Comparison of a Semi-Implicit and a Fully-Implicit Time Integration Method for a Highly Degenerate Diffusion-Reaction Equation Coupled with an Ordinary Differential Equation

by

Eric M. Jalbert

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Science
in
Applied Mathematics

Guelph, Ontario, Canada

ABSTRACT

Comparison of a Semi-Implicit and a Fully-Implicit Time Integration Method for a Highly Degenerate Diffusion-Reaction Equation Coupled with and Ordinary Differential Equation

Eric M. Jalbert
University of Guelph,2015

Advisor
Professor Hermann J. Eberl

A certain class of highly degenerate diffusion equations arises when modelling biofilm growth and propagations. We focus on the cellulolytic *Clostridium thermocellum*, because of its potential in the field of energy biotechnology. From this system a spatially implicit model was proposed in the literature before. Here we study a spatially explicit model. In contrast to other biofilm systems, a special feature of this system is that the growth promoting nutrient is not diffusive but bound in the substratum on which the biofilm grows. As a consequence one obtains a highly degenerate diffusion-reaction equation for the bacteria that is coupled to an ordinary differential equation for nutrients. The degeneracy of the biomass equation introduces gradient blow-up at the interface which makes numerical treatment difficult. For this, a fully-implicit time integration method is formulated so that it generalises a previously used semi-implicit method to solve the problem with increased accuracy. The fully-implicit method uses, at each time-step, a fixed-point iteration to solve the arising nonlinear algebraic equation and can be controlled by the required tolerance for convergence.

This method is validated and tested to investigate numerous issues that arise with numerical computations: mass conservations, preservation of symmetries in the initial data, and convergence with respect to grid refinement. Furthermore, a difference is quantified between the fully-implicit and the simpler semi-implicit methods which it generalises. The trade-off between improved accuracy and increased computational effort is found to be optimal for tolerances that force a single extra iteration of the fully-implicit method.

The numerical method is then used to simulate *C.thermocellum* biofilm formation on cellulose sheets with the main objectives of (i) understanding patterns of biofilm formation and (ii) understanding how including the spatial diffusion terms in the biomass affect the results of the simulations at a reactor-scale. Our simulation results strongly suggest the formation of travelling wave solutions that describe how the biofilm moves across the substratum. To test the effect of the spatial effects on overall biofilm performance, two extremes of initial biomass distributions were simulated. A quantitative difference between the behaviour in both cases is found, but not a qualitative one. This suggests that in applications where spatial heterogeneity is important then a two dimensional spatially explicit model that includes the spatial diffusion must be used instead of the earlier, simple spatially implicit reactor-scale ordinary differential equation model that consolidated the spatial effects with a carrying capacity on the growth term.

Contents

Chapter 1

Introductions

1.1 Introduction

Bacterial biofilms are microscopic depositions of organisms that attach themselves on immersed surfaces whenever environmental conditions can sustain microbial growth. These bacteria, when sessile, surround themselves in a self-produced viscous layer of extracellular polymeric substances. As a result of this, the cells are extraordinarily resistant to mechanical washout or antibiotic attacks. Most bacterial populations live in communities within the extracellular polymeric substances. They can be found in many different aspects of life in both positive ways (wastewater treatments, soil remediation, and groundwater protection) and negative ways (bacterial infections, dental plaque, biocorrosion of facilities and water pipes).

The first biofilm models, like that in Rittmann and McCarty (1980), assumed that biofilms developed as a flat layer and were posed as ordinary differential equations or one-dimensional partial differential equations. This simplified the calculation for the speed of propagation but was limited in non-spatially heterogenous biofilm morphologies. To this end, many models for spatially heterogenous biofilms were proposed. These included stochastic individual based models (Kreft et al., 2001), cellular automata models (Picioreanu et al., 1998), and deterministic continuum models (Eberl et al., 2001). The added complexity helped in modelling more of the multi-dimensional aspects of biofilms, namely the

intrinsic structures that most biofilms grew. These models considered the changes the biofilms made at the meso-scale instead of the reactor-scale which simpler models tended to do.

The recent field of energy biotechonology has led to researching biofilms as a potential means of using plant biomass to generate sustainable fuels. These fuels, such as ethanol, are generated through conversion of terrestrial or aquatic biomass (Lynd, 2008). The main benefits for using these fuel sources is that they produce much less greenhouse gases as seen in Figure 1.1. Using cellulolytic biofilms (cellulose degrading biofilms) to produce biofuels means that only non-edible cellulose is utilized instead of edible plants like corn. *Clostridium thermocellum* is a possible choice for achieving large scale biomass conversion. Because of this there has been a surge of studies based around its behaviours and characteristics. An interesting feature of *C.thermocellum* is that is grows as a thin cellulolytic monolayer and does not produce any extracellular polymeric matrix (Dumitrache et al., 2013a). This is a stark contrast to typical biofilms which are notorious for forming complex mushroom or pillar shaped morphologies.

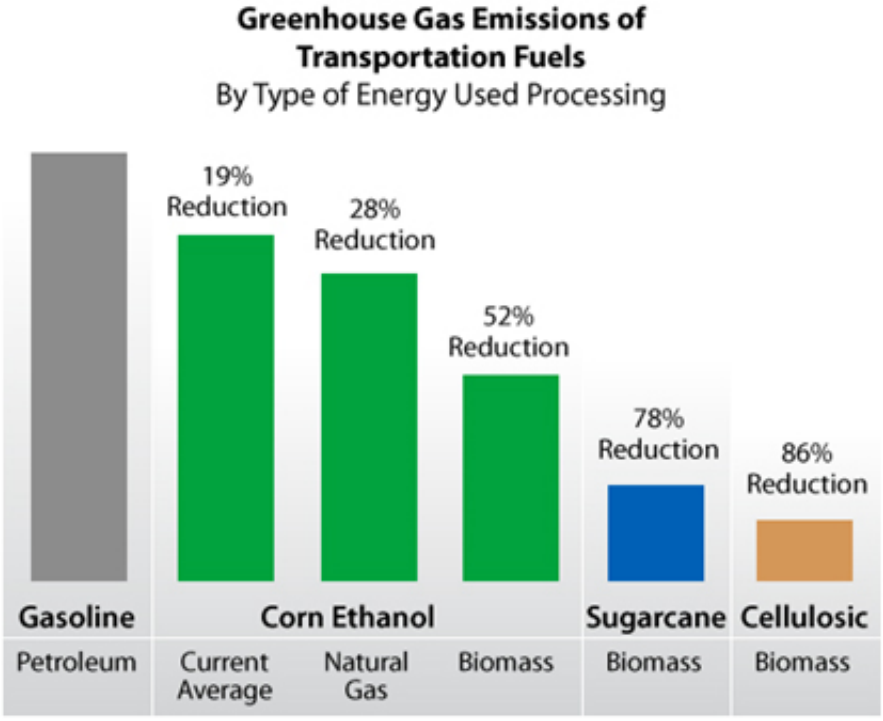


Figure 1.1: A comparison graph of greenhouse gas emissions for transportation fuels produced based on different fuel types. Each are related to the decreased emissions when compared to Petroleum. Graph indicating high greenhouse gas emissions for gasoline, then reductions with corn ethanol and further reductions with sugarcane biomass (78% reduction) and cellulosic biomass (86% reduction). Figure originally from Alternative Fuel Data Center (2014).

Dumitrache et al. (2015) have made a number of *in situ* and *in vitro* observations for *C.thermocellum*. Here they linked the cellulose consumption rate of the bacteria to the rate of CO_2 produced. The experiments ran in a continuous-flow reactor that used Whatman cellulose paper sheets inoculated with *C.thermocellum* strains. The bacteria consumed the fibers of the cellulose sheets as they grew. By tracking only the CO_2 production, they were only focused on the activity of the bacteria at a reactor-scale. The smaller spatial movements of the bacteria were ignored for simplicity of the experiment.

A simple mathematical model for cellulolytic biofilm activity and growth on model cellulosic substrate was proposed in Dumitrache et al. (2015). Here the production of carbon dioxide was used as an indicator of culture metabolism. Because of this indicator, they focused on overall biofilm performance rather than on detailed biofilm structure. This led to a reactor-scale model that attributed the spatial effects into logistic-like growth and carrying capacities to limit the bacteria activity in the models. The model was based on a number of observations on the metabolic activity gathered by online carbon dioxide measurements. For this model, attention was also put on high-resolution imaging of different stages of biofilm development (Dumitrache et al., 2013a) and to the physiological behaviour of substrate modification (Dumitrache et al., 2013b).

The conceptual model of cellulolytic biofilm growth that was followed for their model is shown in Figure 1.2. This model is based on the relation between *C.thermocellum* growth and cellulose sheet consumption. The relation is, when *C.thermocellum* grows there exist cellulose sheet consumption, limiting the area of substratum available for bacteria attachment. Here they express the growth of the biomass in terms of an *ideal* and an *actual* carrying capacity. The ideal carrying capacity, M_∞, refers to the maximum amount of biomass that can be supported when the only limitations are from a deficiency of local space. This value depends on the properties of the substratum and is assumed to be constant since it is independent of the substrate concentration. The actual carrying capacity, M_α, references the amount of biomass that can be supported when the local concentration of substrate mass is limited. This value is not constant since it is a function of the current substrate concentration. In essence, Dumitrache et al. (2015) use the carrying capacity as a means to account for the limitations due to spatial effects.

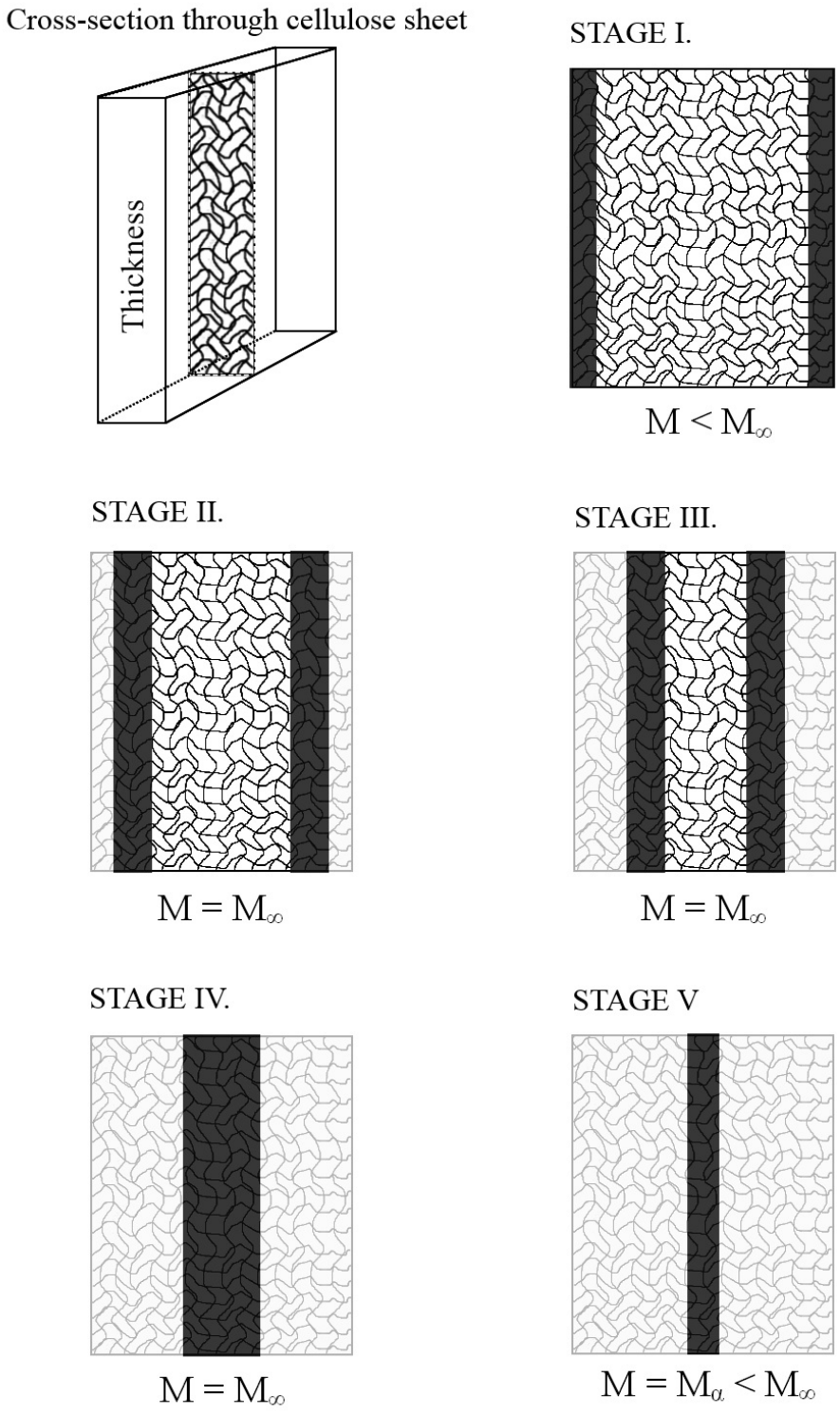


Figure 1.2: Conceptual model of cellulolytic biofilm growth and consumption of cellulose sheets. Attachment and growth occurs on both sides of the sheet, individual monolayers form on each fiber and result in a band of active biofilm (i.e., the effective sessile biomass) M (dark band). The ideal carrying capacity, M_∞, and the actual carrying capacity, M_α, are explained in Dumitrache et al. (2015). Consumed substrate is represented by the light gray areas. Figure originally from Dumitrache et al. (2015).

The model developed by Dumitrache et al. (2015) followed the five different conceptual growth stages of *C.thermocellum*. These stages were:

- Stage I: Independent colonies of cells grow on the matrix of fibers in the substrate. This occurs initially for all the isolated regions of the biofilm.
- Stage II: The biofilm grows inwards. The superficial fibres are consumed and newly unsupported biofilms are released from the cellulose sheet into the aqueous stream.
- Stage III: The active biofilm band stabilizes somewhere around the point where superficial fiber deconstruction rate is the equivalent to the inwards penetration rate of the biofilm.
- Stage IV: The progression of the biofilm band continues until the remaining amount of usable substrate becomes limited.
- Stage V: The remaining cellulose gets consumed without new biofilm being produced. Instead a new generation of non-adherent cells is formed locally.

This formulated a system of ordinary differential equation because of the non-diffusivity of the substrate. These ordinary differential equations resembled traditional growth models in batch cultures more then the typically complex biofilm model seen in studies (Wanner et al., 2005).

The model developed in Wang et al. (2011) focused more on detailing the qualitative aspects of a single *C.thermocellum* colony in terms of spatio-temporal development. The reaction kinetics were ignored as they assumed that when bacteria occupied a new space all the substrate would be immediately utilised. This was completed by use of a nine-neighbour square cellular automata on a 30×15 grid with a single grid cell as an inoculation point. The model results matched the experimental results they gathered. This was the first model to consider the spatial development of *C.thermocellum* at a small scale. Using cellular automata with such a coarse grid gave them a discrete representation of the system they modelled. One purpose of this thesis is to extend this concept to a continuous model similar to Eberl et al. (2001) but instead using the assumptions and growth function that match the behaviour of *C.thermocellum*.

There are problems with trying to extend *C.thermocellum* to a spatially considerate continuum model. Because the substratum used for attachment is consumed with biofilm growth and the stationary nature of the cellulose sheets, this problem becomes significantly different from other biofilm models which are based on the aqueous, free-floating environment where biofilms typically develop. At the meso-scale, our *C.thermocellum* system must model the development of the biofilm along the non-diffusing individual fibers of the cellulose sheet structure. Thus, these two categories of biofilm models differ mainly in their consideration of substrate diffusion, with *C.thermocellum* there is none.

Our problem originates from the ordinary differential equation model from Dumitrache et al. (2015). Here we include the double-degenerate parabolic model of biofilm formation from Eberl et al. (2001) for the spatial consideration. This type of model arises when a volume filling problem has a finite speed of interface propagation (Khassehkhan et al., 2009). A density-dependent diffusion model with reaction terms that match with the behaviour of *C.thermocellum* is what is handled here. The spatial operator for biofilm spreading shows two non-standard diffusion effects:

- A power law degeneracy similar to the porous medium equation for local biomass at the interface of the biofilm (Gurtin, 1977).
- A singularity in the diffusion coefficient when the biofilm approaches maximum biomass density.

Both of these effects together lead to the development of sharp and steep interfaces in the model solutions that mark the separation of the actual biofilm from its liquid environment. Such propagating interface problems in partial differential equations often are difficult to treat numerically.

The mathematical models which focus on the growth dynamics of biological films are usually systems of partial differential equations. These models are built on some of the significant features of biofilm growth observed throughout the practice, such as:

a) the presence of a sharp front of biomass at the solid to fluid transition region,

b) the existence of a threshold of biomass density,

c) the spreading of biomass is only notable when local densities approach the threshold of sustainability,

d) the use of reaction kinetics mechanisms to model the production of biomass,

e) the compatibility of biomass spreading with nutrient transfer and consumption mechanism models.

There exists mathematical background for these systems of equations that prove existence and uniqueness of positive and bounded solutions. However, the complexity of these nonlinear partial differential equations have made providing analytical expressions of the solutions practically impossible, when given biologically meaningful initial conditions.

This forces numerical computational approaches to be taken for simulating the growth of these microbial colonies. Some techniques with the finite-difference method have been used for these problems. This approach has been proposed in Eberl and Demaret (2007) for deterministic models.

In this thesis, the finite-difference method is treated as a semi-implicit numerical method similar to Eberl and Demaret (2007). Here, the method is extended to use a fixed-point iteration to facilitate the computations of the degeneracy from the biomass diffusion. This turns the semi-implicit method into a fully-implicit method. The validity of this method is unknown and put under scrutiny by simulating the *C.thermocellum* system during this thesis. This system has a partial differential equation for the diffusion-reaction equation of biomass and an ordinary differential equation for the non-diffusing substrate concentrations.

1.2 Objectives

There are three main objectives of this thesis:

1. Formulate and test a fully-implicit method for a highly degenerate, highly nonlinear coupled PDE-ODE system modelling *C.thermocellum* biofilms.

2. Compare the fully-implicit method of Objective 1 with a previously introduced semi-implicit method, which it generalizes. This is done based on the trade-off between improved accuracy of the method and increased computational effort.

3. Use the numerical methods developed in Object 1 and 2 to simulate *C.thermocellum* biofilm formation on cellulose sheets, with the goal of understanding better the spatio-temporal dynamics of this system biofilms.

1.3 Thesis Outline

In Chapter 1 of the thesis, a brief background for the research project is provided. The objectives of the thesis and the outline of how each objective will be addressed is also presented here.

In Chapter 2, the underlying model of *C.thermocellum* biofilm growth on cellulose sheets is stated and transformed into a non-dimensional model. This model is a coupled system comprised of a highly-degenerate partial differential equations for bacterial biomass and a ordinary differential equation for the growth limiting substrate.

In Chapter 3, a fully-implicit time-integration scheme is introduced that generalizes an earlier semi-implicit method. The implementation of the method is discussed and validated by testing it against typical settings and running a grid convergence test. Finally a comparison between the fully-implicit method and the earlier semi-implicit method is conducted.

In Chapter 4, the fully-implicit method of Chapter 3 is used to simulate *C.thermocellum* biofilm formation on cellulose fibers. The numerical solution suggests that the model permits travelling wave solutions that describe the spatio-temporal breakdown of the cellulose fibers. The results of the two dimensional simulations are compared qualitatively against a conceptual model of fiber breakdown and against lumped experimental data from the literature.

In Chapter 5, concluding statements are made concerning the main objectives of the thesis, based on the results from chapter 3 and 4. Future extensions of the works done here are also discussed.

Chapter 2

Model Definition

2.1 Model Description

The model used for simulations is based on the deterministic biofilm model developed in Eberl et al. (2001), which was designed for modelling the development of spatially heterogenous biofilm structures. Since *C.thermocellum* grows as a monolayered biofilm and consumes a solid carbon fibrous substrate, there are mechanical differences between the two systems. Our model is based on the following assumptions:

1. The growth of sessile biomass is limited locally by the availability of nutrients and by the availability of colonizable space.

2. The number of cells per unit area of substratum is limited to a finite value because *C.thermocellum* forms only a thin monolayer.

3. Biomass does not spread until its density approaches the physical limit. Near the physical limit it expands spatially into neighbouring regions. Because of this, the physical limit of biomass density is never attained.

4. The carbon fibrous substrate consumed as nutrition for biomass growth is the substratum to which the biofilm attaches. Carbon is bound in the fibers of the substratum and does not diffuse.

The propagation of biofilm is based on the lack of available substrate locally, not the physical degradation of the substratum.

5. Cell death and cell loss into the aqueous environment is assumed to be proportional to cell density.

6. Biomass growth is proportional to substrate consumption.

Assumption 1, 2, 3, 5, and 6 are similar to those made in Eberl et al. (2001). The main difference here is 4; our substrate is sessile. With a sessile substrate, there is no diffusion for the substrate concentration. Another difference is that *C. thermocellum* does not grow from the substratum into the aqueous phase. Instead our biofilm grows across the substratum making this a two dimensional setting.

The model is formulated in a spatial domain $\Omega \subset \mathbb{R}^2$. The independent variables $t > 0$ denote time and $x \in \Omega$ denotes the location within the physical domain. The dependent variables are the local fraction of the surface occupied by biomass $M(t, x)$ and the substrate density $C(t, x)$. The net growth rate of biomass is denoted by $f(C)$ and the substrate consumption rate is denoted by $g(C)$, both are dependent upon available substrate. The diffusion coefficient that describes spatial expansion of biomass is given by the function $d(M)$.

From the above assumptions, a PDE-ODE-coupled system that models *C. thermocellum* growth on carbonous fibres can be formulated as,

$$M_t = \nabla_x \left(d(M) \nabla_x M \right) + f(C)M \tag{2.1}$$

$$C_t = -g(C)M \tag{2.2}$$

where

$$d(M) = d \frac{M^\alpha}{(1 - M)^\beta} \tag{2.3}$$

$$f(C) = u\frac{C}{k_C + C} - n \tag{2.4}$$

$$g(C) = y\frac{C}{k_C + C} \tag{2.5}$$

with all parameters non-negative. Here we have a pair of equations, (2.1) and (2.2), that represent the biomass density and substrate concentration respectively. This is a model for spatially explicit biomass growth. This agrees with assumption 3 since for $0 < M \ll 1$ the spreading effect is negligible but when $0 \ll M \approx 1$ there is considerable spreading. This choice for a spatially considerate model is based on the work done in Khassehkhan et al. (2009). By assumption 1, the only factors affecting the biomass density is growth from nutrient conversion and diffusion from local spatially-full colonized space. For equation (2.3), the density-dependent diffusion equation, d is the diffusion coefficient which controls the magnitude of the diffusion and the parameters α and β are selected to control the strength of the diffusion. For equation (2.3) the diffusion term is shown to have the wanted behaviour since it has a near-zero finite value until $M \to 1$, which leads to $d(M) \to \infty$ as seen in Figure 2.1. The production rate is the difference between simple Monod kinetic growth term, with growth rate u, and a constant death rate term, n, to agree with assumption 1 and 5. Monod kinetic growth was selected, with half-saturation carbon concentration k_C, since it matches the growth of bacteria when limited by available nutrients.

Equation (2.2) describes the consumption of carbon substrate due to biomass growth. Parameter y is the consumption rate, measured in mass carbon per unit time. Substrate consumption is proportional to the local biomass density M. Parameter k_C, same as in the growth term for (2.1), is again the half-saturation carbon concentration. Here assumption 4 and 6 are satisfied since there exists no diffusion term for the substrate and its growth is a scalar multiple of the biomass growth rate.

There has been shown to exist a finite speed of interface propagation for the solutions of these kinds of degenerate problems, where $d(0) = 0$ and $\alpha > 1$ from (2.3) (Jalbert and Eberl, 2014). These problems have a blow up in the biomass gradient at the interface because of the degeneracy that exists there. For this system, we have $M < 1$ always since the diffusion when $M \approx 1$ is great enough to always ensure this (Jalbert and Eberl, 2014).

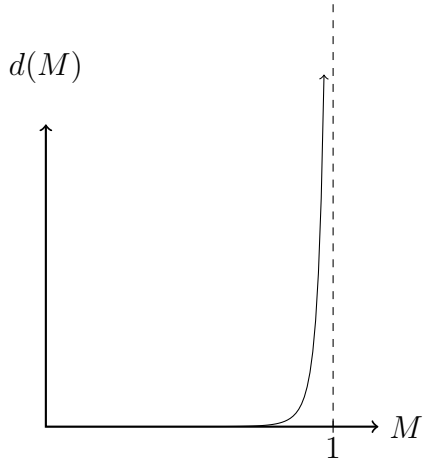


Figure 2.1: A graph of $d(M) = d\frac{M^\alpha}{(1-M)^\beta}$ showing the way diffusion increases asymptotically as $M \to 1$.

The dimensions of the parameters and variables are in Table 2.1. Note that since we have a two dimensional problem, due to the lack of complex biofilm structures from *C. thermocellum* growth, the spatial considerations are all strictly for area and not volume, as is typically done for biofilm modelling.

Description	Symbol	Dimensions
Spatial region	Ω	NA
Time	t	$[days]$
Location in Ω	$x = (x_1, x_2)$	$[meters^2]$
Biomass fraction	M	$[-]$
Substrate concentration	C	$\left[\frac{grams}{meters^2}\right]$
Diffusion coefficient	d	$\left[\frac{meters^2}{days}\right]$
Density-dependent exponent	α	$[-]$
Density-dependent exponent	β	$[-]$
Growth rate	u	$[days^{-1}]$
Half-saturation carbon concentration	k_C	$\left[\frac{grams}{meters^2}\right]$
Maximum consumption rate	y	$\left[\frac{gramscarbon}{days}\right]$
Death constant	n	$[days^{-1}]$

Table 2.1: List of parameters and their dimensions

The model (2.1), (2.2) is completed by boundary conditions for the biomass density, M, and initial conditions for both M and substrate concentration C. For M we pose homogeneous Neumann boundary conditions such that,

$$\partial_n M = 0, \quad x \in \partial\Omega. \tag{2.6}$$

The initial conditions for the biomass density is,

$$M(0,x) = M_0(x), \quad x \in \Omega, \tag{2.7}$$

where $0 \leq M_0(x) < 1$ and $M_0(x)$ non-zero in specific pockets on the substratum. These are specified below for each individual, simulation experiments. The initial conditions for the substrate concentration is,

$$C(0,x) = C_\infty, \quad x \in \Omega, \tag{2.8}$$

where C_∞ describes the initial carbon density in the substratum.

2.2 Nondimensionalization

To help facilitate the analysis of this system, the full removal of all physical units is preferred and so we nondimensionalize the parameters. From this point on, we assume that Ω is a square two dimensional region of length, L. Here the parameters used are: the biomass growth rate, u; the length of the region, L; and the maximum density for biomass and substrate, M_∞ and C_∞. The biomass density fraction represents the current density of biomass divided by the maximum biomass density, M_∞. From using the following parameter changes, the system can be made unitless.

$$\chi = \frac{x}{L} \implies L\nabla_\chi = \nabla_x \tag{2.9}$$

$$\tau = ut \implies \frac{1}{u}d\tau = dt \tag{2.10}$$

$$\mathcal{C} = \frac{C}{C_\infty} \tag{2.11}$$

$$\delta = \frac{1}{uL^2}d \tag{2.12}$$

$$\kappa = \frac{k_C}{C_\infty} \tag{2.13}$$

$$\nu = \frac{n}{uC_\infty} \tag{2.14}$$

$$\gamma = \frac{M_\infty}{C_\infty}y \tag{2.15}$$

Using these, (2.1) and (2.2) can be simplified and nondimensionalized into,

$$M_\tau = \nabla_\chi \left(D(M)\nabla_\chi M\right) + F(\mathcal{C})M \tag{2.16}$$

$$\mathcal{C}_\tau = -G(\mathcal{C})M, \tag{2.17}$$

where,

$$\begin{aligned} D(M) &= \delta\frac{M^\alpha}{(1-M)^\beta} \\ F(\mathcal{C}) &= \frac{\mathcal{C}}{\kappa+\mathcal{C}} - \nu \\ G(\mathcal{C}) &= \gamma\frac{\mathcal{C}}{\kappa+\mathcal{C}}. \end{aligned} \tag{2.18}$$

with only $\delta, \alpha, \beta, \kappa, \nu, \gamma$ as model parameters. For convenience, we henceforth use

$$C := \mathcal{C}, \quad x := \chi, \quad t := \tau. \tag{2.19}$$

Each of the dimensionless parameters in (2.18) have a biological representation based on the trans-

formations done. The parameter δ is the dimensionless biomass motility coefficient. It affects the change in biomass from adjacent biomass sources, a greater δ results in faster biofilm expansion. The parameter κ is the half-saturation point, it is exactly the value for which substrate concentration results in 0.5-optimum growth rate. Parameter ν is the decay and loss rate for biomass. These can be from starvation in cases where substrates are depleted or from loss into the aqueous environment. Lastly, γ is the dimensionless maximum substrate consumption rate. It signifies the ratio of substrate consumed to biomass growth. Here, a larger γ value results in more substrate being consumed to produce the same amount of biomass.

With (2.16) being reduced to these parameters the numerical analysis become more simplified while still retaining the same significance in results. From this point on, since L, the length of the region, was used for nondimensionalization Ω is now the unit square.

Chapter 3

Numerical Methods

3.1 Discretization

In order to approximate the solution for (2.16) spatial and temporal discretizations must be made. First the equations are discretized in time,

$$\frac{M^{k+1} - M^k}{\Delta t} = \nabla_x (D(M^{k+1}) \nabla_x M^{k+1}) + F(C^{k+1}) M^{k+1}, \tag{3.1}$$

$$\frac{C^{k+1} - C^k}{\Delta t} = \frac{1}{2}(G(C^{k+1}) M^{k+1} + G(C^k) M^k). \tag{3.2}$$

Here, (3.1) follows the ideas of the Backwards Euler Method; (3.2) follows Trapezoidal Rule (Burden and Faires, 2010). The index variable k has been introduced in (3.1) - (3.2) such that $M^k(x) \approx M(t^k, x)$, allowing an approximation at a certain time, t^k, to be used; this changes the spatial-temporal continuum model into a spatial continuum model with discrete temporal time steps.

Now, only (3.1) requires spatial considerations since the substrate does not diffuse across the region. The spatial discretization will be through the Finite Difference Method as described in Saad (2003). Here, a uniform $n \times m$ grid is used to discretize Ω. Since all the calculations will be done on the grid intersections the discretization will be grid-point based. This means that a $n \times m$ grid implies there are $(n-1) \times (m-1)$ grid boxes. The distance between grid points is the same in both x_1 and x_2

dimensions; we have $\Delta x_1 = \Delta x_2 = \Delta x$. Since we work on a nondimensionalized domain, and we known the number of grid boxes in our region, we have that $\Delta x = \frac{1}{n-1}$. A five-point stencil is used to approximate the solution of (3.1) at each grid point. This spatial discretization allows the use of i and j to index across the region such that $x_{1_i} = i * \Delta x$ for $i \in \{0, 1, \ldots, n-1\}$ and $x_{2_j} = j * \Delta x$ for $j \in \{0, 1, \ldots, m-1\}$. To index the grid point, i and j are used such that $M^k_{i,j} \approx M(t^k, x_{1_i}, x_{2_j})$. To account for the dependency on neighbouring grid points, we introduce σ as the index pair from the set

$$\mathcal{N}_{ij} = \{n_{ij}, e_{ij}, s_{ij}, w_{ij}\} \tag{3.3}$$

where,

$$\begin{aligned} n_{ij} &= \begin{cases} (i, j+1) & \text{if } j < m \\ (i, j-1) & \text{if } j = m \end{cases} \qquad e_{ij} = \begin{cases} (i+1, j) & \text{if } i < n \\ (i-1, j) & \text{if } i = n \end{cases} \\ s_{ij} &= \begin{cases} (i, j-1) & \text{if } j > 0 \\ (i, j+1) & \text{if } j = 0 \end{cases} \qquad w_{ij} = \begin{cases} (i-1, j) & \text{if } i > 0 \\ (i+1, j) & \text{if } i = 0 \end{cases}. \end{aligned} \tag{3.4}$$

With $\mathcal{N}_{ij}$ and σ we can account for the difference in boundary points and interior points. The different branches of (3.4) are based on the second order discretisation of the Neumann boundary conditions.

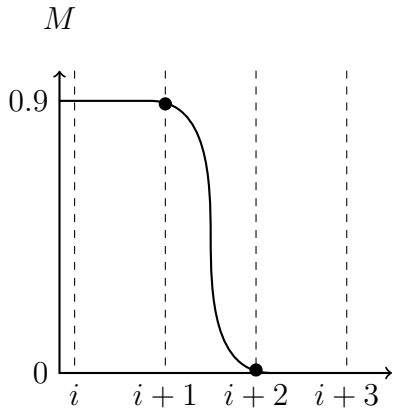
The equation for (3.1), after following the spatial discretization for finite-difference method listed in Saad (2003), is

$$\frac{M^{k+1}_{i,j} - M^k_{i,j}}{\Delta t} = \frac{1}{\Delta x^2} \sum_{\sigma \in \mathcal{N}_{ij}} \left(\frac{D(M^{k+1}_\sigma) + D(M^{k+1}_{i,j})}{2} \right) \cdot \left(M^{k+1}_\sigma - M^{k+1}_{i,j} \right) + F(C^{k+1}_{i,j}) M^{k+1}_{i,j} \tag{3.5}$$

For completeness, we spatially discretize (3.2) as

$$\frac{C^{k+1}_{i,j} - C^k_{i,j}}{\Delta t} = \frac{1}{2}(G(C^{k+1}_{i,j}) M^{k+1}_{i,j} + G(C^k_{i,j}) M^k_{i,j}). \tag{3.6}$$

Notice that for (3.5), the arithmetic mean of the diffusion function, D, is taken because of the steep gradient at the interface. Since this discretization requires $D(M)$ to be evaluated at points that lie



Geometric: $\sqrt{D(0.9) * D(0)} = 0$

Interpolation: $1 \gg D\left(\frac{0.9+0}{2}\right) \approx 0$

Arithmetic: $\frac{D(0.9)+D(0)}{2} = \frac{D(0.9)}{2} \neq 0$

Figure 3.1: An illustration of the three methods for approximating the value of $D(M)$ at ghost points located in between the existing grid points. The two black circles represent the two points inconsideration for the calculations of geometric mean, interpolation of values, and arithmetic mean. The problem with Geometric mean is that $D(0)$ evaluates to 0, resulting in no diffusion effects. With interpolation, the value of $D(0.45)$ is near-zero because $M \ll 1 \implies D(M) \approx 0$. For the arithmetic mean, since $D(0.9)$ is a larger value then the other methods some spatial diffusion actually work.

in between the existing grid points, which do not exist, an approximation is made. Between the three common choices for approximating a point (arithmetic mean, geometric mean, and interpolation) arithmetic mean is the best suited for this situation. This is illustrated in Figure 3.1, where the geometric mean and interpolation approximation are shown to result in a zero diffusion term, thus stopping the spreading of the biomass. Taking the arithmetic mean eliminates this result because the average value of D would not be zero at the interface.

To simplify the spatial indexing, the grid functions are converted into vector functions by use of a bijective mapping defined as:

$$\begin{array}{rccc} \pi: & \{1,\ldots,n\} \times \{1,\ldots,m\} & \to & \{1,\ldots,nm\} \\ & (i,j) & \mapsto & \pi(i,j) \end{array} \tag{3.7}$$

Now, a single index can be used to iterate over the vector, l. Lexigraphical ordering is used here resulting in a bijective function $\pi(i,j) = j + (i-1)m$. This gives the system,

$$\frac{M_l^{k+1} - M_l^k}{\Delta t} = \frac{1}{\Delta x^2} \sum_{\sigma \in \mathcal{N}_{\pi^{-1}(l)}} \left(\left(\frac{D(M_{\pi(\sigma)}^{k+1}) + D(M_l^{k+1})}{2} \right) \cdot \left(M_{\pi(\sigma)}^{k+1} - M_l^{k+1} \right) \right) + F(C_l^{k+1}) M_l^{k+1} \tag{3.8}$$

$$\frac{C_l^{k+1} - C_l^k}{\Delta t} = \frac{1}{2}(G(C_l^{k+1}) M_l^{k+1} + G(C_l^k) M_l^k). \tag{3.9}$$

3.2 Solution Technique

Assuming the values for C and M at time level k are known, (3.8) and (3.9) are a coupled system of $2nm$ highly nonlinear equation for $2nm$ unknown M_l^{k+1}, C_l^{k+1}. To solve this coupled system, we define a fixed point iteration which we apply to (3.8) and (3.9). In a single time step, the solutions for M and C can be solved using the previous time step solution in the follow manner:

$$\frac{M_l^{(p+1)} - M_l^k}{\Delta t} = \frac{1}{\Delta x^2} \sum_{\sigma \in \mathcal{N}_{\pi^{-1}(l)}} \left(\frac{D(M_{\pi(\sigma)}^{(p)}) + D(M_l^{(p)})}{2} \right) \cdot \left(M_{\pi(\sigma)}^{(p+1)} - M_l^{(p+1)} \right) + F(C_l^{(p)}) M_l^{(p+1)} \tag{3.10}$$

$$\frac{C_l^{(p+1)} - C_l^k}{\Delta t} = \frac{1}{2}(G(C_l^{(p+1)}) M_l^{(p+1)} + G(C_l^k) M_l^k) \tag{3.11}$$

where $(p) \in (0, 1, 2, \ldots)$, $(p+1)$ is the next iteration solution fo p, and k is the solution of the previous timestep. An initial guess is made such that,

$$M_l^{(p)} := M_l^k, \quad C_l^{(p)} := C_l^k. \tag{3.12}$$

The fixed-point iteration is stopped when convergence is achieved. This is when the difference between consecutive iterations is below a selected tolerance, i.e.

$$\sum_{l=1}^{nm} \left(\left| M_l^{(p+1)} - M_l^{(p)} \right| + \left| C_l^{(p+1)} - C_l^{(p)} \right| \right) < tol. \tag{3.13}$$

At the end of the fixed-point iteration, the number of iterations is recorded as P, and we define,

$$M_l^{k+1} := M_l^{(P)}, \quad C_l^{k+1} := C_l^{(P)}. \tag{3.14}$$

In this fixed point format, given by (3.10) - (3.11), the equations can be rearranged and solved by conventional methods.

In each iteration step, (3.10) is a simultaneous linear system for the nm unknown $M_l^{(p+1)}$. From this a linear system of equations can be created following Saad (2003).

From (3.10), we get the matrix equation:

$$A^{(p)} M^{(p+1)} = \frac{1}{\Delta t} M^k. \tag{3.15}$$

Here, $A^{(p)}$ is a five-diagonal $nm \times nm$ matrix, defined as

$$A^{(p)} = \begin{bmatrix} a_{1,1} & a_{1,2} & & a_{1,m} & & & & & \\ a_{2,1} & \ddots & \ddots & \ddots & \ddots & & & & \\ a_{n,1} & \ddots & \ddots & \ddots & \ddots & \ddots & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & a_{nm-n,nm} & \\ & & & \ddots & \ddots & \ddots & \ddots & a_{nm-1,nm} & \\ & & & & a_{nm,nm-m} & & a_{nm,nm-1} & a_{nm,nm} \end{bmatrix} \tag{3.16}$$

where each $a_{i,j}$ is the coefficient for specific grid indices based on (3.10).

Proposition 3.2.1. *If* $\Delta t < \left(F(C_l^{(p)})\right)^{-1}$ *then the matrix* A *is positive definite and symmetric.*

Proof. Matrix A is positive definite if all the eigenvalues are positive. Using the Gershgorin's Circle theorem described by Varga (2004), the eigenvalues can be shown to be positive. Gershgorin's Circle theorem tells us that each eigenvalue must be contained in the union of all Gershgorin's discs (Varga, 2004). There exist one disc for each row of A, with radius equal to the sum of the off-diagonals and center equal to the value of the main diagonal. By showing that, independently on all rows, the sum of the off-diagonals values is less then the diagonal value we have that all the Gershgorin's discs must be in the positive region when the main diagonal is positive. This can be shown by manipulating (3.10) for just the main diagonal element,

$$\sum_{\sigma \in \mathcal{N}_{\pi^{-1}(l)}} \left(\left(\frac{D(M_{\pi(\sigma)}^{(p)}) + D(M_l^{(p)})}{2\Delta x^2} \right) - F(C_l^{(p)}) + \frac{1}{\Delta t} \right) M_l^{(p+1)} \tag{3.17}$$

and for just the off-diagonal elements,

$$\sum_{\sigma\in\mathcal{N}_{\pi^{-1}(l)}}\left(\left(\frac{D(M^{(p)}_{\pi(\sigma)})+D(M^{(p)}_{l})}{2\Delta x^2}\right)M^{(p+1)}_{\pi(\sigma)}\right). \tag{3.18}$$

Comparing the coefficients to the vector elements in (3.17) - (3.18) results in the inequality nesseccary for Gershgorin's Circle theorem.

$$\left|\sum_{\sigma\in\mathcal{N}_{\pi^{-1}(l)}}\frac{D(M^{(p)}_{\pi(\sigma)})+D(M^{(p)}_{l})}{2\Delta x^2}\right| < \left|\sum_{\sigma\in\mathcal{N}_{\pi^{-1}(l)}}\left(\frac{D(M^{(p)}_{\pi(\sigma)})+D(M^{(p)}_{l})}{2\Delta x^2}\right) - F(C^{(p)}_{l})+\frac{1}{\Delta t}\right| \tag{3.19}$$

This simplifies to,

$$\Delta t < \left(F(C^{(p)}_{l})\right)^{-1} \tag{3.20}$$

When this inequality is true, we have that the off-diagonals are smaller then the main diagonal and the main diagonal values are positive thus Gershgorin's Circle theorem says that the eigenvalues are all positive. Therefore we have positive definite when (3.20) is true.

The symmetry of A can be trivially shown if one considers the formation of the diagonals. On a single row, each element corresponds to the adjacent grid points of grid l. As the grid ordering counts along, the elements that are equi-distance from the diagonal actually reference the same grid point. Therefore we have symmetry. □

It is important to remark that the time-step constraint in Proposition 3.2.1 is not a severe constraint, practically. The condition, $\frac{1}{F(C)} < \Delta t$, relates the growth of the biomass to the size of time step selected. In order to resolve any biomass growth, Δt must obviously be chosen smaller then the characteristic time scale of growth ,$\frac{1}{F(C)}$.

Given that A is positive definite and symmetric, the conjugate gradient method can be used to compute the solution.

Proposition 3.2.2. *The matrix A is diagonally dominant when $\Delta t < \left(F(C^{(p)}_{l})\right)^{-1}$.*

Proof. This is trivially shown to be true when one considers (3.19). It was shown that this simplifies to

$$\Delta t < \left(F(C_l^{(p)})\right)^{-1} \tag{3.21}$$

This means that when the above is true the diagonal elements of A will be strictly larger then the sum of off-diagonals. Therefore we have diagonal dominance. □

Since we have A positive definite, symmetric, and diagonally dominant we know that A is an M-matrix. This is important because this ensures that if M^k is non-negative we have that $M^{(p)}$ is also non-negative.

For solving (3.11), the equation can be rearranged into a quadratic form, substituting in $G(C)$ from (2.18)

$$\left(C^{(p+1)}\right)^2 + \left(\kappa - C^k + \frac{\Delta t}{2}\gamma M^{(p+1)} + \frac{\Delta t}{2}\frac{\gamma C^k M^k}{\kappa + C^k}\right) C^{(p+1)} + \left(-\kappa C^k + \frac{\Delta t}{2}\frac{\gamma \kappa C^k M^k}{\kappa + C^k}\right) = 0. \tag{3.22}$$

Using the quadratic equation results in,

$$C^{(p+1)} = \frac{-b \pm \sqrt{b^2 - 4c}}{2} \tag{3.23}$$

for which,

$$\begin{aligned} b &= \kappa - C^k + \frac{\Delta t}{2}\gamma M^{(p+1)} + \frac{\Delta t}{2}\frac{\gamma C^k M^k}{\kappa + C^k} \\ c &= -\kappa C^k + \frac{\Delta t}{2}\frac{\gamma \kappa C^k M^k}{\kappa + C^k} \end{aligned} \tag{3.24}$$

Unless $b^2 - 4c = 0$, we have two different solutions to $C^{(p+1)}$. The problem with that is that if both solutions are positive we have two valid values to be used. Here, we can show that there will always be only one positive solution.

Proposition 3.2.3. *The quadratic equation defined as (3.22) will always have one positive solution and one negative solution for non-zero parameter choices.*

Proof. For the duration of this proof, we let $C := C^{(p+1)}$ to make equations easier to read. Rearranging (3.22) so that all the Δt terms are on the right-hand-side, we get

$$(C)^2 + \left(\kappa - C^k\right) C - \kappa C^k = \left(\frac{\gamma C^k M^k}{2\left(\kappa + C^k\right)} - \left(\frac{\gamma M^{(p+1)}}{2} - \frac{\gamma C^k M^k}{2\left(\kappa + C^k\right)}\right) C\right) \Delta t. \tag{3.25}$$

To simplify the notation, we let $\bar{a} := \frac{\gamma M^{(p+1)}}{2} - \frac{\gamma C^k M^k}{2\left(\kappa + C^k\right)}$ and $\bar{b} := \frac{\gamma C^k M^k}{2\left(\kappa + C^k\right)}$.

We analyze both the left-hand-side and right-hand-side independently by letting $f_l = (C)^2 + \left(\kappa - C^k\right) C - \kappa C^k$ and $f_r = \left(\bar{b} - \bar{a}C\right) \Delta t$. f_l is a quadratic equation with positive concavity everywhere and C-intercept at $-\kappa C^k < 0$. f_r is a line with a slope opposite to the sign of $\bar{a}$ and has C-intercept at $\bar{b}\Delta t > 0$.

There exist four cases here, each visualized in Figure 3.2 It is clear that since the f_l is a quadratic function and f_r is a linear function we have that f_l will attain a larger value at some value of C. Since f_l is quadratic we know there can only exist two intersections between f_l and f_r. Because we always have $f_r(0) > f_l(0)$, we can show, using the intermediate value theorem that there must exist a intersection for both $C > 0$ and $C < 0$. This means that we have exactly one positive solution and one negative solution since there is a maximum of two intersections. □

To determine which branch of (3.23) to use, we look at the function logically. If we know that one positive solution will always exist then the only branch for that to occur is the positive branch. This is because the square root term, $\sqrt{b^2 - 4c}$ will never be negative. The addition of two negative numbers cannot result in a positive solution therefore the positive branch must be used for the positive solution that we desire.

Now that computable solutions for M and C at a single time step have been found, an algorithm to solve for the next time step can be established. Algorithm 1 shows the organization of solving (3.11 - 3.10).

Note that Algorithm 1 actually describes both a fully-implicit and a semi-implicit method for solving (2.16). Recall that the final number of iterations is recorded as P, if $P = 1$ then only a single iteration

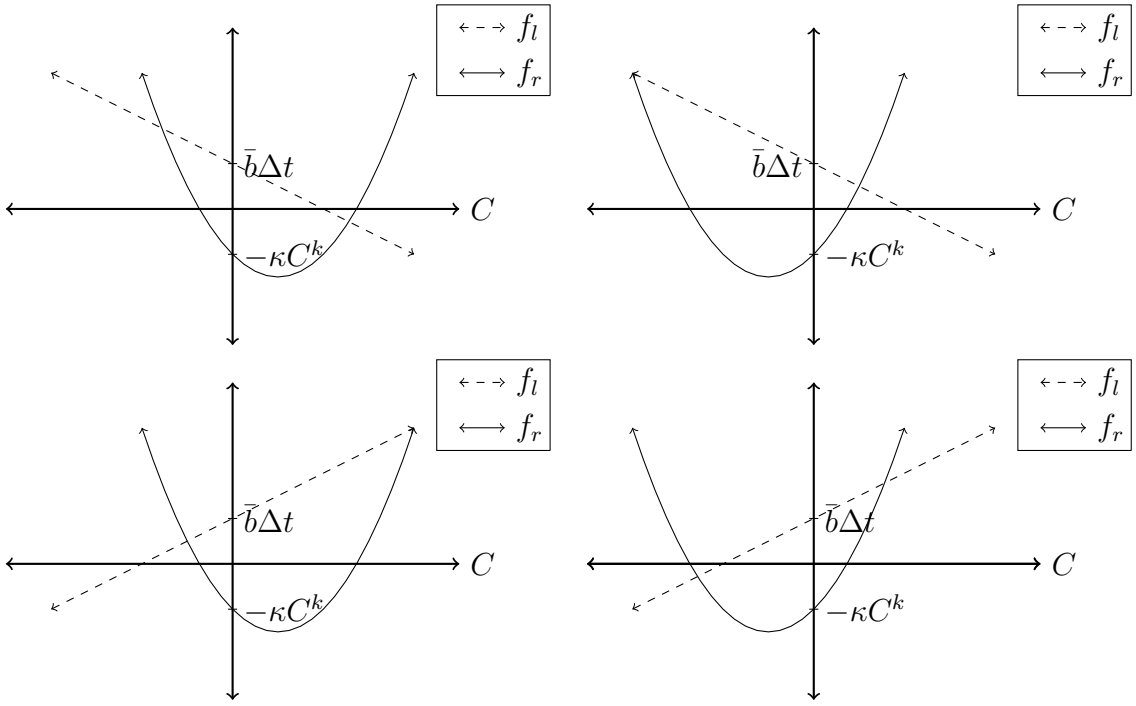


Figure 3.2: Graph of $f_l = (C)^2 + \left(\kappa - C^k\right) C - \kappa C^k$ and $f_r = \left(\bar{b} - \bar{a}C\right) \Delta t$ for all four possible cases. Notice that because $-\kappa C^k < 0$ and $\bar{b}\Delta t > 0$ for all realistic parameter values the two functions will always intersect in the positive C region. The top left graph is for $\bar{a} > 0$ and $\kappa - C^k < 0$. The top right graph is for $\bar{a} > 0$ and $\kappa - C^k > 0$. The bottom left graph is for $\bar{a} < 0$ and $\kappa - C^k < 0$. The bottom right graph is for $\bar{a} < 0$ and $\kappa - C^k > 0$.

Data: M^k, C^k are vectors with values from the previous time step and $p = 0$.
begin
  Let $M^{(p=0)} = M^k$ and $C^{(p=0)} = C^k$;
  while *Convergence is not achieved* **do**
    Solve $A^{(p)} M^{(p+1)} = \frac{1}{\Delta t} M^{(p)}$;
    Solve $C^{(p+1)} = \frac{1}{2}\left(b \pm \sqrt{b^2 - 4c}\right)$;
    Check convergence, (3.13;
    Let $p = p + 1$;
  end
  Let $P := p$;
  Let $M^{(k+1)} = M^{(P)}$ and $C^{(k+1)} = C^{(P)}$;
end

Algorithm 1: Algorithm for the fully-implicit solving of (2.16)

of the algorithm is applied, which correlates to the behaviour of the semi-implicit method. This can be produced by selecting a sufficiently large enough tolerance so that the convergence check, (3.13), is always resolved after the first iteration. The resulting semi-implicit method is effectively the method described in Sirca and Horvat (2012), which was first introduced in Eberl and Demaret (2007).

3.3 Computational Setup

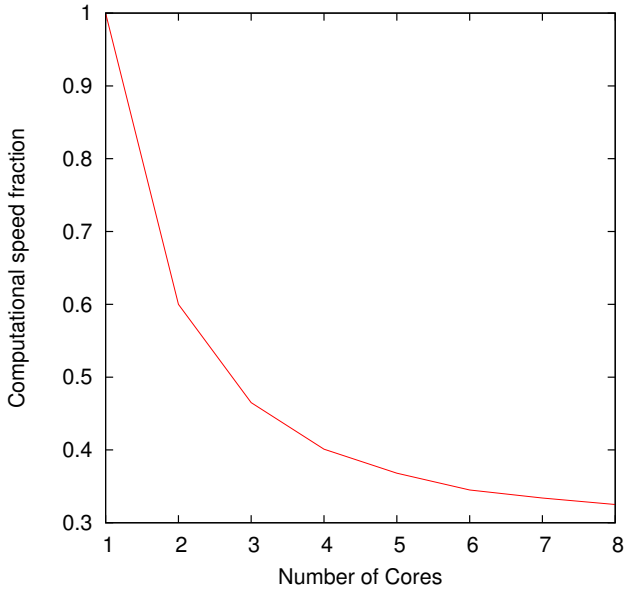


Figure 3.3: A graph showing the computation speed of simulations run with different number of cores. The y-axis represents the fraction of time each core has in comparison to the 1-core result. The simulation is the same setup as the travelling wave simulation that is defined in the next section. The simulation was stopped at $t = 2$ and was run with a grid of 2049×2049.

The implementation of Algorithm 1 was done with Fortran. The system matrix A in (3.16) is stored in the sparse data format called Compressed Diagonal Storage (CDS) (Barrett et al., 1987). For each iteration step the linear system is solved using the Conjugate Gradient method (Saad, 2003).

All the computations were run on a custom built workstation with an Intel Xeon CPU E5-2650 (1.2 GHz, 20MB cache size) and 32 GB RAM under Red Hat Enterprise Linux Server release 6.5 (Santiago). Running the computations with OpenMP, took advantage of 4 out of the 16 threads of the Intel Xeon CPU, with 2 threads to each core. The choice of 4 cores is that the computational gain for each additional core becomes less significant after 4, as shown in Figure 3.3. The GNU Fortran compiler,

version 4.4.7, was used for all computations; the compiler arguments were

```
-O3 -fdefault-real-8 -fopenmp
```

3.4 Method Validation

With a defined method and computational setup we assess the behavior and accuracy of the method in a variety of simulations. An examination of a typical simulation will show if the expected behaviour is observed. A convergence analysis for the method can be done to confirm that solutions from different grid sizes approach a single solution as they become more precise. This convergence test will also show the thresholds for an accurate simulation result, to help reduce the computation times. Once the fully-implicit method has been tested, it can be compared against the semi-implicit method.

3.4.1 Basic Simulations

Using Algorithm 1, simple scenarios can be tested as a first verification on the method.

A simple test would be to check if the spatial discretization can preserve specific characteristics of the solutions. One example of this would be seeing if a 1D initial condition could be preserved as time progresses. Having all of the biomass on one boundary of Ω, for example across the y-axis, would qualify as a 1D initial condition. These initial conditions will be defined as:

$$M(0,x,y) = \begin{cases} -\left(\frac{h}{d^4}\right)x^4 + h & \text{, if } y \leq d \\ 0 & \text{, otherwise} \end{cases} \tag{3.26}$$
$$C(0,x,y) = 1$$

where $h = 0.1$ and $d = \frac{5}{128}$. Here, h and d represent the height and depth of the inoculation site.

The solution shown in Figure 3.4 shows that the 1D characteristic of the biomass stays at a later time.

Another characteristic to observe would be if a spherical initial condition remains spherical. Using

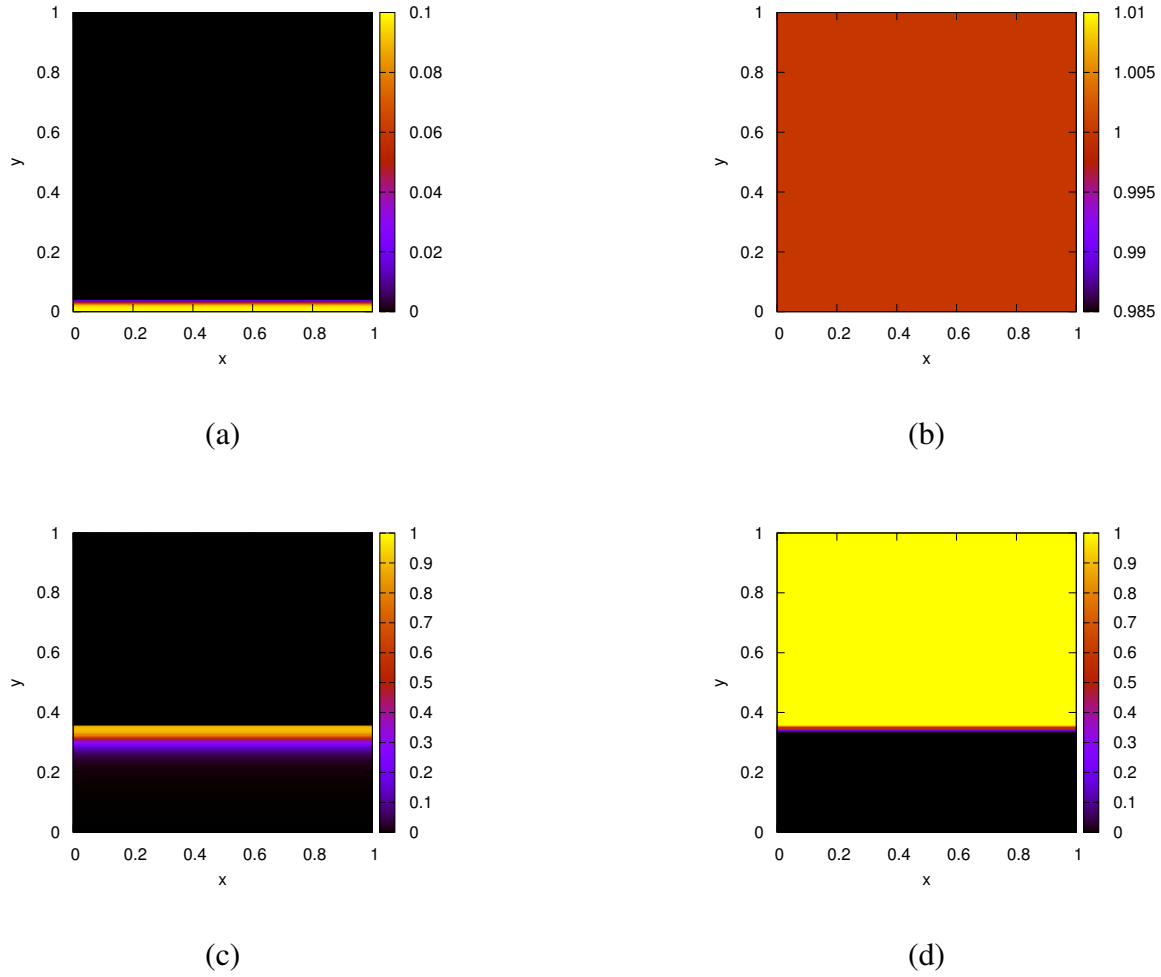
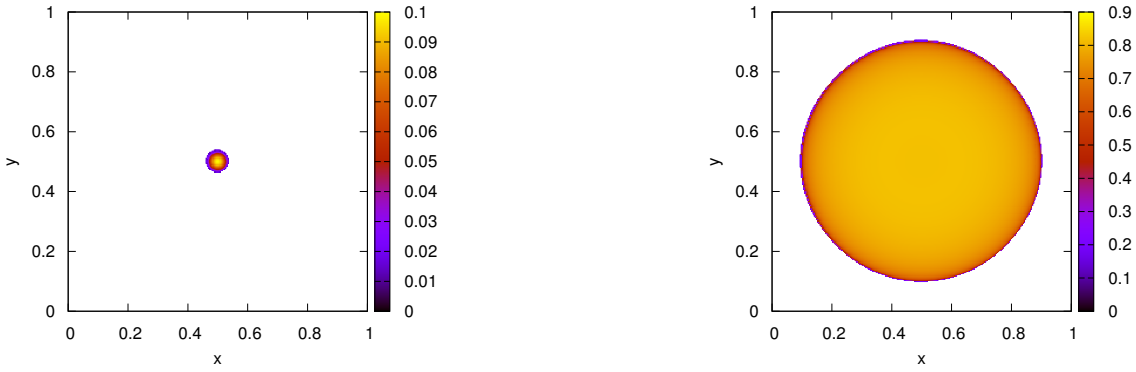


Figure 3.4: Solutions for (ac) M and (bd) C with 1D initial conditions defined in (3.26) at (ab) $t = 0$ and (cd) $t = 40$. Computed with a 1025×1025 grid and a time step of $\Delta t = 10^{-3}$.



(a) (b)

Figure 3.5: Solutions for M with spherical initial conditions defined by (3.27) at (a) $t = 0$ and (b) $t = 40$. Computed with a 1025×1025 grid and a time step of $\Delta t = 10^{-3}$.

initial conditions for the biomass,

$$M(0, x, y) = \begin{cases} -\frac{h}{d^2}\left((x-0.5)^2 + (y-0.5)^2\right) + h & \text{, if } (x-0.5)^2 + (y-0.5)^2 < d^2 \\ 0 & \text{, otherwise} \end{cases}, \tag{3.27}$$

a test can be tried to see if the spherical nature of the solution is kept as time progresses. We still have $C(0, x, y) = 0$ here. The solution shown in Figure 3.5 shows that the spherical shape of the solution is maintained at later times.

Both Figure 3.4 and Figure 3.5 increase the confidence that the spatial discretization did not introduce any loss of characteristics for the solutions.

The global mass conservation may be lost from possible sources or sinks of biomass caused by the implementation. To ensure this is not the case, the total amount of biomass can be used to compare the simulated amount against the theoretical amount. However, the total biomass cannot be exactly determined with the given growth rate function. This means that there will not be anything to measure the validity of the simulation solution against. If we let the growth rate be some constant, a, the exact total biomass can be calculated.

The expected total biomass, $T_M(t)$, can be found by using the divergence theorem and integrating over the region from equation (2.1) with $F(C) = a$,

$$\begin{aligned}
\frac{\partial M}{\partial t} &= \nabla_x(D(M)\nabla_x M) + aM \\
\implies \int_\Omega \frac{\partial M}{\partial t}\, dA &= \int_\Omega (\nabla_x(D(M)\nabla_x M)) + aM\, dA \\
\implies \int_\Omega \frac{\partial M}{\partial t}\, dA &= \int_{\partial\Omega} (D(M)\nabla_n M \cdot n)\, dA + \int_\Omega aM\, dA \\
\implies \int_\Omega \frac{\partial M}{\partial t}\, dA &= \int_{\partial\Omega} (\nabla_x(D(M)(0) \cdot n)) + \int_\Omega aM\, dA \\
\implies \frac{\partial}{\partial t}\int_\Omega M\, dA &= a\int_\Omega M\, dA \\
\text{Let } T_M &= \int_\Omega M\, dA \\
\implies \frac{\partial T_M}{\partial t} &= aT_M \\
\implies T_M(t) &= T_M(0)e^{at}
\end{aligned} \tag{3.28}$$

Numerically, this is computed by grid-wise summation,

$$T_M(t^k) \approx T_M^k = \frac{\sum_i^n \sum_j^m M_{i,j}^k}{nm}. \tag{3.29}$$

The simulation setup used will be analogous to that used for Figure 3.5. The one difference will be that the simulation here is ran for a longer time to allow the biomass to diffuse along the boundary, showing the boundary effects.

From Figure 3.6 we can see that the total biomass only differs between the computed value and the theoretical value by a relative error less then 0.003. The cases where the error becomes significant are from the region being completely filled with biomass, at which point diffusion is no longer possible. The error fluctuates violently here because of this. This suggests that the method does not introduce any significant sources or sinks of biomass at the boundary of the region.

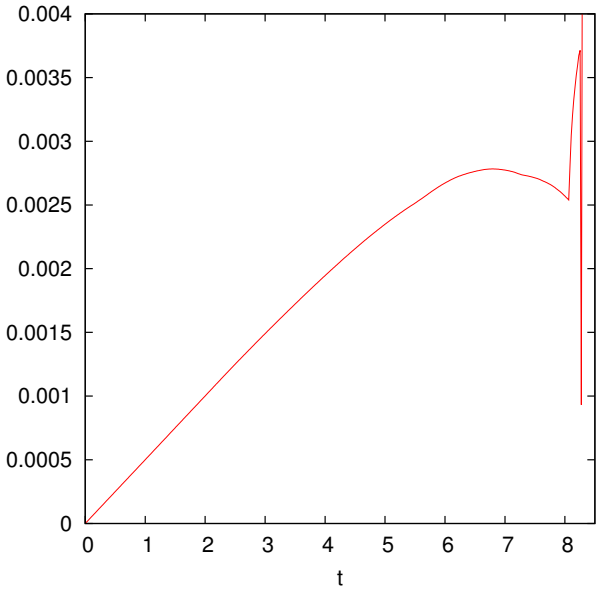


Figure 3.6: Plot of the relative error, $\frac{|f_1-f_2|}{|f_2|}$, between the computed total biomass, $f_1 = T_M(t)$, and the theoretical total biomass, $f_2 = y_0 e^x$. The changes after $t = 8$ are from the biomass having completely filled the region Ω. This means that there is no physical space for the biomass to occupy and thus the growth slows down to a stop.

3.4.2 Convergence Analysis

To validate the accuracy of the method, convergence analyses on the spatial discretizations will need to be made. Then the comparison between the semi- and fully-implicit method established in Algorithm 1 can investigated. First, a metric must be formed to enable consistent comparisons between different simulation solutions. This metric will be referred to as the normed difference. Only M will be considered for the normed difference calculations. This is because C depends on M and including it does not qualitatively change the results.

3.4.2.1 Normed Difference Computations

The normed difference is computed by taking the relative normed-difference between two solution in the following fashion:

$$\epsilon_{sol} = \frac{||u_1 - u_2||}{||u_2||} \tag{3.30}$$

where u_1 represents one simulation solution and u_2 references the solution that is expected to be more accurate. The theoretical accuracy of u_2 derives from the fact that most comparisons will be

done between solutions where one is trivially expected to be more precise. For our purposes, the solutions we compare will typically vary in only Δx or between semi- and fully- implicit. These are understood to have the relation that a smaller Δx, and that the fully-implicit method with the highest tolerance is to be more accurate. There is an assumption that both u_1 and u_2 have the same number of grid points, so that the difference can be taken grid-wise. In the case where there are difference between the number of grid points of u_1 and u_2, the coarser grid point refinement is for both u_1 and u_2. This is done by projecting the finer grid onto the coarser grid.

The results of the normed difference computations, named ϵ_{sol}, is a numerical value for the difference between two solutions. This depends on the norm used during the computations. Here three norms will be used:

$$\ell_1 : ||u||_1 = \frac{1}{nm}\sum_{i}^{nm}|u_i| \tag{3.31}$$

$$\ell_2 : ||u||_2 = \frac{1}{nm}\sqrt{\sum_{i}^{nm}(u_i)^2} \tag{3.32}$$

$$\ell_\infty : ||u||_\infty = \max_{i=1,\dots,nm}|u_i| \tag{3.33}$$

These different norms will all be used to create a broader understanding of the normed difference. This creates three distinct values for ϵ_{sol}, named ϵ_{ℓ_1}, ϵ_{ℓ_2}, and ϵ_{ℓ_∞}; each named for the norm used during the computation. Note that these norms are for the vector of the solution, through the use of the grid-ordering $\pi(i, j)$.

3.4.2.2 Grid Size Convergence

To observe the validity of the method, a test on the convergence of solutions based on the spatial discretization is done. This will involve using the same simulation described in (3.26) due to the simplicity.

The convergence will be tracked with only two forms of ϵ_{sol}; ϵ_1 and ϵ_2. This is because the value of ϵ_∞ does not vary with the grid size, since the wave front has a steep interface and tends to lead to

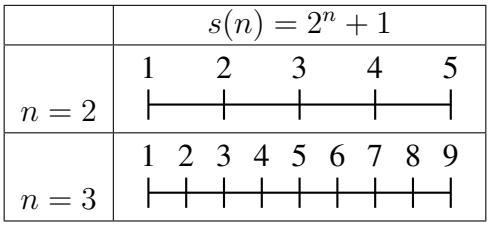


Figure 3.7: Visualization in 1D to illustrate the choice of $s(n) = 2^n + 1$ instead of 2^n for the grid size selection. Here is can be seen that successive grid size selections using $s(n)$ line up on certain grid points and when using 2^n no grid points are equivalent.

inconsistent changes in normed difference. Since the use of ϵ_∞ is not a suitable method for measuring the normed difference, the inconsistency does not suggest an invalidity with the method. Because of the difference in the number of grid points between different solutions, u_1 and u_2, only the grid points in the coarser refinement will be used. This places a limitation on the selection of grid-sizes since there must be some grid points locations that are the same for two different chosen grid sizes. For this purpose, we define the function $s(n) = 2^n + 1$ for $n \in \mathbb{N}$ to be used as the grid size selection function. Now certain grid points will match without the use of linear interpolation, as illustrated in Figure 3.7.

Using the same simulation setup as was done in Figure (3.4), solutions resulting from different grid sizes based on $s(n)$ are computed for $n = 5, 6, \ldots, 12$. In this case, when calculating $\epsilon_{sol} = \frac{|u_1 - u_2|}{|u_2|}$, we let u_1 be the grid size under investigation and u_2 be the solutions of the most refined grid size, $n = 12$. This would show the converging solutions for smaller grid sizes because the change to the finest grid size will be monotonically decreasing.

The results from Figure 3.8 show that the solutions converge as the grid size become refined with the grid size.

3.5 Comparison of Semi-implicit and Fully-implicit Method

We are now in a position to compare the fully-implicit time-integration scheme that we introduced here with the semi-implicit time-integration that has been used for problems with a diffusive substrate in the literature (Khassehkhan et al., 2009). Recall that the semi-implicit method is a special case

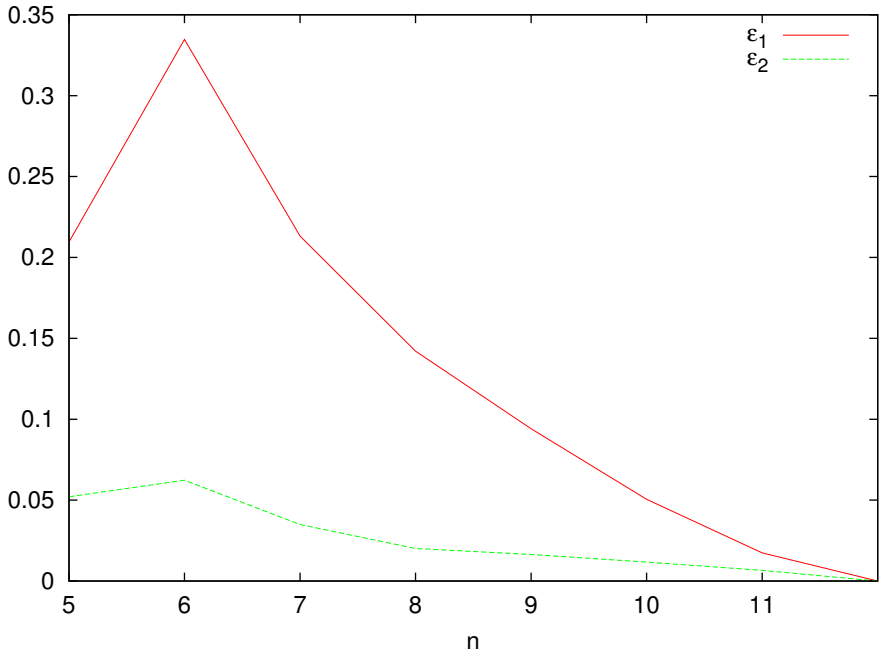


Figure 3.8: Plot showing the convergence of solutions based on changes in Δx. The computations are of ϵ_{ℓ_1} and ϵ_{ℓ_2} with grid-size following $s(n) = 2^n + 1$.

of the fully-implicit method if the non-linear iteration is stopped after one step. This can be forcibly achieved, for example, by choosing a very large tolerance threshold for the non-linear iteration.

The simulation used is the same as described in (3.4) and is stopped at $t = 40$. The comparison will be on multiple metrics: the average number of iterations of Algorithm 1, the value of ϵ_1 and ϵ_2, the computation time of the simulation, and the height of the wave peak.

The average number of iterations are tracked so that the amount of work done for each tolerance can be better quantified. It is based on the average number of iterations of the fully-implicit method taken over all the times steps, from $t = 0$ to the current t. This value is used since it represents the number of iterations in a way that is easy to read and understand. As the tolerance decreases the amount of iterations the algorithm must perform will increase, the degree of increase will help relate the amount of work.

The value of ϵ_1 and ϵ_2 act as a measure of accuracy. Here, these values quantify the difference between the solution of the semi-implicit method (Recall, Tol. $= 1.0e - 0$) which is u_1 and the solution of the fully-implicit method for different tolerances as u_2. This results in a relative difference from the

semi-implicit method and would show the change in solution as the tolerance decreases. Each row of Table 3.1 refers to the u_1 values used in the comparison. Each difference was taken at the last time step.

Along with accuracy, the simulation time is tracked. This is because it represents another metric for which the viability of the fully-implicit method can be verified. Intuitively there should be a decrease in the normed difference with the fully-implicit method as the value for tol decreases. Therefore, this needs to be weighted against the cost of computational intensity and the increase of the simulation time.

The location of the wave peak is a tracked quality of the solution that reveals how consistent the results are. The wave peak is described here as the maximum value of the solution at the final time step calculated. The ultimate goal is that the simulation solutions be converging towards the exact solution. To see this here the x-coordinate of the wave peak is tracked as well as the height of the wave peak.

The results of the method comparison can be seen in Table 3.1.

Tol.	Avg. Iter.	ϵ_1	ϵ_2	Time	Wave Height
1.0e-0	1.0000	0.000000000000	0.000000000000	12.1830	0.96366123
1.0e-1	1.0000	0.000000000000	0.000000000000	12.2080	0.96366123
1.0e-2	1.0000	0.000000000428	0.000000000167	12.3379	0.96366123
1.0e-3	1.0000	0.000000000428	0.000000000167	12.2310	0.96366123
1.0e-4	1.0000	0.000000001098	0.000000000329	12.3200	0.96366123
1.0e-5	1.9650	0.002573969658	0.001066499658	18.9869	0.96391491
1.0e-6	2.0000	0.002574057907	0.001066505860	19.0910	0.96391479
1.0e-7	2.0018	0.002573959764	0.001066498736	19.0940	0.96391492
1.0e-8	2.5856	0.002577916965	0.001066781565	20.5169	0.96390966
1.0e-9	2.9012	0.002577461054	0.001066759099	21.3080	0.96390979
1.0e-10	3.2278	0.002581334868	0.001067029069	22.2280	0.96390490
1.0e-11	16.0990	0.002632955188	0.001070709373	57.5589	0.96383105
1.0e-12	36.3184	0.002647234923	0.001071748013	113.9940	0.96380854
1.0e-13	57.6812	0.002648733848	0.001071857564	173.8489	0.96380614

Table 3.1: Results from running simulations with different Tol.

There are a number of observations that can be made from these results.

- A direct relationship between computation time and average number of iterations exists.
- There is no significant difference between solutions unless the tolerance is set high enough to force multiple iterations. This means the semi-implicit method results in a tolerance between 10^{-4} and 10^{-5} as any smaller tolerance does not require additional iterations.
- The differences in Wave Height are a result of additional iterations and monotonically approach a specific value as the tolerance becomes smaller.
- The greatest gain in accuracy while weighing the increased computation time is from a tolerance around 10^{-5} at which point only one extra iteration is completed.
- The main takeaway is that the semi-implicit method results in 4 digits of accuracy and when a second iteration is forced (from additional tolerance) a 5^{th} digit is gained for the 50% increase in computation time.
- After 36 iterations a 6^{th} digit of accuracy is gained for a 900% increase in computation time.

Chapter 4

Simulation Results

4.1 Typical Simulation

A typical simulation refers to the parameter values and the choice of initial condition. It will show the behaviour of the system under normal circumstances and help reveal interesting characteristics.

The typical initial condition attempts to emulate the biological situation of biomass growing inwards on a sheet of cellulose. This will show how the biomass moves and how two separate masses interact when merging. The initial condition used will initialize a number of random spherical inoculation points near the $y = 0$ and $y = 1$ axis. We let (x_r, y_r) be the random point used as the center for inoculation. To separate the inoculation points we have $x_r \in [0, 1]$ and $y_r \in [0, 0.1] \cup [0.9, 1]$. The number of inoculation points are the same for both the $y = 0$ region and $y = 1$ region. Multiple inoculation points combine additively. Each spherical inoculation point is computed as,

$$M(0, x, y) = \frac{-h}{d^2}\left((x - x_r)^2 + (y - y_r)^2\right) + h, \quad M \geq 0. \tag{4.1}$$

Note that $M(0, x, y)$ is for points within circles of radius d centered at (x_r, y_r), otherwise $M(0, x, y) = 0$. For the substratum we have $C(0, x, y) = 1$ everywhere.

The choice of parameter value is based on the default values given in Table A.1. There are 40 inocu-

lation points on each side, totalling 80. The fully-implicit method is used here with $tol = 10^{-8}$.

4.1.1 Biomass Ratio

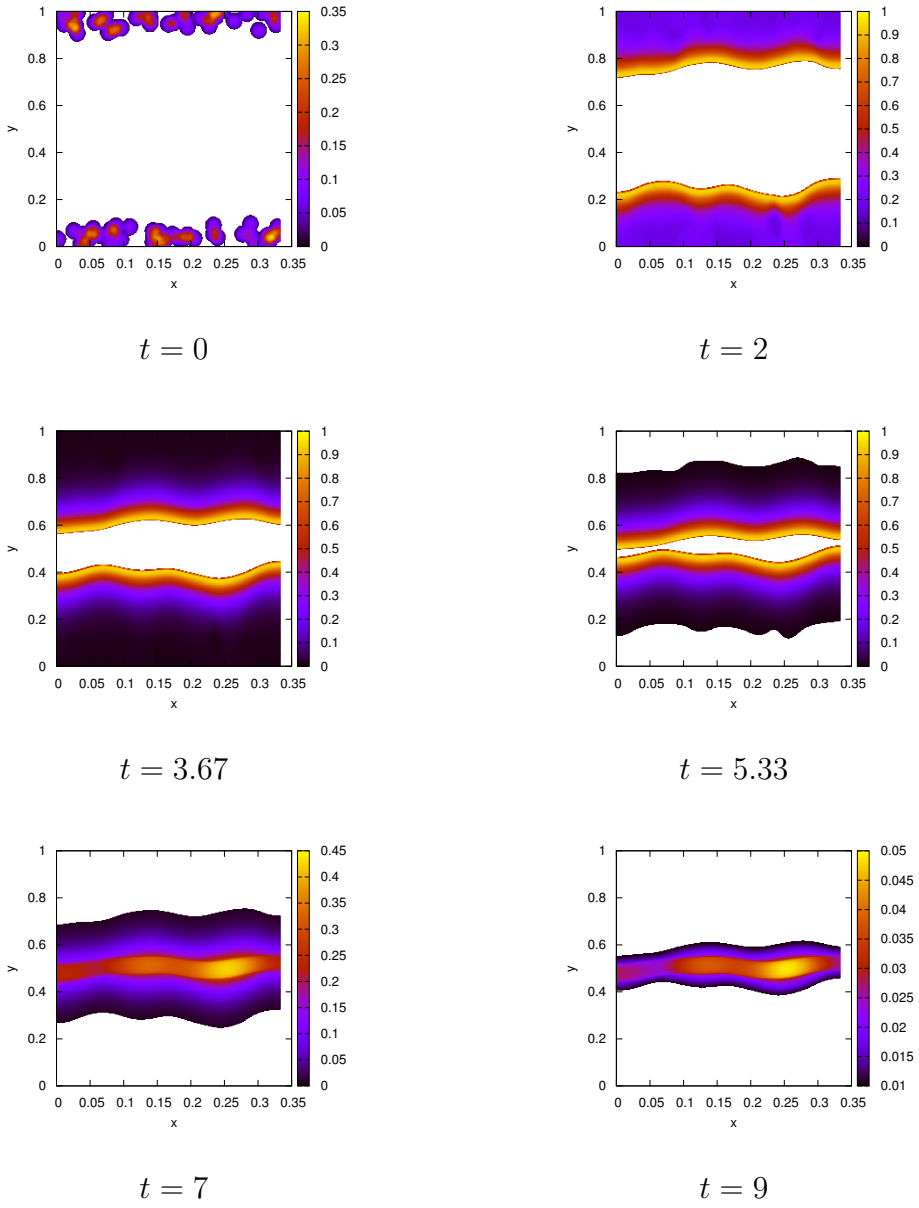


Figure 4.1: A graph showing the M solution of a typical simulation at different time steps. The initial condition is 80 random spherical inoculation points evenly divided between each of the $y = 0$ and $y = 1$ sides. A 513×513 grid was used.

Figure 4.1 shows the time evolution of M for the simulation. Here, the random inoculations on both sides of the region propagate towards each other and eventually combine at the center. By looking at

$t = 2$, $t = 3.67$, and $t = 5.33$ it appears as though the wave front is moving with a constant shape and at a constant speed. This suggest that there may be the existence of a travelling wave solution which will be explored in the next section.

One important feature to notice is that the time evolution in Figure 4.1 matches the conceptual model proposed in Dumitrache et al. (2015). This model can be seen in Figure 1.2. The different stages of the conceptual model can be observed in our simulation results:

- Stage I: $t = 2$ and $t = 3.67$ show the biomass growing towards the center of the sheet, which is the center white area.
- Stage II/III: $t = 5.33$ shows the consumed substrate region as the outer white.
- Stage IV: Not shown. Only occurs at the moment when the two bands first collide and the biomass concentration at that point still remains at the actual carrying capacity.
- Stage V: $t = 7$ and $t = 9$ show the combined center band, now at a biomass concentration lower then the actual carrying capacity.

4.1.2 CO_2 Production

Some important quantities to track are the total amount of biomass, M, and substrate, C. These approximations across the discretization will be called $T_M(t)$ and $T_C(t)$ to represent the total biomass and total substrate, respectively. The computation for these values can be done by integrating over the region, Ω:

$$T_M(t) = \int_\Omega M dA, \quad T_C(t) = \int_\Omega C dA \tag{4.2}$$

These values can be seen in Figure 4.3 (bd) for $T_M(t)$ and (c) for $T_C(t)$.

Since *C. thermocellum* produces CO_2 as the substrate is consumed, we can track the production of CO_2. Following the idea from Dumitrache et al. (2015), we can equate the change in production of

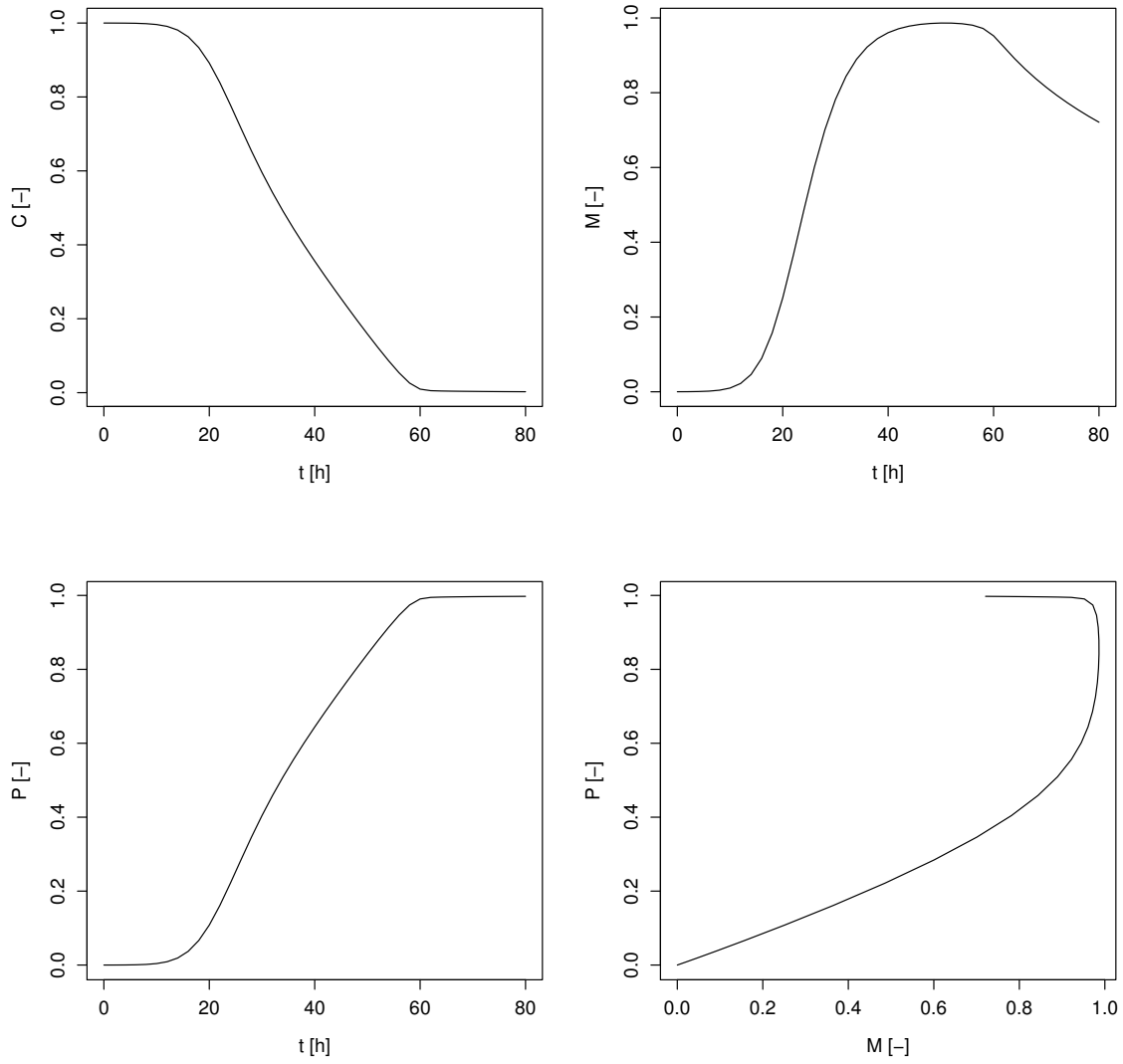


Figure 4.2: A typical model simulation from the simple ODE model. Shown are substrate concentration C (top left, normalised), effective sessile biomass M (top right, relative to the ideal carrying capacity M_∞), and CO_2 product P (bottom left, in moles) as functions of time t;. Also shown is the product P vs sessile biomass M (bottom right, in moles). Figure originally from Dumitrache et al. (2015).

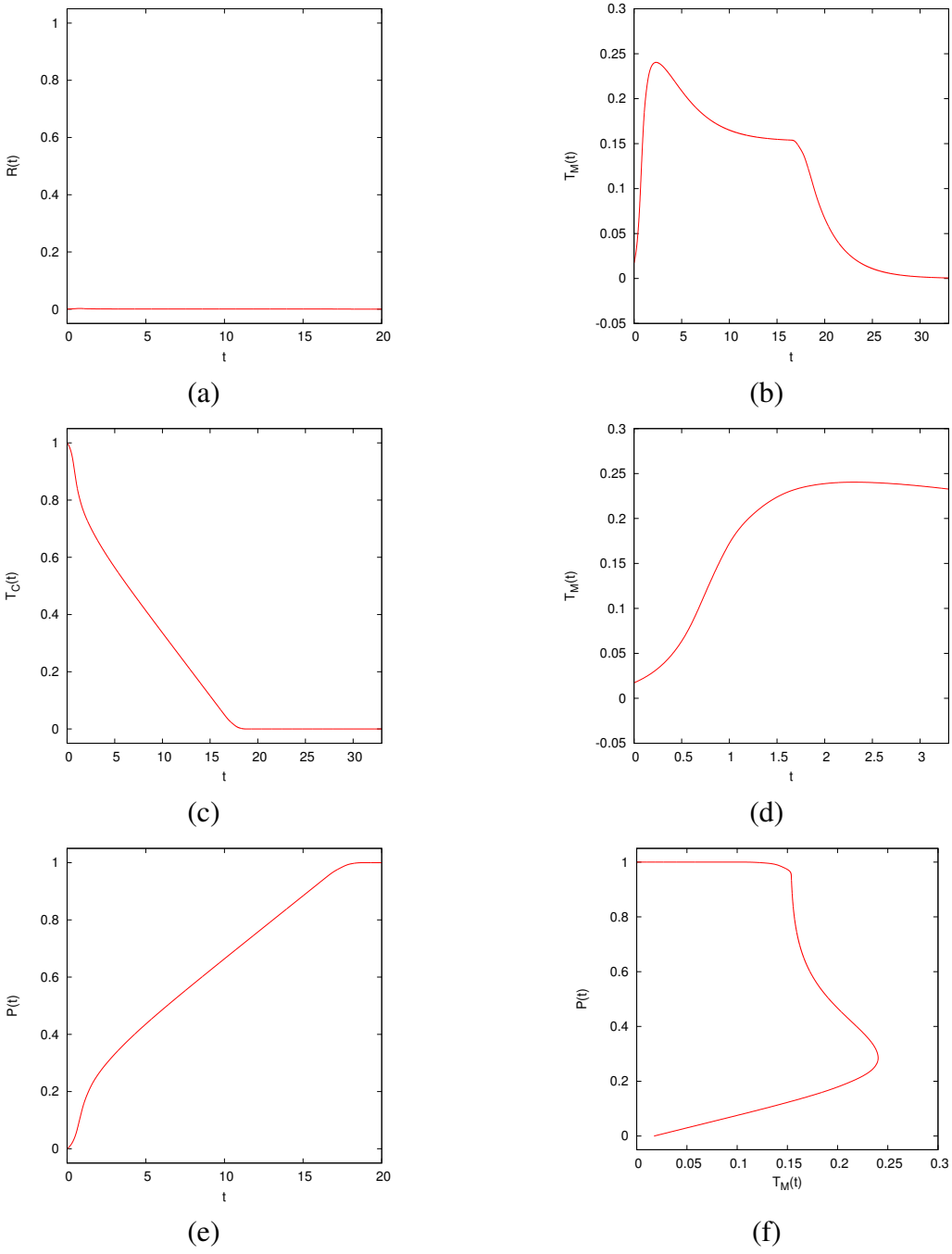


Figure 4.3: Total value of certain qualities from the typical simulation. Here we have: (a) $\mathcal{R}(t)$, the rate of CO_2 production, (b) total M as a function of time, (c) total C as a function of time, (d) total M as a function of time zoomed in from $t = 0$ to $t = 3$, (e) $\mathcal{P}(t)$, the total CO_2 produced, (f) $\mathcal{P}(t)$ as a function of total M. All the graphs are from the same simulation with initial condition of 40 random spherical inoculation points along the $y = 0$ side of the region and another 40 on $y = 1$. A grid of 257×257 was used for this graph. Default parameter set (Appendix A) was used except for $\delta = 10^{-8}$.

CO_2 as time changes by the following equation:

$$p_t = \rho G(C)M. \tag{4.3}$$

To get the amount of CO_2 produced at a specific time we get,

$$\mathcal{R}(t) = \int_\Omega p_t dA = \int_\Omega \rho G(C)M dA. \tag{4.4}$$

From this we can get the more useful value, the total CO_2 produced until this point.

$$\mathcal{P}(t) = \int_0^t \mathcal{R}(s)ds. \tag{4.5}$$

The CO_2 amount is calculated by letting $\rho = 1$ and using the numerically computed values for $G(C)M$ as a measure. For the same simulation as Figure 4.1, the CO_2 information can be seen in Figure 4.3 (a e).

The results from Figure 4.3 (c d e f) seems to match the results from the ordinary differential equation model proposed in Dumitrache et al. (2015). Their results can be seen in Figure 4.2. It is important to note that in our system $T_M = 1$ means that Ω is completely filled with biomass. However, in Dumitrache et al. (2015) they scaled the biomass to the ideal carrying capacity of biomass, i.e. they have $T_M = 1$ when all the biomass is in stage II or III. The overall result from this experiment is that the spatial two dimension model confirms the conceptual model from Dumitrache et al. (2015) based on which the reactor-scale model was formulated. The reactor-scale model consolidated the spatial effects into the carrying capacity of the growth and yet still managed to agree with the results of the actual spatial model.

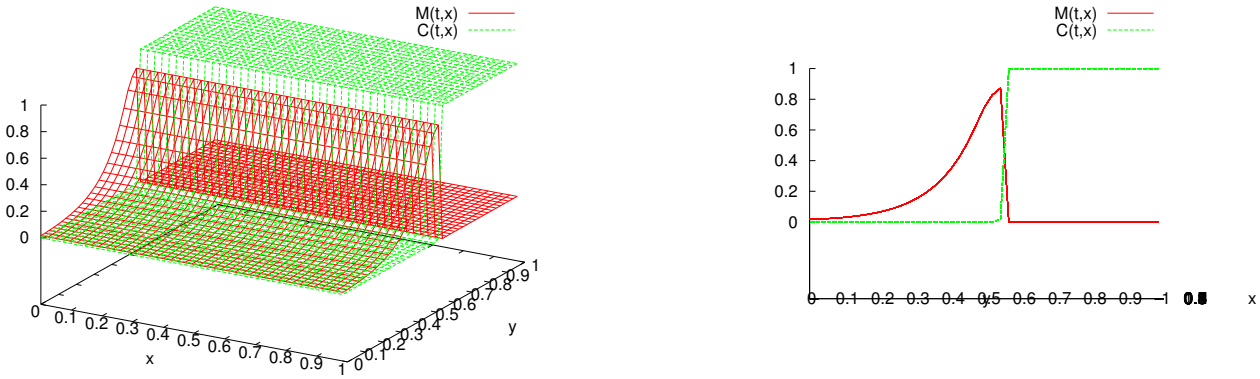
4.2 Travelling Wave Analysis

4.2.1 Spatial Simplification

To simplify the travelling wave analysis we reduce the spatial dimensions to that of a one dimensional problem. This can be done if initial conditions that are homogenous with respect to y are chosen. The purpose of this spatial simplification is that this will speed up the computations considerable. It will also make visualizations easier as certain figures would become too cluttered in two dimensions. What is done here is more of a pseudo-reduction of dimensions. By reducing the grids from an $n \times m$ grid to an $n \times 4$ grid we have changed the way the problem size scales with finer grids. The problem is still two dimensional, just now one dimension has been reduced to only 4 grid points of accuracy instead of m points. This does not effect the final result since we only apply this change to problems with appropriate initial conditions. These initial conditions are homogenous in the y direction and thus we do not have any fluctuation between y values for a given x value.

One main benefit of changing the grid from $n \times m$ to $n \times 4$ is that the growth of the problem with respect to the resolution of the grid is reduced dramatically. This changes the problem from a $O(n^2)$ problem to a $O(n)$. Using the one dimensional travelling wave initial conditions, (3.26), one simulation is computed with a 513×513 grid, seen at Figure 4.4, and another with a 513×4 grid, seen at Figure 4.6.

Before any changes to the grid can be made, it must be confirmed that fluctuations are sufficiently small. To this end, the standard deviation is used as a measure. The standard deviation is calculated along the y-direction for each x value. This gives a numerical quantity for the measure of dispersal each y value has with another. Here, we use the sample standard deviation for the sole reason that this single simulation does not represent its own population. Initially, at $t = 0$ the standard deviation is 0 everywhere (DATA NOT SHOWN). At $t = 40$, Figure 4.5 show the standard deviation of each y value. After many time steps have passed the amount of spread is always less then 10^{-14}, which is an acceptable degree of consistency. Note that the main inconsistency is at the wave front, around



(a) (b)

Figure 4.4: Graph of (a) 3D view of $M(t, x, y)$ and $C(t, x, y)$, (b) Side profile view of $M(t, x, y)$ and $C(t, x, y)$ at $t = 40$.

723 $x = 5.75$, which is mainly because of the sharp change in values.

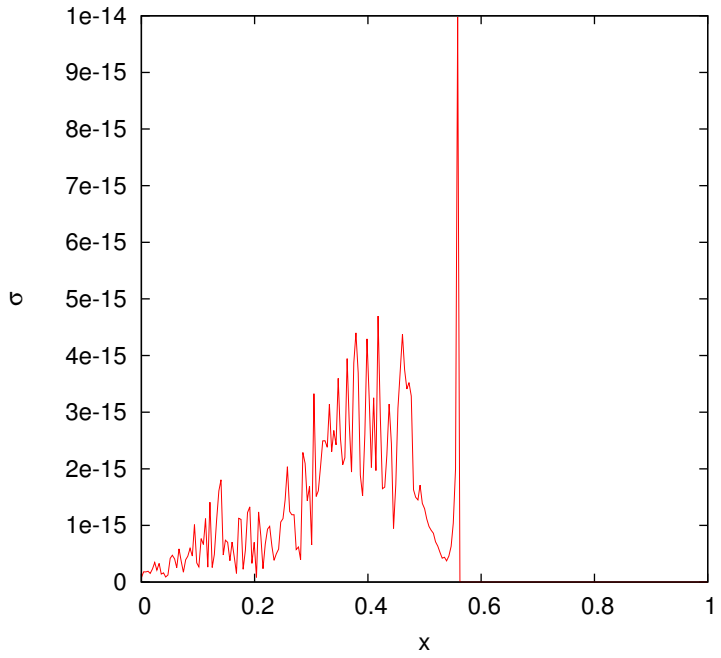


Figure 4.5: The standard deviation at the same time as the above graphs

724 When simulations are computed with a $n \times 4$ grid, a two dimensional solution is still computed.
725 With regards to visualizations, side profiles could be used on these solutions to present pseudo-one
726 dimensional visualizations; this is not ideal. To visualize the solutions in true one dimension we use
727 $\bar{M}$ and $\bar{C}$ as averaged values of the solutions along the y-axis. This is computed after the solution has

been determined and is independent of the actual computations for M and C. So by taking the average of the points along the y-axis we can get a two dimensional plot as seen in Figure 4.6.

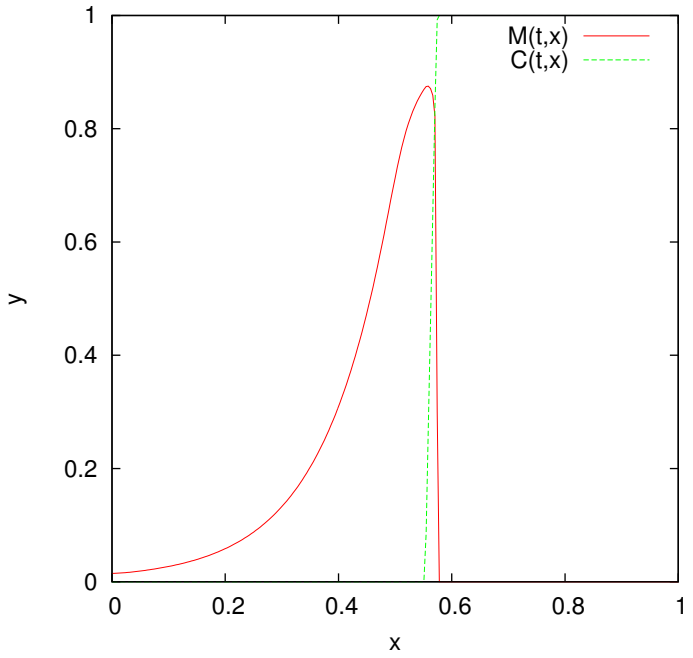


Figure 4.6: Graph of M(2,y) and C(2,y), now reduced to a one dimensional solution.

This means that the system (2.16) - (2.18) can be reduced to a one dimensional problem. With initial conditions that are homogenous with respect to y, we can greatly reduce the required number of grid points in the one axis. Once the y-axis reduced, we can also ignore it for visualizations, only using the x-z axis and plotting the values of $\bar{M}$ and $\bar{C}$.

4.2.2 Travelling Wave Solution

Classical travelling wave solutions are solutions that propagate with an *a priori* unknown constant speed without any change in shape. This means that the solutions can be defined as

$$M(t,x) = M(x - ct) \tag{4.6}$$

Figure 4.7 shows the time evolution of the single time snapshot from Figure 4.6. Given the above definition and by looking at the consistent appearance of the solution, it suggests that it is a travelling

wave. It is clear here that the shape of the solution is consistent enough to suggest the existence of a travelling wave solution.

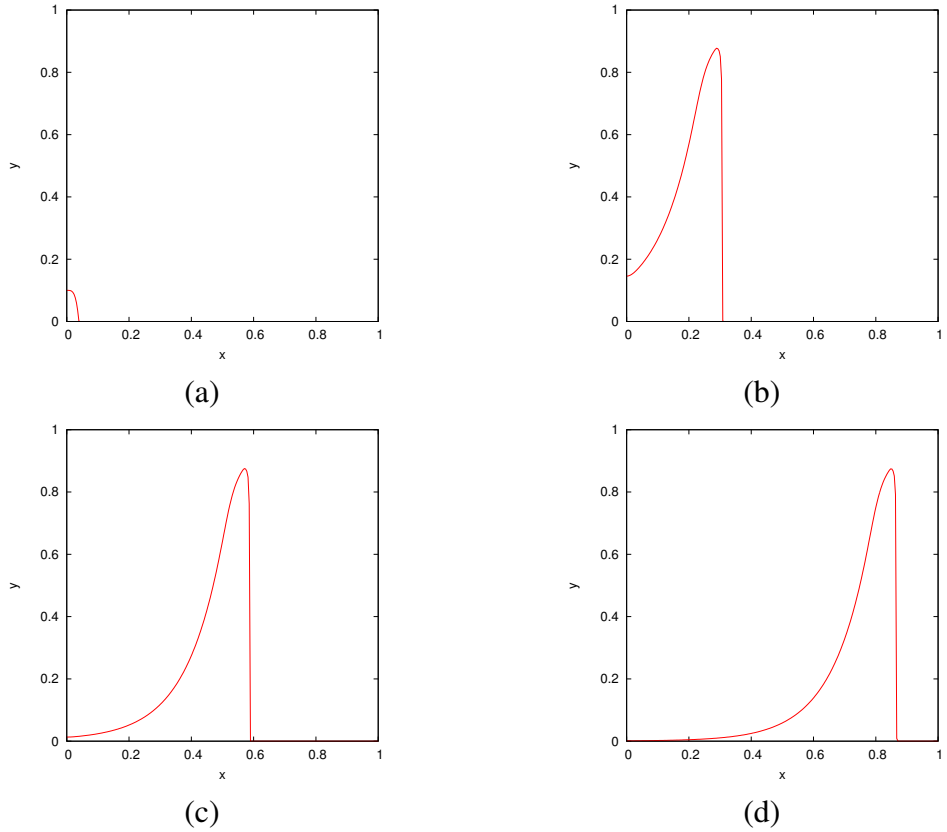


Figure 4.7: Solutions of $M(x,t)$ and $C(x,t)$ at (a) t = 0, (b) t = 20, (c) = 40, (d) = 60. This was run on a 513×4 grid.

The existence of a travelling wave solution for this simulation can be confirmed if the solution $M(x,t)$ can be shown as $M(x-ct)$, where c is the *a priori* unknown wave speed. This can be graphically confirmed by horizontally translating the solution at different time steps on top of each other. If the superimposed solutions are of similar shape and have been translated by multiples of the same value then evidence of a travelling wave would be shown. This would suggest that the value used for horizontal translations is an approximation for the wavespeed c. We can numerically approximate the value for c by looking at how fast the peak of the wave travels. The location of the wave peak is the x coordinate that corresponds to the largest M value. Recall that we are dealing with a pseudo-one dimensional problem, so there does not need to be any consideration for an (x,y) coordinate. For this case, we used the GNUPLOT software to fit a linear model, $f(x) = mx + b$, to the last half of the wave peaks path, seen in Figure 4.8. The last half of the values were used instead of the whole set

of values because only for the former do we have a fully formed travelling wave. The value of m in $f(x)$ is the approximation for the wave speed, c.

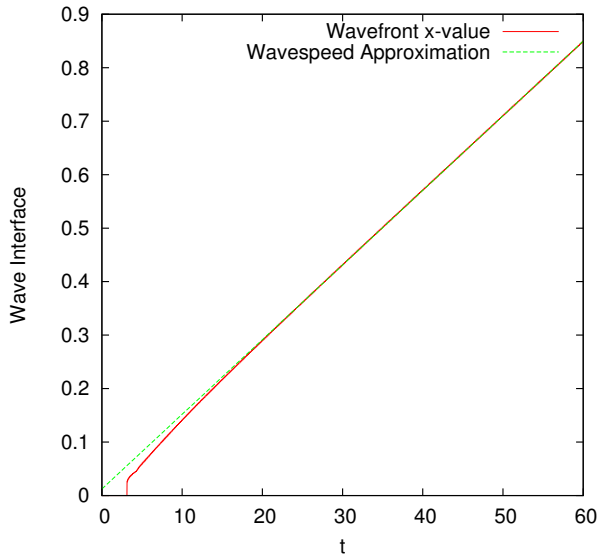


Figure 4.8: The x location of the wave peak as a function of t. The red line is the wave peak location extracted from the simulation results. The green line is the function $f(x) = cx + b$ with c as the wave speed, found by fitting the model to the second half of x values. The simulation results used here are from the solution shown in the previous Figure.

With an approximation for c, the solutions of Figure 4.7 can be represented as $M(x - c(t_0 - t_n))$, where $t_0 = 60$ is a reference point for the other time steps. The values of t_n are the times for the other solutions. By translating along the x-axis multiple solution profiles can be superimposed, as seen in Figure 4.9. The shape of each time step is very similar throughout, only differing slightly at the tail.

Based on the above evidence, we can say that a travelling wave solution has been suggested to exist numerically for a single initial condition and particular set of parameters. This leads to two logical extensions, looking at the stability of the travelling wave solution based on initial condition and investigating the effect the parameters have on the travelling wave solution.

4.2.3 Travelling Wave Stability

Based on the previous example, there seems to exist a travelling wave solution for the intial condition given in (3.26). The next step is looking at how different initial conditions could still result in a travelling wave solution. For this we specifically look at the stability of the solution, does it attract

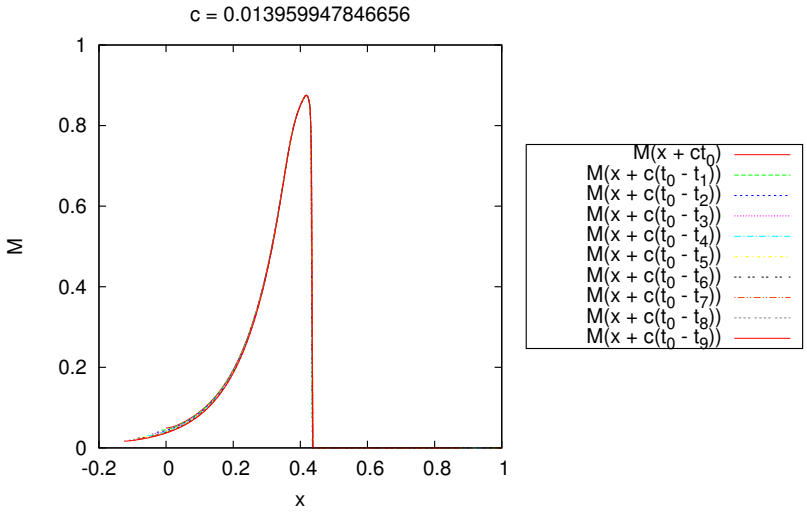


Figure 4.9: Solutions of M that are represented as $M(x - ct)$ *a priori*. The multiple time steps are translated on top of another by horizontal movements of $c(t_0 - t_n)$ for each time step.

each nearby solution into becoming a travelling wave solution or do only special cases become travelling wave solutions. This will help confirm that the existence of the travelling wave solution is not dependent on the single choice of initial condition.

To test this we take an initial condition that is not inherently one dimensional and see if it approaches the one dimensional property. The choice of IC is to have multiple random spherical inoculation points along the $y = 0$ side of the region. Specifically, we use (x_r, y_r) to represent the center of each random inoculation point. Here $x_r \in \mathcal{R}$ and $y_r \in [0, 0.1]$.

The equation used for each random spherical inoculation point is,

$$M = \frac{-h}{d^2}\left((x - x_r)^2 + (y - y_r)^2\right) + h, \quad M \geq 0. \tag{4.7}$$

Random inoculation points add to each other if they overlap. After all the inoculation points have been generated, every value is divide by the total amount of biomass. This lets the initial condition become a representation for the distribution of random inoculation points in terms of the total amount generated. A time evolution of the simulation with the above initial condition can been seen in Figure 4.10. Here it can be observed that the solution M appears to slowly converge to a one dimensional

problem. This cannot be fully seen since the wave propagation reaches the end of the region before it can become fully one dimensional.

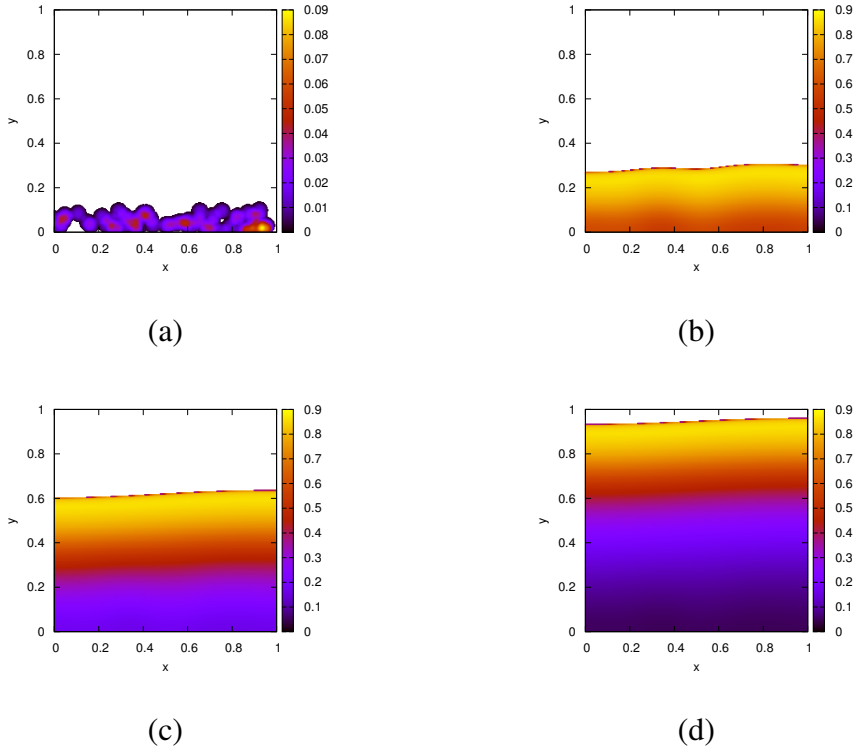


Figure 4.10: Plots of the simulation with random spherical inoculation points centered in the region $(x, y) \in [0, 0] \times [1, 0.1]$. The solutions are shown at (a) $t = 0$, (b) $t = 10$, (c) $t = 20$, and (d) $t = 30$. Each solution is computed on a 513×513 grid.

We can quantitatively see the behaviour of this convergence by calculating the measure of spread at the wave front. This can be achieved by calculating the standard deviation of y coordinated for each x coordinate. By tracking the largest y value with a non-zero M for each x value we can generate a sample data set of the wavefront. The wavefront is used instead of other points of interest, such as the wave peak, because it is the most consistent of characteristics that can be easily tracked. As seen in Figure 4.5, the wave peak had the largest spread among all other values.

Taking the sample standard deviation of this set results in the measure of spread for the wavefront. The sample standard deviation was used since this one example does not represent the whole population of solutions. The idea is that, for a solution that converges to one-dimensionality, the y location of the wavefront should be converging to similar values. This means that the standard deviation would

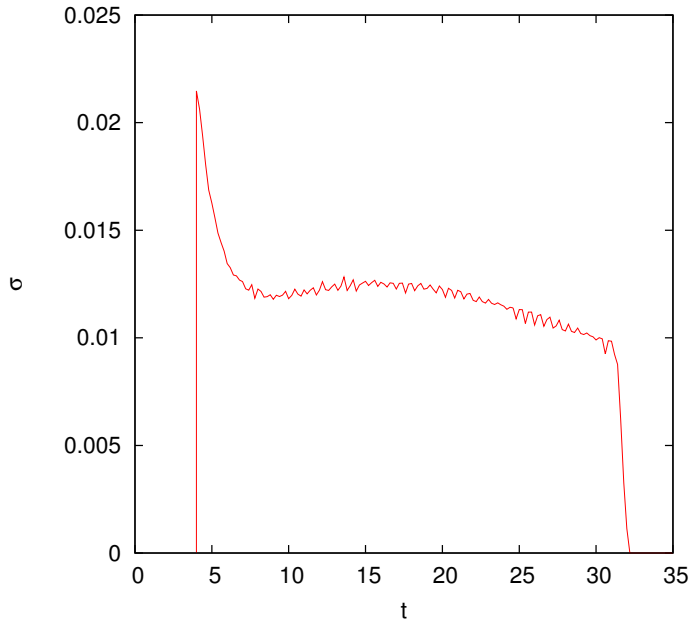


Figure 4.11: The standard deviation of the wavefront interface as a function of time. The wavefront references the largest y coordinate with $M > 0.001$ for each x coordinate. The choice of using $M > 0.001$ is because we want to ignore the small values (10^{-100}) that arise from the diffusion right at the wave front. This simulation is the same as the previous Figure.

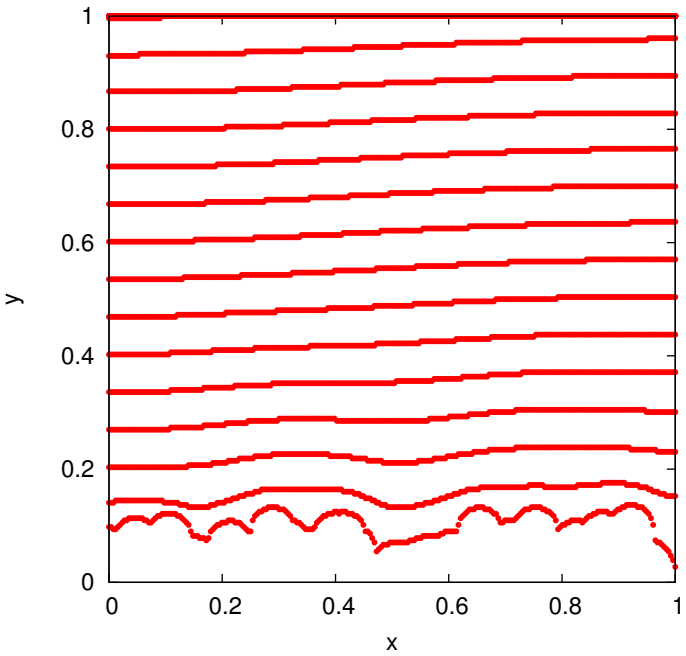


Figure 4.12: The wavefront shape of multiple time steps. Each wavefront has a difference in time by 2, i.e. they are at $t = 4, 6, 8, \ldots, 58, 60$. The simulation results are the same as the previous Figures, using the default parameter values with a grid size 513×513.

converge to zero. The standard deviation of the wavefront as a function of time of the simulation ran in Figure 4.10 can be seen in Figure 4.11. Here is shown that the solution is converging to zero, however not monotonically.

For the numerical computation of the wavefront, the largest y values greater then 0.001 was used instead of 0. The reason is that there are very small values of around 10^{-200} that arise due to the diffusion that were not adequate representations of the wavefront.

Another interesting item to investigate is the actual shape of the wavefront. Figure 4.12 shows only the wavefront shape for multiple time steps. The wavefront shape is the same dataset of points used to calculate the standard deviation of the wavefront interface. Of interest is that the wavefront seems to move at a constant speed, since each wavefront shown is equidistance from the next.

4.2.4 Parameter Effect on Wave Speed

The travelling wave solutions seen before have all existed for a single set of parameters. Here the four main system parameters, δ, κ, ν, and γ are independently varied and the effect on the travelling wave solutions are observed. From this we can also see how the wave speed of the travelling wave solution changes as a function of the different model parameters.

For this an automated script was created that checks, for each time step, if M is travelling wave solution based on the solution M from a number of time steps previous. From this check, a wave speed needs to be approximated based on the distance between the two solutions. If this approximated wave speed is matched throughout all the x values, then a travelling wave solution is assumed to exist at that time step. With this script, we can try the same simulation as Figure 4.7 with different parameter values.

For each parameter, δ, κ, ν, γ, the range of values chosen were arbitrarily. Figure 4.13 shows the results of the wave speed for each parameter changes. Mainly it was so that the solution did not propagate too fast and hit the end of the region before developing into a full travelling wave solution. Generally when the travelling wave does not form it is because the wave front propagates to the end

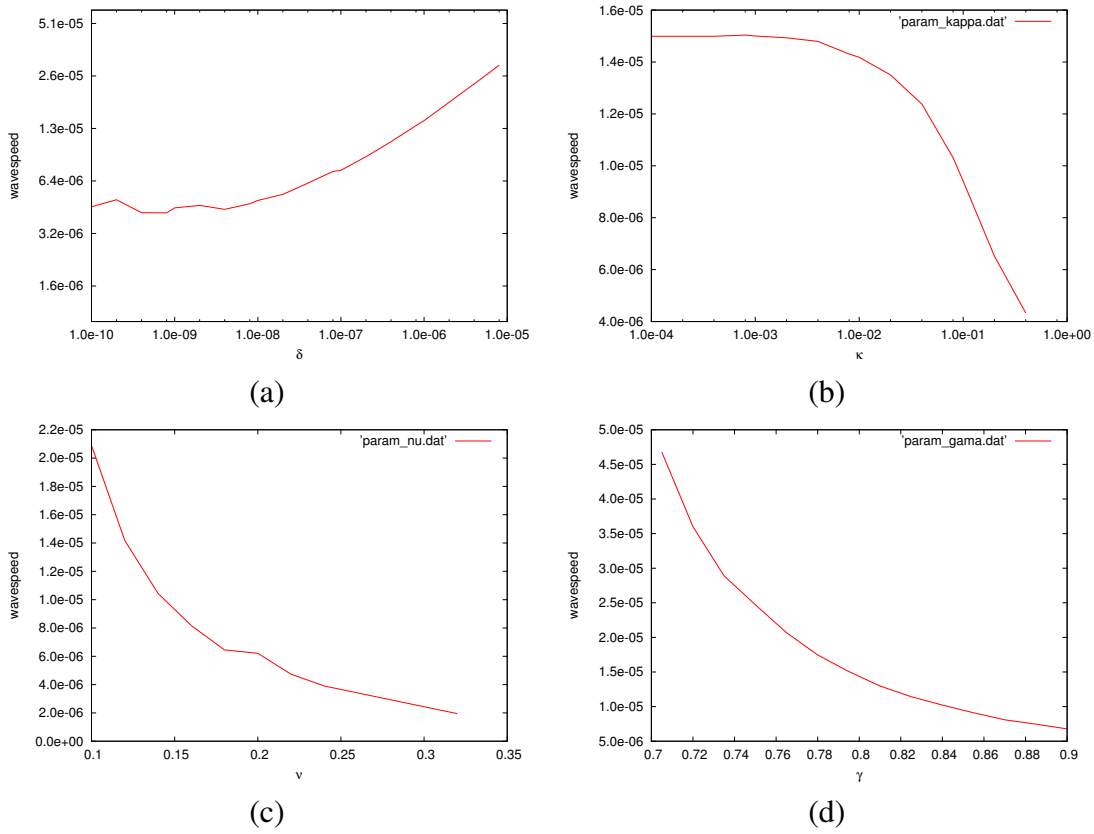


Figure 4.13: The value of c as parameter (a) δ, (b) κ, (c) ν, and (d) γ are changed. Note that (a) and (b) have logscales due to the selection of parameter values. Each of these were calculated with the same setup as the travelling solution previously done. The grid size for each was 513×4 and a time step of $\Delta t = 0.001$ was used.

of the region faster then the tail of the travelling wave can decrease to 0. In the case of ν, any larger values than the selected range resulted in biomass that died faster then it could grow, and thus no travelling wave solution exists. There did not appear to be any cases were a travelling wave solution could not form.

The behaviour of the wave speed as a result of changing the parameters can be explained by the biological meaning of each parameter. For δ, the diffusion constant, a large value results in a larger local biomass growth as a result of diffusion. This speeds up the spreading of the biomass and the propagation of the interface. For κ, the half-saturation concentration, this value depicts at what substrate concentration we achieve half-maximum growth for the biomass. When this value is large, the required amount of substrate for optimal growth speed is increased and thus the overall growth of the biofilm is slowed down. For ν, the decay and loss rate of biomass, a larger value results in more biomass being ejected from the system and thus the amount of biomass available to grow become smaller and growth propagations are slowed. For γ, the biomass yield coefficient, a larger value correlates to a substrate that is quickly consumed which produces less total biomass for growth. This inhibits the propagation of the biomass interface and lowers the wave speed. The simulated values recorded in Figure 4.13 agree with the expected behaviour for each parameter.

4.3 Spatial Effects

There are many spatial effects that can exist here because of the diffusion term in the biomass. We now try observing any differences in the behaviour of the system based on spatially different initial conditions. This will show if there is any noticeable differences in the behaviour of the system based solely on the placement of initial biomass. There will be two methods to compare the different simulations: visually and by monitoring the amount of CO_2 produced. The visual inspection is a logical comparison to use, the CO_2 is chosen since it essentially provides a measure of the activity in the system. The CO_2 is also used in the experiments conducted in Dumitrache (2014). Looking at the CO_2 production, a lumped measurement of the biomass activity, shows whether local spatial effects change the global reactor scale behaviour.

To measure the difference, two simulations will be run with varying initial conditions. One simulation will have the initial condition evenly spread along one side of the region. The other will have all the initial condition clumped in single corner. This will replicate the two possible extremes for the location of biomass.

Since the simulation is comparing the difference between two initial conditions, the initial amount of total biomass, $T_M(0)$, must be the same between both simulations. The equally dispersed initial condition needs to have as much surface area exposed as reasonably possible. To this end, num spherical inoculation points, equidistance from each other, are used along the $y = 0$ side of Ω. Here we use (x_e, y_e) as the center of the evenly distributed inoculation points. The value of (x_e, y_e) depends on the number of points chosen for the simulation. Each spherical inoculation point is computed as,

$$M(0, x, y) = \frac{-h}{d^2}((x - x_e)^2 + (y - y_e)^2) + h, \quad M \geq 0. \tag{4.8}$$

Note that $M(0, x, y)$ is for points within circles of radius d centered at (x_e, y_e), otherwise $M(0, x, y) = 0$. For the substratum we have $C(0, x, y) = 1$ everywhere.

For the clumped initial condition, we choose another spherical inoculation point so that it is similar to the evenly distributed initial condition. Here, we need only a single inoculation point, centered at the corner $(0, 0)$. The choice of inoculation center is so that the initial biomass is as concentrated as possible, while still retaining a spherical shape. The initial condition for the clumped biomass is as follows,

$$M = \frac{-1}{(2hd^2 \cdot num)^{\frac{1}{3}}}(x^2 + y^2) + (2hd^2 \cdot num)^{\frac{1}{3}}. \tag{4.9}$$

Here the coefficients have been chosen so that the two initial conditions have the same amount of biomass. The value of num is to represent the number of inoculation points in the evenly distributed initial condition.

Figure 4.14 - 4.15 shows the time evolution of both initial conditions. Here we arbitrarily select $num = 6$.

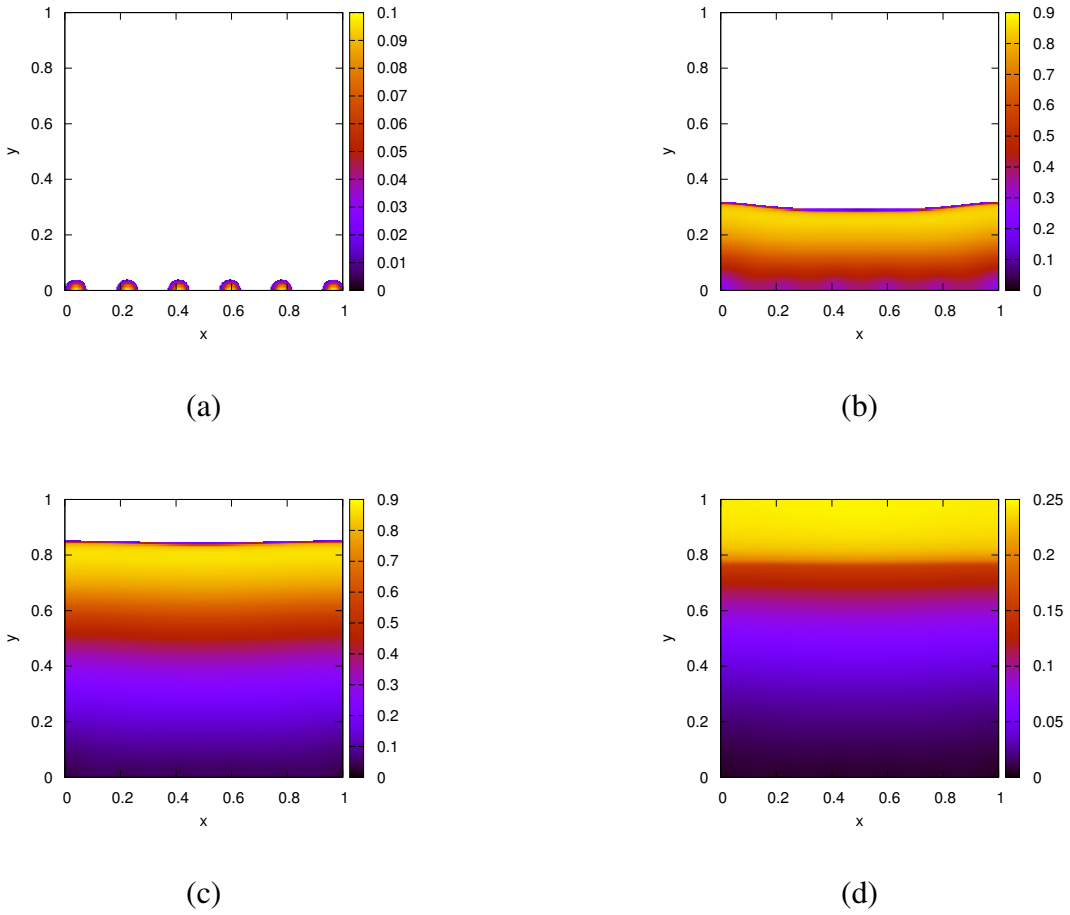


Figure 4.14: This shows the time evolution for the evenly distributed initial condition at (a) $t = 0$, (b) $t = 16$, (c) $t = 32$, (d) $t = 48$. Here default parameters are used on a grid size of 513×513.

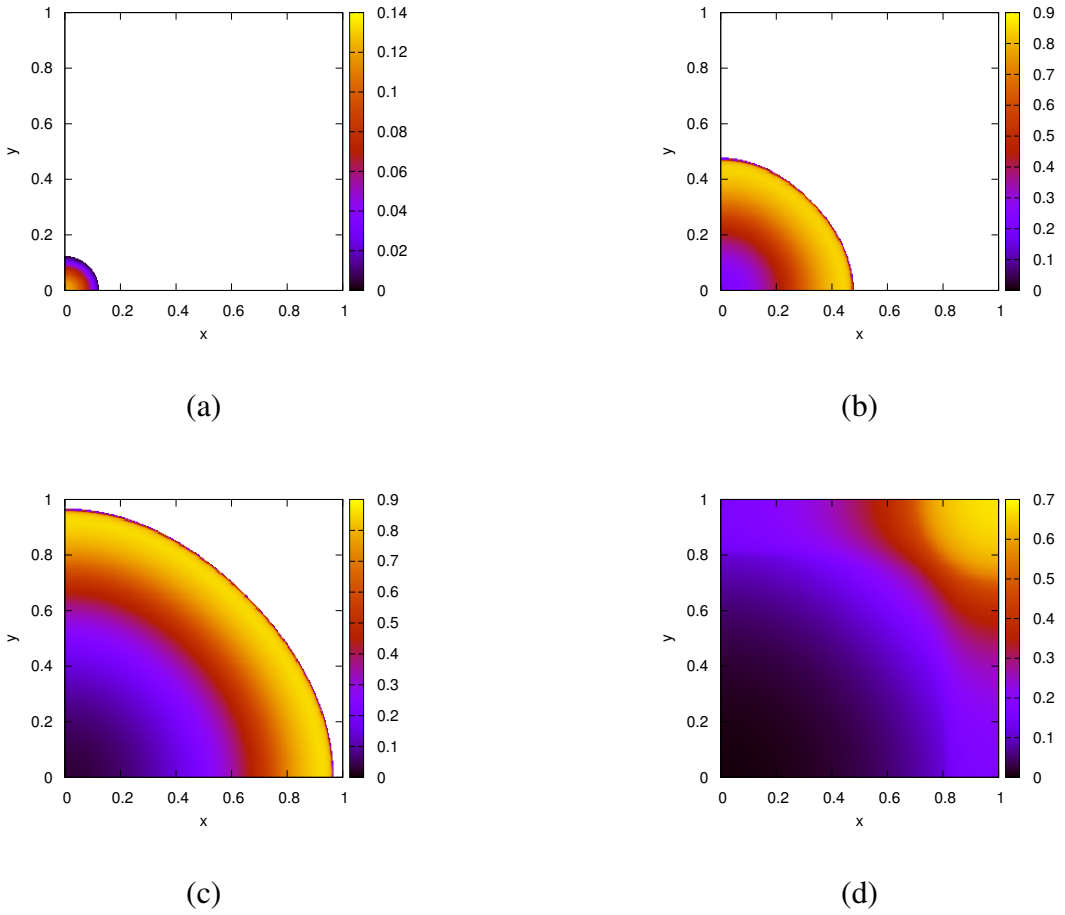
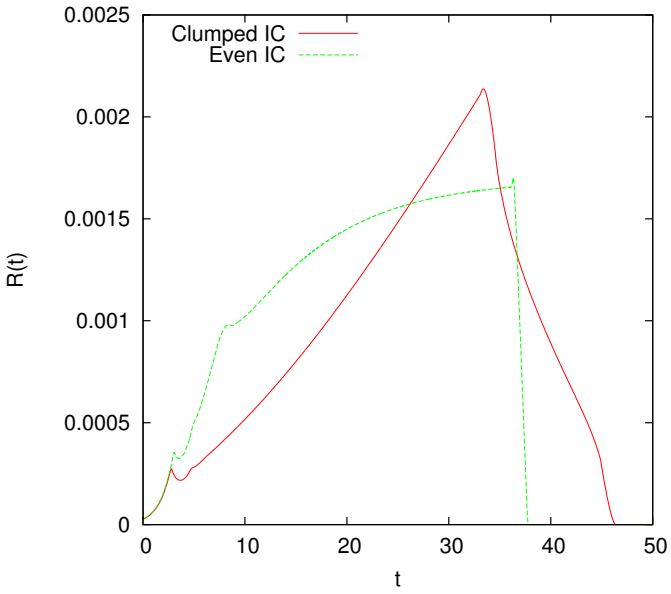
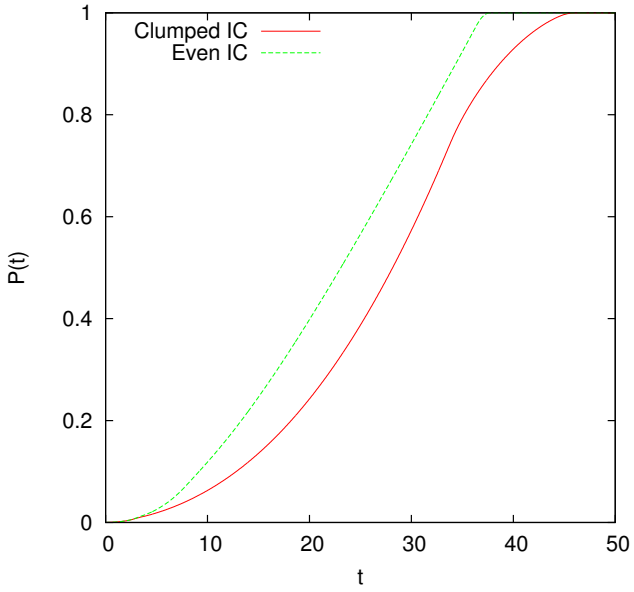


Figure 4.15: This shows the time evolution for the clumped initial condition at (a) $t = 0$, (b) $t = 16$, (c) $t = 32$, (d) $t = 48$. Here default parameters are used on a grid size of 513×513.



(a)



(b)

Figure 4.16: A plot of (a) the rate of CO_2 production, $\mathcal{R}(t)$, and (b) the total amount of CO_2 produced until time t, $\mathcal{P}(t)$. These are extracted from the same simulation done in the two previous figures.

It becomes easier to see the difference between the two spatially different problems when CO_2 production is taken into account. In Figure 4.16 it is clear that there is a substantial difference between the CO_2 production of both cases. This shows that the initial distribution of biomass can affect a lumped, global measurement and change the reactor-scale behaviour when compared to a meso-scaled system.

Chapter 5

Conclusions

5.1 Lessons Learned

- From the numerics chapter of the thesis, the validity and usefulness of a newly developed fully-implicit method was investigated. By comparing it to the standard semi-implicit method, which it extends, it was determined that a significant accuracy gain results from a single extra iteration of the fully-implicit method. Multiple iterations increase this gain since the increase in accuracy is positively correlated to the number of iterations performed. However, the computational effort required from the fully-implicit method grows exponentially with lower tolerance. The ratio for solution accuracy when weighed against heavier computation times suggests that two iterations of the fully-implicit method is best (one extra from the semi-implicit method). This resulted in an extra digit of accuracy at the cost of approximately 150% the computational effort.

- From the simulation chapter of the thesis, a number of useful characteristics were observed in the system. The existence of travelling wave solutions was strongly suggested from all the evidence gathered. The stability of this wave suggests that it always exists, but this cannot be verified due to the analytic complexity of the problem. Testing two sets of initial conditions, chosen at opposing extremes of spatial spreading (all biomass in one location versus equally

distributed across one boundary), and measuring the CO_2 production showed a large difference between the two solutions at a reactor-scale. This suggests that two dimensional models are better for accurately mimicking the behaviour of the system.

5.2 Future Work

- A full travelling wave analysis could be conducted. The existence of travelling wave solutions might be proven and the travelling wave speed could be solved for in terms of parameters. The issue with this is the high degeneracy of the diffusion term. One way of handling this would be to try regularising the system to eliminate the degeneracy.

- The reaction parameters of the diffusion-reaction model from the experimental data in Dumitrache et al. (2015) could be determined. This is an ill-posed problem since the data is lumped for only CO_2 production rates; there is no data for C, M, and any initial data for M.

- The two dimensional model can be upscaled to obtain a reactor-scale model as in Dumitrache et al. (2015) where the spatial effects were accounted for by introducing the concept of a carrying capacity. This could be accomplished by using volume averaging to consolidate the spatial terms in with the growth terms.

- A rigorous proof could be attempted, showing that the nonlinear iteration of the fully-implicit method always converges or that the conditions required for convergence is so far missing.

References

Alternative Fuel Data Center (2014). Ethanol vehicle emissions. Online; accessed 1-December-2015.

Barrett, R., Berry, M., T.F., C., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H. (1987). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, 1st edition.

Burden, R. and Faires, J. (2010). *Numerical Analysis*. Brooks/Cole, 9th edition.

Butcher, J. (2008). *Numerical Methods for Ordinary Differential Equations*. Wiley, 2nd edition.

Dumitrache, A. (2014). *Understanding Biofilms of Anaerobic, Thermophilic and Cellulolytic Bacteria: A Study towards the Advancement of Consolidated Bioprocessing Strategies*. PhD thesis, University of Toronto.

Dumitrache, A., Eberl, H., Wolfaardt, G., and Allen, D. (2015). Mathematical modeling to validate on-line co_2 measurements as a metric for cellulolytic biofilm activity in continuous-flow bioreactors. *Biochemical Engineering Journal*, 101:55–67.

Dumitrache, A., Wolfaardt, G., Allen, D., Liss, S., and Lynd, L. (2013a). Form and function of clostridium thermocellum biofilms. *Applied and Environmental Microbiology*, 79:231–239.

Dumitrache, A., Wolfaardt, G., Allen, D., Liss, S., and Lynd, L. (2013b). Tracking the cellulolytic activity of clostridium thermocellum biofilms. *Biotechnology for Biofuels*, 6:175.

Eberl, H. and Demaret, L. (2007). A finite difference scheme for a doubly degenerate diffusionreaction equation arising in microbial ecology. *Electronic Journal of Differential Equations*, CS15:77–95.

927 Eberl, H., Khassehkhan, H., and Demaret, L. (2010). A mixed-culture model of a probiotic biofilm
928 control system. *Computational and Mathematical Methods in Medicine*, 11(2):99–118.

929 Eberl, H., Parker, D., and van Loosdrecht, M. (2001). A new deterministic spatio-temporal continuum
930 model for biofilm development. *Journal of Theoretical Medicine*, 3:161–175.

931 Efendiev, M., Eberl, H., and Zelik, S. (2002). Existence and longtime behaviour of solutions of a
932 nonlinear reaction-diffusion system arising in the modeling of biofilms. *RIMS Kokyuroko*, 1258:49–
933 71.

934 Gurtin, M.E., M. (1977). On the diffusion of biological populations. *Mathematical Biosciences*,
935 33:35–49.

936 Jalbert, E. and Eberl, H. (2014). Numerical computation of sharp travelling waves of a degenerate
937 diffusion-reaction equation arising in biofilm modelling. *Communications in Nonlinear Science*
938 *and Numerical Simulation*, 19(7):2181–2190.

939 Khassehkhan, H. and Eberl, H. (2008). Modeling and simulation of a bacterial biofilm that is con-
940 trolled by ph and protonated lactic acids. *Computation and Mathematical Methods in Medicine*,
941 9(1):47–67.

942 Khassehkhan, H., Hillen, T., and Eberl, H. (2009). A nonlinear master equation for a degenerate
943 diffusion model of biofilm growth. *Lecture Notes in Computer Science*, 5544:735–744.

944 Kreft, J.-U., Picioreanu, C., Wimpenny, J., and van Loosdrecht, M. (2001). Individual-based mod-
945 elling of biofilms. *Microbiology*, 147(11):2897–2912.

946 Lynd, L. (2008). Energy biotechnology. *Current Opinion in Biotechnology*, 19(3):199–201.

947 Macías-Díaz, J., Macías, S., and Medina-Ramíez, I. (2013). An efficient nonlinear finite-difference
948 approach in the computational modeling of the dynamics of a nonlinear diffusion-reaction equation
949 in microbial ecology. *Computational Biology and Chemistry*, 47:24–30.

950 Noguera, D., Pizarro, G., Stahl, D., and Rittmann, B. (1999). Simulation of multispecies biofilm
951 development in three dimensions. *Water Science and Technology*, 39:123–130.

952 Picioreanu, C., van Loosedrecht, M., and Heijnen, J. (1998). A new ombined differential-discrete cel-
953 lular automaton approach for biofilm modeling: application for growth in gel beads. *Biotechnology*
954 *and Bioengineering*, 57:718–731.

955 Rittmann, B. and McCarty, P. (1980). Model of steady-state-biofilm kinetics. *Biotechnology and*
956 *Bioengineering*, 22(11):2343–2357.

957 Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied
958 Mathematic, 2nd edition.

959 Sirca, S. and Horvat, M. (2012). *Computational Methods for Physicistsl: Compendium for Students*.
960 Springer.

961 Varga, R. (2004). *Geršgorin and His Circles*. Berlin: Springer-Verlag.

962 Wang, Z.-W., Lee, S.-H., Elkins, J., and Morrell-Falvey, J. (2011). Spatial and temporal dynamics
963 of cellulose degradation and biofilm formation by caldicellulosiruptor obsidiansis and clostridium
964 thermocellum. *AMB Express*, 1:30.

965 Wanner, O., Eberl, H., Morgenroth, E., Noguera, D., Picioreanu, C., Rittmann, B., and van Loos-
966 drecht, M. (2005). *Mathematical modeling of biofilms*. IWA Scientific and Technical Report no
967 18., UK: IWA Publishing.

Appendix A

Default Parameter Values

The default parameter values used for simulations are listed in the follow table. Unless stated, every simulation uses these values.

Parameter	Symbol	Value
-	α	4
-	β	4
Decay-Loss rate	ν	0.12
Half-concentration rate	κ	10^{-3}
Yield rate	γ	0.80
Diffusion constant	δ	10^{-4}
Initial condition height	h	0.1
Initial condition depth	d	$\frac{5}{127}$
Number of grid points	nm	513×513
Grid size	Δx	$\frac{1}{513}$
Time step	Δt	10^{-3}

Table A.1: The listing of default parameter values used for most simulations.

Appendix B

Source Code

Listing B.1: PDE-ODEsolver.f90 main source code

```
!=============================================================================
!   PDE - ODE Coupled Solver
!-----------------------------------------------------------------------------
!     This fortran code solves the PDE - ODE coupled system that describes
!   the growth of Clostridium Thermocellum and its consumption of the carbon
!   substrait.
!
!   The system is based off the following system:
!     M_t = nabla (D(M) nabla M ) + f(C,M)
!     C_t = - g(C,M)
!   where M is the biomass of C. Thermocellum and C is the concentration of
!   Carbon. D(M) is the diffusion coeffient for the biomass movement. f(C,M)
!   is the growth and death term for the biomass. g(C,M) is the consumption
!   of carbon substrait.
!-----------------------------------------------------------------------------
!     The method of solving is by trapzidral rule for C, and by finite
!   difference for M
!=============================================================================

program cThermoPDEODE
!    use omp_lib
    implicit none
    INTERFACE
        subroutine solveLSDIAG(n,ndiag,ioff,a,sol,rhs,nit,err1,err2,stopcrit)
        !-----------------------------------------------------------------------
        ! input:  n       problem size
        !         ndiag:  number of diagonals
        !         ioff:   offsets (distance of sub diagonals to main diagonal)
        !         Mnew:      matrix values
        !         sol:    initial guess for iteration ( overwritten by result)
        !         rhs:    the righ hand side of the linear system
        !         nit:    max num of iter to be carried out (overwritten)
        !         err1:   tol for 1st stopping criterion (will be overwritten)
        !         err2:   tol for 2nd stopping criterion (will be overwritten)
        ! output: sol:    solution of Ax=b
        !         nit:    number of iterations taken
        !         err1:   computed value for 1st stopping criterion
        !         err2:   computed value for 2nd stopping criterion
        !         stopcrit: value that tells which stopping criti activated
        !-----------------------------------------------------------------------
```

```
1014          implicit none
1015          integer, intent(in)                 :: n,ndiag
1016          real,dimension(n,ndiag),intent(in):: a
1017          integer, dimension(ndiag),intent(in)::ioff
1018          real,dimension(n),intent(in)        :: rhs
1019          real,dimension(n),intent(inout)     :: sol
1020          real,intent(inout)                  :: err1,err2
1021          integer,intent(inout)               :: nit
1022          integer,intent(out)                 :: stopcrit
1023          end subroutine
1024      END INTERFACE
1025
1026      !======================
1027      ! Variable Descriptions
1028      !======================
1029
1030      ! filename Variables
1031      character(100) :: filename
1032
1033      ! Problem Parameters
1034      real :: kappa,delta,nu,gama
1035      integer :: alpha,beta
1036      real :: depth,height
1037      integer :: fSelect, dSelect, gSelect, MinitialCond
1038      integer :: num_innocu_points
1039
1040      ! Numerical Method Parameters
1041      integer :: pSize,row,col,n,ndiag,nit,nOuts
1042      real :: tEnd,tDel,xDel,e1,e2,eSoln,eTrav
1043      integer :: true2D, checkTrav
1044
1045      ! Solution variables
1046      real,dimension(:),allocatable  :: C, M
1047
1048      ! Reporting variables
1049      real :: avgIters, maxIters
1050      real :: avgNit, maxNit
1051      integer :: startTime, endTime, clock_rate, clock_max
1052      real :: elapsedTime
1053
1054      write(*,*) "Enter parameter file name: ex. 'parameter.txt' "
1055      write(*,'(A)', advance="no") "    "
1056      read(*,*) filename
1057
1058      write(*,*) "Setting Parameters that shouldn't change"
1059      ndiag = 5
1060      write(*,'(A, I5)') "     ndiag = ", ndiag
1061
1062      write(*,*) "Getting problem size from file"
1063      call getProbSize(pSize, true2D, checkTrav, filename, len(filename))
1064      if (true2D == 1) then
1065        write(*,*) "    Problem is 2D"
1066        col = 4
1067      else if (true2D == 0) then
1068        write(*,*) "    Problem is 3D"
```

```
      col = pSize
    end if
    write(*,*) "    pSize = ", pSize
    row = pSize
    n = row * col
    write(*,*) "    row   = ", row
    write(*,*) "    col   = ", col
    write(*,*) "    n     = ", n

    write(*,*) "Opening Parameter file"
    call paramSet(depth, height, num_innocu_points, alpha, beta, kappa, &
                  gama, nu, delta, nOuts, tEnd, tDel, e1, e2, eTrav, &
                  eSoln, fSelect, dSelect, gSelect, MinitialCond, filename, &
                  len(filename))
    xDel = 1/real(row)
    nit = n * 100
    write(*,*) "Parameters set:"
    write(*,*) "    depth       = ", depth
    write(*,*) "    height      = ", height
    write(*,*) "    alpha       = ", alpha
    write(*,*) "    beta        = ", beta
    write(*,*) "    gama        = ", gama
    write(*,*) "    nu          = ", nu
    write(*,*) "    delta       = ", delta
    write(*,*) "    nOuts       = ", nOuts
    write(*,*) "    tEnd        = ", tEnd
    write(*,*) "    tDel        = ", tDel
    write(*,*) "    e1          = ", e1
    write(*,*) "    e2          = ", e2
    write(*,*) "    eSoln       = ", eSoln
    write(*,*) "    nit         = ", nit
    write(*,*) "    xDel        = ", xDel
    write(*,*) "    fSelect     = ", fSelect
    write(*,*) "    dSelect     = ", dSelect
    write(*,*) "    gSelect     = ", gSelect
    write(*,*) "    MinitialCond= ", MinitialCond
   write(*,*) "Allocating the size of arrays"
    allocate(C(n),M(n))
    write(*,'(A,I12,A)') "    C and M are now dimension(", n, ") arrays"

    write(*,*) "Setting Initial Conditions"
    call setInitialConditions(C,M,row,col,n,depth,height,xDel,MinitialCond, &
                              num_innocu_points)

    write(*,*) "Starting Solver"
    call system_clock(COUNT_RATE=clock_rate, COUNT_MAX=clock_max)
    call system_clock(startTime)
    call solveOrder2(tEnd,nOuts,tDel,n,row,col,M,C,xDel,&
       gama,nu,alpha,beta,kappa,delta,ndiag,e1,e2,nit,eSoln,dSelect,&
       fSelect,gSelect,avgIters,maxIters,avgNit,maxNit,true2D,checkTrav,eTrav)
    call system_clock(endTime)

    ! Convert times into seconds
    elapsedTime = real(endTime - startTime)/real(clock_rate)

```

```
1124
1125      ! Report Statistics
1126      write(*,*) "-------------------------------------------------------------"
1127      write(*,*) "Statsitcs:"
1128      write(*,*) "-------------------------------------------------------------"
1129      write(*,*) "Time to compute = ", elapsedTime
1130      write(*,*) ""
1131      write(*,*) "Avg Iters for iterating betn. soln. =", avgIters
1132      write(*,*) "Max Iters for iterating betn. soln. =", maxIters
1133      write(*,*) "Avg Iters for linear solver =", avgNit
1134      write(*,*) "Max Iters for linear solver =", maxNit
1135      write(*,*) "Writing statistics to file"
1136      call reportStats(avgIters,maxIters,avgNit,maxNit,elapsedTime)
1137
1138      write(*,*) "Execution completed"
1139
1140  end program
1141
1142
1143  !=============================================================================
1144  !   Sets the problem size and checks for auxillary settings
1145  !-----------------------------------------------------------------------------
1146  !     This must be done first since the problem size is used in the variable
1147  !   declaration of the arrays.
1148  !     The auxillary settings are if the problem is 2D in nature; this means
1149  !   that the computation time can be drastically reduced by letting col = 4
1150  !   instead of col = pSize.
1151  !     Also, the option for checking for the existance of travelling wave
1152  !   solutions is done here.
1153  !=============================================================================
1154  subroutine getProbSize(pSize, true2D, checkTrav, filename, nameLen)
1155      implicit none
1156      integer,intent(out) :: pSize, true2D, checkTrav
1157      integer,intent(in)  :: nameLen
1158      character(nameLen),intent(in) :: filename
1159      character :: dum   ! Dummy variable
1160      write(*,*) filename
1161      open(UNIT = 19, FILE = filename, STATUS = "old", ACTION = "read")
1162      read(19,*); read(19,*); read(19,*)  ! Skip first 3 lines
1163      read(19,*) dum, dum, pSize
1164      read(19,*) dum, dum, true2D
1165      read(19,*) dum, dum, checkTrav
1166      close(19)
1167  end subroutine getprobSize
1168
1169
1170  !=============================================================================
1171  !   Sets the all the parameter values
1172  !-----------------------------------------------------------------------------
1173  !     Opens the parameter.txt file, which holds all the parameter values and
1174  !   function uses, and sets the variables accordingly.
1175  !     Takes in all the parameters on entry and outputs them with the appropriate
1176  !   value.
1177  !=============================================================================
1178  subroutine paramSet(depth, height, numInnocu, alpha, beta, kappa, &
```

```
                    gama, nu, delta, nOuts, tEnd, tDel, e1, e2, eTrav, &
                    eSoln, fSelect, dSelect, gSelect, MinitialCond, filename, &
                    nameLen)
    implicit none
    real, intent(out) :: depth, height, kappa, delta, nu
    real, intent(out) :: tEnd, tDel, e1, e2, eSoln, eTrav, gama
    integer, intent(out) :: alpha, beta,  numInnocu, nOuts
    integer, intent(out) :: fSelect, dSelect, gSelect, MinitialCond
    integer, intent(in) :: nameLen
    character(nameLen), intent(in) :: filename

    character :: dum, dum2      ! Dummy variable

    open(UNIT = 19, FILE = filename, STATUS = "old", ACTION = "read")
    ! Skip first 6 lines
    read(19,*); read(19,*); read(19,*); read(19,*); read(19,*); read(19,*)

    read(19,*) dum, dum2, depth
    read(19,*) dum, dum2, height
    read(19,*) dum, dum2, numInnocu
    read(19,*) dum, dum2, alpha
    read(19,*) dum, dum2, beta
    read(19,*) dum, dum2, nu
    read(19,*) dum, dum2, kappa
    read(19,*) dum, dum2, gama
    read(19,*) dum, dum2, delta
    read(19,*) dum, dum2, nOuts
    read(19,*) dum, dum2, tEnd
    read(19,*) dum, dum2, tDel
    read(19,*)                          ! Skip xDel
    read(19,*) dum, dum2, e1
    read(19,*) dum, dum2, e2
    read(19,*) dum, dum2, eSoln
    read(19,*) dum, dum2, eTrav
    read(19,*)                          ! Skip nit
    read(19,*); read(19,*); read(19,*)  ! Skip 3 lines
    read(19,*) dum, dum2, fSelect
    read(19,*) dum, dum2, dSelect
    read(19,*) dum, dum2, gSelect
    read(19,*) dum, dum2, MinitialCond

    close(19)

end subroutine paramSet


!=======================================================================
!   Sets the initial conditions for the system
!-----------------------------------------------------------------------
!   For C, we have homogenous initial conditions, trivial to do
!   For M, the innoculation point has a smooth curve to avoid sharp numerical
!     artifacts in the system. This is done with a polynomial f(x) = a*x^8+b,
!     this function is computed based on the depth and height parameters,
!     calculating b = height and a = b/(depth)^8
!=======================================================================
```

```
1234 subroutine setInitialConditions(C,M,row,col,n,depth,height,xDel,MinitialCond, &
1235                                 num_innocu_points)
1236     implicit none
1237     integer,intent(in) :: row,col,n,MinitialCond, num_innocu_points
1238     real,intent(in) :: depth, height, xDel
1239     real,dimension(n),intent(out) :: C,M
1240 
1241     integer :: i,j,x,y, xi, yi, p
1242     real :: f,a       ! function for IC curve
1243     real :: innoc     ! Placeholder for new IC value at point
1244     real :: total     ! Counts total innoculation height
1245     real :: mIC, cIC      ! Used to store previous simulation values while reading
1246     real :: xr, yr
1247 
1248     C = 1.; j = 0; M = 0
1249 
1250     ! 2D trav wave smooth curve
1251     if (MinitialCond == 1) then
1252         a = -height/(depth)**4
1253         !$omp parallel do private(f) shared(height,a)
1254         do i = 1, n
1255           x = (i-1)/col
1256           f = a*(x*xDel)**4 + height
1257           M(i) = f
1258           if (f .LE. 0) M(i) = 0
1259         enddo
1260         !$omp end parallel do
1261 
1262     ! homogenous everywhere
1263     else if (MinitialCond == 2) then
1264         M = height
1265 
1266     ! 3D initial conditions
1267     else if (MinitialCond == 3) then
1268         a = -height/(depth)**2
1269         !$omp parallel do private(f,x,y) shared(height,a)
1270         do i = 1, n
1271           x = MOD(i-1, col)
1272           y = i / row
1273           f = a*((x*xDel-0.5)*(x*xDel-0.5) + (y*xDel-0.5)*(y*xDel-0.5)) + height
1274           M(i) = f
1275           if (M(i) .LE. 0) M(i) = 0
1276         enddo
1277         !$omp end parallel do
1278 
1279     !(Random innoclation points over whole domain [3D])
1280     else if (MinitialCond == 4) then
1281       a = -height/(depth*depth)
1282       call init_random_seed()
1283       do i = 1, num_innocu_points
1284         call random_number(xr)
1285         call random_number(yr)
1286         do j = 1, n
1287           if (M(j) .LE. 0) then
1288             x = MOD(j-1, col)
```

```
            y = j / row
            M(j) = M(j) + a*((x*xDel-xr)*(x*xDel-xr) + (y*xDel-yr)*(y*xDel-yr)) + height
            if (M(j) .LE. 0) M(j) = 0
          endif
        enddo
      enddo

    !(Random innoculation points on y=0 side [3D])
    else if (MinitialCond == 5) then
      a = -height/(depth*depth)
      call init_random_seed()
      do i = 1, num_innocu_points
        call random_number(xr)
        call random_number(yr)
        yr = yr*0.1
        do j = 1, n
            x = MOD(j-1, col)
            y = j / row
            innoc = a*((x*xDel-xr)*(x*xDel-xr) + (y*xDel-yr)*(y*xDel-yr)) + height
            if (innoc .GE. 0) M(j) = M(j) + innoc
        enddo
      enddo
        ! Divide innoculation height by average non-zero value
      i = 0
      do j = 1, n
        if (M(j) .GT. 0) then
          i = i + 1
          total = total + M(j)
        endif
      enddo
      do j = 1, n
        M(j) = M(j) * height/(total/real(i))
      enddo


    ! sharp IC. homogenous in y-dir
    else if (MinitialCond == 6) then
      do i = 1, n
        x = (i-1) / col
        if ( x*xDel .LE. depth) then
          M(i) = height
        endif
      enddo

    ! pertubations in y-dir; check trav.wave. stability
    else if (MinitialCond == 7) then
      call init_random_seed()
      do i = 1, n
        if ( i*xDel/col .LE. depth) then
          call random_number(xr)
          M(i) = xr*0.2
        endif
      enddo

    ! Test the grid ordering/ printing
```

```
1344      else if (MinitialCond == 8) then
1345       ! do i = 1, row
1346       !   do j = 1, col
1347       !     if (i*xDel .LE. depth) then
1348       !       p = j + (i-1)*col
1349       !       M(i) = real(p)/real(n)/4.0
1350       !     endif
1351       !   enddo
1352       ! enddo
1353        do i = 1, n
1354          M(i) = real(i)/real(n)/10
1355        enddo
1356
1357      !(Evenly spaced innoculation points on y=0 side [3D])
1358      else if (MinitialCond == 9) then
1359        a = -height/(depth*depth)
1360        do i = 1, num_innocu_points
1361          xr = depth + (i-1)*2*depth + real((i-1)*(1-num_innocu_points*depth*2))/real(num_inno
1362          do j = 1, n
1363            if (M(j) .LE. 0) then
1364              x = MOD(j-1, col)
1365              y = j / row
1366              M(j) = M(j) + a*((x*xDel-xr)*(x*xDel-xr) + (y*xDel)*(y*xDel)) + height
1367              if (M(j) .LE. 0) M(j) = 0
1368            endif
1369          enddo
1370        enddo
1371
1372      ! Random innoculation points on y=0 and y = 1
1373      else if (MinitialCond == 10) then
1374        a = -height/(depth*depth)
1375        call init_random_seed()
1376        do i = 1, 2*num_innocu_points
1377          call random_number(xr)
1378          call random_number(yr)
1379          yr = yr*0.1
1380          if (i .GT. num_innocu_points) then
1381            yr = yr + 0.9 ! For y = 1 side
1382          endif
1383          do j = 1, n
1384              x = MOD(j-1, col)
1385              y = j / row
1386              innoc = a*((x*xDel-xr)*(x*xDel-xr) + (y*xDel-yr)*(y*xDel-yr)) + height
1387              if (innoc .GE. 0) M(j) = M(j) + innoc
1388          enddo
1389        enddo
1390
1391      ! Read IC from output file... (Continues a previous sim) CANNOT GET WORKING>>>>
1392  !     else if (MinitialCond == 11) then
1393  !       open(UNIT = 119, FILE = "initialCond.dat", STATUS = "old", ACTION = "read")
1394  !       read(119,*) ! Skip first line
1395  !       do i = 1, n/2+1
1396  !         read(119,*) innoc, innoc, innoc, innoc
1397  !       enddo
1398  !       do i = n/2+1, n      <<<<< THIS IS THE PROBLEM AREA, Can't divide the region in two
```

```
!         read(119,*) innoc, innoc, mIC, cIC ! innoc is used as a dummy variable here
!         M(i) = mIC
!         C(i) = cIC
!       enddo
!       close(119)
!

    ! Clumped up innoculation at origin. Same total as MinitCond == 9
    else if (MinitialCond == 12) then
      innoc = (2*height*depth*depth*num_innocu_points) ** (1.0/3.0)
      a = -1.0/(innoc)
      do j = 1, n
        x = MOD(j-1, col)
        y = j / row
        M(j) = a*((x*xDel)*(x*xDel) + (y*xDel)*(y*xDel)) + innoc
        if (M(j) .LE. 0) M(j) = 0
      enddo

    ! Evenly Spaced innocu point in cubes (exact total)
    else if (MinitialCond == 13) then
      xr = real(col)/real(num_innocu_points)
      do j = 1, n
        x = MOD(j-1, col)
        y = j / row
        if (y*xDel .LE. depth .AND. MOD(x+xr/4.0, xr) >= xr/2.0) M(j) = height
      enddo

    ! Clumped up like ==12, but exact cube
    else if (MinitialCond == 14) then
      xr = (num_innocu_points * depth * height * real(col)/(2*real(num_innocu_points))) **
      do j = 1, n
        x = MOD(j-1, col)
        y = j / row
        if (y*xDel .LE. xr .AND. x*xDel .LE. xr) M(j) = xr
      enddo

    endif

end subroutine setInitialConditions


!=============================================================================
!   Function f(C,M)
!=============================================================================
subroutine fFunc(M,C,f,kappa,nu,fSelect)
    implicit none
    integer, intent(in) :: fSelect
    real, intent(in) :: M,C,kappa,nu
    real, intent(out) :: f
    real :: eps
    eps = C*0.00000001
    if (fSelect == 1) f = C/(kappa+C)-nu
    if (fSelect == 2) f = C/(kappa+C)
    if (fSelect == 3) f = C/(kappa*M+C+eps)-nu
    if (fSelect == 4) f = C/(kappa*M+C+eps)
```

```
    if (fSelect == 5) f = 1
    if (fSelect == 6) f = C/(kappa+C)*(1-M/(C+eps))-nu
end subroutine fFunc


!=======================================================================
!   Function d(M)
!=======================================================================
subroutine dFunc(M,d,delta,alpha,beta,dSelect)
    implicit none
    integer,intent(in) :: alpha, beta, dSelect
    real, intent(in) :: M, delta
    real, intent(out) :: d
    if (dSelect == 1) d = delta
    if (dSelect == 2) d = delta * M** alpha
    if (dSelect == 3) d = delta * M**alpha / ((1 - M)**beta)
end subroutine dFunc


!=======================================================================
!   Solves the solution, C
!-----------------------------------------------------------------------
!   Uses the trapizoidal rule for an ODE and solves for C.
!   Different gSelects give different values for b and c.
!   gSelect = 1 --> g = gama*C/(kappa+C)
!=======================================================================
subroutine solveC(M,Mnew,Csol,Cnew,n,kappa,gama,tDel,gSelect)
    implicit none
    integer,intent(in) :: n,gSelect
    real,intent(in) :: kappa,tDel,gama
    real,dimension(n),intent(in) :: M,Mnew,Csol
    real,dimension(n),intent(out) :: Cnew

    integer :: i,j    ! grid index
    integer :: g      ! current grid point
    real :: b,c       ! quadratic equation terms
    real :: f         ! f(C,M) value at gridpoint
    real :: aux

    real :: r,s
    r = 1
    s = 0.5

    aux = tDel*0.5*gama

    if (gSelect == 1) then
     !$omp parallel do private(b,c) shared(kappa,Csol,aux,Mnew,M)
      do i = 1,n
        b = kappa - Csol(i) + aux*(Mnew(i) + Csol(i)*M(i)/(kappa+Csol(i)) )
        c = -kappa*Csol(i) + aux*kappa*Csol(i)*M(i)/(kappa + Csol(i))
        Cnew(i) = 0.5 * (-b + SQRT(b*b - 4 * c))
      enddo
     !$omp end parallel do
    end if

```

```
1509      end subroutine solveC
1510
1511
1512  !=====================================================================
1513  !   Generates matrix M in diagonal format for the current timestep
1514  !=====================================================================
1515  subroutine GenMatrixM(M,C,MatrixM,Mioff,Mrhs,row,col,n,ndiag,delta,nu,&
1516                        alpha,beta,kappa,tDel,xDel,dSelect,fSelect)
1517      implicit none
1518      integer,intent(in) :: row,col,n,ndiag,alpha,beta,dSelect,fSelect
1519      real,intent(in) :: delta,kappa,xDel,tDel,nu
1520      real,dimension(n),intent(in) :: M,C
1521
1522      real,dimension(n,ndiag),intent(out) :: MatrixM
1523      real,dimension(n),intent(out) :: Mrhs
1524      integer,dimension(ndiag),intent(out) :: Mioff
1525
1526      real :: xCof
1527      real :: f
1528      real,dimension(n) :: diff
1529      !integer :: x,y,g
1530      integer :: i
1531
1532      real :: tDela
1533      tDela = tDel      ! Used for testing purposes
1534
1535      xCof = 1/(xDel*xDel)
1536
1537      Mioff = (/ -col, -1, 0, 1, col/)
1538      MatrixM(:,:) = 0
1539
1540      ! Compute all the diffusion coefficients
1541      !$omp parallel do shared(M,diff,delta,alpha,beta,dSelect)
1542      do i = 1,n
1543          call dFunc(M(i), diff(i), delta, alpha, beta, dSelect)
1544      enddo
1545      !$omp end parallel do
1546
1547      !!$omp parallel do shared(MatrixM,xCof,diff,Mrhs,tDel,M,C,kappa,nu,fSelect) private(i,
1548      !$omp parallel do private(f)
1549      do i = 1,n
1550          if (i .LE. col) then
1551              MatrixM(i,5) = MatrixM(i,5) - xCof*0.5*(diff(i+col)+diff(i))
1552              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+col)+diff(i))
1553          else
1554              MatrixM(i,1) = MatrixM(i,1) - xCof*0.5*(diff(i-col)+diff(i))
1555              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-col)+diff(i))
1556          endif
1557
1558          if (MOD(i,col) == 1) then
1559              MatrixM(i,4) = MatrixM(i,4) - xCof*0.5*(diff(i+1)+diff(i))
1560              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+1)+diff(i))
1561          else
1562              MatrixM(i,2) = MatrixM(i,2) - xCof*0.5*(diff(i-1)+diff(i))
1563              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-1)+diff(i))
```

```
1564          endif
1565
1566          if (MOD(i,col) == 0) then
1567              MatrixM(i,2) = MatrixM(i,2) - xCof*0.5*(diff(i-1)+diff(i))
1568              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-1)+diff(i))
1569          else
1570              MatrixM(i,4) = MatrixM(i,4) - xCof*0.5*(diff(i+1)+diff(i))
1571              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+1)+diff(i))
1572          endif
1573
1574          if  (i .GE. n-col) then
1575              MatrixM(i,1) = MatrixM(i,1) - xCof*0.5*(diff(i-col)+diff(i))
1576              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-col)+diff(i))
1577          else
1578              MatrixM(i,5) = MatrixM(i,5) - xCof*0.5*(diff(i+col)+diff(i))
1579              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+col)+diff(i))
1580          endif
1581
1582          call fFunc(M(i),C(i),f,kappa,nu,fSelect)
1583          MatrixM(i,3) = MatrixM(i,3) - f + (1/tDel)
1584          Mrhs(i) = M(i)/tDel
1585      enddo
1586      !$omp end parallel do
1587
1588  end subroutine GenMatrixM
1589
1590
1591  !=============================================================================
1592  !   Solve Order 2
1593  !-----------------------------------------------------------------------------
1594  !   1. Solves for M_{i+1} using C_i and M_i
1595  !   2. Solves for C_{i+1} using C_i, M_i, and M_{i+1}
1596  !   ... repeat until convergence
1597  !=============================================================================
1598  subroutine solveOrder2(tEnd,nOuts,tDel,n,row,col,M,C,xDel,&
1599      gama,nu,alpha,beta,kappa,delta,ndiag,e1,e2,nit,eSoln,dSelect,fSelect,&
1600      gSelect,avgIters,maxIters,avgNit,maxNit,true2D,checkTrav,eTrav)
1601    implicit none
1602    integer,intent(in) :: nOuts,n,row,col,alpha,beta,ndiag
1603    integer,intent(in) :: dSelect,fSelect,gSelect,true2D,checkTrav
1604    real,intent(in) :: tEnd,tDel,xDel,kappa,delta,nu,gama
1605    real,intent(in) :: eSoln,eTrav
1606    real,intent(inout) :: e1,e2
1607    integer,intent(inout) :: nit
1608    real,dimension(n),intent(out) :: M,C
1609    real,intent(out) :: avgIters,maxIters,avgNit,maxNit
1610
1611    integer :: counter        ! Counts the num. of iter. in system solver
1612    integer :: endLoop        ! endLoop = 1 -> solving loop can stop
1613    integer :: filter         ! Controls frequency of outputs written
1614
1615    real :: stored_e1, stored_e2
1616
1617    real :: totalMassM        ! Total Biomass over region
1618    real :: totalMassC        ! Total Substrait over region
```

```
1619    real :: prevMassC          ! Total Substrait from X timesteps ago
1620
1621    real,dimension(n) :: Mnew
1622    real,dimension(n) :: Cnew
1623    real,dimension(n,ndiag) :: MatrixM
1624    real,dimension(n) :: Mrhs
1625    real,dimension(n) :: Cprev, Mprev
1626    integer,dimension(ndiag) :: Mioff
1627    integer :: stopcritria
1628    integer :: stat
1629    real :: diffC, diffM
1630    integer :: countIters
1631    real :: peak, height, intfac  ! Track Interface and wave peak
1632    integer :: travExist          ! 1 if trav wave exist at this timestep
1633    real :: waveSpeed             ! calculated wavespeed at current timestpe
1634    real,dimension(n) :: Mprev_10 ! M from 10-ish timesteps ago, for travCheck
1635
1636    stored_e1 = e1
1637    stored_e2 = e2
1638
1639    filter = int(tEnd/(nOuts*tDel))
1640    endLoop = 0
1641    counter = 0
1642    countIters = 0
1643    avgIters = 0
1644    avgNit = 0
1645    Mprev_10 = M  ! To initiatize for travCheck
1646    travExist = 0
1647
1648    prevMassC = 1
1649
1650    open(UNIT = 124, IOSTAT = stat, FILE = "total.dat", STATUS = "old")
1651    if (stat .EQ. 0) close(124, STATUS = "delete")
1652    open(UNIT = 120, FILE = "total.dat", POSITION = "append", ACTION = "write")
1653
1654    open(UNIT = 124, IOSTAT = stat, FILE = "COprod.dat", STATUS = "old")
1655    if (stat .EQ. 0) close(124, STATUS = "delete")
1656    open(UNIT = 126, FILE = "COprod.dat", POSITION = "append", ACTION = "write")
1657
1658    if (true2D == 1) then
1659      open(UNIT = 124, IOSTAT = stat, FILE = "peakInfo.dat", STATUS = "old")
1660      if (stat .EQ. 0) close(124, STATUS = "delete")
1661      open(UNIT = 121, FILE = "peakInfo.dat", POSITION = "append", ACTION = "write")
1662    endif
1663
1664    if (checkTrav == 1) then
1665      open(UNIT = 124, IOSTAT = stat, FILE = "travCheck.dat", STATUS = "old")
1666      if (stat .EQ. 0) close(124, STATUS = "delete")
1667      open(UNIT = 138, FILE = "travCheck.dat", POSITION = "append", ACTION = "write")
1668    endif
1669
1670    write(*,*) "   time    avgIter  maxIter      avgNit  maxNit        avgM
1671  avgC"
1672
1673    do while((counter) * tDel <= tEnd)
```

```
1674      ! Output every 100 times more then nOuts for the peak info
1675      if (MOD(counter, int(filter/100)+1) == 0) then
1676        ! Get total M and C
1677        call calcMass(M,totalMassM,n,row,col)
1678        call calcMass(C,totalMassC,n,row,col)
1679        write(120,*) counter*tDel, totalMassM, totalMassC
1680
1681        ! CO_2 production: t, current produced CO2, total produced CO2
1682        write(126,*) counter*tDel, prevMassC - totalMassC, 1 - totalMassC
1683        prevMassC = totalMassC
1684
1685        ! peak-interface, only availbale for 2D graphs
1686        if (true2D == 1) then
1687          call calcPeakInterface(M, row, col, peak, height, intfac)
1688          write (121,*) tDel*counter, peak, height, intfac
1689        end if
1690      endif
1691      if (true2D == 1 .AND. checkTrav == 1 .AND. MOD(counter, int(filter))==0) then
1692        call checkTravWave(M, Mprev_10, row, col, travExist, wavespeed, height, eTrav)
1693        wavespeed = wavespeed/filter ! Makes wavespeed independent of number of outputs
1694        write (138,*) tDel*counter, travExist, wavespeed
1695        Mprev_10 = M
1696      endif
1697
1698      ! Write to file / report Total Mass
1699      if (MOD(counter, filter) == 0) then
1700        if (true2D == 1) then
1701          call printToFile2D(n,row,col,M,C)
1702        else
1703          call printToFile(n,row,col,M,C)
1704        end if
1705
1706        if (counter == 0) then
1707          write(*,'(F8.2,F12.2,I8,F12.2,I8,F12.6,F12.6)') 0.0, 0.0, &
1708            int(maxIters), 0.0, int(maxNit), totalMassM, totalMassC
1709        else
1710          write(*,'(F8.2,F12.2,I8,F12.2,I8,F12.6,F12.6)') tDel*counter, &
1711            real(avgIters/counter), int(maxIters), real(avgNit/avgIters), &
1712            int(maxNit),totalMassM, totalMassC
1713        endif
1714      endif
1715
1716      diffC = 1
1717      diffM = 1
1718      countIters = 0
1719      Cprev = C
1720      Mprev = M
1721
1722      do while(diffC + diffM > eSoln)
1723          nit = 100*n
1724          e1 = stored_e1
1725          e2 = stored_e2
1726
1727          ! Solve M
1728          call GenMatrixM(Mprev,C,MatrixM,Mioff,Mrhs,row,col,n,ndiag,delta,nu,&
```

```
1729                      alpha,beta,kappa,tDel,xDel,dSelect,fSelect)
1730          call solveLSDIAG(n,ndiag,Mioff,MatrixM,Mnew,Mrhs,nit,e1,e2,stopcritria)
1731
1732          ! Solve C
1733          call solveC(Mprev,Mnew,Cprev,Cnew,n,kappa,gama,tDel,gSelect)
1734
1735          ! Solve CO_2 production
1736
1737
1738          if(nit > maxNit) maxNit = nit
1739          avgNit = avgNit + nit
1740
1741          call calcDiff(diffC, C, Cnew, row, col)
1742          call calcDiff(diffM, M, Mnew, row, col)
1743
1744          C = Cnew
1745          M = Mnew
1746          countIters = countIters+1
1747
1748          if (countIters > 10000) then
1749            write(*,*) "[!] Over 10000 iterations in one timestep"
1750            write(*,*) "[!] Solutions not converging. Exit!"
1751            stop
1752          end if
1753  !        write(*,*) countIters, diffC, diffM, C(12),M(12)
1754       enddo
1755       if(countIters > maxIters) maxIters = countIters
1756       avgIters = avgIters + countIters
1757
1758       counter = counter + 1
1759     enddo
1760     avgNit = avgNit/(avgIters) ! avgIters right now is the total
1761     avgIters = avgIters/counter
1762
1763  !  ! Print final solution
1764  !  if (true2D == 1) then
1765  !    call printToFile2D(n,row,col,M,C)
1766  !  else
1767  !    call printToFile(n,row,col,M,C)
1768  !  end if
1769  !  write(*,'(F8.2,F12.2,I8,F12.2,I8,F12.6,F12.6)') tDel*counter, &
1770  !          real(avgIters), int(maxIters), real(avgNit), &
1771  !          int(maxNit),totalMassM, totalMassC
1772
1773     close(120)
1774     close(138)
1775     close(126)
1776     if (true2D == 1) then
1777       close(121)
1778     endif
1779
1780  end subroutine solveOrder2
1781
1782
1783  !==============================================================================
```

```
1784 !   Calculate the Total Mass
1785 !------------------------------------------------------------------------------
1786 !   Uses Riemann Sums kind of concept to check if the program runs correctly
1787 !   since the total mass of the region can be computed and checked.
1788 !==============================================================================
1789 subroutine calcMass(X,totalMass,n,row,col)
1790     implicit none
1791     integer,intent(in) :: n,row,col
1792     real,dimension(n),intent(in) :: X
1793 
1794     real,intent(out) :: totalMass
1795 
1796     integer :: i,j,g
1797 
1798     totalMass = 0
1799 
1800     !$omp parallel do reduction(+:totalMass)
1801     do i=1,n
1802       totalMass = totalMass + X(i)
1803     enddo
1804     !$omp end parallel do
1805 
1806     totalMass = totalMass / n
1807 
1808 end subroutine calcMass
1809 
1810 
1811 !==============================================================================
1812 !   Prints out the solution in a format accepted by gnuplot, used for graphing
1813 !------------------------------------------------------------------------------
1814 !   Runs through the grid, row-by-row. The MOD and filter act to reduce the
1815 !     number of grid points written, useful when comparing different grid
1816 !     sizes.
1817 !------------------------------------------------------------------------------
1818 !   Input:
1819 !     n      =  The problem size
1820 !     row    =  The number of rows
1821 !     col    =  The number of columns
1822 !     M      =  The solution vector for biomass
1823 !     C      =  The solution for substrait
1824 !   Output:
1825 !     none
1826 !==============================================================================
1827 subroutine printToFile(n,row,col,M,C)
1828     implicit none
1829 
1830     integer,intent(in) :: n, row, col
1831     real,dimension(n),intent(in) :: M,C
1832 
1833     integer :: p
1834     integer :: i,j
1835     integer :: stat
1836     integer :: filter
1837 
1838     !----------------------------------------
```

```
1839      ! Deletes the old output file if it exist
1840      !----------------------------------------
1841      open(UNIT = 123, IOSTAT = stat, FILE = "output.dat", STATUS = "old")
1842      if (stat .EQ. 0) close(123, STATUS = "delete")
1843
1844      open(UNIT = 11, FILE = "output.dat", POSITION = "append", ACTION = "write")
1845
1846      filter = 1
1847      if (row .gt. 257) then
1848        filter = (row-1)/256
1849      endif
1850
1851      do i=1,row
1852        if (MOD(i-1, filter) == 0) then
1853          do j=1,col
1854            if (MOD(j-1, filter) == 0) then
1855              p = (j + (i-1)*col)
1856              write(11,*) real(j-1)/real(col-1), &
1857                          real(i-1)/real(row-1), M(p), C(p)
1858            endif
1859          enddo
1860          write(11,*) ' '
1861        endif
1862      enddo
1863      write(11,*)
1864
1865
1866  end subroutine printToFile
1867
1868
1869  !=============================================================================
1870  !   Prints out the solution in a 2D format, used for graphing the Travelling
1871  !     Wave Example.
1872  !-----------------------------------------------------------------------------
1873  !   Runs through the grid, row-by-row. The MOD and filter act to reduce the
1874  !     number of grid points written
1875  !   Unique here is that the average of the x-axis is taken so that the system
1876  !     can be reduced to just y. Also written are the max and min for each y
1877  !     value; this is used for showing that the system can be reduced.
1878  !-----------------------------------------------------------------------------
1879  !   Input:
1880  !     n       =  The problem size
1881  !     row     =  The number of rows
1882  !     col     =  The number of columns
1883  !     M       =  The solution vector for biomass
1884  !     C       =  The solution for substrait
1885  !   Output:
1886  !     none
1887  !=============================================================================
1888  subroutine printToFile2D(n,row,col,M,C)
1889     implicit none
1890
1891     integer,intent(in) :: n, row, col
1892     real,dimension(n),intent(in) :: M,C
1893
```

```
1894      integer :: p
1895      integer :: i,j
1896      integer :: stat
1897      integer :: filter
1898      real :: averageM, averageC
1899      real :: maxM, minM, maxC, minC
1900      real :: y
1901
1902      !-----------------------------------------
1903      ! Deletes the old output file if it exist
1904      !-----------------------------------------
1905      open(UNIT = 124, IOSTAT = stat, FILE = "2D_output.dat", STATUS = "old")
1906      if (stat .EQ. 0) close(124, STATUS = "delete")
1907
1908      open(UNIT = 12, FILE = "2D_output.dat", POSITION = "append", ACTION = "write")
1909
1910      filter = 1
1911      if (row .gt. 257) then
1912        filter = (row-1)/256
1913      endif
1914
1915      do, i=1,row
1916        if (MOD(i-1, filter) == 0) then
1917          averageM = 0
1918          averageC = 0
1919          maxM = 0
1920          maxC = 0
1921          minM = 1
1922          minC = 1
1923          do, j=1,col
1924              p = j + (i-1)*col
1925              averageM = averageM + M(p)
1926              averageC = averageC + C(p)
1927              if(M(p) .ge. maxM) then
1928                  maxM = M(p)
1929              endif
1930              if(M(p) .le. minM) then
1931                  minM = M(p)
1932              endif
1933              if(C(p) .ge. maxC) then
1934                  maxC = C(p)
1935              endif
1936              if(C(p) .le. minC) then
1937                  minC = C(p)
1938              endif
1939          enddo
1940          averageM = averageM/(col)
1941          averageC = averageC/(col)
1942          y = real(i-1)/real(row-1)
1943          write(12,'(f20.12,f20.12,f20.12)') y,averageM,averageC
1944  !wrte(12,'(f14.10,f14.10,f14.10,f14.10,f14.10,f14.10,f14.10)') y, averageM, averageC, minM,
1945        endif
1946      enddo
1947      write(12,*)
1948
```

```
1949
1950 end subroutine printToFile2D
1951
1952
1953 !================================================================================
1954 !   Calculates the difference between each grid point for the solutions at
1955 !       different iterations
1956 !--------------------------------------------------------------------------------
1957 !   Input:
1958 !     row    =  The number of rows
1959 !     col    =  The number of columns
1960 !     C      =  The previous solution for substrait
1961 !     Cnew   =  The current solution for substrait
1962 !   Output:
1963 !     diff   =  The average difference between the two C's
1964 !================================================================================
1965 subroutine calcDiff(diff, C, Cnew, row, col)
1966     integer, intent(in) :: row, col
1967     real, dimension(row*col), intent(in) :: C, Cnew
1968     real, intent(out) :: diff
1969
1970     integer :: i
1971
1972     diff = 0
1973     !$omp parallel do reduction(+:diff)
1974     do i=1,row*col
1975         diff = diff + abs(C(i) - Cnew(i))
1976     enddo
1977     !$omp end parallel do
1978     diff = diff/real(row*col)
1979
1980 end subroutine calcDiff
1981
1982
1983 !================================================================================
1984 !   Calculates the peak and interface info at a single timestep
1985 !--------------------------------------------------------------------------------
1986 !   Input:
1987 !     row    =  The number of rows
1988 !     col    =  The number of columns
1989 !     M      =  The solution for biomass, array size (n)
1990 !   Output:
1991 !     peak   =  Peak location
1992 !     height =  Peak height
1993 !     intfac =  Interface location
1994 !================================================================================
1995 subroutine calcPeakInterface(M, row, col, peak, height, intfac)
1996     implicit none
1997     integer, intent(in):: row,col
1998     real, dimension(row*col), intent(in) :: M
1999     real, intent(out) :: peak, height, intfac
2000
2001     real :: hei
2002     real :: y
2003     integer :: i,j,p
```

```
2004
2005      hei = 0
2006      do, i=1,row
2007        y = real(i-1)/real(row-1)
2008        do, j=1,col
2009            p = (j + (i-1)*col)
2010            if (M(p) >= hei) then
2011                peak = y
2012                hei = M(p)
2013            endif
2014            if (M(p) > 0.1) then
2015                intfac = y
2016            endif
2017        enddo
2018      enddo
2019      height = hei
2020
2021  end subroutine
2022
2023
2024
2025  !=====================================================================
2026  !   Check if there is evidence of a travelling wave solution
2027  !---------------------------------------------------------------------
2028  !   Makes an educated guess for the wavespeed (which would be incorrect
2029  !   by, at worst, 2*eTrav) and then uses this wavespeed to check if the
2030  !   difference between M and Mprev is consistently wavespeed +- eTrav
2031  !     Reports 1 or 0 for travExists based on if traveilling exists (1)
2032  !   or not (0). Also reports the approximated wavespeed.
2033  !=====================================================================
2034  subroutine checkTravWave(M, Mprev, row, col, travExist, wavespeed, height, eTrav)
2035    implicit none
2036    integer, intent(in) :: row, col
2037    real, intent(in) :: height, eTrav
2038    real, dimension(row * col), intent(in) :: M, Mprev
2039
2040    integer, intent(out) :: travExist
2041    real, intent(out) :: wavespeed
2042
2043    integer :: i, wavePoint1, wavePoint2
2044    real :: diff  ! placeholder for difference between M and Mprev
2045    wavePoint1 = -1
2046    wavePoint2 = -1
2047
2048    do, i=row,1,-1
2049      ! -3 because each column is 4 and I want to start at 1
2050      if (M(i*col-3) > 0.09 - eTrav .AND. M(i*col-3) < 0.09 + eTrav) then
2051        wavePoint1 = i
2052        exit
2053      endif
2054    enddo
2055
2056    do, i=row,1,-1
2057      if (Mprev(i*col-3) > 0.09 - eTrav .AND. Mprev(i*col-3) < 0.09 + eTrav) then
2058        wavePoint2 = i
```

```
      exit
    endif
  enddo

  ! Here the wavespeed is in 'i' units
  wavespeed = abs(wavePoint1 - wavePoint2)


  travExist = 1
  do i = 1, row-int(wavespeed)
   !write(*,*) wavespeed, i,  M((i - int(wavespeed))* col - 3 ), Mprev(i*col - 3)
    diff = abs(M((i+int(wavespeed)) * col - 3) - Mprev(i * col - 3))
    if ( diff > eTrav*10 ) travExist = 0
  enddo
  if (wavespeed == 0) travExist = 0 ! Can't have trav wave with 0 speed

  ! Here wavespeed is converted to dimensionless units over X time
  wavespeed = wavespeed / float(row)


end subroutine checkTravWave



!=============================================================================
!   Writes a bunch of statistics to file
!-----------------------------------------------------------------------------
!   Input:
!       avgIters = average number of iterations from between solutions
!       maxIters = maximum number of iterations from between solutions
!       avgNit   = average number of iterations from linear solver
!       maxNit   = maximum number of iterations from linear solver
!       time     = time to complete solveOrder
!   Output:
!       write everything to the file.
!=============================================================================
subroutine reportStats(avgIters,maxIters,avgNit,maxNit,time)
  implicit none
  real,intent(in)::avgIters,maxIters,avgNit,maxNit
  real,intent(in)::time
  integer :: stat

  open(UNIT = 125, IOSTAT = stat, FILE = "statReport.dat", STATUS = "old")
  if (stat .EQ. 0) close(125, STATUS = "delete")
  open(UNIT = 128, FILE = "statReport.dat", POSITION = "append", ACTION = "write")

  write(128,*) "Statsitcs:"
  write(128,*) "-------------------------------------------------------------"
  write(128,*) "Time to compute = ", time
  write(128,*) ""
  write(128,*) "Avg Iters for iterating betn. soln. =", avgIters
  write(128,*) "Max Iters for iterating betn. soln. =", maxIters
  write(128,*) "Avg Iters for linear solver =", avgNit
  write(128,*) "Max Iters for linear solver =", maxNit

```

```
2114   close(128)
2115
2116 end subroutine
2117
2118
2119
2120 subroutine amuxd (n,x,y,diag,idiag,ioff)
2121 !-----------------------------------------------------------------------
2122 !         Mnew times a vector in Diagonal storage format (DIA)
2123 !         f90/f95 version of the sparskit f77 subroutine
2124 !-----------------------------------------------------------------------
2125 ! multiplies a matrix by a vector when the original matrix is stored
2126 ! in the diagonal storage format.
2127 !-----------------------------------------------------------------------
2128 !
2129 ! on entry:
2130 !----------
2131 ! n      = row dimension of Mnew
2132 ! x      = real array of length equal to the column dimension of
2133 !          the Mnew matrix.
2134 ! ndiag  = integer. The first dimension of array adiag as declared in
2135 !          the calling program.
2136 !          (obscolete (=n always)
2137 ! idiag  = integer. The number of diagonals in the matrix.
2138 ! diag   = real array containing the diagonals stored of Mnew.
2139 ! idiag  = number of diagonals in matrix.
2140 ! diag   = real array of size (ndiag x idiag) containing the diagonals
2141 !
2142 ! ioff   = integer array of length idiag, containing the offsets of the
2143 !           diagonals of the matrix:
2144 !           diag(i,j) contains the element a(i,i+ioff(j)) of the matrix.
2145 !
2146 ! on return:
2147 !-----------
2148 ! y      = real array of length n, containing the product y=Mnew*x
2149 !
2150 !-----------------------------------------------------------------------
2151 implicit none
2152     integer, intent(in)::  n, idiag
2153     integer, intent(in),dimension(idiag) :: ioff
2154     real, dimension(n), intent(in) :: x
2155     real, dimension(n,idiag), intent(in) :: diag
2156     real, dimension(n), intent(out) :: y
2157     integer :: j, io, i1, i2, i
2158
2159     !$omp parallel shared(y,diag,x,n) private(j,io,i1,i2)
2160
2161     !!$omp workshare
2162     !$omp do
2163     do i=1,n
2164         y(i)=0.
2165     enddo
2166     !$omp enddo
2167     !!$omp end workshare
2168
```

```
2169      do j=1, idiag
2170           io = ioff(j)
2171           i1 = max0(1,1-io)
2172           i2 = min0(n,n-io)
2173           !$omp do
2174           do i=i1,i2
2175                y(i) = y(i)+diag(i,j)*x(i+io)
2176           enddo
2177           !$omp end do
2178      enddo
2179      !$omp end parallel
2180
2181 end subroutine amuxd
```