin tends toward $\alpha = \beta$. The parameter constraints are summarized in Table (2.1).

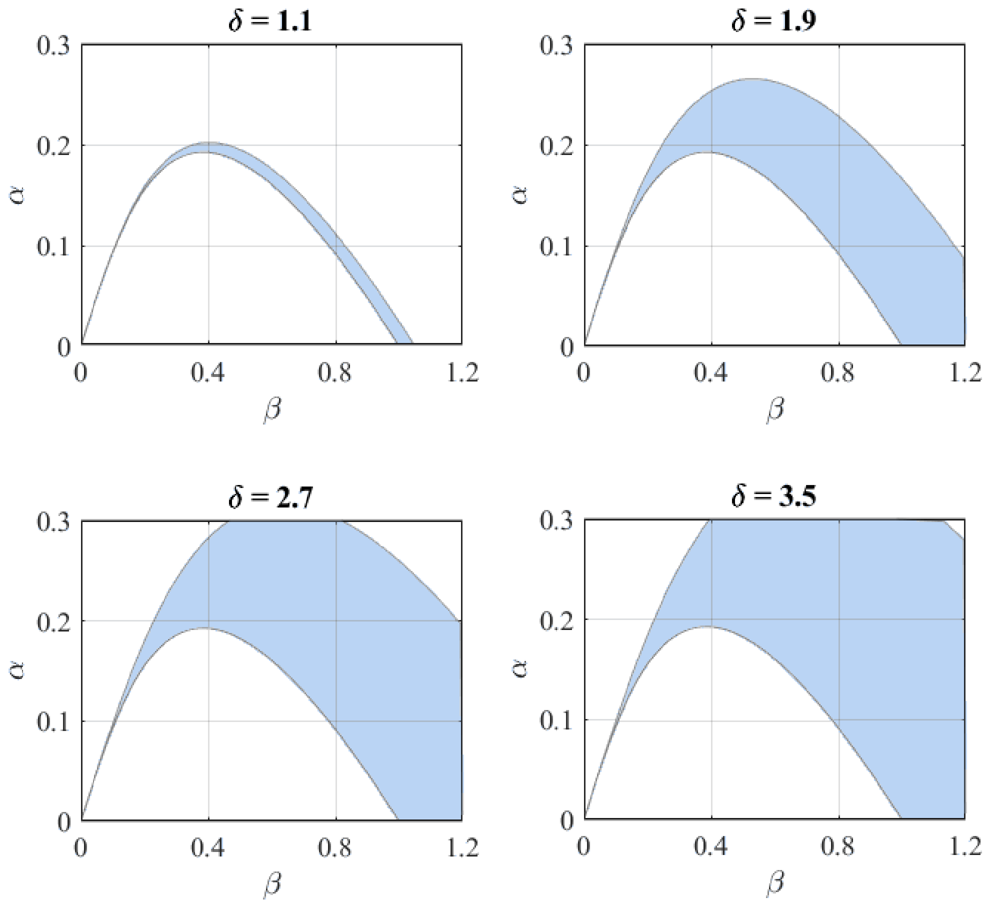


Figure 2.2. Instability regions for varying δ in the model system when diffusion is present.

Table 2.1. Parameter Constraints For Turing Instabilities

Generic Constraint	**Parameter Constraint**
$f_u + g_v < 0$	$\beta - \alpha < (\alpha + \beta)^3$
$\delta f_u + g_v > 0$	$\alpha < \beta$ and $\delta(\beta - \alpha) > (\alpha + \beta)^3$
$4\delta_c(f_u g_v - f_v g_u) - (\delta_c f_u + g_v)^2 > 0$	$\big(\delta(\beta - \alpha) - (\alpha + \beta)^3\big)^2 > 4\delta(\alpha + \beta)^4$

2.6 Analytical Solutions on a Sphere

Having established several parameter constraints, we can now solve the eigenvalue problems stated in (2.15) using the standard partial differential equation method of separation of variables. We can begin with a domain on the surface of the unit sphere. For this analysis, we are only concerned with the steady state patterns that arise from exciting specific wavenumbers when $\dot{W} = 0$. Therefore, we will only be examining the spatial variables. Here, the following boundary conditions and initial conditions are set as:

$$r = 1, \quad -\frac{\pi}{2} < \theta < \frac{\pi}{2}, \quad -\pi < \phi < \pi \tag{2.24}$$

To solve the spatial eigenvalue problems from Equation (2.15), we solve their corresponding eigenfunctions. The Laplace-Beltrami eigenvalue problem is given by:

$$\Delta_\Gamma W = -k^2 W = \mu \tag{2.25}$$

It takes a 3-dimensional argument in spherical coordinates and eliminates the dependence on the radial vector. If we use the unit sphere, then there is no term representing the radius to evaluate. However, since we are examining growing domains, it is prudent to mention that radii not equal to one will have a scaling effect on the solution interval values. Consider:

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial W}{\partial r}\right) + \frac{1}{r^2 \sin\phi}\frac{\partial}{\partial \phi}\left(\sin\phi\frac{\partial W}{\partial \phi}\right) + \frac{1}{r^2 \sin^2\phi}\frac{\partial^2 W}{\partial \theta^2} = k^2 W = \mu \tag{2.26}$$

But without any change in the radial dimension, the partial derivatives with respect to r are zero, leaving:

$$\frac{1}{\sin\phi}\frac{\partial}{\partial\phi}\left(\sin\phi\frac{\partial W}{\partial\phi}\right)+\frac{1}{\sin^2\phi}\frac{\partial^2 W}{\partial\theta^2}=r^2k^2W=\mu \tag{2.27}$$

If we evaluate the wavenumber as r^2k^2, then we find that the k values for the Turing instability interval (2.23) change to $r^2\cdot(k_-,k_+)$. This observation becomes extremely useful when we discuss growing domains in Section 5.2. For now, it is sufficient to evaluate the system's eigenfunctions for a domain were the unit sphere.

Since our solution has the form $W(\theta,\phi,t)=x(\theta)y(\phi)z(t)$, the results are the two Sturm-Liouville equations:

$$\frac{d}{d\theta}\left(\sin\theta x'(\theta)\right)+\left(k^2\sin\theta-\frac{\mu}{\sin\theta}\right)x(\theta)=0 \qquad \text{and} \qquad y''(\phi)-\mu y(\phi)=0 \tag{2.28}$$

We find a periodic solution for $y(\phi)$, since $y(\pi)=y(-\pi)$, and likewise for the derivative $y'(\phi)$. This leads to solutions in sine and cosine, given in their exponential form below. The function in $x(\theta)$ follows the form of the Legendere equation. We now have the 2 eigenfunctions for W in θ and ϕ:

$$x_{mn}(\theta)=P_n^m(\cos\theta) \qquad \text{and} \qquad y_m(\phi)=e^{im\phi} \tag{2.29}$$

with $m=0,1,2,...$ and $n\geq m$. The eigenvalue μ is equal to m^2, and the wave numbers are necessarily $k^2=n(n+1)$, which satisfied the the Legendre Polynomial in θ. Together, the eigenfunctions in x and y are defined as the spherical harmonics $Y_n^m(\theta,\phi)=P_n^m(\cos\theta)e^{im\phi}$, and have many applications in dynamical systems [12]. Since we already established an interval for k in which the Turing Region exists, we can define the solution in terms of the maximum and minimum values of k. Let $k_{\min}$ designate the smallest integer value in (k_-,k_+), and let $k_{\max}$ be the greatest integer value in (k_-,k_+). Then the analytical solution can be defined as:

$$W(\theta,\phi)=\sum_{m=0}^{\infty}\sum_{n=k_{\min}\geq m}^{k_{\max}}Y_n^m(\theta,\phi) \tag{2.30}$$

While (2.30) represents the overall picture of the eigenfunction solutions, we are more interested in specific values of m and n which excite certain wavenumbers. Our goal is to show that we can derive patterns from the eigenfunction solutions and then duplicate them numerically. By plotting the eigenfunctions $Y_n^m(\theta,\phi)$ on the surface of a sphere, we can excite wavenumbers k^2 using values of m and n. Figure 2.3 below shows several examples of eigenfunction solution approximation visualizations for varying values of m and n. The visualizations use a color scheme where dark purple represents relatively lower values, and light green indicates relatively greater values. In this context, we are more interested in the patterns rather than the values they represent. For these images, an open-source MATLAB® function [35] was used to calculate the values for each 3D point, then the data was copied into a .vtk file for visualization in ParaView®. More discussion on comparing the eigenfunction solutions to the numerical solutions of the model system is discussed more in Chapter 3.

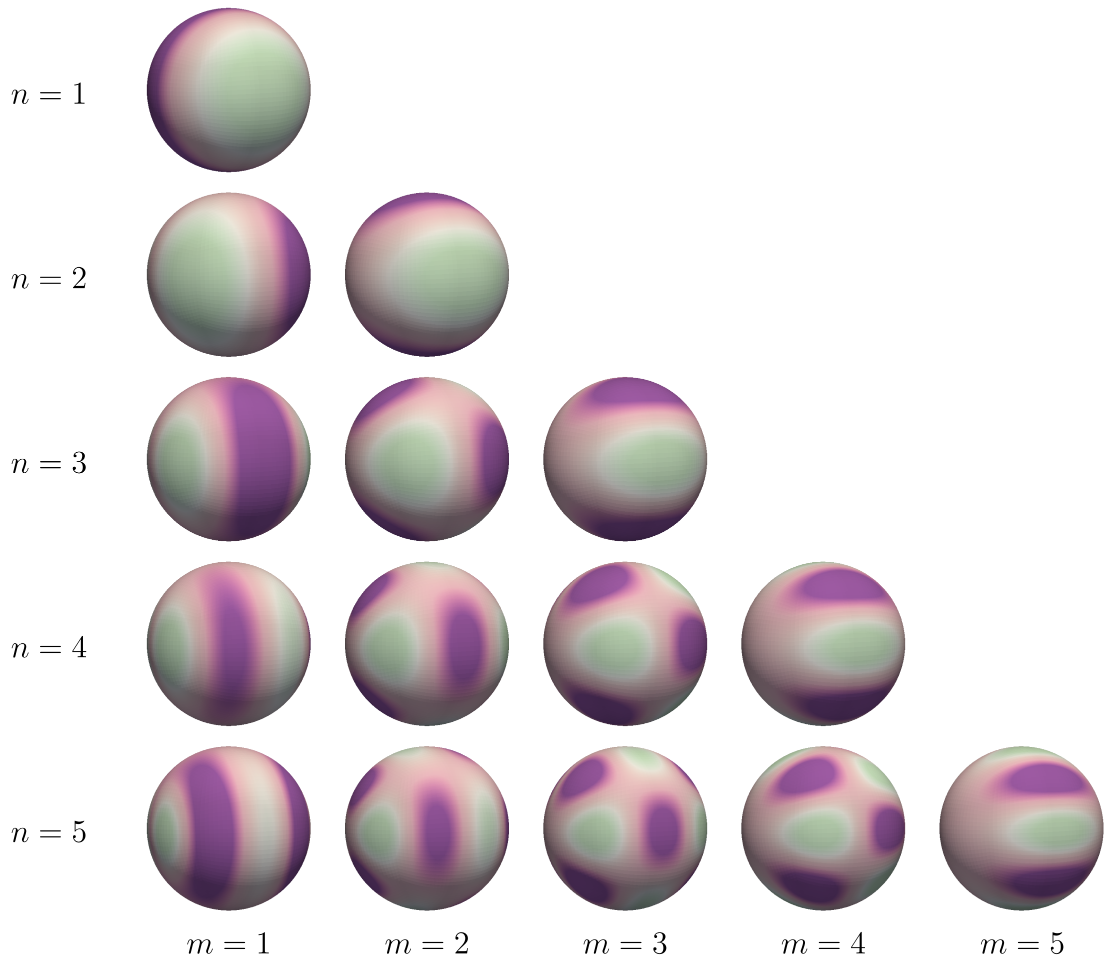


Figure 2.3. Eigenfunctions of spherical harmonics on the surface of a sphere

CHAPTER 3
Numerical Approximation of the Model Equations

The model system does not have an analytic solution for irregular geometries, so a finite element numerical approximation is used. First, we will describe the finite element method in which a domain is split into a discretized mesh and the differential equation is solved on each division. Next, we will discuss how I discretized the system in time using an implicit-explicit (IMEX) combination scheme. Then we will examine how varying the parameter γ corresponds to a comparable growth of the domain. Finally, this chapter will discuss the pattern visualizations for varying γ on the surface of a sphere.

3.1 The Surface Finite Element Method

Solving the model equations using the finite element method (FEM) uses the Laplace-Beltrami operator for a 2D domain in a 3D space. The discretized space is represented numerically by a linear system of matrices, and for parabolic problems, the square matrices are conveniently formatted as symmetric and positive definite. This process has already received thorough analysis [10], so only a brief overview will be given here.

To begin, recall the system of differential equations on the 2D domain Γ, now given Neumann boundary conditions:

$$\frac{\partial u}{\partial t} - \Delta_\Gamma u = \gamma\, f(u,v) \qquad \frac{\partial v}{\partial t} - \delta\Delta_\Gamma v = \gamma\, g(u,v) \qquad \text{with} \qquad \hat{n}\cdot\nabla_\Gamma u = \hat{n}\cdot\nabla_\Gamma v = 0 \;\text{ on }\; \partial\Gamma \tag{3.1}$$

We treat this as a closed system, therefore there is no flux through the surface. The first steps in applying the surface finite element method are to multiply the equations by a test function and then integrate to establish the weak formulation. The purpose of using a test function of our choosing is to guarantee a solution for the system in the weak formulation. The test function φ must satisfy:

$$\int_\Gamma \varphi\frac{\partial u}{\partial t} - \int_\Gamma \varphi\Delta_\Gamma u = \int_\Gamma \gamma\varphi f(u,v) \qquad\qquad \int_\Gamma \varphi\frac{\partial v}{\partial t} - \int_\Gamma \varphi\delta\Delta_\Gamma v = \int_\Gamma \gamma\varphi g(u,v) \tag{3.2}$$

Note that we can use integration by parts here, and substitute a more easily computed term for Δ_Γ in this context. Green's Theorem states:

$$\int_{\delta\Gamma} a \cdot \nabla_\Gamma b = \int_\Gamma a \cdot \Delta_\Gamma b + \int_\Gamma \nabla_\Gamma a \cdot \nabla_\Gamma b \tag{3.3}$$

We can redily apply Green's Theorem here, and since the flux on the boundary $\partial\Omega$ is zero, we can rewrite (3.2) as:

$$\begin{aligned} \int_\Gamma \varphi \frac{\partial u}{\partial t} + \int_\Gamma \nabla_\Gamma \varphi \cdot \nabla_\Gamma u = \gamma \int_\Gamma \varphi f(u,v) \quad \text{and} \\ \int_\Gamma \varphi \frac{\partial v}{\partial t} + \delta \int_\Gamma \nabla_\Gamma \varphi \cdot \nabla_\Gamma v = \gamma \int_\Gamma \varphi g(u,v) \end{aligned} \tag{3.4}$$

3.1.1 Mesh Discretization

Now recall the discussion of the Laplace-Beltrami operator in Section 2.3. When considering that our domain is not continuous but a mesh of countable cells, we can represent Γ as a discretized space $\mathbb{T}$ with elements K. In this case, $\mathbb{T}$ is a representation of Γ that is partitioned into non-overlapping quadrilateral cells. This work uses both a spherical mesh domain for preliminary studies and validation, as well as a polygonal lung mesh for biological applications. The mesh surface is made of quadrilateral cells. The points where the quadrilateral vertices meet are called nodes. Surrounding each node is a patch of quadrilaterals that each share a node as a vertex. Figure 3.1 shows a close-up area of one such mesh used in this thesis, highlighting a patch of cells sharing a node.

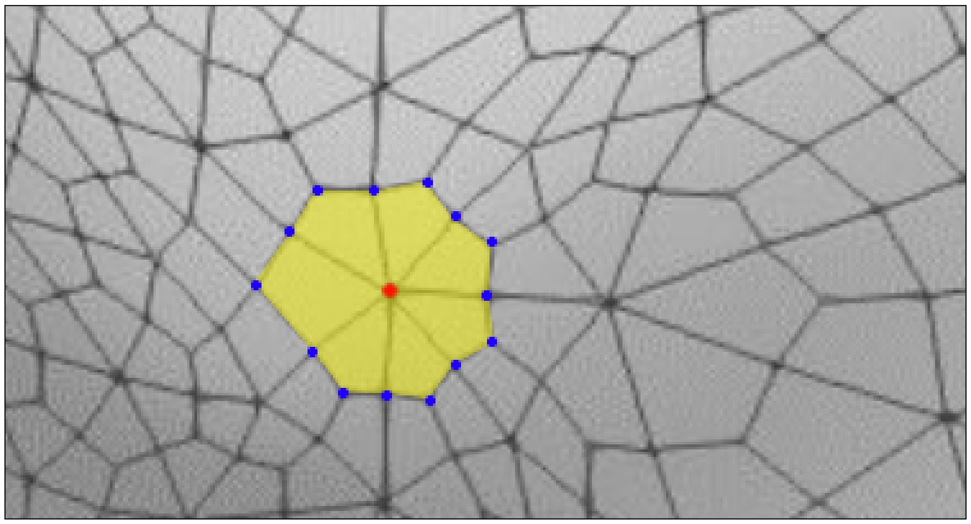


Figure 3.1. Close-up of polygonal lung mesh with quadrilateral elements. The yellow region represents a patch of cells that share a node (red) as one of their vertices (blue).

Using this notation to represent the mesh mathematically, we approximate the system as:

$$
\begin{aligned}
\sum_{K\in\mathbb{T}}\int_K \varphi\frac{\partial u}{\partial t} + \sum_{K\in\mathbb{T}}\int_K \nabla_K\varphi\cdot\nabla_K u &= \gamma\sum_{K\in\mathbb{T}}\int_K \varphi f(u,v)\\
\sum_{K\in\mathbb{T}}\int_K \varphi\frac{\partial v}{\partial t} + \delta\sum_{K\in\mathbb{T}}\int_K \nabla_K\varphi\cdot\nabla_K v &= \gamma\sum_{K\in\mathbb{T}}\int_K \varphi g(u,v)
\end{aligned}
\tag{3.5}
$$

Because our domain is a surface, we must use the tangential gradients when computing the spatial integral (as opposed to the standard gradients for a bulk volume). Recall that the matrices here are symmetric and positive definite. In this context, the tangential gradient is defined for each variable as:

$$
\begin{aligned}
&\nabla_K u = D\mathbf{x}_K G_K^{-1}\nabla u \qquad \nabla_K v = D\mathbf{x}_K G_K^{-1}\nabla v\\
&\text{and}\quad \nabla_K\varphi = D\mathbf{x}_K G_K^{-1}\nabla\varphi = \nabla\varphi^T G_K^{-1} D\mathbf{x}_K^T\\
&\text{with}\quad G_K = D\mathbf{x}_K^T D\mathbf{x}_K \quad\text{and}\quad D\mathbf{x}_K = \frac{\partial\mathbf{x}_K}{\partial\bar{\mathbf{x}}}
\end{aligned}
\tag{3.6}
$$

Furthermore, we can express the spatial elements in terms of a reference element $\hat{K} = [0,1]^2$, which is beneficial for efficiency in numerical implementation. The reference element is derived from the discretization of the domain mesh. Since our surface mesh is made of quadrilaterals, the reference element will be a unit square. We use the reference element to more quickly and efficiently solve the given system on each cell of the mesh. It is easier to first solve each cell of the model on the unit square, then input that solution through a isoparametric mapping that aligns with the spatial properties of an irregular quadrilateral in the mesh. Figure 3.2 shows an example visualization of such a process for a square $[-1,1]^2$.

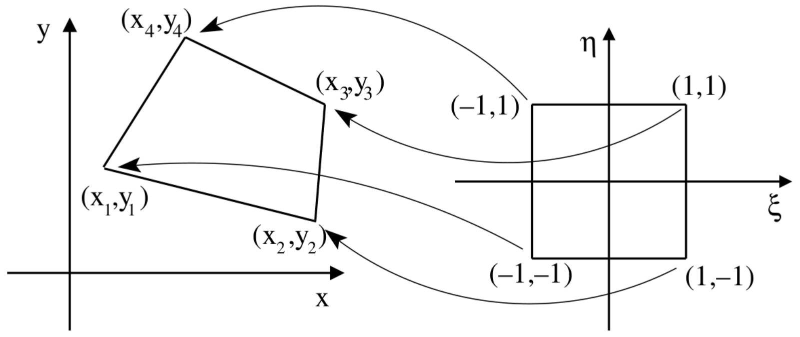


Figure 3.2. Example visualization of the reference element under isoparametric mapping to a quadrilateral on a discretized mesh. Figure adapted with permission from ETH Zurich Universitatstrasse, [P. Arbenz, *Lecture notes for Introduction to finite elements and sparse linear system solving*, September 2017.]

The discretized space can be expressed in terms of the reference element and simplified using the relation $G_K = D\mathbf{x}_K^T D\mathbf{x}_K$.

$$\begin{aligned}\int_K \nabla_K \varphi \cdot \nabla_K u &= \int_{\hat{K}} \left(\nabla\varphi^T G_K^{-1} D\mathbf{x}_K^T\right) D\mathbf{x}_K G_K^{-1} \nabla u = \int_{\hat{K}} \nabla\varphi^T G_K^{-1} \nabla u \\ \int_K \nabla_K \varphi \cdot \nabla_K v &= \int_{\hat{K}} \left(\nabla\varphi^T G_K^{-1} D\mathbf{x}_K^T\right) D\mathbf{x}_K G_K^{-1} \nabla v = \int_{\hat{K}} \nabla\varphi^T G_K^{-1} \nabla v\end{aligned} \tag{3.7}$$

It is prudent to note that this process is referred to as *triangulation*, named from the standard practice in computational mesh generation, which creates surfaces with triangular elements. For the sphere model, we use quadrilaterals within a spherical manifold refined 4 times, yielding a discretized surface with 4^4 quadrilaterals. The lung model is also be composed of quadrilaterals.

3.1.2 Basis Functions

The triangulation process lends itself to numerical representation in vector form. Vertices on the mesh are assigned to positions in a vector, and can be solved in a linear system. With the discretization of space, the variables u and v (representing FGF10 and SHH, respectively), must in turn be discretized. Below is our discretized approximation for u and v, with new variables explained below:

$$u \approx u_K = \sum_j \vartheta_j U_j \qquad v \approx v_K = \sum_j \vartheta_j V_j \tag{3.8}$$

U_j and V_j are unknown coefficients, which we will ultimately be solving for. We now introduce the basis function ϑ_j. The basis function in this context is similar to the linear algebra standard. It is a piecewise polynomial whose purpose is to assign a value to a node with index j, interpolate values for the vertices that share an edge with node j, and return zero for all other vertices. We create a different basis function for each node on the mesh. Recall Figure 3.1 earlier in this section. Our basis function for this particular patch would have a value of one for the red-colored node, and interpolated value for the blue vertices, and zero for every other vertex. In "classical" FEM formulations, low-degree polynomials are used for each basis function. Predictably, higher degree polynomials result in both increased accuracy and increased computing time. For our model, we use a tri-quadratic piecewise approximation for each basis function, given in the form of

$$(ax^2 + bx + c)(dy^2 + ey + f)(gz^2 + hz + i) \tag{3.9}$$

with all real coefficients. Each of the 27 terms ($1,\ x,\ y,\ z,\ xy,\ xz,\ yz,\ xyz...\ x^2y^2x^2$) form a linear combination that composes the basis function. Numerical algorithms are then responsible for the heavy lifting of using the basis functions to solve the model on each cell.

3.1.3 Notation Simplification

Our spatially discretized system is as follows (with the tangential gradient notation omitted for clarity):

$$\begin{aligned}
&\sum_{K\in\mathbb{T}}\sum_{j}\int_K \varphi_i\cdot\vartheta_j\left[\frac{\partial U_j}{\partial t}\right]+\sum_{K\in\mathbb{T}}\sum_{j}\int_K \nabla_K\varphi_i\cdot\nabla_K\vartheta_j\left[U_j\right]=\gamma\sum_{K\in\mathbb{T}}\sum_{j}\int_K \varphi_i f_K(U_j,V_j)\\
&\sum_{K\in\mathbb{T}}\sum_{j}\int_K \varphi_i\cdot\vartheta_j\left[\frac{\partial V_j}{\partial t}\right]+\delta\sum_{K\in\mathbb{T}}\sum_{j}\int_K \nabla_K\varphi_i\cdot\nabla_K\vartheta_j\left[V_j\right]=\gamma\sum_{K\in\mathbb{T}}\sum_{j}\int_K \varphi_i g_K(U_j,V_j)
\end{aligned} \tag{3.10}$$

With these approximations, the test functions φ_i can have a solution for the system at each vertex. For this thesis, it will be sufficient to state that a unique solution to the above system exists as a direct application of the Lax-Milgram Theorem [10]. For ease of communication, we will further simplify the notation here as:

$$\sum_{K\in\mathbb{T}}\sum_{j}\int_K x\cdot y=\Big(\,x,y\,\Big)$$

Now our system is more easily examined when written as

$$\begin{aligned}
\Big(\,\varphi_i,\vartheta_j\,\Big)\frac{\partial U_j}{\partial t}+\Big(\,\nabla_K\varphi_i,\nabla_k\varphi_j\,\Big)U_j &= \gamma\Big(\,\varphi_i,f_K(U_j,V_j)\,\Big)\\
\Big(\,\varphi_i,\vartheta_j\,\Big)\frac{\partial V_j}{\partial t}+\delta\Big(\,\nabla_K\varphi_i,\nabla_k\varphi_j\,\Big)V_j &= \gamma\Big(\,\varphi_i,g_K(U_j,V_j)\,\Big)
\end{aligned} \tag{3.11}$$

Here, it is useful to explain the expansion of the functions f_K and g_K, the discretized versions of the reaction equations seen in (2.4). Because the solutions are in vector format, we must treat the nonlinear term piecewise; that is, multiply each vector term according to its position in the vector.

Since the basis functions need only satisfy the linear system, we want it to be as simple as possible. In addition, α and β are transformed into $\mathbf{a}$ and $\mathbf{b}$, which are simply mono-valued vectors corresponding to the size of the basis function vectors. We can now expand the f_K and g_K terms as follows:

$$\begin{aligned}
\Big(\,\varphi_i,f_K\,\Big)&=\Big(\,\varphi_i,\alpha-u_K+u_K^2v_K\,\Big)=\Big(\,\varphi_i,\mathbf{a}\,\Big)-\Big(\,\varphi_i,\varphi_j\,\Big)\cdot U_j+\Big(\,\varphi_i,\varphi_j\,\Big)\cdot U_j^2V_j\\
\Big(\,\varphi_i,g_K\,\Big)&=\Big(\,\varphi_i,\beta-u_K^2v_K\,\Big)\qquad=\Big(\,\varphi_i,\mathbf{b}\,\Big)-\Big(\,\varphi_i,\varphi_j\,\Big)\cdot U_j^2V_j
\end{aligned} \tag{3.12}$$

Note that the notation $U_j^2 V_j$ represents the nonlinear term vectors multiplied piecewise. Rewriting Equation (3.11) using the substitutions above yields:

$$\begin{aligned}\Big(\varphi_i,\vartheta_j\Big)\frac{\partial U_j}{\partial t}+\Big(\nabla_K\varphi_i,\nabla_k\vartheta_j\Big)U_j&=\gamma\Big[\Big(\varphi_i,\mathbf{a}\Big)-\Big(\varphi_i,\vartheta_j\Big)\cdot U_j+\Big(\varphi_i,\vartheta_j\Big)\cdot U_j^2V_j\Big]\\ \Big(\varphi_i,\vartheta_j\Big)\frac{\partial V_j}{\partial t}+\delta\Big(\nabla_K\varphi_i,\nabla_k\vartheta_j\Big)V_j&=\gamma\Big[\Big(\varphi_i,\mathbf{b}\Big)-\Big(\varphi_i,\vartheta_j\Big)\cdot U_j^2V_j\Big]\end{aligned} \tag{3.13}$$

We can now use matrix notation for each summation, using the following terms:

$$\mathbf{M}=(\varphi_i,\vartheta_j)\qquad \mathbf{L}=(\nabla\varphi_i,\nabla\vartheta_j)\qquad \mathbf{A}=(\varphi_i,\mathbf{a})\qquad \mathbf{B}=(\varphi_i,\mathbf{b})$$

Common nomenclature dictates that **M** is the mass matrix, **L** is the Laplace matrix, and **A** and **B** are the forcing term vectors. The simple form is now:

$$\begin{aligned}\mathbf{M}\cdot\frac{d}{dt}[U_j]+\mathbf{L}\cdot U_j&=\gamma(\mathbf{A}-\mathbf{M}\cdot U_j+\mathbf{M}\cdot U_j^2V_j)\\ \mathbf{M}\cdot\frac{d}{dt}[V_j]+\delta\mathbf{L}\cdot V_j&=\gamma(\mathbf{B}-\mathbf{M}\cdot U_j^2V_j)\end{aligned} \tag{3.14}$$

The spatial discretization is now complete. We can use these results to plug into the temporal discretization, which will be discussed in the next section. We will use the following generalized substitutions:

$$\begin{aligned}u&\to\mathbf{M}\cdot U_j & v&\to\mathbf{M}\cdot V_j\\ \Delta u&\to-\mathbf{L}\cdot U_j & \Delta v&\to-\mathbf{L}\cdot V_j\\ \alpha&\to\mathbf{A} & \beta&\to\mathbf{B}\\ &u^2v\to\mathbf{M}\cdot U_j^2V_j\end{aligned} \tag{3.15}$$

We use the openly sourced *deal.ii* library [3], a differential equation analysis library for C++, to create the spherical triangulated mesh and implement most functions for the finite element method. *Deal.ii* produces a triangulated model with a few simple commands. The use of tangential gradients for surface calculations is more challenging, requiring several nested loops through each vertex on the mesh. References and comments in the code implementing these triangulation procedures can be found in Section 3.4, and the complete code is in Appendix A.

3.2 Implicit-Explicit Time Stepping Scheme

The following time discretization will use both implicit and explicit strategies. There is significant evidence that using a combination of implicit and explicit methods improves the stability and decreases the error in temporal discretization schemes [21]. Further explanation and a summary of improved methods can be found in Appendix A. Once we assemble the IMEX linear system, we will substitute the spatial discretization terms discussed in Section 3.1. To begin, we recall our nondimensionalized system of equations:

$$\frac{\partial u}{\partial t} - \Delta u = \gamma\ (\alpha - u + u^2 v)\ , \qquad \frac{\partial v}{\partial t} - \delta \Delta v = \gamma\ (\beta - u^2 v) \tag{3.16}$$

First, let's examine the first equation in terms of u. We apply a first order implicit Euler method to the time derivative, with the exception of the nonlinear term, which will be kept explicit. This is one of many forms known as the Implicit-Explicit, or IMEX, discretization scheme. Note that k is the time step length and N is the time step index number.

$$\frac{u_N - u_{N-1}}{k} - \Delta u_N = \gamma\ (\alpha - u_N + u_{N-1}^2 v_{N-1}) \tag{3.17}$$

Separating the unknown values with u_n we get

$$u_N + k\left(\gamma u_N - \Delta u_N\right) = k\gamma\left(\alpha + u_{N-1}^2 v_{N-1}\right) + u_{N-1} \tag{3.18}$$

From here, we can solve the equation for v using a slightly more implicit scheme. This is possible because when solving this system numerically, we can solve one equation before the other for each time step. Therefore, the value u_N will be known when solving the equation for v. This yields:

$$\frac{v_N - v_{N-1}}{k} - \delta \Delta v_N = \gamma(\beta - u_N^2 v_{N-1}) \tag{3.19}$$

Again, separating unknown terms yields

$$v_N - k\left(\delta \Delta v_N\right) = k\gamma\left(\beta - u_N^2 v_{N-1}\right) + v_{N-1} \tag{3.20}$$

In operator form, the system simplifies to

$$\begin{aligned} \left[1 + k\left(\gamma - \Delta \cdot\right)\right] u_n &= k\gamma\left(\alpha + u_{N-1}^2 v_{N-1}\right) + u_{N-1} \\ \left[1 - k\delta\Delta\cdot\right] v_n &= k\gamma\left(\beta - u_{N-1}^2 v_{N-1}\right) + v_{N-1} \end{aligned} \tag{3.21}$$

Using this combination of implicit and explicit approaches allows a moderately high level of accuracy without a significantly complex scheme. The issues surrounding solving a model with nonlinear terms makes a fully implicit scheme challenging. We achieved a reasonably stable scheme, so a more complex discretization that better accounts for the nonlinear term was unnecessary. We can apply this temporally discretized system to the spatially discretized result of the finite element method linear system by substituting the results to form a linear system. Combining Equation (3.21) and the substitutions in (3.15), we get:

$$\begin{aligned}\left[(1+k\gamma)\mathbf{M}+k\mathbf{L}\right]U_N &= k\gamma\left(\mathbf{A}+\mathbf{M}U_{N-1}^2V_{N-1}\right)+\mathbf{M}U_{N-1}\\ \left[\mathbf{M}+k\delta\mathbf{L}\right]V_N &= k\gamma\left(\mathbf{B}-\mathbf{M}U_N^2V_{N-1}\right)+\mathbf{M}V_{N-1}\end{aligned} \tag{3.22}$$

We now have a linear system of the form $\mathscr{A}\mathbf{x}=\mathscr{b}$, with:

$$\mathscr{A}=\begin{pmatrix}(1+k\gamma)\mathbf{M}+k\mathbf{L}\\ \mathbf{M}+k\delta\mathbf{L}\end{pmatrix},\quad \mathbf{x}=\begin{pmatrix}U_N\\ V_N\end{pmatrix},\quad \text{and } \mathscr{b}=\begin{pmatrix}k\gamma\left(\mathbf{A}+\mathbf{M}U_{N-1}^2V_{N-1}\right)+\mathbf{M}U_{N-1}\\ k\gamma\left(\mathbf{B}-\mathbf{M}U_N^2V_{N-1}\right)+\mathbf{M}V_{N-1}\end{pmatrix} \tag{3.23}$$

3.3 Adaptive Time Stepping

The stability of this system largely depends on the size of the time step chosen (this is discussed further in Chapter 4). Even though we have a partially implicit scheme, it does not have the robust stability that a fully implicit Euler scheme has. For our numerical implementations, the time step size began at $k_0 = 10^{-3}$, and was adjusted based on an adaptive scheme. Because several simulations ran until $t = 10$ to validate convergence, there was a desire to minimize the number of time steps required for the simulation.

In order to maximize time efficiency and ensure stability, we adopted an adaptive time stepping scheme [21] when solving on the lung mesh. The scheme itself uses a variation of the Euler midpoint method to determine if the system is stable and the time step can be increased, or if the system is heading toward instability and the time step should be decreased. The second order method is given as follows:

$$\begin{aligned}
&F^{(1)} = u_n + k_n f(u_n, v_n) \qquad\qquad G^{(1)} = v_n + k_n g(u_n, v_n)\\
&F^{(2)} = u_n + k_n f\left(u_n + \frac{k_n}{2}F^{(1)}, v_{+}\frac{k_n}{2}G^{(1)}\right)\\
&G^{(2)} = v_n + k_n g\left(u_n + \frac{k_n}{2}F^{(1)}, v_{+}\frac{k_n}{2}G^{(1)}\right)\\
&e_f = \sqrt{\left(F^{(2)} - F^{(1)}\right)^2} \qquad\qquad e_g = \sqrt{\left(G^{(2)} - G^{(1)}\right)}\\
&\chi = \min\left[\left(\frac{k_0}{e_f}\right)^{1/4}, \left(\frac{k_0}{e_g}\right)^{1/4}\right] \qquad \longrightarrow \qquad k_{n+1} = \chi k_n
\end{aligned} \tag{3.24}$$

To prevent drastic time step changes in either direction, we enforced the following constraint:

$$\text{if } \chi > 1.1,\ \chi = 1.1 \quad \text{and} \quad \text{if } \chi < 0.9,\ \chi = 0.9$$

A convergence analysis and simulation validation for testing this method on a lung mesh will be discussed further in Chapter 4, which will go into detail on error analysis and the effect of time stepping constraints.

3.4 *deal.II* Algorithm Details

The C++ library *deal.ii* was used to solve the model in this thesis. The analytic surface finite element method is implemented through a variety of functions designed to discretize a defined space and solve a differential equation on each element. Below is a summary of solving the model equations on the surface of a sphere. The headings for the pseudocode appear as comments in Appendix A.

- **Shape Definition:** We use *deal.ii*'s GridGenerator function to call a hypersphere, which defines only the surface elements of a sphere. It can be refined an arbitrary number of times according to the user's desired degree of precision.
- **Degrees of Freedom:** The degrees of freedom are associated with the edges on each quadrilateral of the mesh. Expressed as a single number, the degrees of freedom represent the number of expansion coefficients (U_j and V_j) that need to be solved in the linear system. This step distributes the degrees of freedom, essentially numbering each quadrilateral edge, as well as establish the size of the solution and forcing term vectors.
- **Sparsity Pattern:** Since each degree of freedom is only dependent on neighboring nodes, the linear system can be represented as a sparse matrix. The sparsity pattern of the matrix is saved in templates for the mass matrix, Laplace matrix, and what will become the system matrix (the $\mathscr{A}$ in the linear system $\mathscr{A}\mathbf{x} = \mathscr{b}$).
- **Surface Assembly:** Because we are solving on a surface, the default shape functions will not be useful. Using the degrees of freedom enumeration, we looped through each cell and defined the tangential gradients for each shape function.
- **System Assembly:** This is the longest section of code, though not terribly complex. Since we have a system of two equations, all the above steps were implemented twice. With the exception of the surface assembly, *deal.ii* had built in functions that handled each process automatically. No such function exists for the time discretization, so we manipulated the mass matrix, Laplace matrix, and forcing terms to create the system matrix and right hand side vector. The IMEX method, as detailed in Section 3.2, was the basis for this manipulation.
- **Solve Timestep:** Here, we use *deal.ii*'s conjugate gradient solver on the linear system. No preconditioner is necessary, since for parabolic problems the matrices are already symmetric and positive definite.

- **Output Solution:** For observing a sphere with a single γ value, we solve until $t = 10$ to observe the steady state. When modeling the evolution of patterns on a growing domain, the steady state is found several times, each with a different value for γ. More on this process is explained in Section 5.2.

3.5 Wavenumber Pattern Replication

Here we expand on the numerical application of the ideas discussed in Section 2.6. An important part of verifying the validity of the model above is showing that the numerical scheme can successfully reproduce the eigenfunction solutions on the surface of a sphere. First, it may be prudent to note that one of the criticisms of Turing Pattern analysis is that resulting patterns rely heavily on initial conditions. Here we use several different initial conditions to excite various wavenumbers, and observe the resulting pattern. Using the steady state initial values, $\big(\alpha+\beta,\ \beta/(\alpha+\beta)^2\big)$, does not produce the desired result. There must be some perturbation from these values to form patterns. This is the deviation from a homogeneous state to which Turing referred. Through reviewing published results [21] and through trial-and-error, we found that the following perturbation from initial conditions readily instigated pattern formation:

Table 3.1. Perturbations from Steady-State Initial Conditions

Variable	Initial Condition
FGF10 (u)	$\alpha+\beta+0.1\cdot\sum_{j=1}^{9}\cos(2\pi jxyz)$
SHH (v)	$\frac{\beta}{(\alpha+\beta)^2}+0.1\cdot\sum_{j=1}^{9}\cos(2\pi jxyz)$

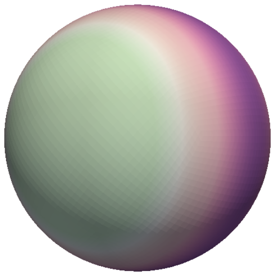
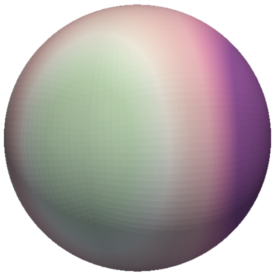
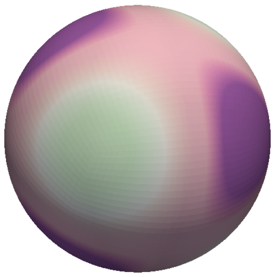
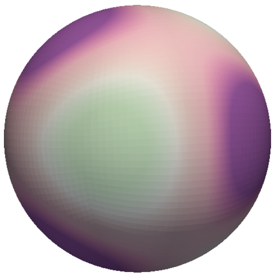
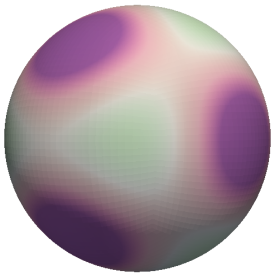
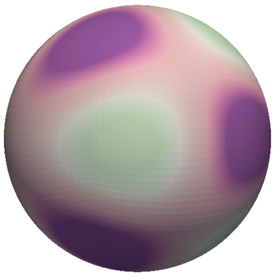
Notice that the maximum value for either perturbation is 0.9. The values of α and β were chosen so that even the largest perturbation possible would still allow initial conditions valid within the Turing instability region. Table 3.2 shows some possible γ values for various excited wavenumbers k^2. Table 3.3 gives a few examples of visualizations for $\alpha = 0.1,\ \beta = 0.9$, and $\delta = 10$. Recall that the values for k_- and k_+ are found by applying (2.23), and that $k^2 = n(n+1)$. Therefore, each wavenumber corresponds to some n value. In addition, each numeric solution matches an eigenfunction solution from Figure 2.3.

Table 3.2. Possible Parameters for the First 6 n Model Eigenvalues

α	β	γ	k^2 Interval	Wavenumber	n Eigenvalues
0.082	0.940	10.377	(1.998, 5.630)	$k^2 = 2$	$n = 1$
0.037	1.065	10.377	(2.0543, 6.3657)	$k^2 = 6$	$n = 2$
0.1	0.9	24.039	(4.8078, 12.0195)	$k^2 = 6,\ 12$	$n = 2$ or $n = 3$
0.014	1.245	24.039	(7.5306, 12.1634)	$k^2 = 12$	$n = 3$
0.134	0.821	70.060	(15.959, 28.050)	$k^2 = 20$	$n = 4$
0.1	0.9	70.060	(14.0120, 35.0300)	$k^2 = 20,\ 30$	$n = 4$ or $n = 5$
0.176	0.538	100.410	(19.7041, 26.0852)	$k^2 = 30$	$n = 5$
0.1	0.9	100.410	(20.0820, 50.2050)	$k^2 = 30,\ 42$	$n = 5$ or $n = 6$

While the numerical model can reproduce wavenumber excitations on the surface of a sphere, there are still some issues with validating accuracy. When comparing patterns between the eigenfunction solutions and numerical solutions, the images were examined using the naked eye. Patterns were matched based on visual similarity between the dark purple regions, which represent relatively lower values, and the light green regions, which represent relatively higher values. Like the eigenfunction solutions, we are more concerned with the pattern produced than the solution values themselves. A numeric image analysis tool may have been better able to match wavenumber patterns to their model counterpart. However, it is clear that our numeric scheme is able to solicit patterning. Further experimentation through parameter manipulation can certainly produce even more matching patterns, however for our purposes a simple proof of concept is sufficient to validate our methods.

Table 3.3. A Comparison Of Numeric and Eigenfunction Solutions For the Model On the Surface of a Sphere

Numeric Solutions ($\alpha = 0.1,\ \beta = 0.9,\ \delta = 10$)	Eigenfunction Solutions
$\gamma = 24.039 \longrightarrow$ $n = 2$ 	$m = 1,\ n = 2$ 
$\gamma = 31.839 \longrightarrow$ $n = 3$ 	$m = 2,\ n = 3$ 
$\gamma = 70.060 \longrightarrow$ $n = 4$ 	$m = 3,\ n = 4$ 

CHAPTER 4
Numerical Simulations of the Model on a Unit Sphere

As discussed in Section 3.1, the finite element method is a technique used to solve many different types of partial differential equations on a discretized domain. Here we will discuss the simulation results of the model on a unit sphere. We will examine the well-posedness, consistency, convergence, and stability of our model system (3.22) on the surface of a sphere.

4.1 The Spherical Domain

Before examining the measures of error for consistency, convergence, and stability, it is useful to establish some details about the domain in question. We will examine an arbitrary pattern on the surface of the sphere for various spatial and time step intervals. Each mesh was created using *deal.ii*'s mesh library. In addition, we will be using solution values for u rather than v for the following analyses. The IMEX scheme used grants significantly more accuracy to v, so for fairness we will measure the system based on the maximum error in u.

The mesh itself has a base shape of a cube, and can be refined an arbitrary number of times according to a spherical manifold. The color scheme of the visualizations assigns light green to concentration values of u (FGF10) and dark purple to concentration values of v (SHH). Upon each refinement, the number of cells in the mesh quadruple, and the maximum diameter among all the cells, h, reduces by about half. We will use h_2 to denote the maximum diameter of a given cell after two refinements from a cube, h_3 for three refinements, et. cetera. Figure 4.1 below shows four mesh densities for reference, while Table 4.1 details the numerical information on each mesh, including number of cells and degrees of freedom. Not pictured is the h_6 sphere, which will only be used as a domain for an exact solution facsimile. The images shown in other sections use h_3 spheres for visualizing patterns.

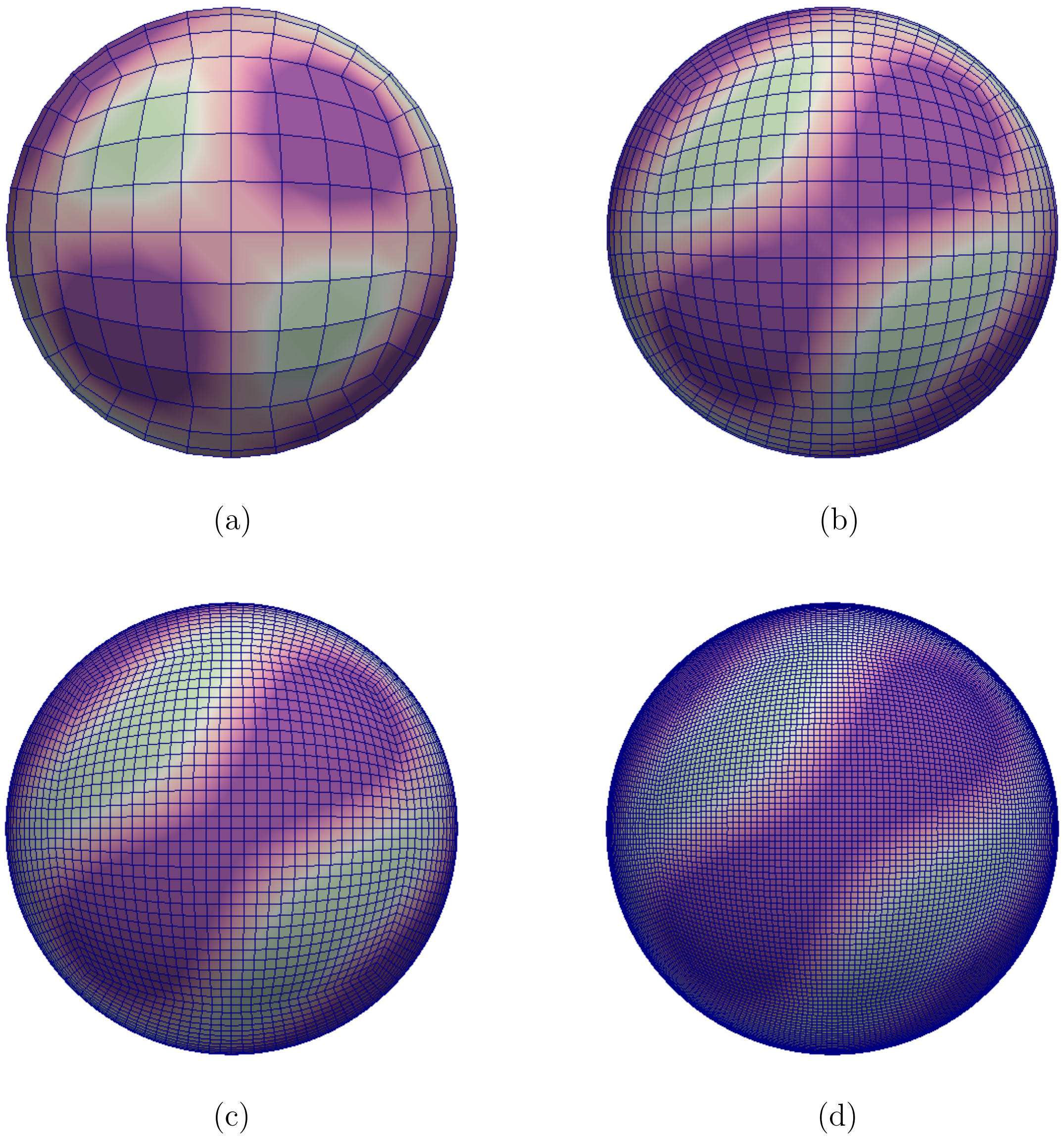


Figure 4.1. Sphere model meshes with various cell densities. (a) h_2, **(b)** h_3, **(c)** h_4, **(d)** h_5.

Table 4.1. Sphere Model Mesh Density Values

	h_2	h_3	h_4	h_5	h_6
Number of Cells	96	384	1,536	6,144	24,576
Degrees of Freedom	386	1538	6,146	24,578	98,306
Maximum Diameter	0.541196	0.277048	0.139239	0.0697337	0.034877

Some visualizations to follow only show data up to $t = 0.5$. It is important to establish that data collected up to this point is sufficient to show steady-state dynamics. To do this, we will observe a standard L_2 norm at $t = 1$, as follows:

$$||u_{N+1} - u_N|| = \sqrt{\sum_{i}^{\dim u} (u_{N+1} - u_N)^2} \tag{4.1}$$

Figure 4.2 shows the differences in L_2 norms for varying mesh densities, each taken using a timestep length of 10^{-3}. The norm takes the difference between time steps $u_{N+1} - u_N$ as the argument, where $i = 1, 2, 3, ... \dim u$, with $\dim u$ being the total number of elements in the solution vector representing u. We can see that as the mesh density increases (or, as the diameter h halves), the steady state norm value decreases by about half.

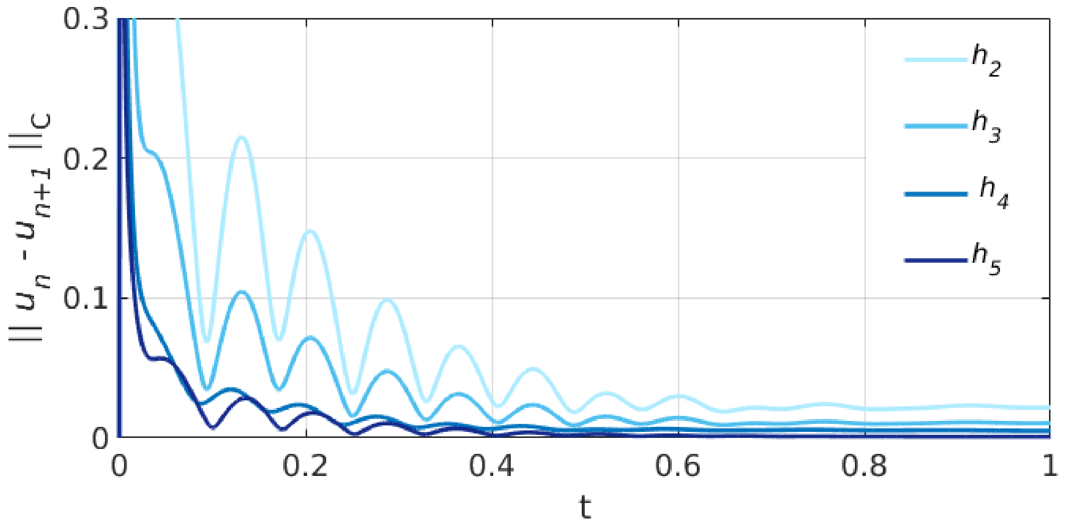


Figure 4.2. Numeric Scheme L_2 norm using time step 10^{-3} for spheres of various mesh densities.

4.2 Well-Posedness and Stability

We can define a well-posed numerical scheme as having three general characteristics [34]:

1. A solution for the problem exists
2. The solution is unique
3. The solution changes continuously as the boundary values and initial conditions change.

We can show existence and uniqueness, which is already established for FEM nonlinear parabolic reaction-diffusion systems [10]. We have already stated that the linear system $\mathscr{A}\mathbf{x} = \mathscr{b}$ defined in Equation (3.23) consists of positive definite matrices. Our matrices are also uniform; the *deal.ii* algorithm sparsity pattern is designed to have the same number of entries for each row. For a linear system with uniformly positive definite matrices, we are guaranteed both existence and uniqueness [9]. Therefore, we know that there is a unique solution to the linear system stated in Equation (3.22).

The third requirement for well-posedness is more challenging to show, and will not be proved here. Note that each timestep of the model linear system is time invariant, which indicates the system is well-posed in general [4]. We rely on proofs formulated in literature to verify that the solution indeed changes continuously [9, 10, 11]. We can now state with confidence that the linear system is well-posed.

Stability demands that the solution is persistent, that is, small perturbations or errors (such as round-off errors) in the data disappear over time. In addition, it implies that a change of the initial and boundary data leads to a comparable change in the numerical solution. Since the model linear system is well-posed, the solution depends continuously on the input. For a discrete scheme, this continuous dependence is analogous to stability [4]. However, a quantitative measure of stability was not found. We discuss more about the conditional stability of the system in Section 4.5.

4.3 Consistency

Consistency in a model scheme requires that when mesh cell size and time step size decrease, the truncation error approaches zero. In other words, the discrete system should be a "good" approximation of the partial differential equation system. For nonlinear time dependent PDEs solved by the finite element method, we must measure consistency by examining the local truncation error for the linear system [7]. For the system in question, we solve each time step via a linear equation. Recall the linear system we set up using the FEM-IMEX scheme:

$$\mathscr{A}\mathbf{x} = \mathscr{b} \tag{4.2}$$

Here $\mathbf{x} = (u^{N+1}, v^{N+1})^T$, and $\mathscr{b}$ is a function of (u^N, v^N). To find the truncation error, we apply a simple check after solving the system:

$$\text{Truncation error} = ||\mathscr{b} - \mathscr{A}\mathbf{x}|| \tag{4.3}$$

The truncation error is already an integral part of the algorithm for solving the linear system (in this case, the conjugate gradient method). We can apply a constraint that forces the solver to continue enumerating until the solution gives a truncation error below the desired amount. This feature also allows us to easily control the accuracy to order $p = (p_1, p_2)$, so long as we constrain the truncation error as being less than both $\mathcal{O}(k^{p_1})$ and $\mathcal{O}(h^{p_2})$. We define order p by the following inequality:

$$||\mathcal{b} - \mathcal{A}\mathbf{x}|| \lessapprox h^{p_1} + k^{p_2} \tag{4.4}$$

We can then say that the numerical scheme is consistent if it is accurate of order $p = (p_1, p_2) > 0$. For this analysis, we constrained the conjugate gradient algorithm to iterate until the truncation error was less than 10^{-20}. Therefore, we can achieve an arbitrarily high accuracy order so long as we are willing to sacrifice the computing time needed to achieve it. Since we keep the truncation error at 10^{-20} for all computational runs, all permutations of h and k in the proceeding analysis are consistent with an order of accuracy of at least (2,2). It may be prudent to note, however, that regardless of initial conditions, consistent schemes never exceeded 200 conjugate gradient iterations.

4.4 Convergence

While consistency examines the discretization of a system, convergence implies that the *solution* to the discrete system is a "good" approximation to the *solution* of the partial differential equation system. Now, we cannot measure convergence in the traditional way, because it requires knowledge of the exact solution. The exact solution to the Schnakenberg system on the surface of a sphere is unknown. However, we can create a facsimile of the exact solution by producing an output using the smallest time step that is computationally reasonable.

If we call this exact solution facsimile W_E while our approximate solution is W_h, then we can say that the solution converges given that the following criteria is satisfied:

$$\Big|\Big| W_h - W_E \Big|\Big| \lessapprox h^{q_1} + k^{q_2} \tag{4.5}$$

However, because of computational limitations, it was not feasible to calculate the difference $W_h - W_E$ for each time step. Instead, we examined the convergence rate based on the following inequality:

$$\Big| \, ||W_h|| - ||W_E|| \, \Big| \leq \Big| \, ||W_h - W_E|| \, \Big| \lessapprox |h^{q_1} + k^{q_2}| = h^{q_1} + k^{q_2} \tag{4.6}$$

Using $\big| \, ||W_h|| - ||W_E|| \, \big|$ is a valid measure so long as $q = (q_1, q_2) > 0$, or the solution converges at rate q. The norms are the same as the one defined in (4.1). We see this graphically in Figure 4.3 below, which shows how the convergence rates change for varying h and k. Note that each successive graph (from right to left) is zoomed in by a factor of 10 for the y-axis.

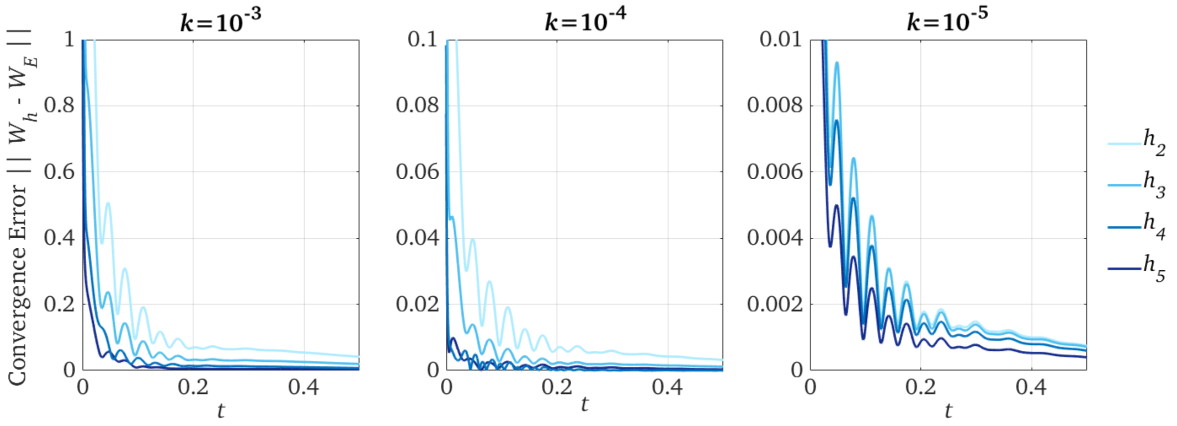


Figure 4.3. Numeric Scheme convergence norms over time for varying spatial discretization (h) and time step (k) units

Convergence norm values are shown in Table 4.2 below, for various values of h and k. Table 4.3 shows the maximum convergence rate for each pair of values (h, k). We can confidently say that all models converge to some degree. Note that convergence rates steadily increase as k decreases.

Table 4.2. Numeric Scheme Convergence Norm Values For Various Spatial and Temporal Parameters

	h_5	h_4	h_3	h_2
$k = 10^{-3}$	1.9e-3	1.3e-3	5.0e-3	1.0e-2
$k = 10^{-4}$	1.8e-4	1.2e-4	4.9e-4	1.0e-3
$k = 10^{-5}$	1.8e-5	1.2e-5	4.9e-5	1.0e-4

Table 4.3. Numeric Scheme Convergence Rates For Various For Various Spatial and Temporal Parameters

	h_5	h_4	h_3	h_2
$k = 10^{-3}$	(10, 10)	(5, 5)	(2, 2)	(1, 1)
$k = 10^{-4}$	(14, 14)	(7, 7)	(3, 3)	(2, 2)
$k = 10^{-5}$	(17, 17)	(8, 8)	(5, 5)	(3, 3)

4.5 Simulation Errors

Each simulation using the above-mentioned time steps and mesh densities produced solutions in line with expected value intervals. However, a problem arises when we come across a scheme that is not consistent, like when solution values for the linear system for a single time step approached infinity when using the conjugate gradient method. This was the case for h_2 using a time step of $k = 10^{-2}$, which indicated that the scheme is only conditionally stable. Indeed, we found that when solving the system on more complex domains, a minimum time step of approximately $k = 10^{-3}$ was needed to endure consistency and thus stability.

Despite the uncertainty that follows such results and the absence of a rigorous proof to precisely quantify stability conditions, we can still move forward with using the FEM-IMEX scheme to produce patterns on the surface of a lung mesh. It is quite easy to tell if there is a consistency failure without sacrificing much computing time. We can examine the truncation error of the linear system over a small handful of iterations (say, 10) if more than 100 iterations are needed. If the average norm between two successive iterations continues to increase, the process can be halted and the timestep restarted using a smaller k value. There were not any solutions produced that were outside a reasonable range of values (between -2 and 2) for any simulations, indicating that so long as the linear system was solvable, the solution is accurate to some degree.

This presents challenges when considering the time step adaptation method presented in Section 3.3. We found that when using such a method with high values of γ (usually over 300 for solving on the lung, and over 700 for solving on a sphere), the solution tended toward infinity sometime after $t = 1$. It was then necessary to limit the k value between 10^{-3} and 10^{-2} to prevent such errors, however this gave a significantly smaller computational reward. More time and further study could yield a much deeper experimental analysis of this scheme, but for this thesis, we will continue to the results.

CHAPTER 5
Fibroblast Growth Factor 10 Expression On a Growing Domain

The following will detail how we varied the parameter γ to mimic the pattern formation that occurs on a growing domain. First, we examined the relationship between pattern emergence and domain size in spheres. We found that the predicted pattern for larger domains could be induced by scaling γ by a factor dependent on the square of the desired radius. We applied the same reasoning to the lung mesh, and induced increasingly complex patterns by scaling the γ parameter.

5.1 Pattern Replication On a Sphere

Before solving on the surface of the lung, we study the relationship between the scaling parameter γ and sphere radius r. Modeling this relationship validates the efficacy of γ in simulating a growing domain. This entails solving the model system several times on a variety of sphere surfaces.

Recall the analytic results when solving the Laplace-Beltrami eigenfunction on the surface of a sphere. We concluded that $k^2 = n(n+1)$, with n being the degree of the Legendre equation. However, increasing the radius of the domain increases the wavenumber interval by an r^2 scale factor. Our experimental results have been consistent with this relationship. As the radius increases, smaller γ values are needed to achieve the same patterns. Recall Equation (2.23). We can now introduce the influence of the radius to give us:

$$r^2k^2 = r^2\gamma \cdot \frac{1}{2\delta}\left[\delta f_u + g_v \pm \sqrt{(\delta f_u + g_v)^2 - 4\delta(f_u g_v - f_v g_u)}\right] \tag{5.1}$$

Where $r^2\gamma$ will be referred to the k-scaling term. An example of this relationship is as follows: Say we have a sphere of radius r_{start}, and would like to replicate some wavenumber k^2 pattern on a sphere of a radius r_{end}. Initially using some value, γ_{start}, we would then need to find a new γ_{end} such that the k-scaling term remains the same:

$$r^2_{\text{end}}\gamma_{\text{end}} = r^2_{\text{start}}\gamma_{\text{start}} \Rightarrow \gamma_{\text{end}} = \gamma_{\text{start}}\frac{1}{s^2} \tag{5.2}$$

With s as the scale factor from r_{start} to r_{end}. In practice, it is useful to use larger spheres for pattern replication. We have found that the transition between patterns is more fluid when we examine a smaller γ range using decreased partition amounts on larger spheres. In addition, there is less oscillation when mapping the L_2 norm on larger spheres (see Figure 5.1). Therefore, our simulations moving forward will use larger domains to simulate the growth from small to large domains. As an example, we used the following procedure to model a growing sphere:

1. Choose the starting model parameters using r_{start} and γ_{start}.
2. Reproduce the starting pattern on a larger radius r_{end} by plugging r_{end} into the pattern replication formula (5.2). From this, obtain γ_{end}.
3. Model the solutions on the larger sphere from γ_{end} to γ_{start}.

We can verify this experimentally by simulating an initial pattern, replicating it on a larger domain, and modeling over the resulting γ interval. For the following models, we allowed sufficient time for the iterations to reach a steady state. Figure 5.1 shows the normal difference sum for three radii, each at $\gamma = 100$. In each case, $t = 10$ was sufficient, as we observed that for any value $t > 10$,

$$\sqrt{\sum_{n\in\mathbb{T}}(u_n - u_{n-1})^2} \ < \ 10^{-3}$$

Note that this difference norm does not imply convergence to a single value, but rather to an oscillating steady state. This is in line with results from literature, and consistent with the behavior of a chemical feedback loop [22, 29, 30].

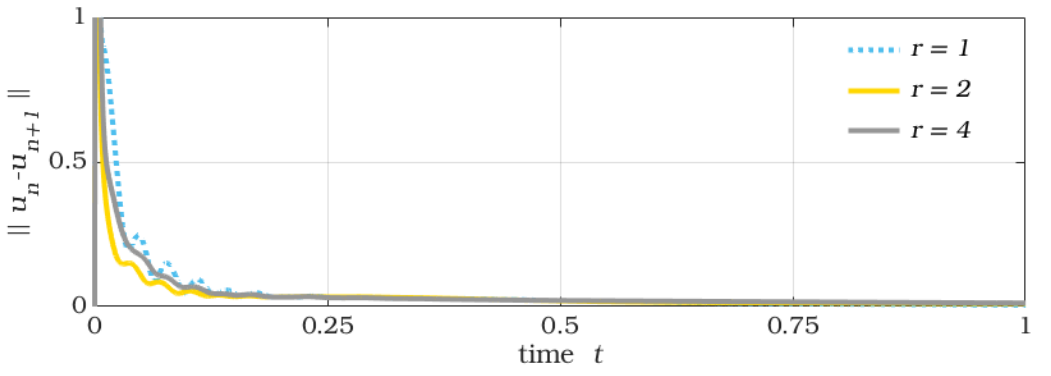
We now have the tools to demonstrate pattern replication. First, we show the results of reproducing a pattern on spheres of different radii in Figure 5.2. Using $r_0 = 0.5$, $r_1 = 1$, and $r_2 = 2$ we were able to reproduce a $k^2 = 20$ pattern by varying γ. Note that the results approximately align with our findings when using r_1 as the reference measurement:

$$\gamma_0 = 160 \approx 41\frac{r_1^2}{r_0^2} \approx 164, \qquad \gamma_1 = 41.98 \approx 41 \qquad \gamma_2 = 11.098 \approx 41\frac{r_1^2}{r_2^2} \approx 10.25 \quad (5.3)$$

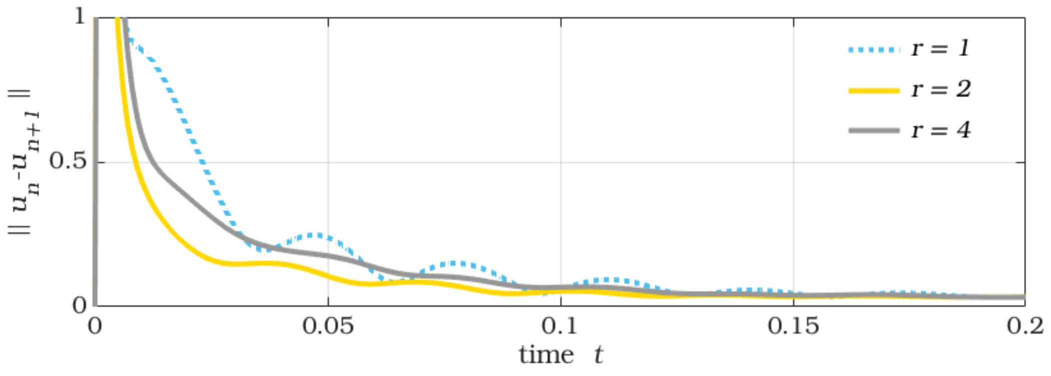
Fitting γ and r observational data to a power curve of the form $\gamma = ar^b$ becomes a reasonable estimate for (5.2). Figure 5.3 and Table 5.1 below show the data points collected for the $k^2 = 20$ pattern. Each data point was recorded based on the best fit solution. The best fit is considered the solution value interval with the smallest range difference from the r_1 model. Such a criteria is necessary to avoid "cherry picking" our results. We found the resulting power model to be useful:

$$\gamma = 41.06r^{-1.963} \qquad (5.4)$$

This model corresponds well with enough with (5.2) that we can proceed confidently with simulating growth. Note that the scaling factor s is with respect to measurements taken at $r = 1$. This model has a sum of square errors value of 3.124 with a 95% confidence interval. We can now reproduce growth on a static sphere.



(a)



(b)

Figure 5.1. Difference norms for FGF10 on spheres of radii 1, 2, and 4, with $\gamma = 100$. (a) View of L_2 norm up to $t = 1$. (b) Close-up showing differences in oscillation of solutions.

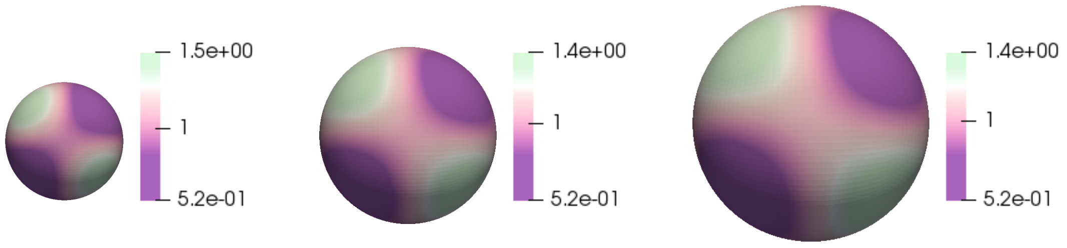


Figure 5.2. Reproduction of a $k^2 = 20$ pattern solution for FGF10 on different-sized spheres. (a) $r = 0.5$ and $\gamma = 160$, (b) $r = 1$ and $\gamma = 41.98$, (c) $r = 2$ and $\gamma = 11.10$.

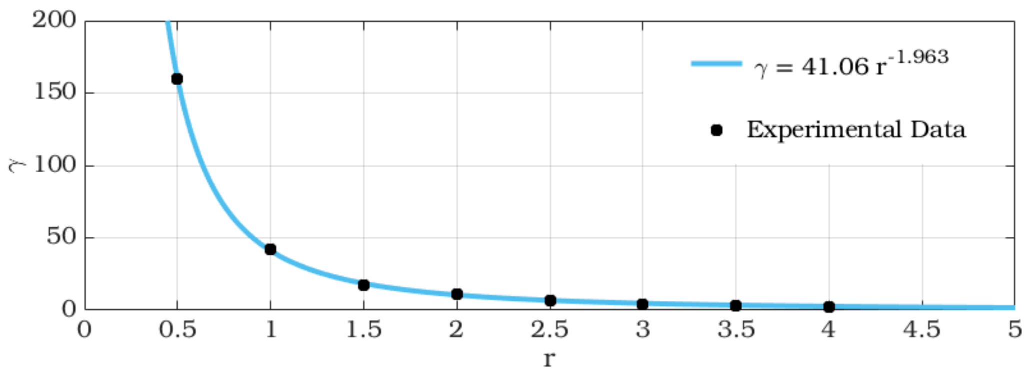


Figure 5.3. Simulation data fit to the power model $\gamma = ar^b$ from x data points

Table 5.1. Experimental and Analytic γ For the $k^2 = 20$ Pattern On Spheres

Radius	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
Experimental γ	160	41.98	17.50	11.10	6.50	4.00	3.20	2.31
Analytic γ	164	41	18.22	10.25	6.56	4.56	3.90	2.56

5.2 Growing Domain Models

To model a growing domain, we use techniques similar to those which replicated a pattern on domains of varying sizes. First, we give the starting and ending results of the growth we wish to simulate. Figure 5.4 shows two spheres with $r = 1$ and $r = 2$, both solved using the same parameters. Our goal is to induce the pattern from the $r = 1$ sphere onto the $r = 2$ sphere, then compare the γ values on each.

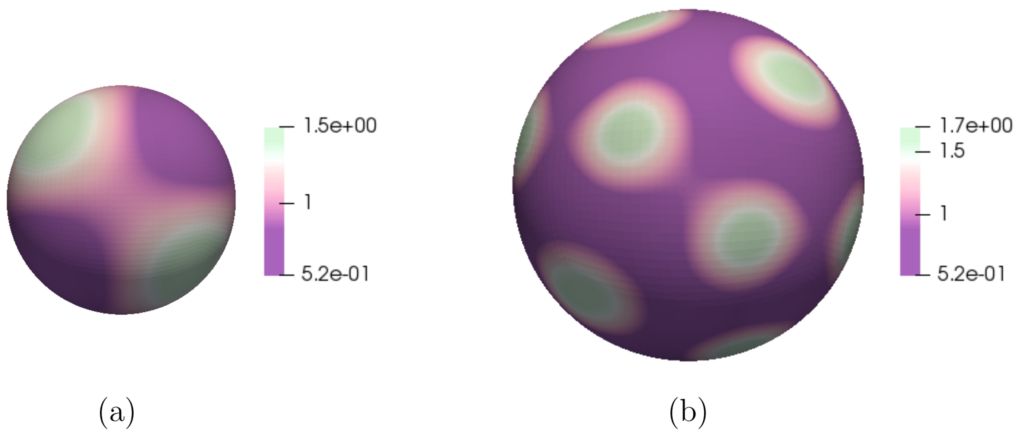


Figure 5.4. Starting and ending u solution patterns for growing domain simulation using $\alpha = 0.1$, $\beta = 0.9$, $\delta = 10$, and $\gamma = 40$. (a) $r = 1$, (b) $r = 2$.

Plugging in the parameters $r_{\text{Start}} = 1$, $\gamma_{\text{start}} = 40$, and $r_{\text{end}} = 2$ into (5.2), we find that the $k^2 = 20$ pattern is replicated by using $\gamma = 10$ on the larger sphere. Now we can compare the patterns that emerge on a sphere using $\gamma = 40$ growing from $r = 1$ to $r = 2$ with the patterns that emerge on a sphere of radius 2 and $\gamma = 10$ to 40.

First we will solve on spheres ranging from $r = 1$ to $r = 2$ using $\gamma = 40$. Then will partition the γ interval (10, 40) and observe the pattern found. Finally we will compare the images side-by-side, choosing patterns that best fit near their counterparts. Figure 5.5 at the end of this section shows the results of this comparison. Each pair is matched with the size-γ formula from (5.2).

The results of the simulation are interesting. While the patterns have a clear correspondence, the third and fourth have a rotational difference. The mathematical reasoning behind the rotations is not clear, but further study may give insight into this phenomenon. For now, we can state that pattern replication is successful. A supplement to this thesis is a movie (sphere-movie.avi) showing the pattern evolution of a sphere with a radius of 2 and a γ range of 0 to 100, in intervals of approximately 0.2. It is included simply to show the smooth evolution between patterns, and how the pattern chosen for these figures were ones that persisted through several values of γ.

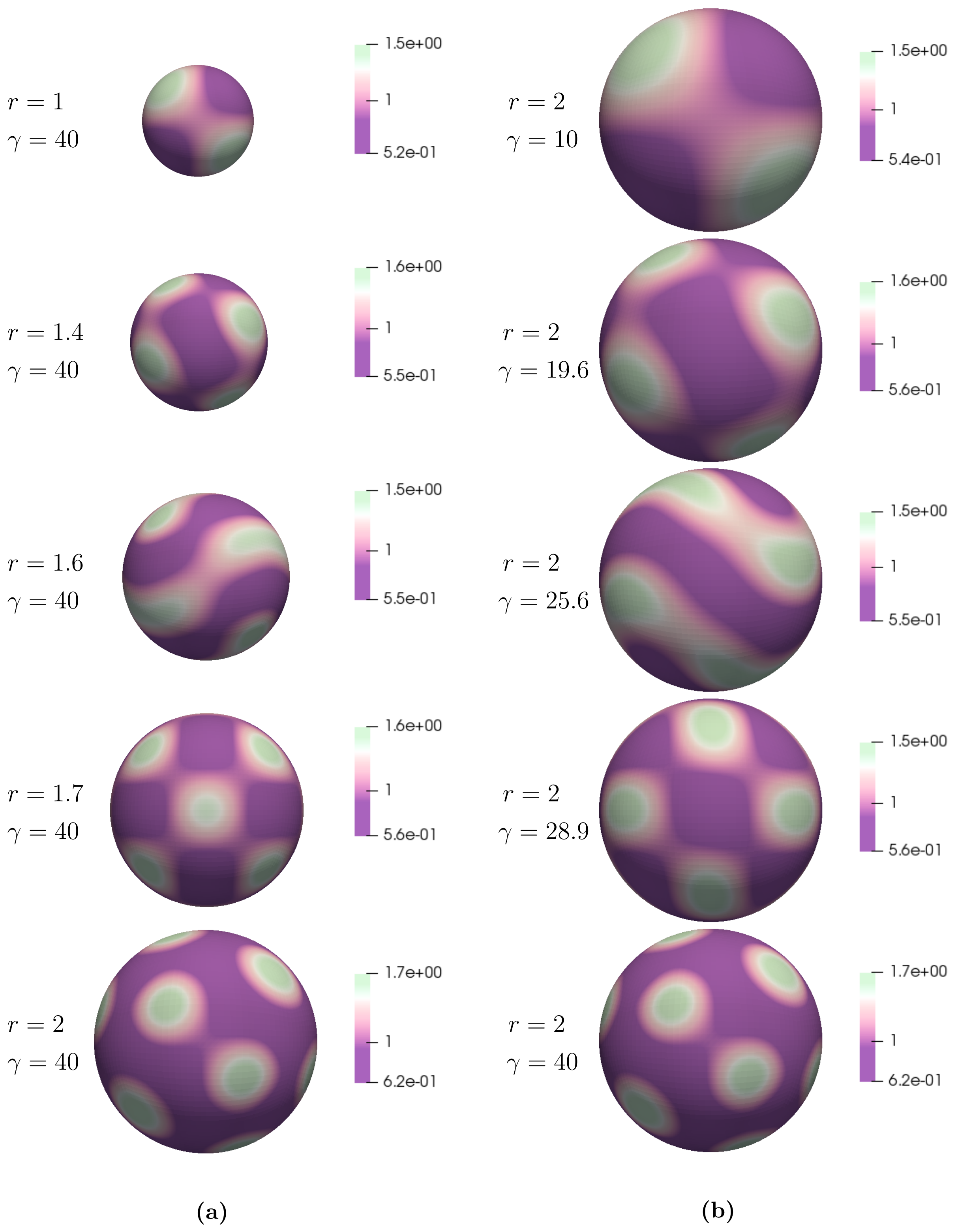


Figure 5.5. Comparison of patterns on a growing and static sphere. (a) Patterns on a growing sphere with the radius increasing from $r = 1$ to $r = 2$, while keeping γ constant. (b) Patterns on a static sphere with γ increasing from $\gamma = 10$ to $\gamma = 40$, while keeping the radius constant

5.3 Lung Mesh Geometry

For the applied simulation, we created a lung mesh from confocal microscopy images processed with Simpleware® and edited with Blender®. The images consisted of 88 slices from a murine lung at an E12.5 development stage, or the pseudoglandular stage. We used a voxel depth of 4.770 microns between each slice. The volume of the lung is approximately 3.52835cm^3, which is consistent with research on mouse lung development [6].

The discretized mesh had a default triangulated surface consisting of triangles. Using the Catmull-Clark division routine, each triangle was sub-divided into 3 quadrilaterals for use with *deal.II*. Figure 5.6 shows a simple diagram of how each cell was subdivided. During this refinement, no additional curvature was implemented, as the complexity of the lung surface made it impractical to define a custom curved manifold. Therefore, each group of 3 quadrilaterals (subdivided from a single triangle) share a plane, and the error calculations remain the same regardless of mesh refinement using this method. Despite this, simulations indicated that the mesh refinement was adequate to get good results.

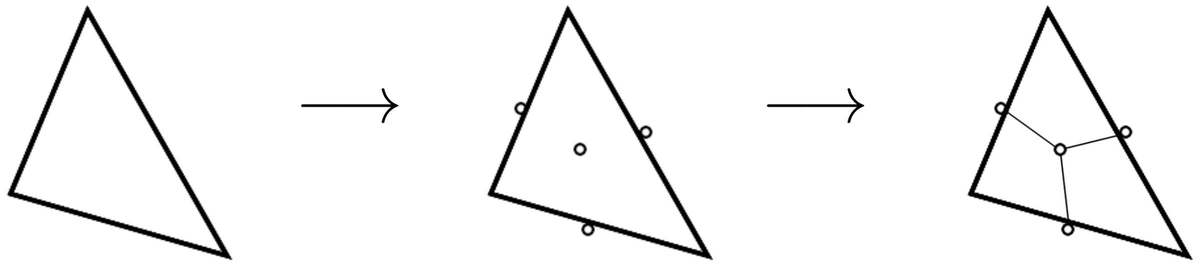


Figure 5.6. Approximate representation of how the lung mesh triangulation surface is subdivided using the Catmull-Clark method.

Figure 5.7 shows the total lung mesh data from the confocal images. For this thesis, only the surface mash data was utilized. We extracted the interior structure and solved on the lung surface only. This mesh contains 63,604 active cells, with 254,416 degrees of freedom. While a more detailed mesh could have been extracted from the confocal images (containing upwards of 1,000,000 cells), the analytic advantage was minimal and the computational disadvantage was significant. No major features were lost when converting the confocal images to a workable mesh. In fact, some noise was removed manually to provide a smoother, more cohesive surface. The final mesh had a maximum diameter length of 0.0435035 cm.

For the model simulations, we used the original mesh and 3 additional enlargements. These enlargements were scaled by a factor of 2, 3, and 4 times the volume from the original size. Each enlargement had the same number of cells and degrees of freedom, and there was no refinement used. Also, the mesh was closed to allow a smooth solution without the need for additional boundary conditions. Such boundary conditions would likely only affect the area near the initial bronchial split. It is reasonable to assume that the pattern formation that occurs in that area is not well matched to biological phenomena, so we chose to negate that area from analysis and focus more on the larger lung surface. Simulations solving the model system on the surface of the mesh is discussed more later in this chapter.

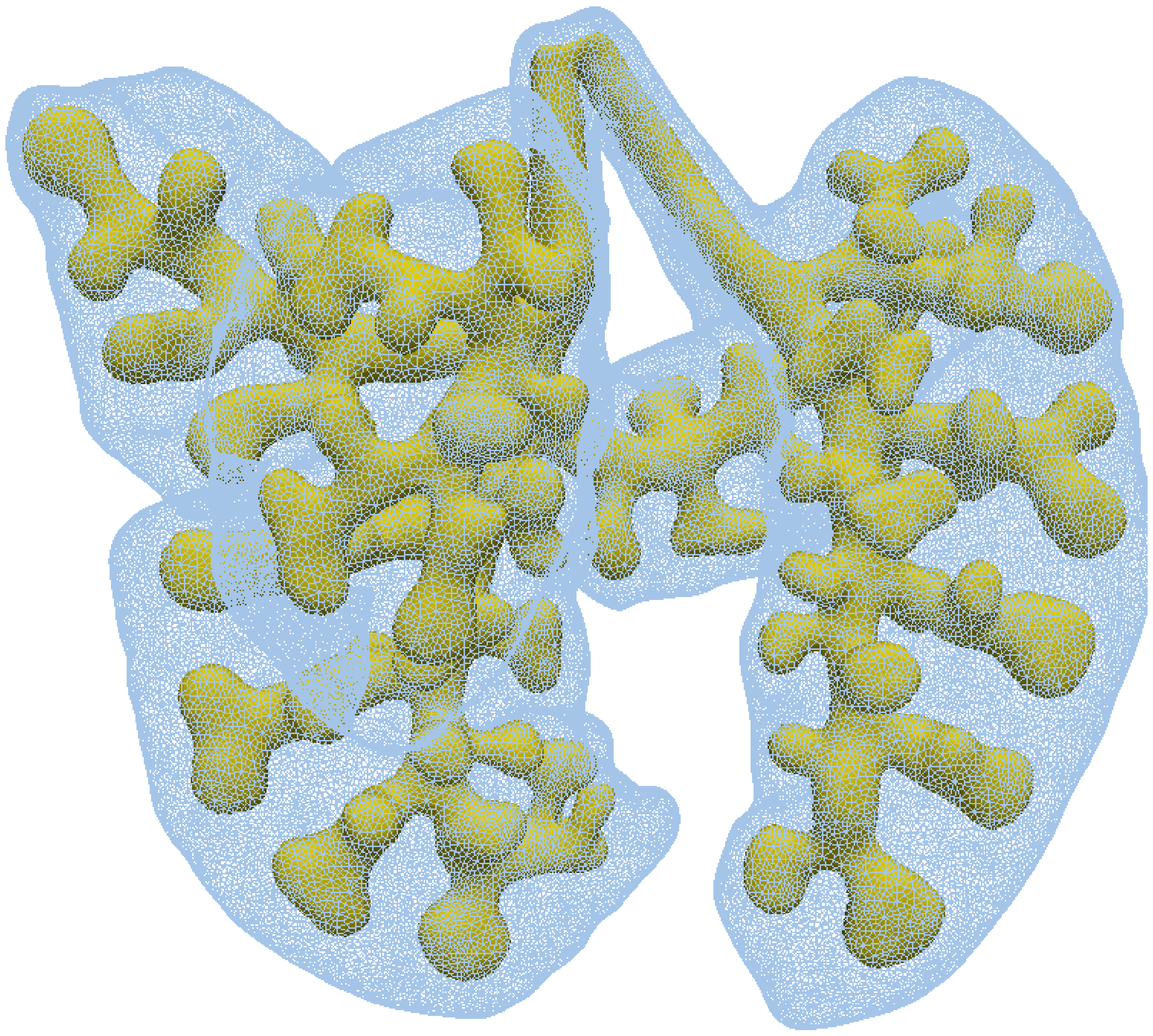


Figure 5.7. Full view of the murine lung mesh used for this model, including interior epithelial branch structure.

5.4 FGF10 Patterns on a Growing Lung

While our simulated growth model on a sphere was reasonably successful, translating those ideas to the lung surface proves more difficult. Since we cannot provide rigorous analytic reasoning specific to simulating a complex domain such as a lung mesh, we must rely on our findings from studying a sphere. Based on the success of that model, we can reasonably say that increasing the γ parameter on the lung may approximate the effects of a growing domain. While we do not have the computational resources to simulate patterns on the lung to the same extent as on a sphere, we can still observe the general behavior of the model solution on a variety of domain sizes.

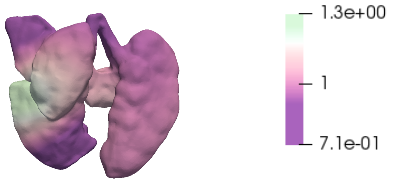
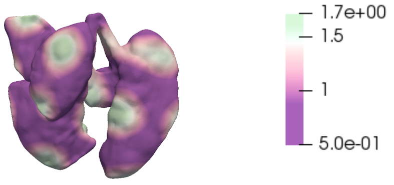
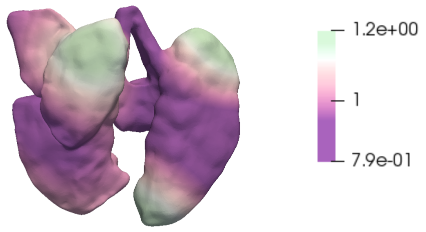
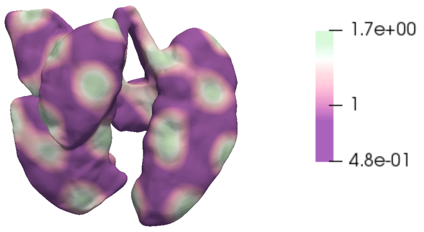
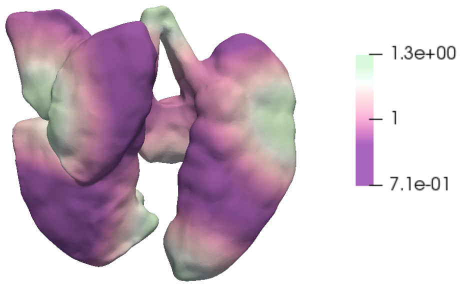
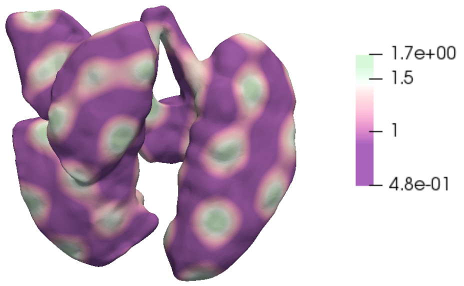
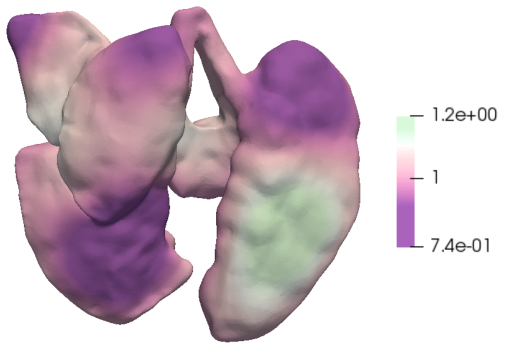
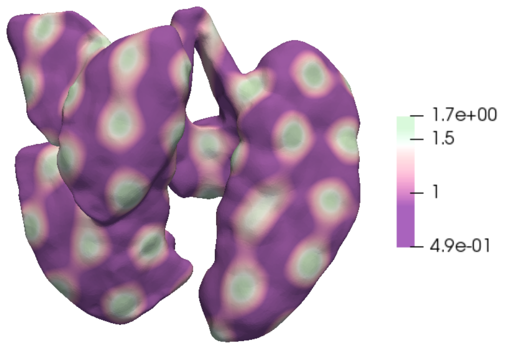
Initial simulations produced working results displaying an increased pattern complexity as γ increased. The parameters $\alpha = 0.1$, $\beta = 0.9$, and $\delta = 10$ were used for initial experimentation. Recall that u (light green) represents FGF10, while v (dark purple) represents SHH. Table 5.2 shows the corresponding patterns for varying values of γ on different sized meshed. Notice that while the exact pattern is not duplicated, the number of FGF10 concentrated areas consistently increases with high γ values.

After several simulations, we were able to loosely correlate pattern replication. Figure 5.8 shows a few examples of these similarities. Again, there were no close pattern matches, but similar levels of pattern complexity were observed. For instance, in the small-lung simulations, when $\gamma = 200$ on the 1x lung and $\gamma = 75$ on the 4x lung, we see similar color blocking, with FGF10 and SHH being expressed in comparable amounts across large areas of the lung surface.

Despite some similarities, there is not enough correlation to fit a model curve in the same way that we did for patterns on the spheres. One reason for the discrepancy could be that when each mesh was scaled, the origin was used as the center point, rather than the center of the lung mesh. This might explain why it is simple to find similar pattern complexity, yet difficult to find actual matching patterns. The position of the mesh in Cartesian space may be the key to accurate pattern replication; yet there may not be any discernible difference when examining the qualitative properties of the solutions. Regardless, the amount of time and computational resources needed to examine the influence of the mesh center in space is beyond the scope of this thesis.

Despite a lack of accurate pattern replication between lung meshed of different sizes, simply noting that increasing γ produces pattern complexity modes similar to that of an increasing domain is sufficient to address real-world applications. The next chapter will compare these patterns to images of real lungs, and discuss possible uses for our model.

Table 5.2. Patterns on Lungs of Varying Sizes ($\alpha = 0.1$, $\beta = 0.9$, and $\delta = 10$)

Scale	$\gamma = 50$	$\gamma = 500$
1x		
2x		
3x		
4x		

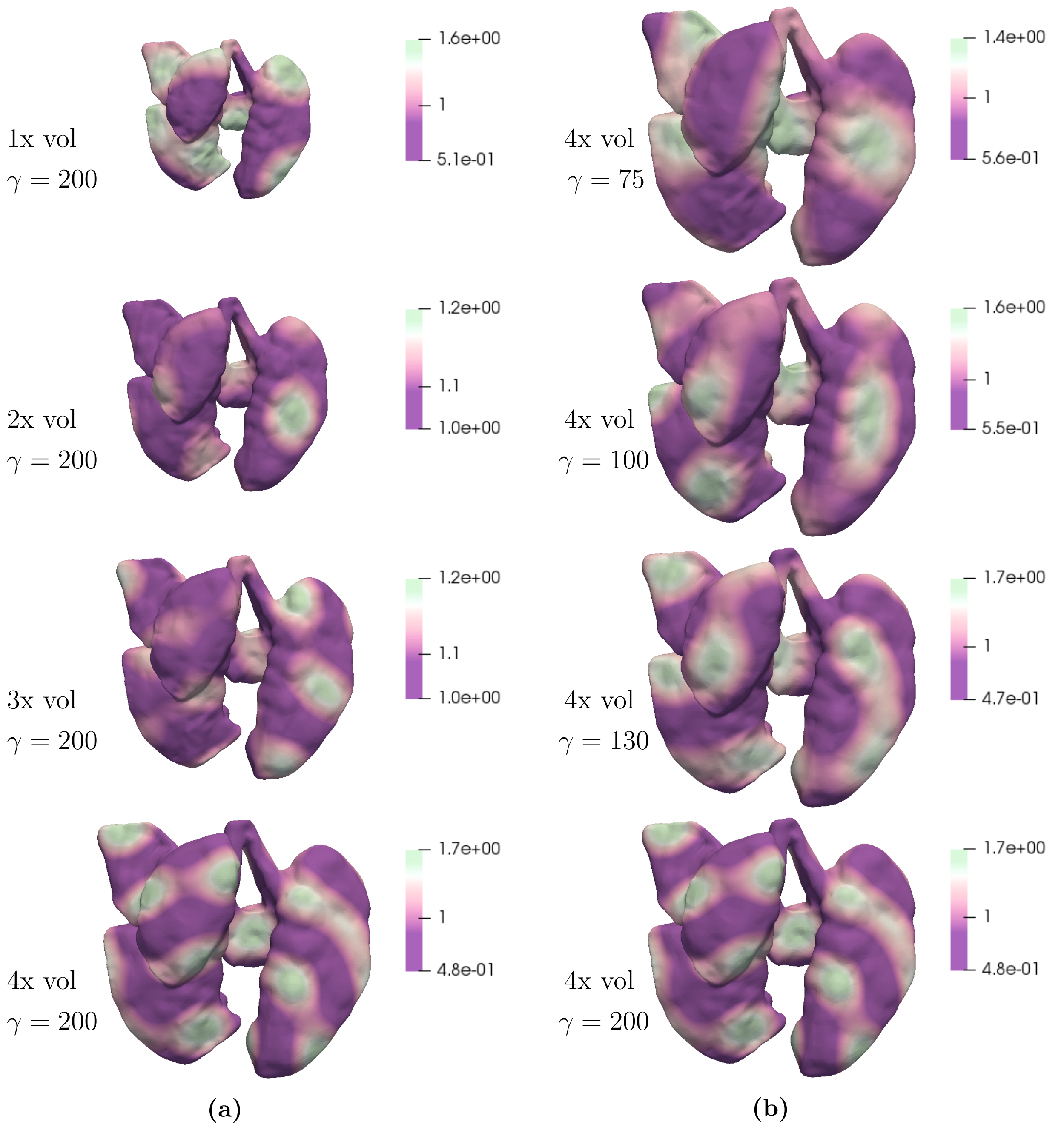


Figure 5.8. Pattern correlation between lungs using parameters $\alpha = 0.1$, $\beta = 0.9$, and $\delta = 10$. (a) Varying lung size from 1x to 4x volume with γ fixed at 200. (b) Fixed lung size at 4x volume with γ varying from 75 to 200.

CHAPTER 6
Murine Lung Applications

We have established several important ideas that validate the study of solving the model system on the surface of the lung. The Turing analysis and eigenfunction plots on a sphere showed that the model is well-posed. The numeric discretization schemes for time and space were shown to be conditionally stable and able to mimic analytic solutions. The emergence of increasingly complex patterns was preceded by either a larger domain or a larger γ value, leading us to conclude that an increase in γ produces pattern solutions comparable to the solutions on an increased domain. Now we can examine empirical data on phases of lung development that correlate to the FGF10 distribution modeled previously.

6.1 Initial Conditions

For Turing analysis, the combination of parameter values is less important than the patterns excited, since different parameter values can excite the same pattern. We will therefore use the same parameters that have been used for simulations throughout the previous chapters: $\alpha = 0.1$, $\beta = 0.9$, and $\delta = 10$. As usual, we will vary γ to excite different patterns.

It is still useful to discuss some details of the dimensionalized parameters and their impact on the model as a whole. Recall the chemical equation and resulting nondimensionalized γ term introduced in Sections 2.1 and 2.2:

$$X \underset{\mathrm{k}_2}{\overset{\mathrm{k}_1}{\rightleftharpoons}} \mathrm{F} \qquad 2\mathrm{F} + \mathrm{S} \xrightarrow{\mathrm{k}_3} 3\,\mathrm{F} \qquad Y \xrightarrow{\mathrm{k}_4} \mathrm{S} \qquad \text{and} \qquad \gamma = \frac{1}{k_2}$$

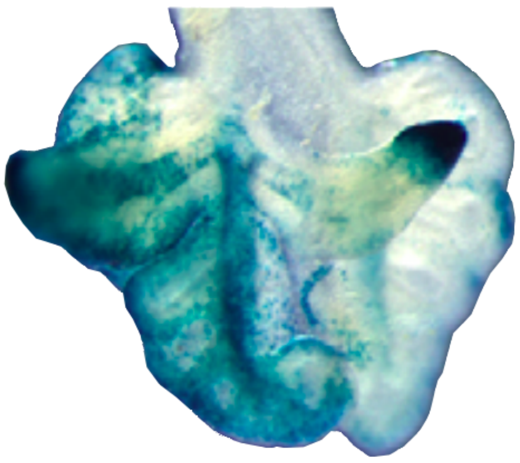
We have already established that for a growing domain simulated on a static domain, we increase γ. Note that the term $\gamma = 1/k_2$ indicates that the presence of the FGF10 precursor substrate X is decreasing with time, thus γ increases with time. This is the expected behavior for biological system development, as growth naturally slows and new branch development eventually halts.

6.2 Modeling Stages of Development

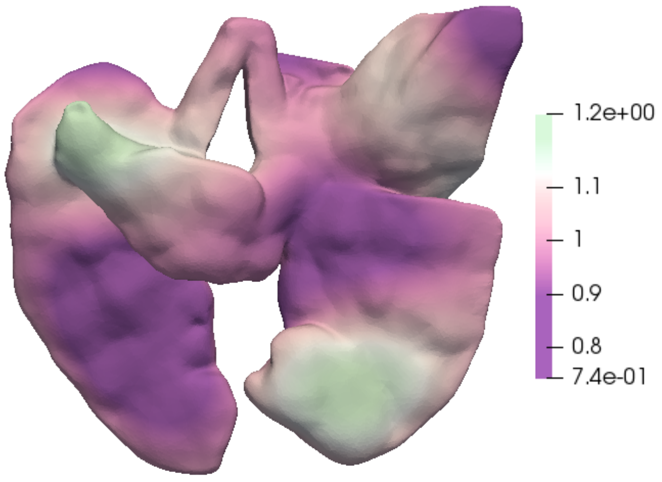
Finally, we reach the culmination of this work: will these simulated patterns have relevance to real-life applications? So far we have looked at images of only one side of the lung. To better compare results, we will now use the opposite side of the lung, which has a more complex structure. First, we examine the early stages of murine lung growth, when FGF10 accumulates at the tips of the middle and post-caval lobes. Such gene distribution has been observed experimentally several times [18, 6, 20]. Figure 6.1a and 6.1b show both the experimental murine lung image and the lung model simulation at an approximate development stage of E12.5, or about 12.5 days, during the early pseudoglandular stage (this occurs at around 30 days in human development). Our simulation shows similar FGF10 accumulation around the post-caval lobe, but not the middle lobe. This is the development stage where rapid growth occurs due to high concentrations of FGF10. Note that the images appear anatomically mirrored. We believe that when forming the mesh, the stack of confocal images may have been photographed upside-down. Rather than mirror the images, we show the original content as-is.

By E13.5, several branching structures have formed. Figure 6.1c and 6.1d shows the experimental and simulated results for this development period. Notice that both images have striated distributions of FGF10 radiating outward and upward from the main bronchial tubes. The "best match" simulation was chosen visually, with the intent to match both the number of striations and the number of FGF10 concentrated areas per striation.

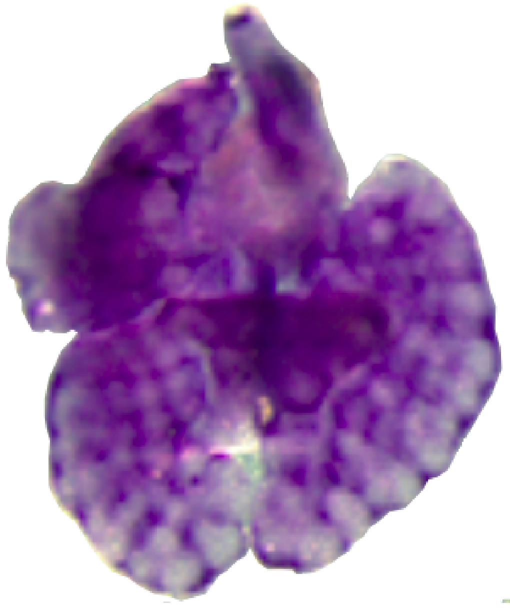
Now we can implement our new technique to study what FGF10 distribution may look like between E12.5 and E13.5. Although our previous simulations are not an exact match, observing how the pattern changes may provide insight into murine lung development. Future experimentation and more thorough simulations may provide more relevant insight. For now, we can observe the theoretical migration of FGF10 about the surface of the lung. Figure 6.2 and 6.3 shows how varying γ changes the surface pattern from what is observed at the E12.5 simulation to what is observed at the E13.5 simulation. We can see the gradual increase in complexity, and such changes inspire us to question how these concentration distributions morph and multiply. From the images, it seems that there is a cycle of concentration areas coming together to form striation patterns, then splitting apart again into a greater number of concentration areas. More on this is discussed in the next section.



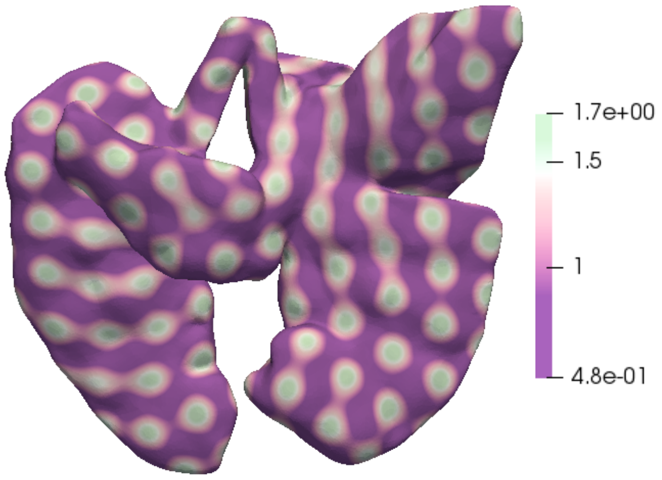
(a)



(b)

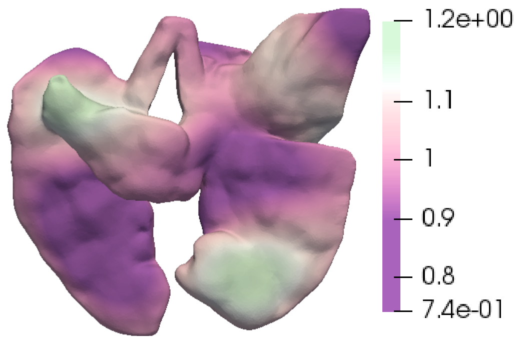


(c)

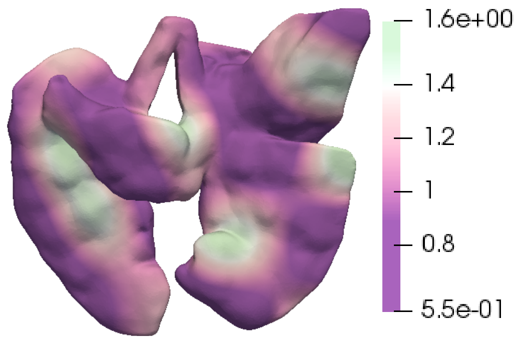


(d)

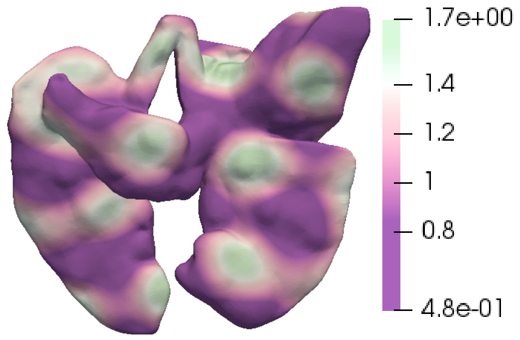
Figure 6.1. A comparison between experimental and simulated FGF10 distribution at E12.5 and E13.5 in the murine lung. (a) FGF10 distribution at E12.5. Dark green indicates areas where FGF10 is more highly concentrated. (b) Model 2x lung at $\gamma = 75$. Green indicates FGF10 concentrated areas. (c) FGF10 distribution at E13.5. Purple indicates areas where FGF10 is more highly concentrated. (d) Simulated 2x lung at $\gamma = 1500$. Green indicates FGF10 concentrated areas. [T. Volckaert, A. Campbell, E. Dill, C. Li, P. Minoo, and S. DeLanghe, *Localized Fgf10 expression is not required for lung branching morphogenesis but prevents differentiation of epithelial progenitors*, Development (Cambridge), 140 (2013), pp. 3731–3742.]



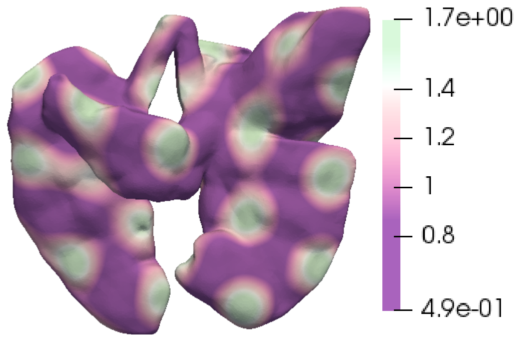
(a) $\gamma = 50$



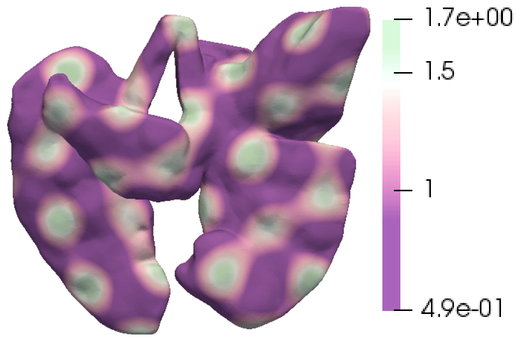
(b) $\gamma = 100$



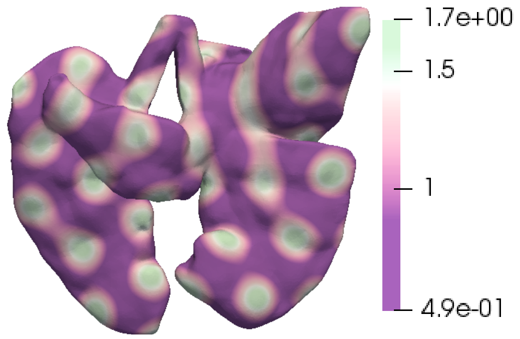
(c) $\gamma = 200$



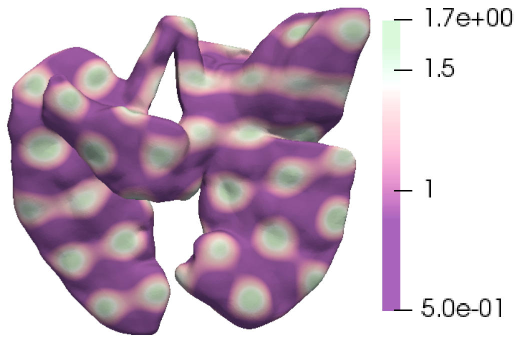
(d) $\gamma = 300$



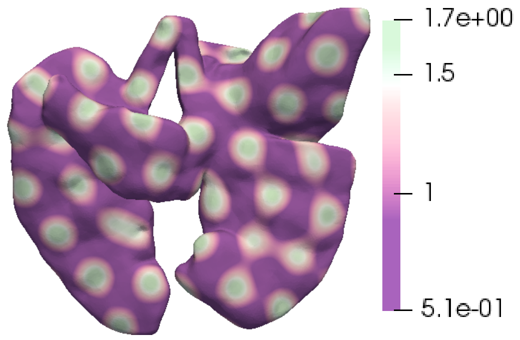
(e) $\gamma = 400$



(f) $\gamma = 500$

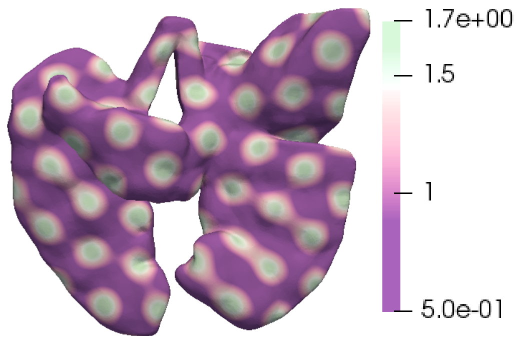


(g) $\gamma = 600$

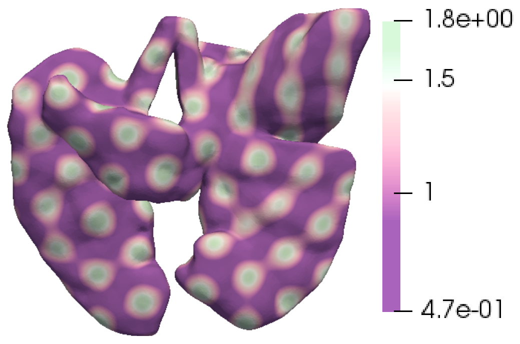


(h) $\gamma = 700$

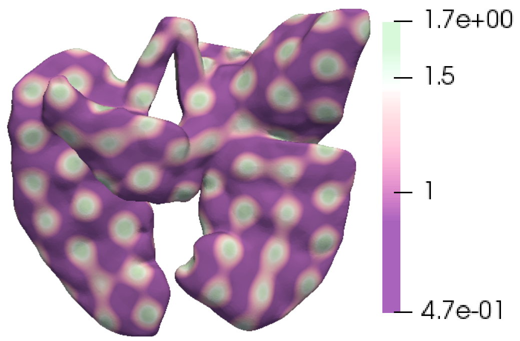
Figure 6.2. Model solutions on 2x lung meshes for γ = 50 to γ = 700



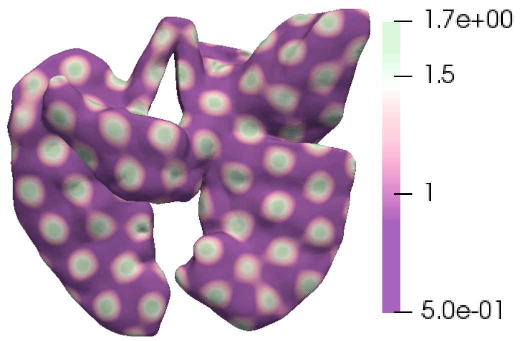
(a) $\gamma = 800$



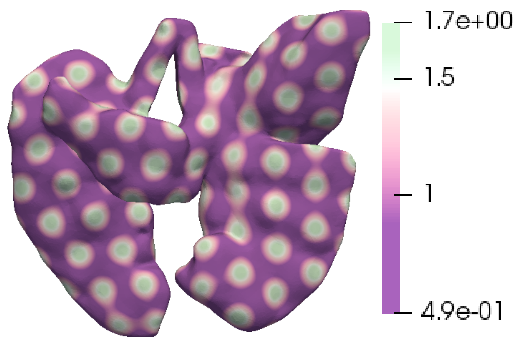
(b) $\gamma = 900$



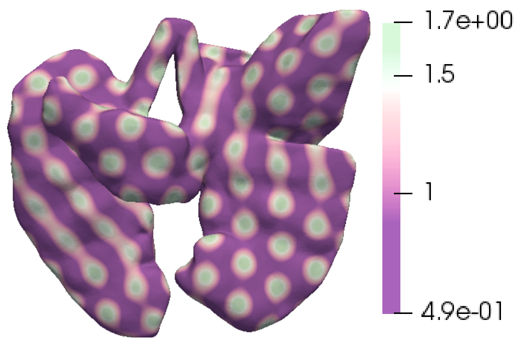
(c) $\gamma = 1000$



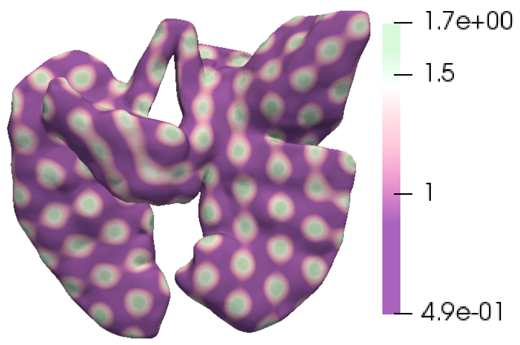
(d) $\gamma = 1100$



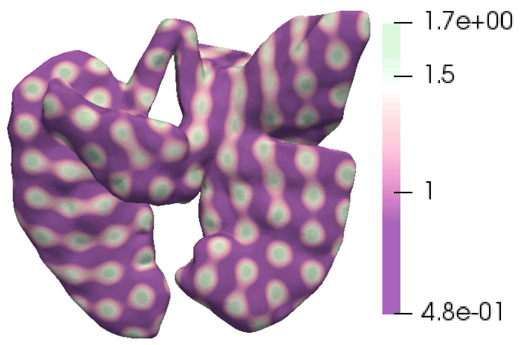
(e) $\gamma = 1200$



(f) $\gamma = 1300$



(g) $\gamma = 1400$



(h) $\gamma = 1500$

Figure 6.3. Model solutions on 2x lung meshes for γ = 800 to γ = 1500

6.3 Discussion of Results

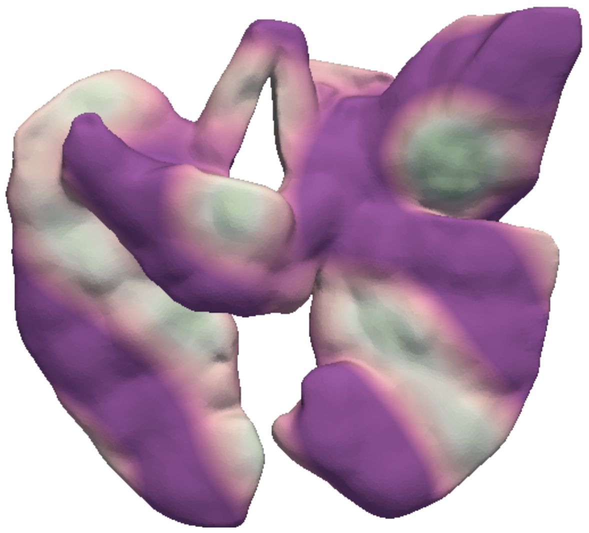
While the results are interesting, we do not yet know how they may contribute to the scientific community's discussion on lung development. There is only a qualitative match between simulated and experimental results, so the application to biological development seems limited. One reason why a match to experimental results may be difficult is the age of the domain. We have a model from E13.5, so simulating E12.5 presents some significant differences in tissue formation. Getting a closer match at E12.5 may not be attainable, simply because the geometry is not what we observe during the stage of development that is being simulated.

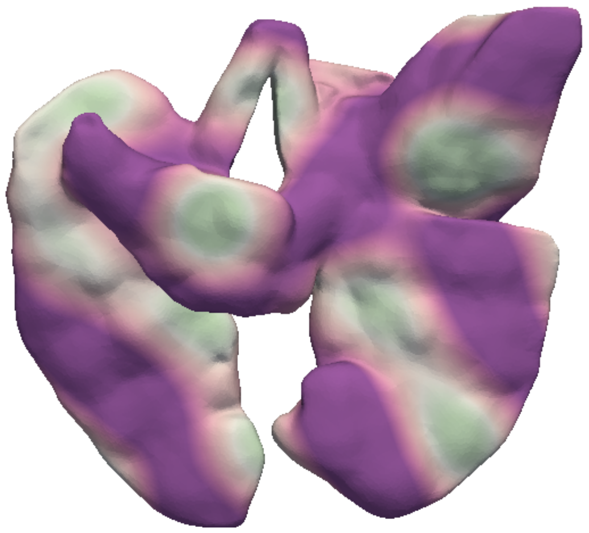
However, there is still much to learn through more varied simulations. A supplement to this thesis is a short movie (lung-movie.avi) containing lung images ranging from $\gamma = 0$ to $\gamma = 2000$, in intervals of 25. The movie provides some insight into how the surface patterns increase in complexity, but a movie with γ intervals at 1 or smaller could render a smoother visualization that would provide much insight into this issue. Unfortunately, that is not currently computationally feasible within a reasonable time frame.

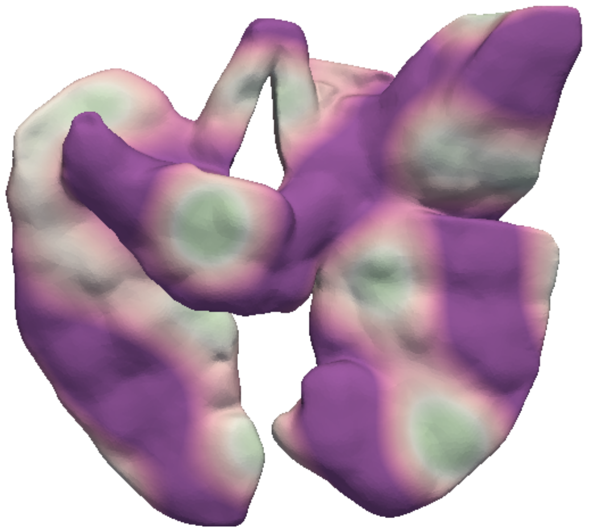
Limitations aside, we can still suggest ideas about how branching occurs based on FGF10 distribution. If FGF10 concentrations areas indeed merge into striations before splitting into several more concentration areas, then we may have insight into how lung branching is promoted. Recall that domain branching occurs during rapid growth. Such a phase may be stimulated by a universal increase of FGF10 on the lung surface. At first glance, it appears that simulations that show more pronounced FGF10 striations may also have FGF10 covering a greater surface area. A quantitative image analysis would be needed to validate such a statement. The direction of the striations may also control which direction branching occurs.

Figures 6.4 and 6.5 show a snippet of what this behavior may look like. In Figure 6.4, the inferior lobe has a very pronounced FGF10 stripe at $\gamma = 135$. As γ increases by 5 in each successive image, the stripe begins to distort and pinch into two separate concentration areas. Notice that in Figure 6.5, the highlighted area, also on the inferior lobe, shows up-down striations forming when $\gamma = 1625$ and $\gamma = 1650$, then they gradually migrate away from one another until they are separate at $\gamma = 1700$.

More experimental data could validate or invalidate an FGF10 concentration pattern migration like the one described above. Currently, there is no known literature to support genetic pattern migration in this context. Hopefully, there will soon be more research in this area to make more informed observations of results such as these.


(a) $\gamma = 135$


(b) $\gamma = 145$


(c) $\gamma = 155$

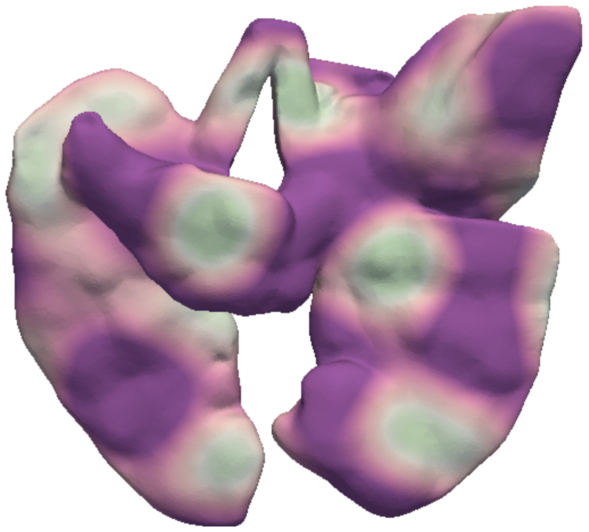
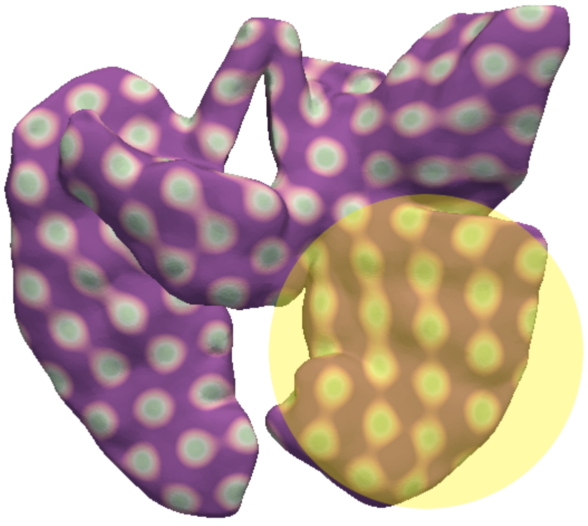
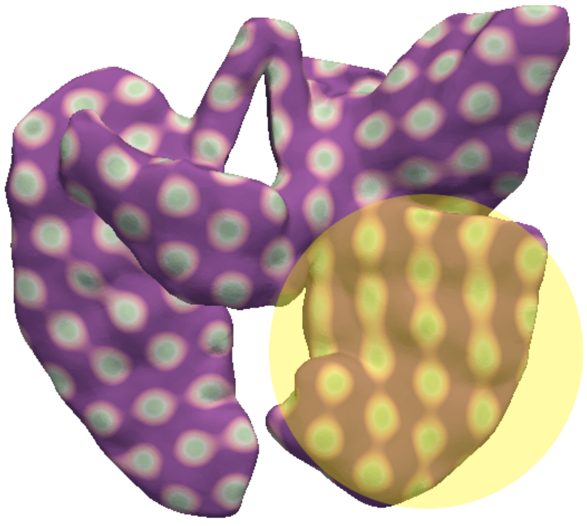
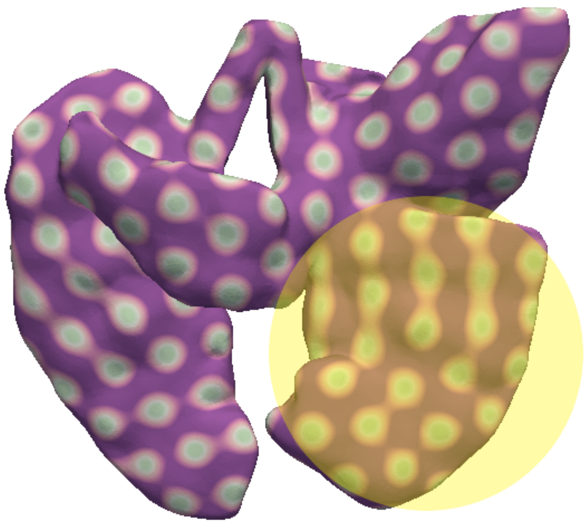

(d) $\gamma = 170$

Figure 6.4. Model solutions on 2x lung meshes for $\gamma = 135$ to $\gamma = 170$


(a) $\gamma = 1625$


(b) $\gamma = 1650$


(c) $\gamma = 1675$

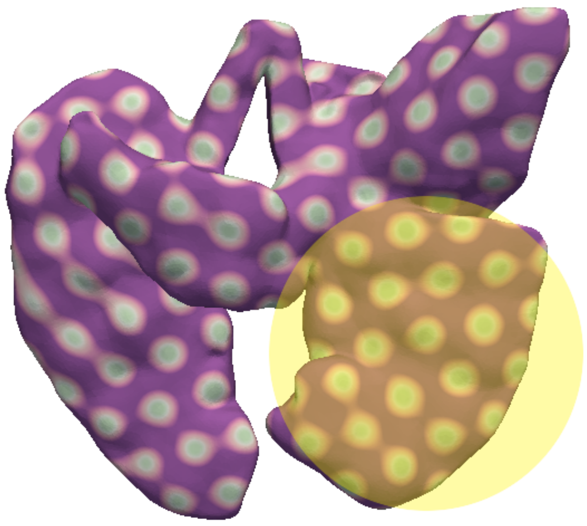

(d) $\gamma = 1700$

Figure 6.5. Model solutions on 2x lung meshes for $\gamma = 1625$ to $\gamma = 1700$

6.4 Conclusion

FGF10 is a key regulator of epithelial lung branching. The concentration and spatiotemporal expression of FGF10 is highly stereotyped. Wet-lab experiments have shown that the complexity of the spatial patterning of FGF10 expressions in embryonic lungs increase with gestational age in mice. The mechanisms that control the spatial expression of FGF10 are not fully understood. The highly complex interactions between different morphogens expressed in embryonic lungs hinder the understanding of morphogen expressions during lung development.

We proposed and implemented a new method to study the spatiotemporal expression of FGF10 at the lung surface. This involved using a system of surface reaction-diffusion equations to describe FGF10 interactions with SHH at the lung surface. Finite element simulations of the model equations on lung geometries of embryonic mice produced FGF10 patterning that are consistent with those reported in the literature.

We simulated FGF10 expression on lung geometries of mice at different stages of embryonic development, with all the model parameters held fixed. We found that the lung surface area increases with lung size and gestational age. Simulation results on smaller lungs produced minimal patterns, and the complexity of the simulated patterns increased with lung size. Therefore, the simulation results identified the lung surface area as a very important regulator of the spatial expression of FGF10.

Future work might involve expanding the model to investigate the dynamic interactions between FGF10 expression at the lung surface and SHH expression at the lung epithelium. This would further improve our understanding of the formation of the lung epithelial structures and highlight potential avenues that should be explored to increase the rate of epithelial branching in underdeveloped fetal lungs.

BIBLIOGRAPHY

[1] *How children's lungs grow*, British Lung Foundation, https://www.blf.org.uk, Accessed: 2020-02-13 (2019).

[2] P. Arbenz, *Lecture notes for Introduction to finite elements and sparse linear system solving*, September 2017.

[3] D. Arndt, W. Bangerth, B. Blais, T. C. Clevenger, M. Fehling, A. V. Grayver, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, R. Rastak, I. Thomas, B. Turcksin, Z. Wang, and D. Wells, *The deal.ii library, version 9.0*, submitted, (2020).

[4] D. N. Arnold, *Stability, Consistency, and Convergence: A 21st Century Viewpoint*, in Feng Kang Distinguished Lecture, 2009.

[5] R. Barreira, C. M. Elliott, and A. Madzvamuse, *The surface finite element method for pattern formation on evolving biological surfaces*, Journal of Mathematical Biology, 63 (2011), pp. 1095–1119.

[6] S. Bellusci, J. Grindley, H. Emoto, N. Itoh, and B. L. Hogan, *Fibroblast Growth Factor 10 (FGF10) and branching morphogenesis in the embryonic mouse lung*, Development, 124 (1997), pp. 4867–4878.

[7] A. Bonito, R. H. Nochetto, and M. S. Pauletti, *Geometrically Consistent Mesh Modification*, SIAM Journal on Numerical Analysis, 48 (2010), pp. 1877–1899.

[8] M. J. Cook, *The Anatomy of the Laboratory Mouse*, Academic Press, 1965.

[9] G. Dziuk and C. M. Elliott, *Surface finite elements for parabolic equations*, Technical Report 4, 2007.

[10] G. Dziuk and C. M. Elliott, *Finite element methods for surface PDEs*, Acta Numerica, 22 (2013), pp. 289–396.

[11] C. M. Elliott, B. Stinner, V. Styles, and R. Welford, *Numerical computation of advection and diffusion on evolving diffuse interfaces*, IMA Journal of Numerical Analysis, 31 (2011), pp. 786–812.

[12] C. Frye and C. J. Efthimiou, *Spherical Harmonics in p Dimensions*, (2012), p. 87.

[13] U. George and S. Lubkin, *Tissue geometry may govern lung branching mode selection*, Journal of Theoretical Biology, (2018).

[14] A. Gierer and H. Meinhardt, *A Theory of Biological Pattern Formation*, Kybernetik, 12 (1972), pp. 30–39.

[15] D. L. Hartl and E. W. Jones, *Genetics: Principles and Analysis*, Jones and Bartlett, Sudbury, Massachusetts, fourth ed., 1998.

[16] M. Herriges and E. E. Morrisey, *Lung development: Orchestrating the generation and regeneration of a complex organ*, Development (Cambridge), 141 (2014), pp. 502–513.

[17] A. L. Krause, A. M. Burton, N. T. Fadai, and R. A. Van Gorder, *Emergent Structures in Reaction-Advection-Diffusion Systems On a Sphere*, technical report, 2018.

[18] A. Lazarus, P. M. Del-Moral, O. Ilovich, E. Mishani, D. Warburton, and E. Keshet, *A perfusion-independent role of blood vessels in determining branching stereotypy of lung airways*, Development, 138 (2011), pp. 2359–2368.

[19] J. Lee and W. Sung, *Emergence and Evolution of Patterns*, technical report, 2000.

[20] J. Lü, K. I. Izvolsky, J. Qian, and W. V. Cardoso, *Identification of FGF10 targets in the embryonic lung epithelium during bud morphogenesis*, Journal of Biological Chemistry, 280 (2005), pp. 4834–4841.

[21] A. Madzvamuse, *Time-stepping schemes for moving grid finite elements applied to reaction–diffusion systems on fixed and growing domains*, Journal of Computational Physics, 214 (2006), pp. 239–263.

[22] A. Madzvamuse, A. H. W. Chung, and C. Venkataraman, *Stability analysis and simulations of coupled bulk-surface reaction–diffusion systems*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 471 (2015), p. 20140546.

[23] D. McCulley, M. Wienhold, and X. Sun, *The pulmonary mesenchyme directs lung development*, jun 2015.

[24] R. J. Metzger, O. D. Klein, G. R. Martin, and M. A. Krasnow, *The branching programme of mouse lung development*, Nature, 453 (2008), pp. 745–750.

[25] E. E. Morrisey, W. V. Cardoso, R. H. Lane, M. Rabinovitch, S. H. Abman, X. Ai, K. H. Albertine, R. D. Bland, H. A. Chapman, W. Checkley, J. A. Epstein, C. R. Kintner, M. Kumar, P. Minoo, T. J. Mariani, D. M. McDonald, Y. S. Mukouyama, L. S. Prince, J. Reese, J. Rossant, W. Shi, X. Sun, Z. Werb, J. A. Whitsett, D. Gail, C. J. Blaisdell, and Q. S. Lin, *Molecular determinants of lung development*, in Annals of the American Thoracic Society, vol. 10, apr 2013.

[26] J. D. Murray, *Mathematical Biology II: Spatial Models and Biomedical Applications*, Springer, third ed., 2000.

[27] C. V. Pepicelli, P. M. Lewis, and A. P. McMahon, *Sonic hedgehog regulates branching morphogenesis in the mammalian lung*, Current Biology, 8 (1998), pp. 1083–1086.

[28] M. Roth-Kleiner and M. Post, *Genetic Control of Lung Development*, Neonatology, 84 (2003), pp. 83–88.

[29] W. Sarfaraz and A. Madzvamuse, *Classification of parameter spaces for a reaction-diffusion model on stationary domains*, Chaos, Solitons and Fractals, 103 (2017), pp. 33–51.

[30] W. Sarfaraz and A. Madzvamuse, *Domain-dependent stability analysis and parameter classification of a reaction-diffusion model on spherical geometries*, European Journal of Applied Mathematics, (2018).

[31] J. C. Schittny, *Development of the lung*, Cell and Tissue Research, 367 (2017), pp. 427–444.

[32] J. Schnakenberg, *Simple chemical reaction systems with limit cycle behaviour*, Journal of Theoretical Biology, 81 (1979), pp. 389–400.

[33] C. Schnatwinkel and L. Niswander, *Multiparametric image analysis of lung-branching morphogenesis*, Developmental Dynamics, 242 (2013), pp. 622–637.

[34] E. Tadmor, *A review of numerical methods for nonlinear partial differential equations*, Bulletin of the American Mathematical Society, 49 (2012), pp. 507–554.

[35] J. M. Tordera, *Spherical harmonics*, MathWorks MATLAB®, (2020).

[36] A. M. Turing, *The Chemical Basis of Morphogenesis*, Philosophical Transactions of the Royal Society of London, 237 (1952), pp. 37–72.

[37] M. Unbekandt, P. M. del Moral, F. G. Sala, S. Bellusci, D. Warburton, and V. Fleury, *Tracheal occlusion increases the rate of epithelial branching of embryonic mouse lung via the FGF10-FGFR2b-Sprouty2 pathway*, Mechanisms of Development, 125 (2008), pp. 314–324.

[38] C. Varea, J. L. Aragó, and R. A. Barrio, *Turing patterns on a sphere*, Physical Review, 60 (1999), pp. 4588–4592.

[39] D. Warburton, A. El-Hashash, G. Carraro, C. Tiozzo, F. Sala, O. Rogers, S. D. Langhe, P. J. Kemp, D. Riccardi, J. Torday, S. Bellusci, W. Shi, S. R. Lubkin, and E. Jesudason, *Lung Organogenesis*, vol. 27, 2010, pp. 73–158.

APPENDIX

C++ Code Using deal.ii

C++ Code Using deal.ii

```
#include <deal.II/base/utilities.h>
#include <deal.II/base/quadrature_lib.h>
#include <deal.II/base/function.h>
#include <deal.II/base/logstream.h>

#include <deal.II/lac/vector.h>
#include <deal.II/lac/full_matrix.h>
#include <deal.II/lac/dynamic_sparsity_pattern.h>
#include <deal.II/lac/sparse_matrix.h>
#include <deal.II/lac/solver_cg.h>
#include <deal.II/lac/solver_control.h>
#include <deal.II/lac/precondition.h>
#include <deal.II/lac/affine_constraints.h>

#include <deal.II/grid/tria.h>
#include <deal.II/grid/manifold_lib.h>
#include <deal.II/grid/grid_refinement.h>
#include <deal.II/grid/grid_out.h>
#include <deal.II/grid/grid_in.h>
#include <deal.II/grid/tria_accessor.h>
#include <deal.II/grid/tria_iterator.h>

#include <deal.II/dofs/dof_handler.h>
#include <deal.II/dofs/dof_accessor.h>
#include <deal.II/dofs/dof_tools.h>

#include <deal.II/fe/fe_q.h>
#include <deal.II/fe/fe_values.h>
#include <deal.II/fe/mapping_q.h>

#include <deal.II/numerics/data_out.h>
#include <deal.II/numerics/vector_tools.h>
#include <deal.II/numerics/matrix_tools.h>

#include <fstream>
```

```
#include <iostream>
#include <cstdlib>
#include <math.h>


namespace Schnakenberg
{
  using namespace dealii;

  double epsilon = 0.1;
  double alpha   = 0.1;
  double beta    = 0.9;

  double delta   = 10.0;
  double gamma   = 300.0;

  template <int spacedim>
  class ReactionDiffusionEquation
  {
    public:
    ReactionDiffusionEquation
    (const unsigned degree = 2);
    void run();

    private:
    static constexpr unsigned int dim = spacedim - 1;

    void setup_system();
    void assemble_surface();
    void solve_time_step_U();
    void solve_time_step_V();
    void calculate_norm();
    void output_results(std::string) const;
    void compute_error() const;

    Triangulation<dim, spacedim>    triangulation;
    FE_Q<dim, spacedim>             fe;
    DoFHandler<dim, spacedim>       dof_handler;
    MappingQ<dim, spacedim>         mapping;

    AffineConstraints<double> constraints;

    SparsityPattern      sparsity_pattern;
    SparseMatrix<double> mass_matrix;
    SparseMatrix<double> laplace_matrix;
    SparseMatrix<double> system_matrix_u,
    system_matrix_v;

    Vector<double> solution_u, solution_v;
    Vector<double> exact_u, exact_v;
    Vector<double> old_solution_u, old_solution_v;
    Vector<double> system_rhs_u, system_rhs_v;
    Vector<double> u_squared_v;
    Vector<double> normal_u, normal_v;
    Vector<double> forcing_terms;

    double       time;
    double       time_step;
    unsigned int timestep_number;

    unsigned int skip_step;

    std::ofstream printfile;
  };



  template <int dim>
  class InitialValues_U : public Function<dim>
  {
    public:
    InitialValues_U () : Function<dim>() {}

    virtual double value (const Point<dim> &p,
    const unsigned int component = 0) const;
  };
```

```

template <>
double InitialValues_U<3>::value (const Point<3> &p,
const unsigned int) const
{
  using numbers::PI;
  double sum = 0;

  for(unsigned int i=1; i<9; i++)
  {
    sum += std::cos(2*PI*i*p[0]*p[1]+p[2]);
  }

  return (alpha + beta + epsilon*sum);
}

template <int dim>
class InitialValues_V : public Function<dim>
{
  public:
  InitialValues_V () : Function<dim>() {}

  virtual double value (const Point<dim> &p,
  const unsigned int component = 0) const;
};

template <>
double InitialValues_V<3>::value (const Point<3> &p,
const unsigned int) const
{
  using numbers::PI;
  double sum = 0;

  for(unsigned int i=1; i<9; i++)
  {
    sum += std::cos(2*PI*i*p[0]*p[1]+p[2]);
  }

  return (beta / std::pow(alpha+beta,2) + epsilon*sum);
}



template <int dim>
class RightHandSide : public Function<dim>
{
  public:
  RightHandSide() : Function<dim>() {}

  virtual double value(const Point<dim> &p,
  const unsigned int component = 0) const override;
};

template <>
double RightHandSide<3>::value(const Point<3> &p,
const unsigned int) const
{
  return 1;
}



template <int spacedim>
ReactionDiffusionEquation<spacedim>::
ReactionDiffusionEquation(const unsigned degree):
  fe(degree),
  dof_handler(triangulation),
  mapping(degree),
  time(0.0),
  time_step(1.0 / 1000.0),
  timestep_number(0),
  skip_step(100)
  {}
```

```
template <int spacedim>
void ReactionDiffusionEquation<spacedim>::setup_system()
{
  //// Shape Definition ////

  std::string import_name =
  "./models/final-positive-1x.obj";
  GridIn<dim, spacedim> grid_in;

  grid_in.attach_triangulation(triangulation);
  grid_in.read_assimp(import_name, -1, true);

  //// Degrees of Freedom ////

  dof_handler.distribute_dofs(fe);

  std::cout << std::endl << "Number of active cells: "
            << triangulation.n_active_cells() << std::endl
            << "Number of degrees of freedom: "
            << dof_handler.n_dofs() << std::endl;

  constraints.clear();
  DoFTools::make_hanging_node_constraints(
  dof_handler, constraints);
  constraints.close();

  //// Sparsity Pattern ////

  DynamicSparsityPattern dsp(dof_handler.n_dofs());
  DoFTools::make_sparsity_pattern(dof_handler,
                                  dsp,
                                  constraints,
                                  true);
  sparsity_pattern.copy_from(dsp);

  mass_matrix.reinit(sparsity_pattern);
  laplace_matrix.reinit(sparsity_pattern);

  system_matrix_u.reinit(sparsity_pattern);
  system_matrix_v.reinit(sparsity_pattern);

  MatrixCreator::create_mass_matrix(dof_handler,
  QGauss<dim>(fe.degree+1),
  mass_matrix);
  MatrixCreator::create_laplace_matrix(dof_handler,
  QGauss<dim>(fe.degree+1),
  laplace_matrix);

  solution_u.reinit(dof_handler.n_dofs());
  solution_v.reinit(dof_handler.n_dofs());

  old_solution_u.reinit(dof_handler.n_dofs());
  old_solution_v.reinit(dof_handler.n_dofs());

  system_rhs_u.reinit(dof_handler.n_dofs());
  system_rhs_v.reinit(dof_handler.n_dofs());

  u_squared_v.reinit(dof_handler.n_dofs());
  normal_u.reinit(dof_handler.n_dofs());
  normal_v.reinit(dof_handler.n_dofs());

  forcing_terms.reinit(solution_u.size());
}


//// Surface Assembly ////

template <int spacedim>
void ReactionDiffusionEquation<spacedim>::assemble_surface()
{
  system_matrix_u = 0;
  system_matrix_v = 0;
  system_rhs_u    = 0;
  system_rhs_v    = 0;

  const QGauss<dim> quadrature_formula(2 * fe.degree);
```

```
    FEValues<dim, spacedim> fe_values(mapping,
                                      fe,
                                      quadrature_formula,
                                      update_values |
                                      update_gradients |
                                      update_quadrature_points |
                                      update_JxW_values);

    const unsigned int dofs_per_cell = fe.dofs_per_cell;
    const unsigned int n_q_points =
    quadrature_formula.size();

    FullMatrix<double> cell_matrix(
    dofs_per_cell, dofs_per_cell);
    Vector<double>     cell_rhs(dofs_per_cell);

    std::vector<double>
    rhs_values(n_q_points);
    std::vector<types::global_dof_index>
    local_dof_indices(dofs_per_cell);

    const RightHandSide<spacedim> rhs_function;

    for (const auto &cell :
    dof_handler.active_cell_iterators())
    {
      cell_matrix = 0;
      cell_rhs    = 0;

      fe_values.reinit(cell);

      rhs_function.value_list(fe_values.get_quadrature_points(),
                              rhs_values);

      for (unsigned int i = 0; i < dofs_per_cell; ++i)
        for (unsigned int j = 0; j < dofs_per_cell; ++j)
          for (unsigned int q_point = 0;
               q_point < n_q_points; ++q_point)
```

```
              cell_matrix(i, j) +=
                   fe_values.shape_grad(i, q_point) *
                   fe_values.shape_grad(j, q_point) *
                   fe_values.JxW(q_point);

        for (unsigned int i = 0; i < dofs_per_cell; ++i)
          for (unsigned int q_point = 0;
               q_point < n_q_points;
               ++q_point)
            cell_rhs(i) +=
                 fe_values.shape_value(i, q_point) *
                 rhs_values[q_point] *
                 f e_values.JxW(q_point);

        cell->get_dof_indices(local_dof_indices);
        for (unsigned int i = 0; i < dofs_per_cell; ++i)
        {
          for (unsigned int j = 0; j < dofs_per_cell; ++j)
          system_matrix_u.add(local_dof_indices[i],
          local_dof_indices[j],
          cell_matrix(i, j));
          system_rhs_u(local_dof_indices[i]) +=
          cell_rhs(i);
          system_rhs_v(local_dof_indices[i]) +=
          cell_rhs(i);
        }
      }

      system_matrix_v.copy_from(system_matrix_u);
      system_rhs_v=system_rhs_u;

      std::map<types::global_dof_index, double>
      boundary_values;
      VectorTools::interpolate_boundary_values(
                                  mapping, dof_handler, 0,
                                  Functions::ZeroFunction<spacedim>(),
                                  boundary_values);

```

```
  MatrixTools::apply_boundary_values(boundary_values,
                                     system_matrix_u,
                                     solution_u,
                                     system_rhs_u,
                                     false);

  MatrixTools::apply_boundary_values(boundary_values,
                                     system_matrix_v,
                                     solution_v,
                                     system_rhs_v,
                                     false);
}


//// Solve Timestep ////
template <int spacedim>
void ReactionDiffusionEquation<spacedim>::solve_time_step_U()
{
  SolverControl solver_control(10000,
  1e-20 * system_rhs_u.l2_norm());
  SolverCG<>    cg(solver_control);

  PreconditionSSOR<> preconditioner;
  preconditioner.initialize(system_matrix_u, 1.0);

  cg.solve(system_matrix_u,
  solution_u,
  system_rhs_u,
  preconditioner);

  constraints.distribute(solution_u);
}



template <int spacedim>
void ReactionDiffusionEquation<spacedim>::solve_time_step_V()
{
  SolverControl solver_control(10000,
  1e-20 * system_rhs_v.l2_norm());
  SolverCG<>    cg(solver_control);

  PreconditionSSOR<> preconditioner;
  preconditioner.initialize(system_matrix_v, 1.0);

  cg.solve(system_matrix_v,
  solution_v,
  system_rhs_v,
  preconditioner);

  constraints.distribute(solution_v);
}



template<int dim>
void ReactionDiffusionEquation<dim>::calculate_norm()
{
  double norm_u = 0;
  double norm_v = 0;
  normal_u = solution_u;
  normal_u.add(-1, old_solution_u);
  normal_v = solution_v;
  normal_v.add(-1, old_solution_v);

  for(unsigned int i = 0; i < normal_u.size(); i++)
  {
    norm_u += std::pow(normal_u[i],2);
    norm_v += std::pow(normal_v[i],2);
  }

  norm_u = std::pow(norm_u, 0.5);
  norm_v = std::pow(norm_v, 0.5);
  printfile << timestep_number << ","
            << norm_u << ","
            << norm_v << "\n";
```

```
}


//// Output Solution ////

template <int spacedim>
void ReactionDiffusionEquation<spacedim>::
output_results(std::string gamma_string) const
{
  DataOut<dim, DoFHandler<dim, spacedim>> data_out;

  data_out.attach_dof_handler(dof_handler);
  data_out.add_data_vector(solution_u, "U",
                           DataOut<dim,
                           DoFHandler<dim,
                           spacedim>>::type_dof_data);
  data_out.add_data_vector(solution_v, "V",
  DataOut<dim,
          DoFHandler<dim,
          spacedim>>::type_dof_data);
  data_out.build_patches(mapping, mapping.get_degree());

  const std::string image_filename =
              "./solutions/lung-1x-g-" +
              gamma_string +
  Utilities::int_to_string(
              timestep_number/skip_step, 3) +
              ".vtk";
  std::ofstream output(image_filename);
  data_out.write_vtk(output);
}



template <int spacedim>
void ReactionDiffusionEquation<spacedim>::run()
{
  setup_system();
```

```
    assemble_surface();

    double gamma_whole = std::floor(gamma);
    double gamma_decim = std::floor((gamma -
                         gamma_whole)*1000);
    std::ostringstream g_object;
    g_object << std::setw(3) << std::setfill('0')
             << gamma_whole
             << std::setw(3) << std::setfill('0')
             << gamma_decim;
    std::string gamma_string = g_object.str();

    std::string norm_filename =
          "./lung-solutions/lung-norm-1x-" +
          gamma_string + ".csv";
    printfile.open(norm_filename);
    printfile << "0,0,0\n";

    std::map<types::global_dof_index, double> boundary_values;
    VectorTools::interpolate_boundary_values(mapping,
                                 dof_handler,
                                 999,
                                 Functions::ZeroFunction<spacedim>(),
                                 boundary_values);

    Vector<double> tmp;
    tmp.reinit(solution_u.size());

    VectorTools::interpolate(dof_handler,
    InitialValues_U<spacedim>(),
    old_solution_u);

    VectorTools::interpolate(dof_handler,
    InitialValues_V<spacedim>(),
    old_solution_v);

    solution_u = old_solution_u;
    solution_v = old_solution_v;

    std::cout << "t = " << time << std::endl;

    output_results(gamma_string);

    while (time <= 10)
    {

      time += time_step;
      ++timestep_number;

      if (timestep_number % skip_step == 0)
      std::cout << "t=" << time << std::endl;

      //// System Assembly ////

      // forcing_terms = gamma
      RightHandSide<spacedim> rhs_function;
      VectorTools::create_right_hand_side(mapping,
      dof_handler,
      QGauss<dim>(fe.degree + 1),
      rhs_function,
      forcing_terms,
      constraints);
      forcing_terms *= gamma;


      // u_squared_v = u^2*v
      std::transform(old_solution_u.begin(),
                     old_solution_u.end(),
                     old_solution_u.begin(),
                     tmp.begin(),
                     std::multiplies<double>() );
      std::transform(tmp.begin(),
                     tmp.end(),
                    old_solution_v.begin(),
                     u_squared_v.begin(),
                     std::multiplies<double>() );

      // system_rhs_u = k*gamma*(alpha*F - M*U_{n-1} +
      // M*U_{n-1}*U_{n-1}*V_{n-1}) + M*U_{n-1}
      system_rhs_u = forcing_terms;
      system_rhs_u *= time_step*alpha;
      mass_matrix.vmult(tmp, old_solution_u);
      system_rhs_u.add(1-time_step*gamma, tmp);
      mass_matrix.vmult(tmp, u_squared_v);
      system_rhs_u.add(time_step*gamma, tmp);

      // system_matrix_u = M + k*A
      system_matrix_u.copy_from(mass_matrix);
      system_matrix_u.add(time_step, laplace_matrix);

      constraints.condense(system_matrix_u, system_rhs_u);

      MatrixTools::apply_boundary_values(boundary_values,
                   system_matrix_u,
                   solution_u,
                   system_rhs_u);

      solve_time_step_U();

      // u_squared_v = u^2*v (again)
      std::transform(solution_u.begin(),
                     solution_u.end(),
                     solution_u.begin(),
                     tmp.begin(),
                     std::multiplies<double>() );
      std::transform(tmp.begin(),
                     tmp.end(),
                     old_solution_v.begin(),
                     u_squared_v.begin(),
                     std::multiplies<double>() );

      // system_rhs_v = k*gamma*(beta*F -
      // M*U_{n}*U_{n}*V_{n-1})+ M*V_{n-1}
      system_rhs_v = forcing_terms;
```

```
      system_rhs_v *= time_step*beta;
      mass_matrix.vmult(tmp, old_solution_v);
      system_rhs_v.add(1, tmp);
      mass_matrix.vmult(tmp, u_squared_v);
      system_rhs_v.add(-time_step*gamma, tmp);

      // system_matrix_v = M  + d*k*A
      system_matrix_v.copy_from(mass_matrix);
      system_matrix_v.add(time_step*delta, laplace_matrix);


      constraints.condense(system_matrix_v, system_rhs_v);
      MatrixTools::apply_boundary_values(
                          -boundary_values,
                          system_matrix_v,
                          solution_v,
                          system_rhs_v);
      solve_time_step_V();

      if (timestep_number % skip_step == 0)
      output_results(gamma_string);

      calculate_norm();
      old_solution_u = solution_u;
      old_solution_v = solution_v;
    }
  }
}

int main()
{
  using namespace dealii;
  using namespace Schnakenberg;
  ReactionDiffusionEquation<3> rd_equation_solver;
  rd_equation_solver.run();

  return 0;
}
```