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-Reactor 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

Abstract ii

1 Introductions **1**
1.1 Introduction 1
1.2 Objectives 8
1.3 Thesis Outline 8

2 Model Definition **10**
2.1 Model Description 10
2.2 Nondimensionalization 14

3 Numerical Methods **17**
3.1 Discretization 17
3.2 Solution Technique 20
3.3 Computational Setup 25
3.4 Method Validation 26
3.5 Comparison of Semi-implicit and Fully-implicit Method 33

4 Simulation Results **36**
4.1 Typical Simulation 36
4.2 Travelling Wave Analysis 42
4.3 Spatial Effects 52

5 Conclusions **57**
5.1 Lessons Learned 57
5.2 Future Work 58

Bibliography **59**

A Default Parameter Values **62**

B Source Code **63**

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 poised 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 lead to researching biofilms as a potential means of using plant biomass to generate sustainable fuels. These fuels are generated by conversion of terrestrial or aquatic biomass into fuels such as carbon dioxide (CO_2) or ethanol (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 biofilms as a source of biofuel means that the crop leftover, cellulose, is utilized so that no edible goods are used like with corn ethanol production. *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.

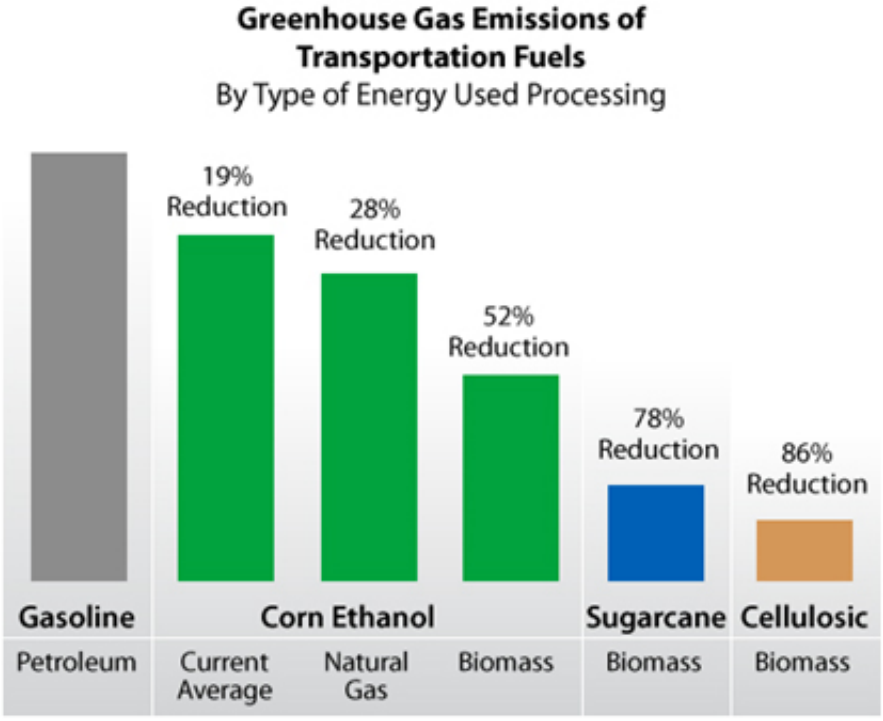


Figure 1.1: A comparison graph of greenhouse gas emissions for transportation fuels produced based on different fuel types. Each are related to the decreased emissions when compared to Petroleum. Graph indicating high greenhouse gas emissions for gasoline, then reductions with corn ethanol and further 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 lead 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.

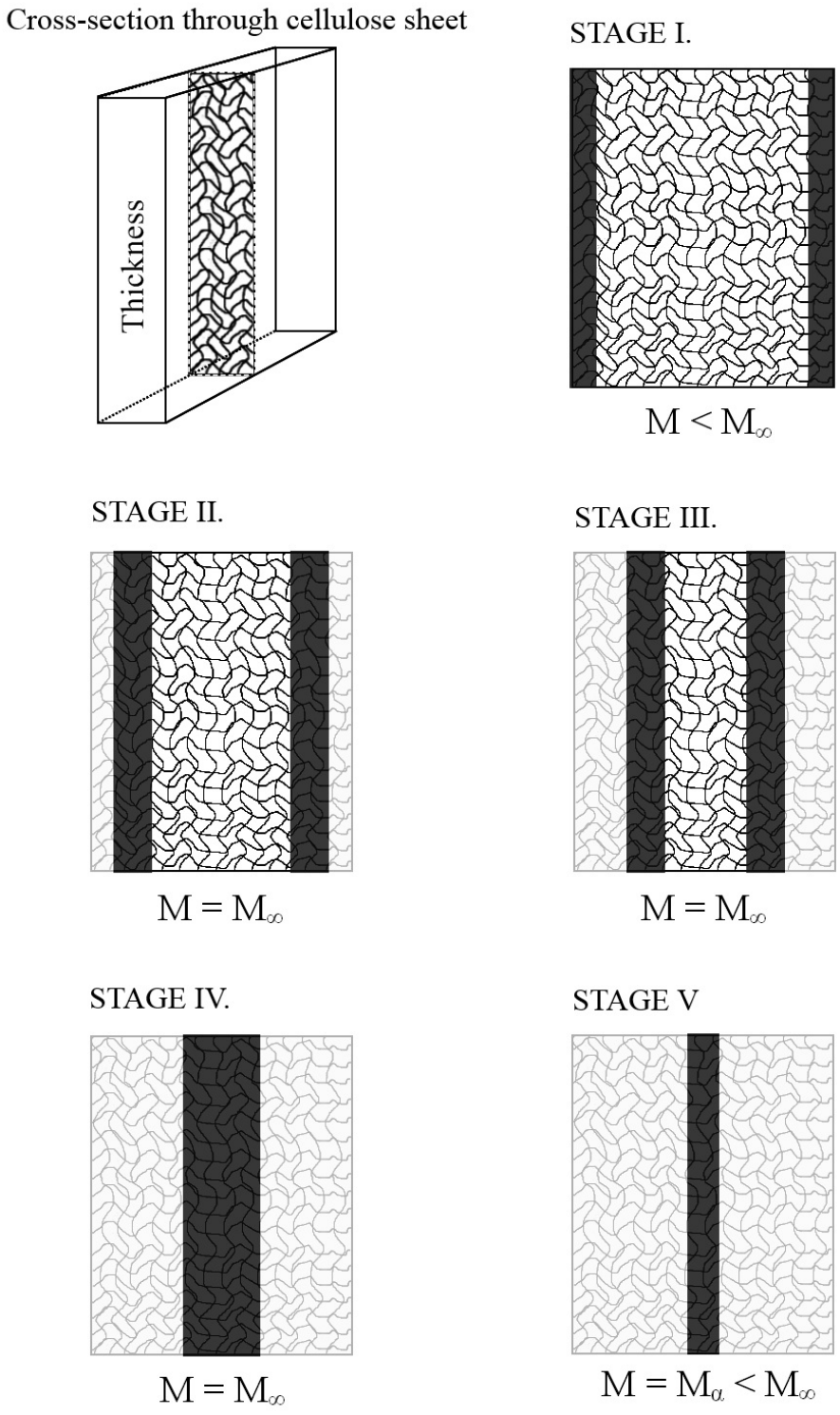


Figure 1.2: Conceptual model of cellulolytic biofilm growth and consumption of cellulose sheets. Attachment and growth occurs on both sides of the sheet, individual monolayers form on each fiber and result in a band of active biofilm (i.e., the effective sessile biomass) M (dark band). The ideal carrying 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. In this thesis, one purpose is to extend this concept to a continuous model similar to Eberl et al. (2001) but instead using 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 arise 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 exist mathematical background for these systems of equation 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 simulation 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 and as an explicit numerical method in Macías-Díaz et al. (2013).

In this thesis, the finite-difference method is treated as a semi-implicit numerical method similar to Eberl and Demaret (2007). However, a new extension is made where multiple solutions are found using a fixed-point iteration in a single time step. By this the terms of the system are split into forward-difference terms and backwards-difference terms. The forward-difference terms are the terms that make finding an analytic solution difficult. For each iteration of the fixed-point iteration, the forward-difference terms are updated to a new approximation of their *a priori* value. This turns the semi-implicit method into a fully-implicit method since the next time step is reached once all the forward-difference terms become sufficiently close to their true value for the next time step. 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 degenrate, highly nonlinear coupled PDE-ODE system modelling *C.thermocellum*.

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.

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 simulation *C.thermocellum* biofilm formation on cellulose fibers. The numerical solution suggests that the model permits travelling wave

229 solutions that describe the spatio-temporal breakdown of the cellulose fibers. The results of the two
230 dimensional simulations are compared qualitatively against a conceptual model of fiber breakdown
231 and against lumped experimental data from the literature.

232 In Chapter 5, the general findings of chapter 3 and 4 are listed for. A number of potential extensions
233 are discussed for future works.

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 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 Ω. 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, in dependence of available substrate we denote by $f(C)$, the substrate consumption rate by $g(C)$. 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

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

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

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

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 the spatial spreading of biomass. For $0 < M << 1$ the spreading effect is negligible but when $0 << M \approx 1$ we have considerable spreading. The work done by Khassehkhan et al. (2009) shows this to be an ideal choice for modelling the spreading of biofilms. By assumption 1, the only factors effecting the biomass density is growth from nutrient conversion and diffusion from local spatially-full colonized space. For the density-dependent diffusion equation (2.3), δ is the diffusion coefficient, which controls the magnitude of this term, and the parameters α and β are selected to control the strength of the diffusion. This agrees with assumption 3 since (2.3) has a near-zero value until when $M \to 1$, 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, 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, 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 exist no diffusion term for the substrate and its growth is a scalar multiple of the biomass growth rate.

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.

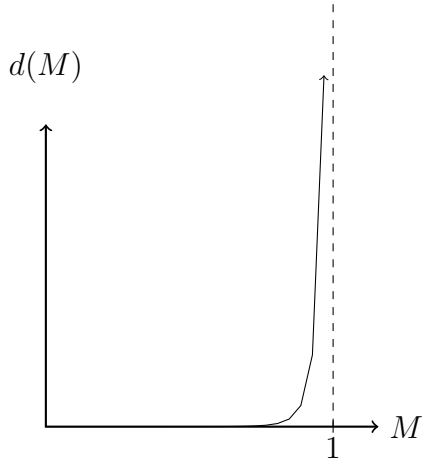


Figure 2.1: A graph of $d(M) = d\frac{M^\alpha}{(1-M)^\beta}$ showing the way diffusion increases asymptotically as $M \to 1$.

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

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.

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.

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. 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.

$$\chi = \frac{x}{L} \implies Ld\chi = dx \tag{2.9}$$

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

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

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

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

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

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

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

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

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

319 where,

320
$$\begin{aligned} D(M) &= \delta\frac{M^\alpha}{(1-M)^\beta} \\ F(\mathcal{C}) &= \frac{\mathcal{C}}{\kappa+\mathcal{C}} - \nu \\ G(\mathcal{C}) &= \gamma\frac{\mathcal{C}}{\kappa+\mathcal{C}}. \end{aligned} \tag{2.18}$$

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

322
$$C := \mathcal{C}, \quad x := \chi, \quad t := \tau. \tag{2.19}$$

Each of the dimensionless parameters in (2.18) have a biological representation based on the transformations 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.

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.

For this system, we let the nondimensional spatial region of consideration, $\Omega = [0, 1] \times [0, 1]$, be square. 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.

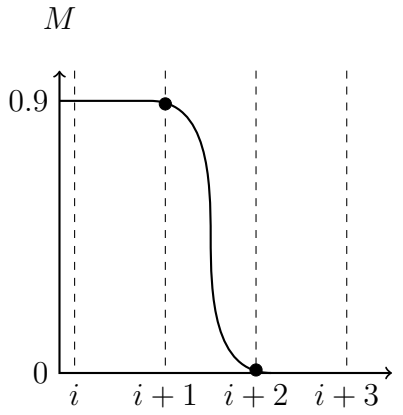
The equation for (3.1), after following the spatial discretization for finite-difference method listed in Saad (2003), is

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

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: $D\left(\frac{0.9+0}{2}\right) = 0$

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, arithmetic mean, and interpolation values.

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:

$$\begin{array}{rccc} \pi: & \{0,\ldots,n\}\times\{0,\ldots,m\} & \rightarrow & \{0,\ldots,nm\} \\ & (i,j) & \rightarrow & \pi(i,j) \end{array} \tag{3.7}$$

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,

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

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

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:

$$M_l^{(p)} := M_l^k, \quad C_l^{(p)} := C_l^k, \tag{3.10}$$

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:

$$\frac{M_l^{(p+1)} - M_l^k}{\Delta t} = \frac{1}{\Delta x^2} \sum_{\sigma \in \mathcal{N}_{ij}} \left(\frac{D(M_{\pi(\sigma)}^{(p)}) + D(M_l^{(p)})}{2} \right) \cdot \left(M_{\pi(\sigma)}^{(p+1)} - M_l^{(p+1)} \right) + F(C_l^{(p)}) M_l^{(p+1)} \tag{3.11}$$

$$\frac{C_l^{(p+1)} - C_l^k}{\Delta t} = \frac{1}{2}(G(C_l^{(p+1)}) M_l^{(p+1)} + G(C_l^k) M_l^k) \tag{3.12}$$

where $(p) \in (0, 1, 2, \ldots)$ 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.

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

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 (3.11) - (3.12), the equations can be rearranged and solved by conventional methods.

In each iteration step, (3.11) 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). For each grid point, l a linear system is defined as:

$$\frac{1}{\Delta t}M_l^k = \sum_{\sigma\in\mathcal{N}_{ij}}\left(\frac{D(M_{\pi(\sigma)}^{(p)}) + D(M_l^{(p)})}{2\Delta x^2}\cdot M_{\pi(\sigma)}^{(p+1)}\right) + \sum_{\sigma\in\mathcal{N}_{ij}}\left(\left(\frac{D(M_{\pi(\sigma)}^{(p)}) + D(M_l^{(p)})}{2\Delta x^2}\right) - F(C_l^{(p)}) + \frac{1}{\Delta t}\right)M_l^{(p+1)}. \tag{3.15}$$

From (3.15), we get the matrix equation:

$$A^{(p)}M^{(p+1)} = \frac{1}{\Delta t}M^k. \tag{3.16}$$

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

$$A^{(p)} = \begin{bmatrix} a_{1,1} & a_{1,2} & & a_{1,m} & & & & & & \\ a_{2,1} & \ddots & \ddots & & \ddots & & & & & \\ a_{n,1} & \ddots & \ddots & \ddots & & \ddots & & & & \\ & \ddots & \ddots & \ddots & \ddots & & \ddots & & & \\ & & \ddots & \ddots & \ddots & \ddots & & \ddots & & a_{nm-n,nm} \\ & & & \ddots & \ddots & \ddots & \ddots & & \ddots & a_{nm-1,nm} \\ & & & & \ddots & \ddots & \ddots & \ddots & a_{nm,nm-m} & a_{nm,nm-1} & a_{nm,nm} \end{bmatrix} \tag{3.17}$$

where each a is the coefficients for specific grid indices based on (3.15).

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

Proof. Matrix A is positive definite if all the eigenvalues are positive. Using the Gershgorin Circle theorem described by Varga (2004), the eigenvalues can be shown to be positive if, independently on all rows, the sum of the off-diagonals values is less then the diagonal value. This can be verified. From (3.15) it can be said that,

$$\left|\sum_{\sigma\in\mathcal{N}_{ij}}\left(\frac{D(M_{\pi(\sigma)}^{(p)})}{\Delta x^2}\right)\right| < \left|\left(\sum_{\sigma\in\mathcal{N}_{ij}}\left(\frac{D(M_{\pi(\sigma)}^{(p)})}{\Delta x^2}\right) - F(C_l^{(p)}) + \frac{1}{\Delta t}\right)\right|. \tag{3.18}$$

This simplifies to,

$$\Delta t < \left(F(C_l^{(p)})\right)^{-1} \tag{3.19}$$

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 are actually reference to the same grid point. Therefore we have symmetry. □

It is important to remark that this condition for which A is positive definite and symmetric is practically not a severe constraint. 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 dominate when $\Delta t < \left(F(C_l^{(p)}\right)^{-1}$.*

Proof. This is trivially shown to be true when one considers (3.18). It was shown that this simplifies to

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

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 dominate 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 (3.12), the equation can be rearranged into a quadratic form, substituting in $G(C)$ from (2.18)

$$\left(C^{(p+1)}\right)^2 + \left(\kappa - C^k + \frac{\Delta t}{2}\gamma M^{(p+1)} + \frac{\Delta t}{2}\frac{\gamma C^k M^k}{\kappa + C^k}\right) C^{(p+1)} + \left(-\kappa C^k + \frac{\Delta t}{2}\frac{\gamma \kappa C^k M^k}{\kappa + C^k}\right) = 0. \tag{3.21}$$

Using the quadratic equation results in,

$$C^{(p+1)} = \frac{-b \pm \sqrt{b^2 - 4c}}{2} \tag{3.22}$$

for which,

$$\begin{aligned} b &= \kappa - C^k + \frac{\Delta t}{2}\gamma M^{(p+1)} + \frac{\Delta t}{2}\frac{\gamma C^k M^k}{\kappa + C^k} \\ c &= -\kappa C^k + \frac{\Delta t}{2}\frac{\gamma \kappa C^k M^k}{\kappa + C^k} \end{aligned} \tag{3.23}$$

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 (3.21) 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 (3.21) so that all the Δt terms are on the right-hand-side, we get

$$(C)^2 + \left(\kappa - C^k\right) C - \kappa C^k = \left(\frac{\gamma C^k M^k}{2\left(\kappa + C^k\right)} - \left(\frac{\gamma M^{(p+1)}}{2} - \frac{\gamma C^k M^k}{2\left(\kappa + C^k\right)}\right) C\right) \Delta t. \tag{3.24}$$

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, from the combinations of $\bar{a} \geq 0, \bar{a} < 0$ and $\kappa - C^k \geq 0, \kappa - C^k < 0$. These four cases are 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

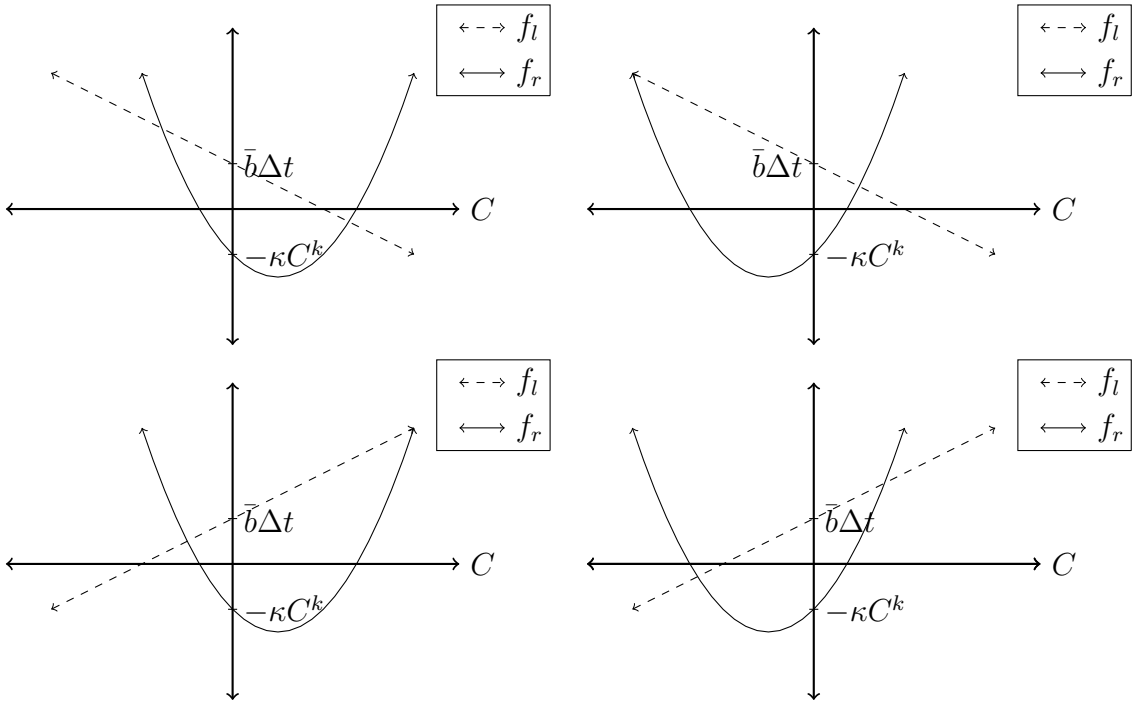


Figure 3.2: Graph of $f_l = (C)^2 + \left(\kappa - C^k\right) C - \kappa C^k$ and $f_r = \left(\bar{b} - \bar{a}C\right) \Delta t$ for all four possible cases. Notice that because $-\kappa C^k < 0$ and $\bar{b}\Delta t > 0$ for all realistic parameter values the two functions will always intersect in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

$f_r(0) > f_l(0)$, 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 (3.22) 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 (3.12 - 3.11).

Note that Algorithm 1 actually describes both a fully- and semi- implicit method for solving (2.16).

Data: M^k, C^k are vectors with values from the previous time step and $p = 0$.
begin
  Let $M^{(0)} = M^k$ and $C^{(0)} = C^k$;
  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;
    Let $C^{(p)} = C^{(p+1)}$;
    Let $M^{(p)} = M^{(p+1)}$;
    Let $p = p + 1$;
  end
  Let $M^{(k+1)} = M^{(P)}$ and $C^{(k+1)} = C^{(P)}$;
end

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

If $P = 1$ then only a single iteration of the algorithm is applied, which correlates to a semi-implicit method would behave. This can be produced by selecting a sufficiently large enough tolerance so that the convergence check is always resolved after the first iteration. The resulting semi-implicit method is effectively the same as one described in Sirca and Horvat (2012).

3.3 Computational Setup

The implementation of Algorithm 1 was done with Fortran. The system is stored in diagonal format, since A has five distinct diagonals.

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 selection of 4 threads is because the computation time decrease becomes less efficient after more then 4 threads. The GNU Fortran compiler, version 4.4.7, was used for all computations; the compiler arguments were

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

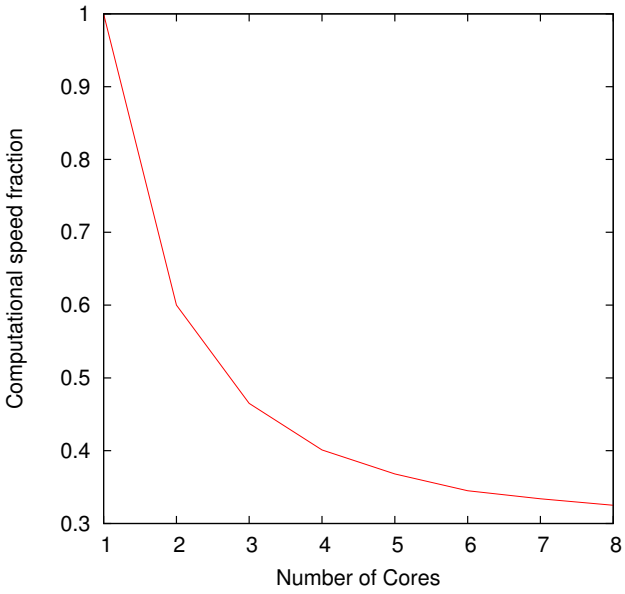


Figure 3.3: Computation speed of simulations ran with different number of cores. The y-axis represents the fraction of time each core has in comparison to the 1-core result. Same simulation setup as the travelling wave simulation. Ran until $t = 2$ with a grid of 2049×2049.

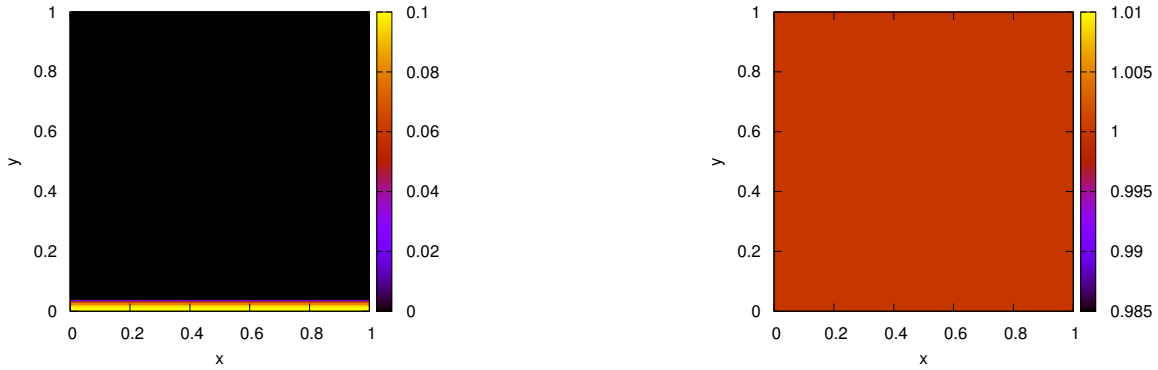
3.4 Method Validation

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.

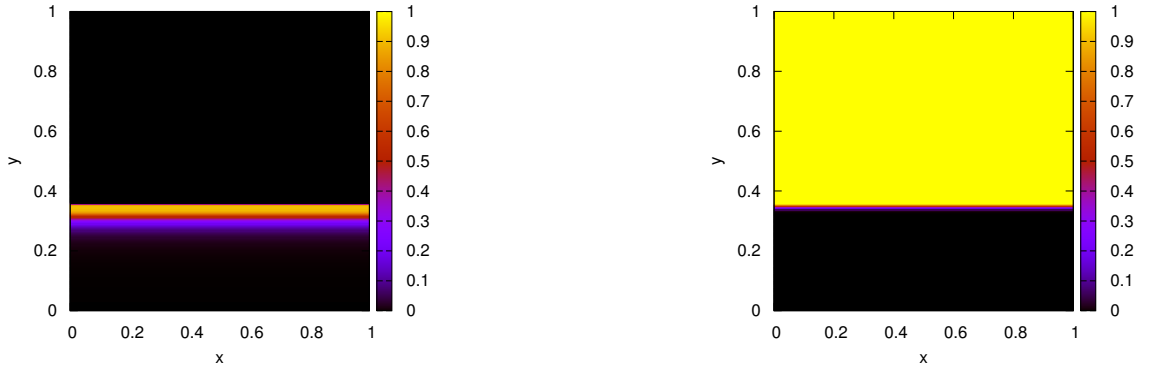
3.4.1 Basic Simulations

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



(a) (b)



(c) (d)

Figure 3.4: Solutions for (ac) M and (bd) C with 1D initial conditions defined in (3.25) at (ab) $t = 0$ and (cd) $t = 40$. Computed with a 1025×1025 grid and a time step of $\Delta t = 10^{-3}$.

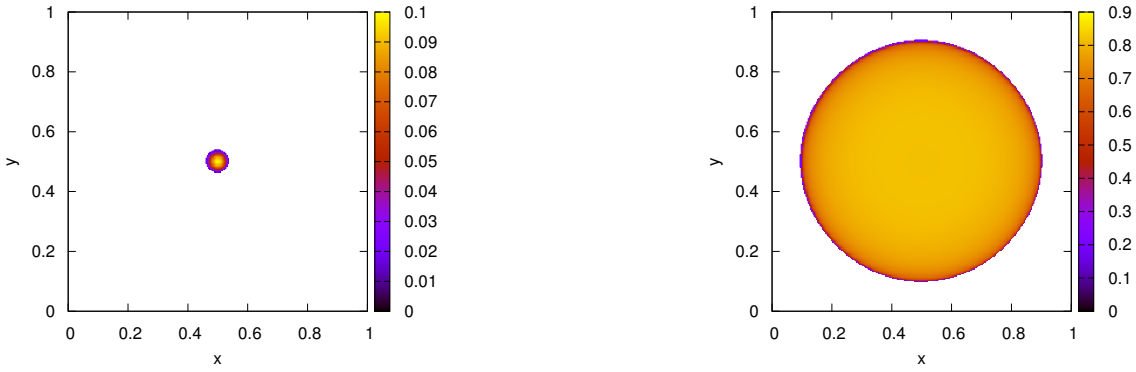
qualify as a 1D initial condition. These initial conditions will be defined as:

$$M(0,x,y) = \begin{cases} -\left(\frac{h}{d^4}\right) x^4 + h & \text{, if } y \leq d \\ 0 & \text{, otherwise} \end{cases} \tag{3.25}$$

$$C(0,x,y) = 1$$

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.



(a) (b)

Figure 3.5: Solutions for M with spherical initial conditions defined by (3.26) at (a) $t = 0$ and (b) $t = 40$. Computed with a 1025×1025 grid and a time step of $\Delta t = 10^{-3}$.

Another characteristic to observe would be if a spherical initial condition remains spherical. Using initial conditions for the biomass,

$$M(0, x, y) = \begin{cases} -\frac{h}{d^2}\left(x - 0.5\right)^2 + \left(y - 0.5\right)^2) + h & \text{, if } \left(x - 0.5\right)^2 + \left(y - 0.5\right)^2 < d^2 \\ 0 & \text{, otherwise} \end{cases}, \tag{3.26}$$

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.

Given the boundary conditions and spatial discretization, there could be a possible source or sink of biomass when it must diffuse along the boundary of the region. 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 the biomass over the region,

$$\begin{aligned} T_M &= \int_\Omega (\nabla_x (D(M)\nabla_x M) + aM \; dA) \\ &= \int_\Omega \nabla_x (D(M)\nabla_x M) \; dA + \int_\Omega aM \; dA \\ &= \int_{\partial\Omega} D(M)\nabla_n M \cdot n \; dn + \int_\Omega aM \; dA \\ &= \int_{\partial\Omega} D(M)(0) \cdot n \; dn + \int_\Omega aM \; dA \\ &= \int_\Omega aM \; dA \\ &= T_M(0)e^{at} \end{aligned} \tag{3.27}$$

Numerically, this is computed by grid-wise summation,

$$T_M(t^k) \approx T_M^k = \frac{\sum_i^n \sum_j^m M_{i,j}^k}{nm}. \tag{3.28}$$

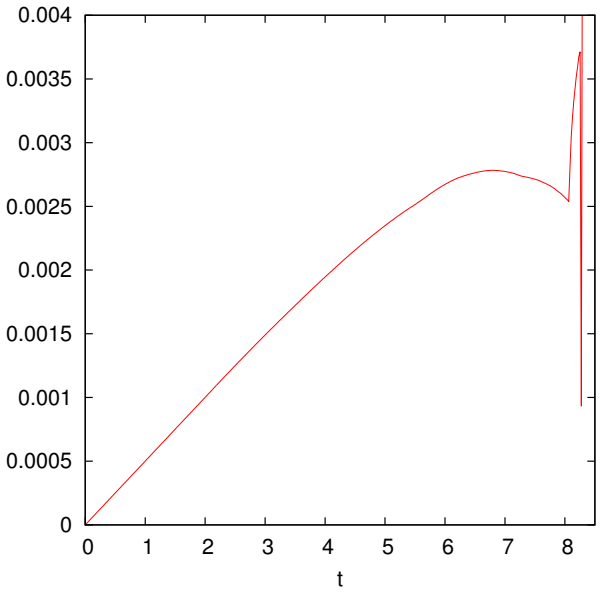


Figure 3.6: Plot of the relative error, $\frac{|f_1 - f_2|}{|f_2|}$, between the computed total biomass, $f_1 = T_M(t)$, and the theoretical total biomass, $f_2 = y_0 e^x$. The changes after $t = 8$ are from the biomass having completely filled the region Ω. This means that there is no physical space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

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.

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:

$$\epsilon_{sol} = \frac{||u_1 - u_2||}{||u_2||} \tag{3.29}$$

where u_1 represents one simulation solution and u_2 represents the solution that is theoretically 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.

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:

$$\ell_1 : ||u||_1 = \frac{1}{nm} \sum_{i}^{nm} |u_i| \tag{3.30}$$

$$\ell_2 : ||u||_2 = \frac{1}{nm} \sqrt{\sum_{i}^{nm} (u_i)^2} \tag{3.31}$$

$$\ell_\infty : ||u||_\infty = \max_{i=1,\dots,nm} |u_i| \tag{3.32}$$

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 for 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 (3.25) 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_∞ doesn't vary with the grid size, since the wave front has a steep interface and tends to lead to 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

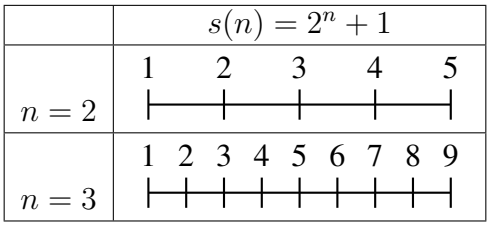


Figure 3.7: Visualization in 1D to illustrate the choice of $s(n) = 2^n + 1$ instead of 2^n for the grid size selection. Here is can be seen that successive grid size selections using $s(n)$ line up on certain grid points and when using 2^n no grid points are equivalent.

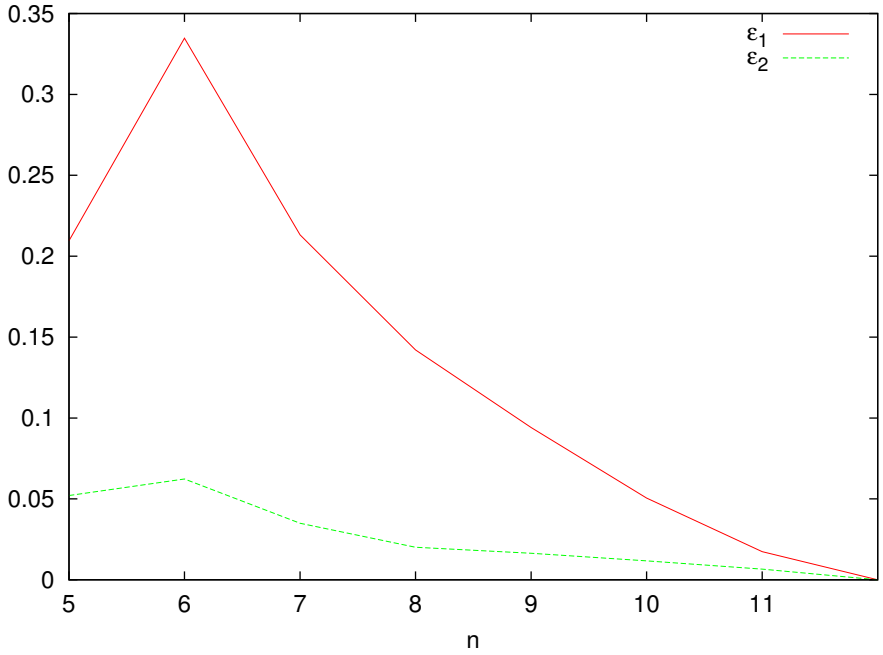


Figure 3.8: Plot showing the convergence of solutions based on changes in Δx. The computations are of ϵ_{ℓ_1} and ϵ_{ℓ_2} with grid-size following $s(n) = 2^n + 1$.

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. The non-monotonicity for $n = 5$ is because the grid has become too large for predictable calculations, which is an acceptable error since such a coarse grid would never be used for computational simulations.

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. Recall that the semi-implicit method is a special case 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 an idea of the extra work can be formed. This average is based on the average number of iteration of the fully-implicit method 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 correspond to the difference between a pair of solutions, u_1 and u_2. The choice of u_2 here is the semi-implicit methods solution. 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. Theoretically 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 positive 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. So the semi-implicit method results in a tolerance between 10^{-4} and 10^{-5} since any tolerance forced below that does not require addition iterations.
- The differences in Wave Height are a result of addition 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 compute time.

- After 36 iterations a 6^{th} digit of accuracy is gained for a 900% increase in compute 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 the 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 in a collision. 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-

643 lation points of both sides, totalling 80. The fully-implicit method is used here with $tol = 10^{-8}$.

644 4.1.1 Biomass Ratio

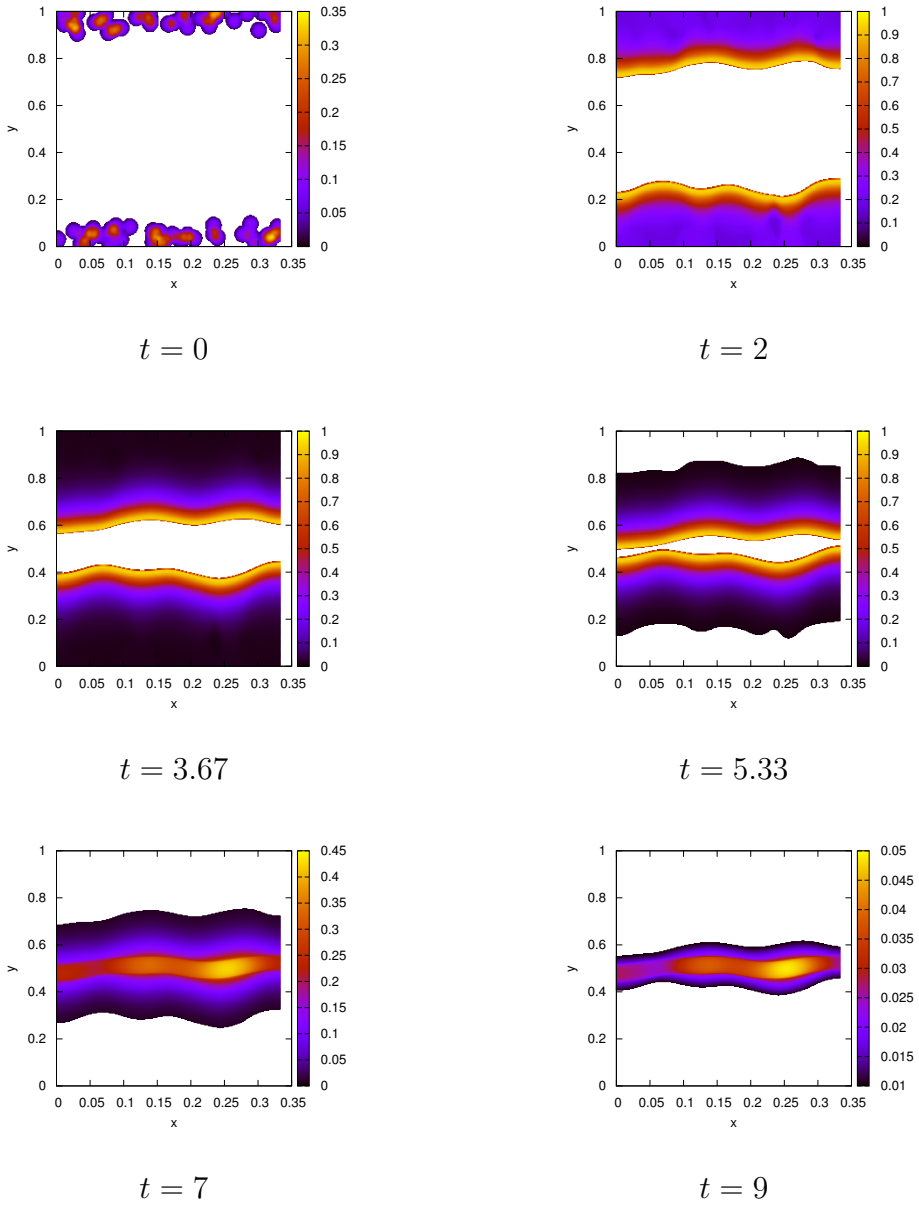


Figure 4.1: A graph showing the M solution of a typical simulation at different time steps. The initial condition is 80 random spherical inoculation points evenly divided between each of the $y = 0$ and $y = 1$ sides. A 513×513 grid was used.

645 Figure 4.1 shows the time evolution of M for the simulation. Here, the random inoculations on both

646 sides of 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.

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 band 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 values 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 (2014), we can equate the change in production of CO_2

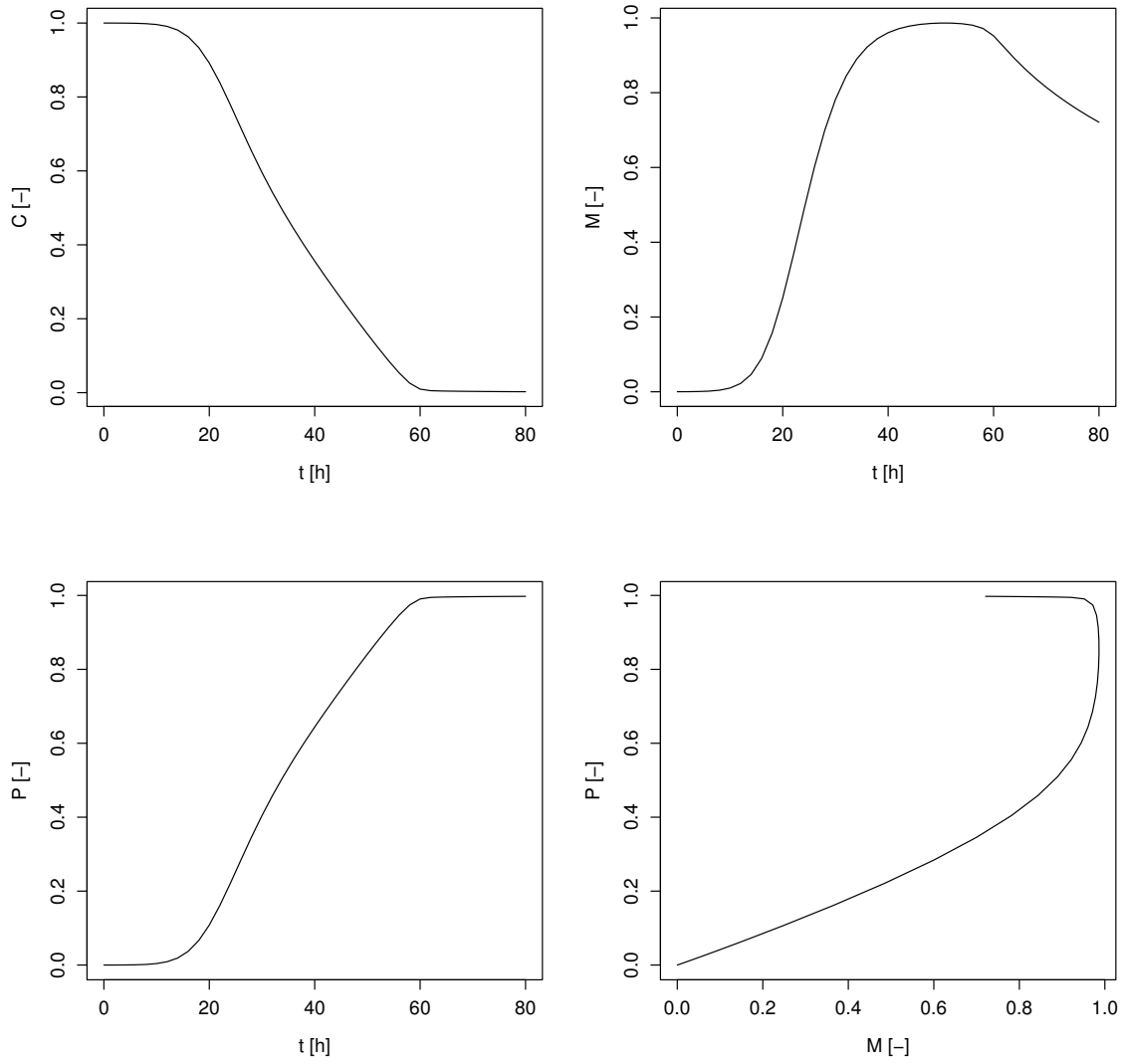


Figure 4.2: A typical model simulation from the simple ODE model. Shown are substrate concentration C (top left, normalised), effective sessile biomass M (top right, relative to the ideal carrying capacity M_∞), and CO_2 product P (bottom left, in moles) as functions of time t;. Also shown is the product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

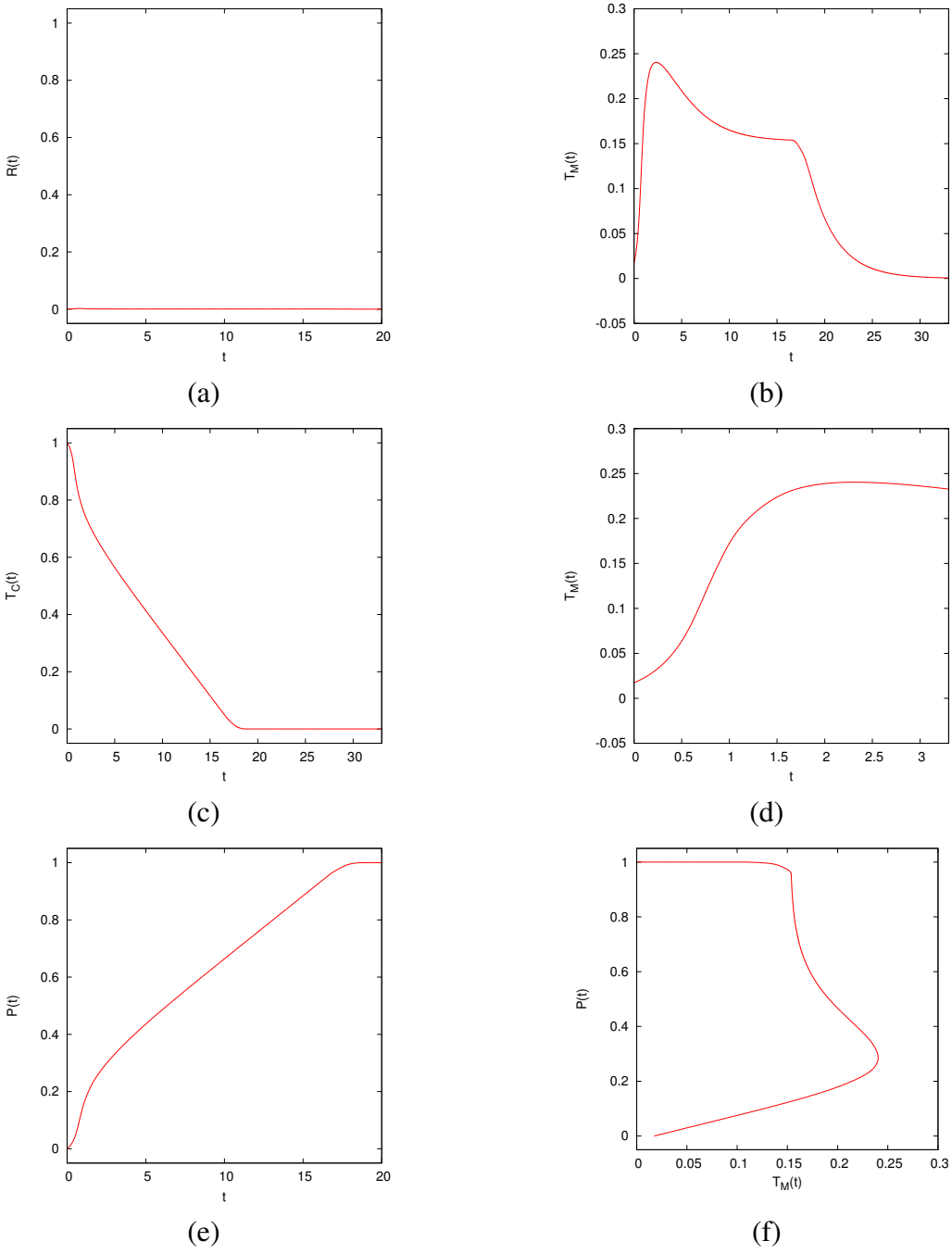


Figure 4.3: Total value of certain qualities from the typical simulation. Here we have: (a) $\mathcal{R}(t)$, the rate of CO_2 production, (b) total M as a function of time, (c) total C as a function of time, (d) total M as a function of time zoomed in from $t = 0$ to $t = 3$, (e) $\mathcal{P}(t)$, the 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 $\delta = 10^{-8}$.

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 (2014). 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.

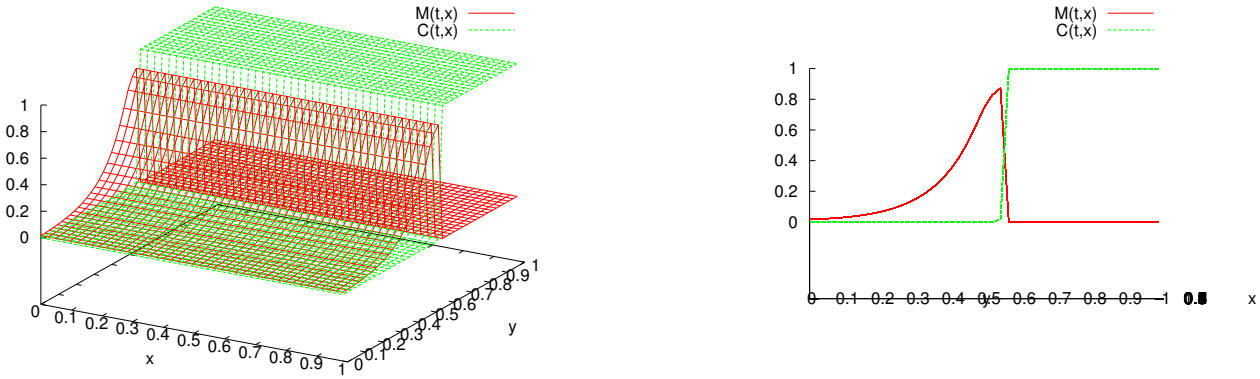
4.2 Travelling Wave Analysis

4.2.1 Spatial Simplification

To simplify the travelling wave analysis we reduce the spatial dimensions to that of a 1D 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 2D. 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 2D, 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 travelling wave 1D initial conditions, (3.25), 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 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 $x = 5.75$, which



(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 $t = 40$.

is mainly because of the sharp change in values.

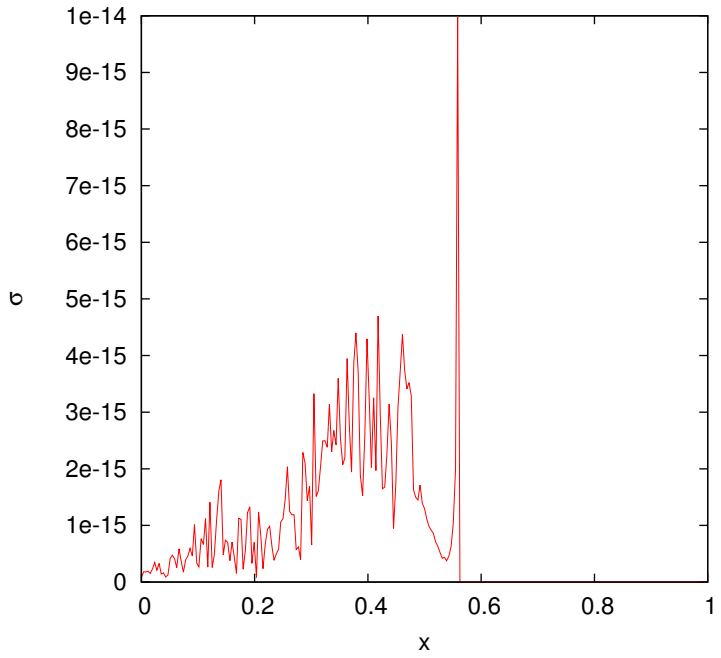


Figure 4.5: The standard deviation at the same time as the above graphs

When simulations are computed with a $n \times 4$ grid, they are still 2D problems. With regards to visualizations, side profiles could be used on these solutions to present pseudo-1D visualization but this is not ideal. To visualize the solutions in true 1D the we use $\bar{M}$ 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 2D plot as seen in Figure 4.6.

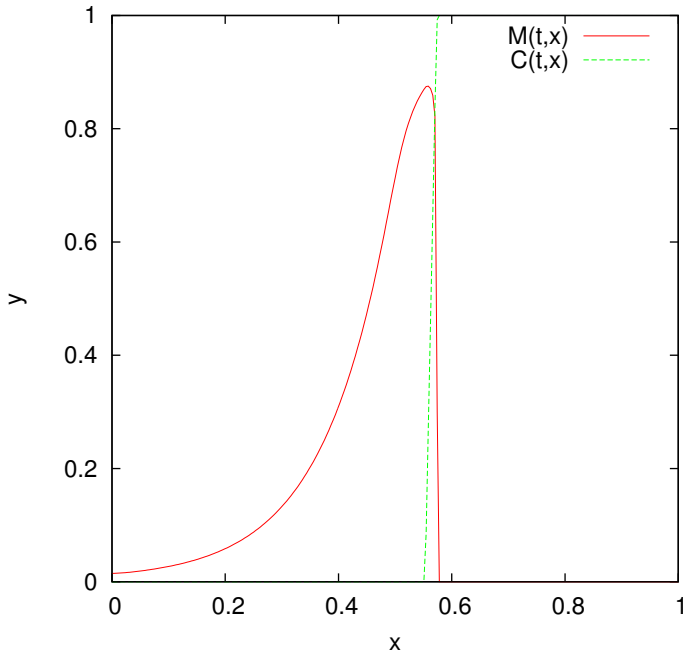


Figure 4.6: Graph of M(2,y) and C(2,y), now reduced to a 2D plot.

This means that the system (2.16) - (2.18) can be reduced to a $1D$ problem. With initial conditions that are homogenous with respect to y, we can greatly reduce the accuracy 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

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

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.

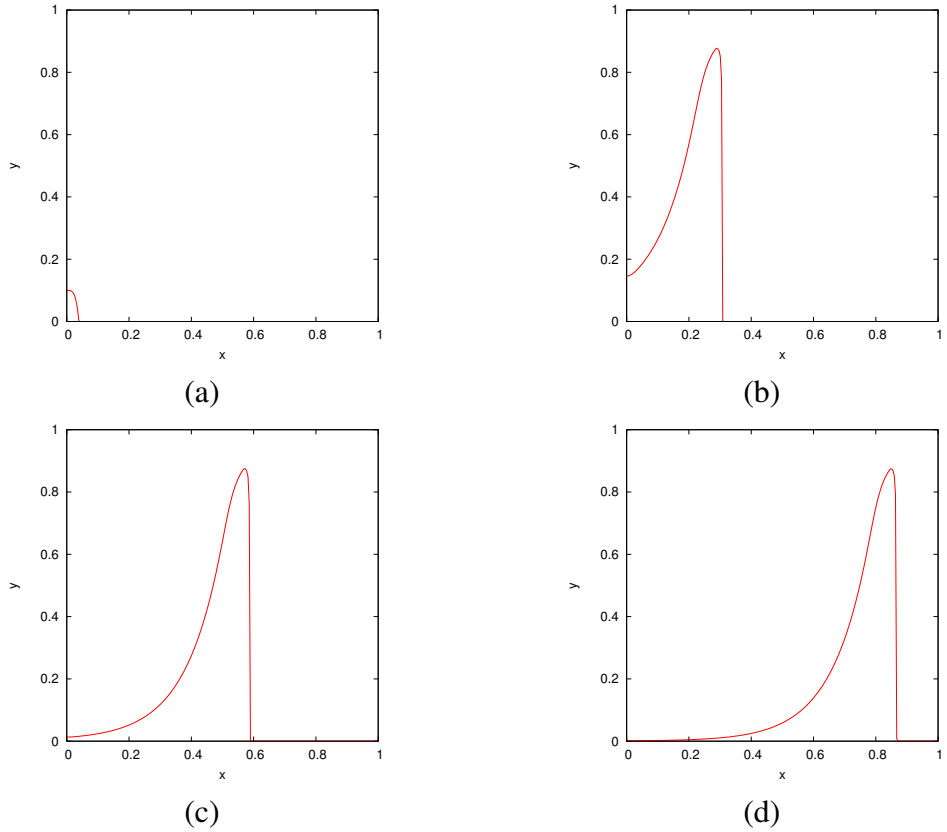


Figure 4.7: Solutions of $M(x,t)$ and $C(x,t)$ at (a) t = 0, (b) t = 20, (c) = 40, (d) = 60. This was run on a 513×4 grid.

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. Visually, multiple time steps horizontally translated onto each other would show this. If the horizontal translations are all multiples of the same number then we have shown that a constant speed exists. If the shape of all the time steps match then a constant shape then strong evidence that a constant shape exists would be shown. 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-1D 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

740 value of m in $f(x)$ is the approximation for the wave speed, c.

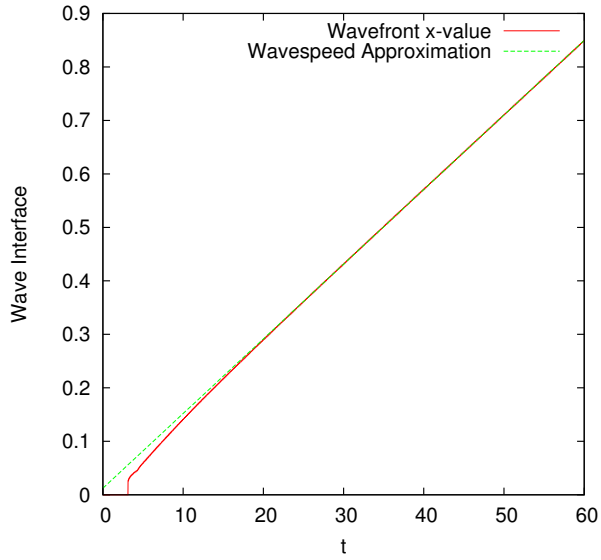


Figure 4.8: The x location of the wave peak as a function of t. The red line is the wave peak location extracted from the simulation results. The green line is the function $f(x) = cx + b$ with c as the wave speed, found by fitting the model to the second half of x values. The simulation results used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

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

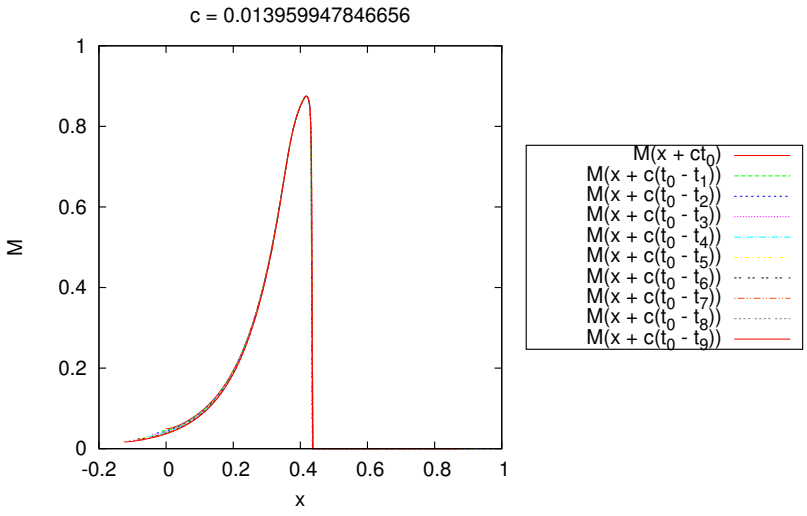


Figure 4.9: Solutions of M that are represented as $M(x - ct)$ *a priori*. The multiple time steps are translated on top of another by horizontal movements of $c(t_0 - t_n)$ for each time step.

Based on the above evidence, we can say that a travelling wave solution has been shown to exist 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. 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 nearby solution into becoming a travelling wave solution or is it only for specific cases that one results. This will help confirm that the existence of the travelling wave solution is not depended 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 to the one dimension 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 1D problem. This cannot be fully seen since the wave propagation reaches the end of the region before it can become fully one dimensional.

We can quantitatively see the behaviour of this convergence by calculating the measure of spread at

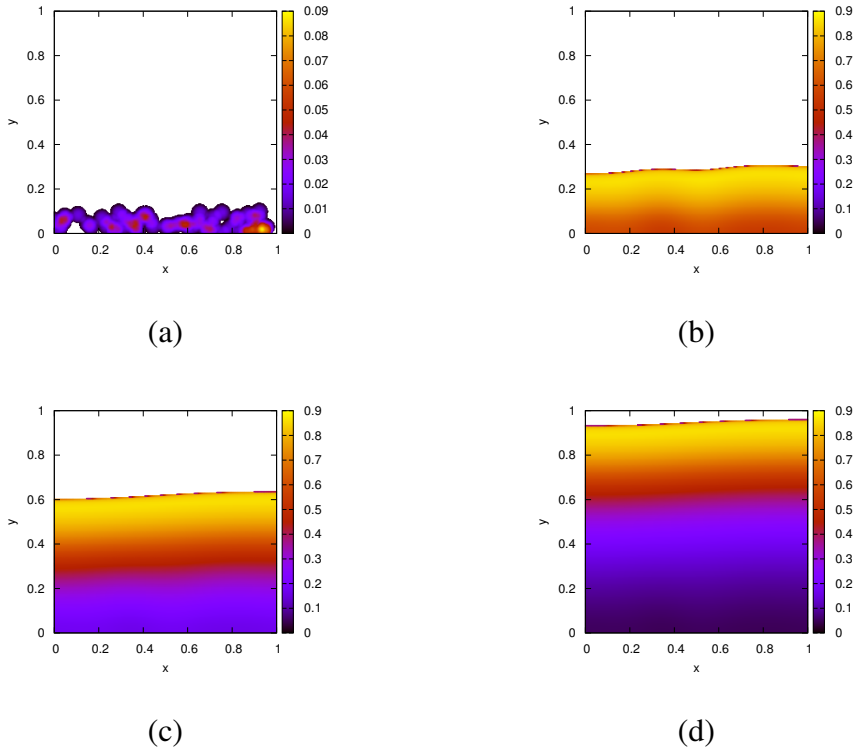


Figure 4.10: Plots of the simulation with random spherical inoculation points centered in the region $(x, y) \in [0, 0] \times [1, 0.1]$. The solutions are shown at (a) $t = 0$, (b) $t = 10$, (c) $t = 20$, and (d) $t = 30$. Each solution is computed on a 513×513 grid.

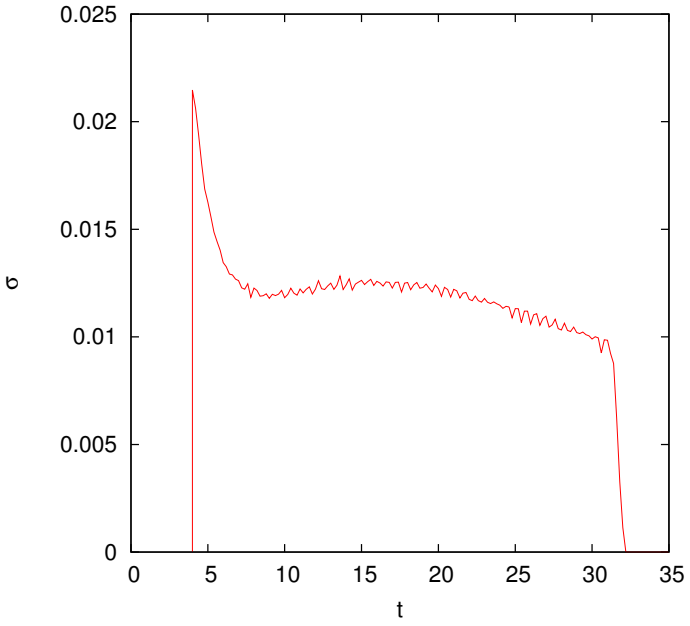


Figure 4.11: The standard deviation of the wavefront interface as a function of time. The wavefront references the largest y coordinate with $M > 0.001$ for each x coordinate. The choice of using $M > 0.001$ is because we want to ignore the small values (10^{-100}) that arise from the diffusion right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

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.

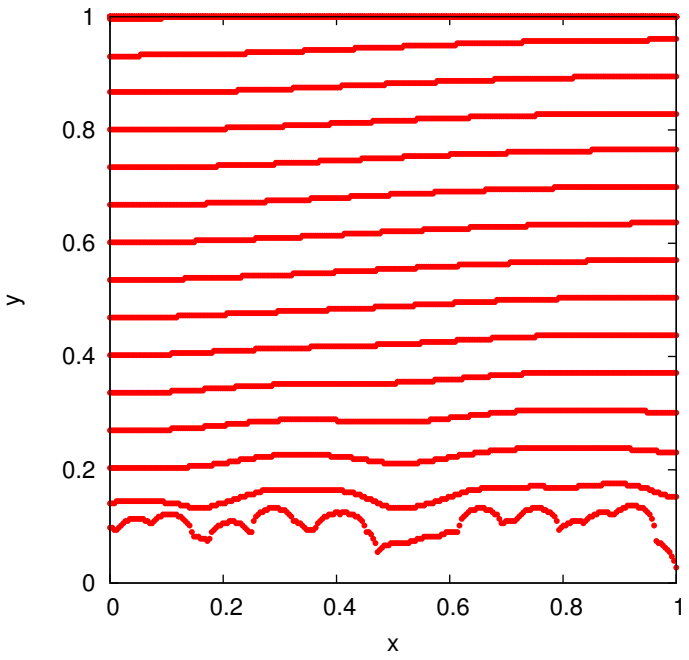


Figure 4.12: The wavefront shape of multiple time steps. Each wavefront has a difference in time by 2, i.e. they are at $t = 4, 6, 8, \ldots, 58, 60$. The simulation results are the same as the previous Figures, using the default parameter values with a grid size 513×513.

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 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 shows 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 independent 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.

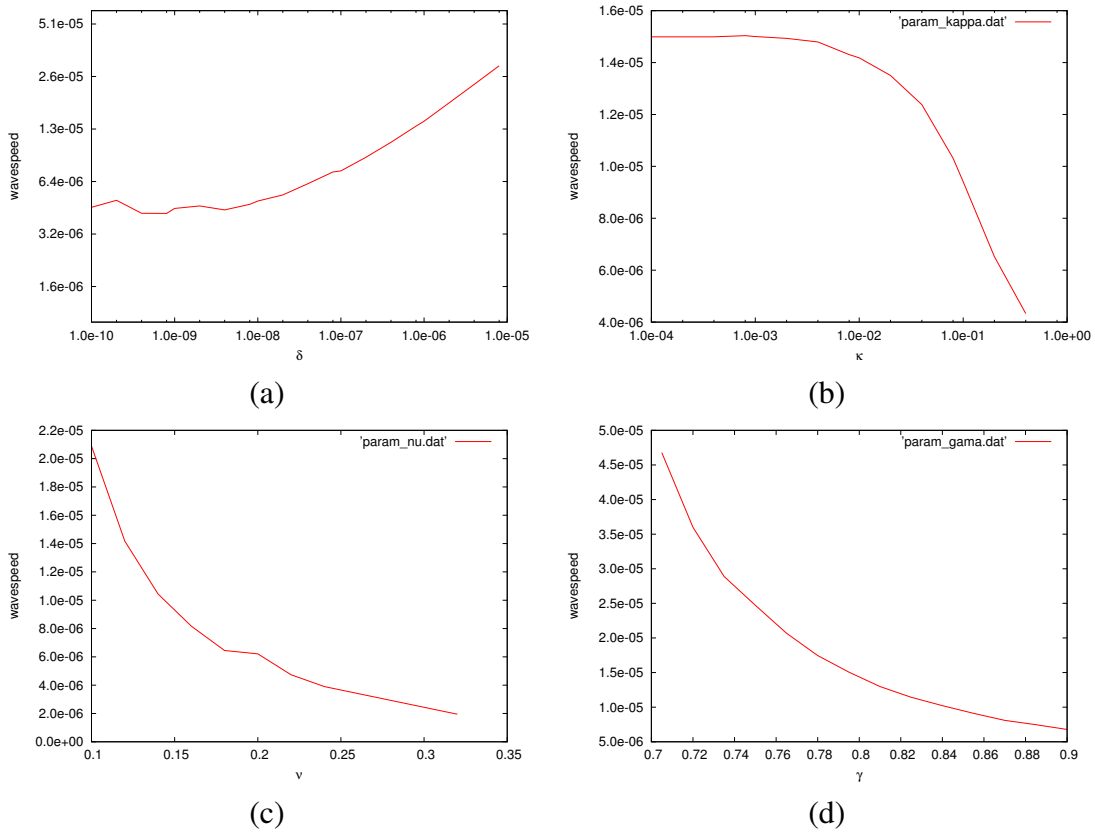


Figure 4.13: The value of c as parameter (a) δ, (b) κ, (c) ν, and (d) γ are changed. Note that (a) and (b) have logscales due to the selection of parameter values. Each of these were calculated with the same setup as the travelling solution previously done. The grid size for each was 513×4 and a time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

For this an automated script was created that checks, for each time step, if M is travelling wave solution based on the solution of 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 of the region faster then the tail of the travelling wave can decrease to 0. In the case of ν, any larger values then 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 inhibit the propagation of the biomass interface and lowers the wave speed. The simulated values recorded in Figure 4.13 agrees with the expected behaviour for each parameter.

4.3 Spatial Effects

There are many spatial effect 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

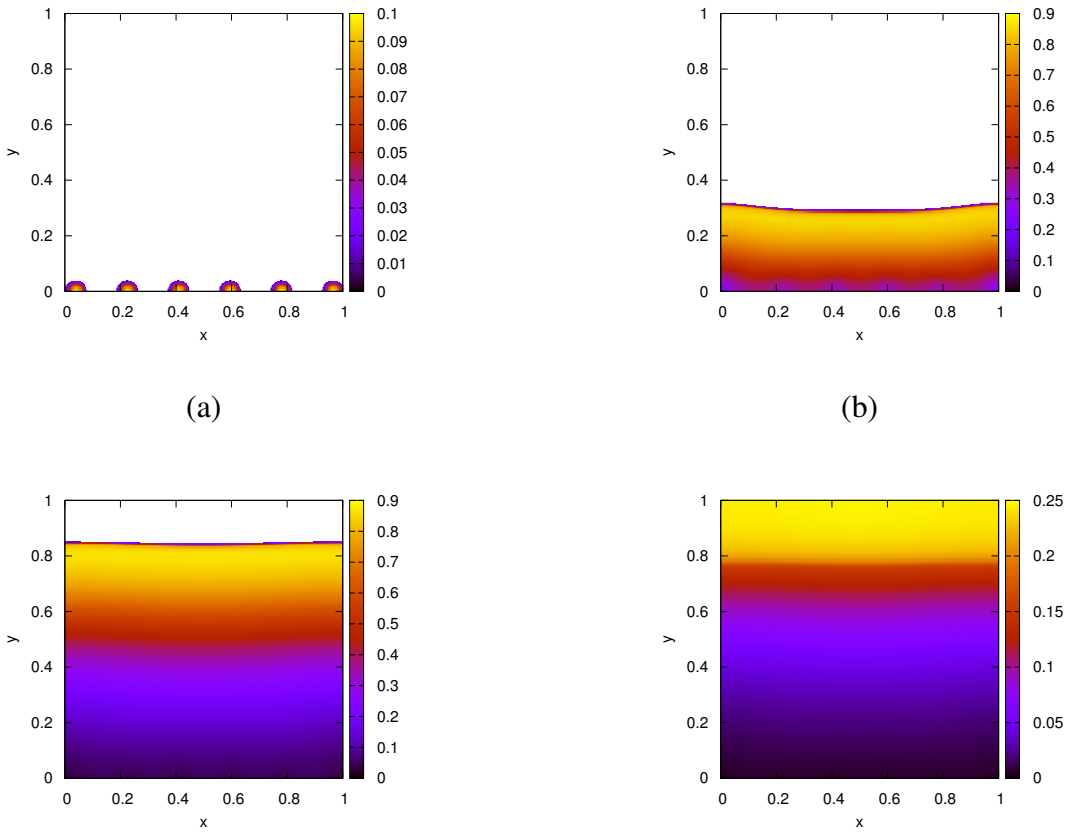


Figure 4.14: This shows the time evolution for the evenly distributed initial condition at (a) $t = 0$, (b) $t = 16$, (c) $t = 32$, (d) $t = 48$. Here default parameters are used on a grid size of 513×513.

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

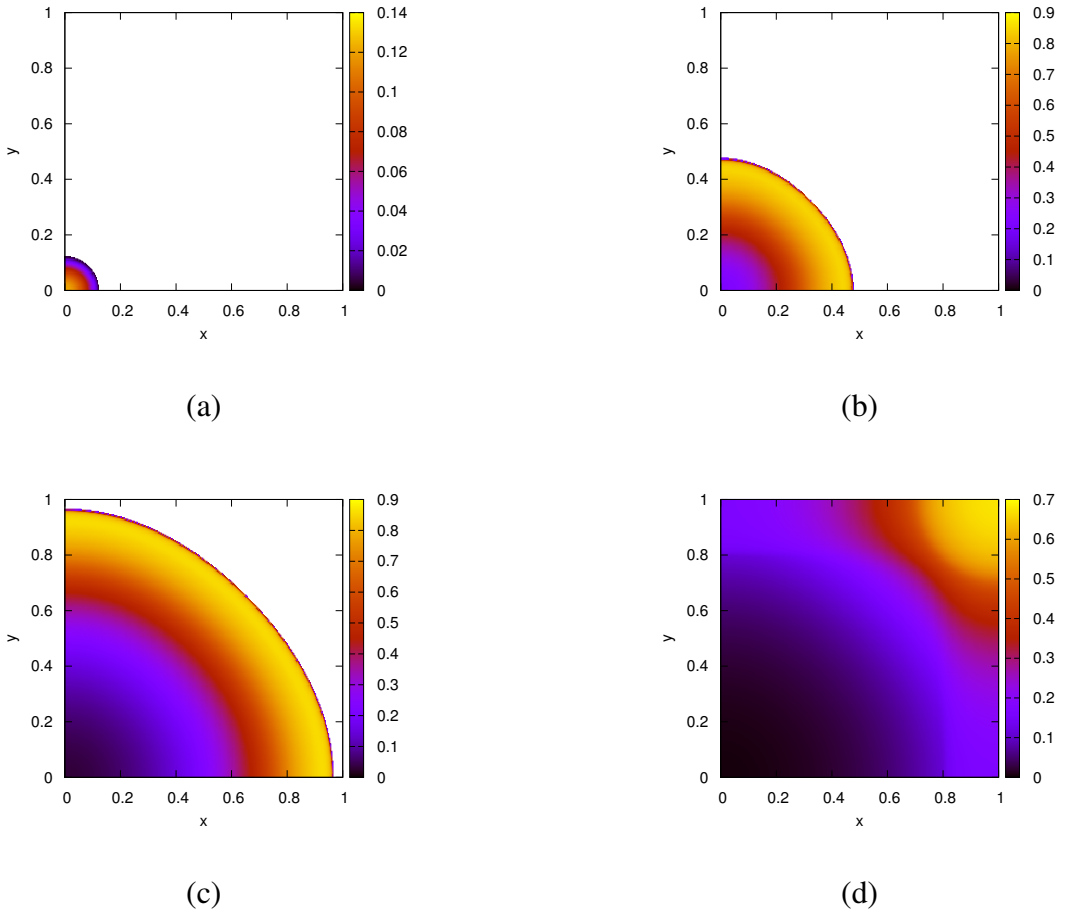
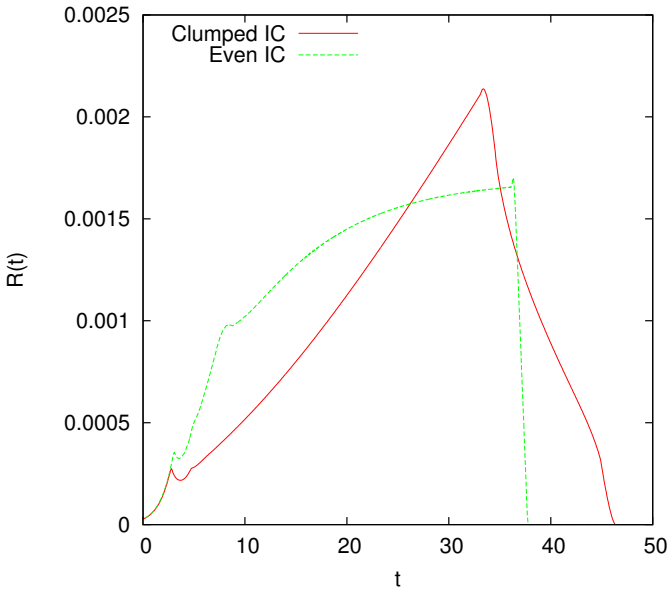


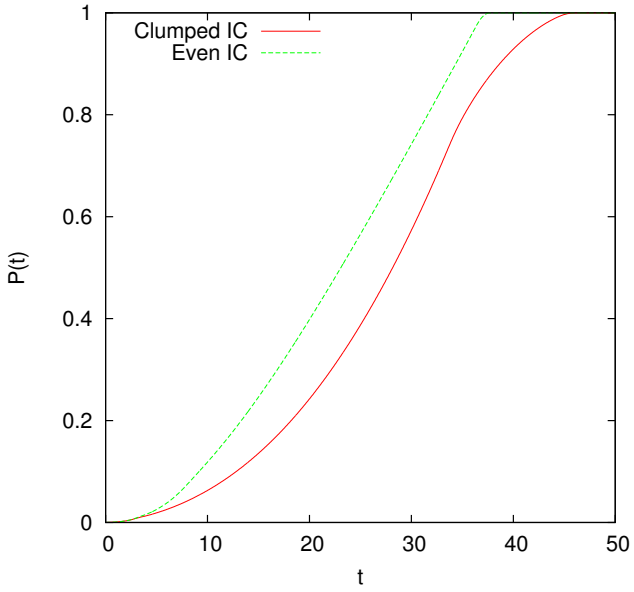
Figure 4.15: This shows the time evolution for the clumped initial condition at (a) $t = 0$, (b) $t = 16$, (c) $t = 32$, (d) $t = 48$. Here default parameters are used on a grid size of 513×513.

852 6.

853 It becomes easier to see the difference between the two spatially difference problems when CO_2
854 production is taken into account. In Figure 4.16 it is clear that there is a substantial difference between
855 the CO_2 production of both cases. This shows that the initial distribution of biomass can affect a
856 lumped, global measurement and change the reactor-scale behaviour. From this, two dimensional
857 models that take spatial consideration at a meso-scale can lead to different results.



(a)



(b)

Figure 4.16: A plot of (a) $\mathcal{R}(t)$ and (b) $\mathcal{P}(t)$. These are extracted from the same simulation done in the two previous figures.

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, of 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 fully 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 dependence 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.

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

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

917 Efendiev, M., Eberl, H., and Zelik, S. (2002). Existence and longtime behaviour of solutions of a
918 nonlinear reaction-diffusion system arising in the modeling of biofilms. *RIMS Kokyuroko*, 1258:49–
919 71.

920 Gurtin, M.E., M. (1977). On the diffusion of biological populations. *Mathematical Biosciences*,
921 33:35–49.

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

925 Khassehkhan, H. and Eberl, H. (2008). Modeling and simulation of a bacterial biofilm that is con-
926 trolled by ph and protonated lactic acids. *Computation and Mathematical Methods in Medicine*,
927 9(1):47–67.

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

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

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

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

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

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

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

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

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

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

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

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

Appendix A

Default Parameter Values

The default parameter values used for simulation 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
        !-----------------------------------------------------------------------
```

```
1000          implicit none
1001          integer, intent(in)                 :: n,ndiag
1002          real,dimension(n,ndiag),intent(in):: a
1003          integer, dimension(ndiag),intent(in)::ioff
1004          real,dimension(n),intent(in)        :: rhs
1005          real,dimension(n),intent(inout)     :: sol
1006          real,intent(inout)                  :: err1,err2
1007          integer,intent(inout)               :: nit
1008          integer,intent(out)                 :: stopcrit
1009          end subroutine
1010      END INTERFACE
1011
1012      !======================
1013      ! Variable Descriptions
1014      !======================
1015
1016      ! filename Variables
1017      character(100) :: filename
1018
1019      ! Problem Parameters
1020      real :: kappa,delta,nu,gama
1021      integer :: alpha,beta
1022      real :: depth,height
1023      integer :: fSelect, dSelect, gSelect, MinitialCond
1024      integer :: num_innocu_points
1025
1026      ! Numerical Method Parameters
1027      integer :: pSize,row,col,n,ndiag,nit,nOuts
1028      real :: tEnd,tDel,xDel,e1,e2,eSoln,eTrav
1029      integer :: true2D, checkTrav
1030
1031      ! Solution variables
1032      real,dimension(:),allocatable  :: C, M
1033
1034      ! Reporting variables
1035      real :: avgIters, maxIters
1036      real :: avgNit, maxNit
1037      integer :: startTime, endTime, clock_rate, clock_max
1038      real :: elapsedTime
1039
1040      write(*,*) "Enter parameter file name: ex. 'parameter.txt' "
1041      write(*,'(A)', advance="no") "    "
1042      read(*,*) filename
1043
1044      write(*,*) "Setting Parameters that shouldn't change"
1045      ndiag = 5
1046      write(*,'(A, I5)') "     ndiag = ", ndiag
1047
1048      write(*,*) "Getting problem size from file"
1049      call getProbSize(pSize, true2D, checkTrav, filename, len(filename))
1050      if (true2D == 1) then
1051        write(*,*) "    Problem is 2D"
1052        col = 4
1053      else if (true2D == 0) then
1054        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)

```

```
1110
1111      ! Report Statistics
1112      write(*,*) "-------------------------------------------------------------"
1113      write(*,*) "Statsitcs:"
1114      write(*,*) "-------------------------------------------------------------"
1115      write(*,*) "Time to compute = ", elapsedTime
1116      write(*,*) ""
1117      write(*,*) "Avg Iters for iterating betn. soln. =", avgIters
1118      write(*,*) "Max Iters for iterating betn. soln. =", maxIters
1119      write(*,*) "Avg Iters for linear solver =", avgNit
1120      write(*,*) "Max Iters for linear solver =", maxNit
1121      write(*,*) "Writing statistics to file"
1122      call reportStats(avgIters,maxIters,avgNit,maxNit,elapsedTime)
1123
1124      write(*,*) "Execution completed"
1125
1126  end program
1127
1128
1129  !=============================================================================
1130  !   Sets the problem size and checks for auxillary settings
1131  !-----------------------------------------------------------------------------
1132  !     This must be done first since the problem size is used in the variable
1133  !   declaration of the arrays.
1134  !     The auxillary settings are if the problem is 2D in nature; this means
1135  !   that the computation time can be drastically reduced by letting col = 4
1136  !   instead of col = pSize.
1137  !     Also, the option for checking for the existance of travelling wave
1138  !   solutions is done here.
1139  !=============================================================================
1140  subroutine getProbSize(pSize, true2D, checkTrav, filename, nameLen)
1141      implicit none
1142      integer,intent(out) :: pSize, true2D, checkTrav
1143      integer,intent(in)  :: nameLen
1144      character(nameLen),intent(in) :: filename
1145      character :: dum   ! Dummy variable
1146      write(*,*) filename
1147      open(UNIT = 19, FILE = filename, STATUS = "old", ACTION = "read")
1148      read(19,*); read(19,*); read(19,*)  ! Skip first 3 lines
1149      read(19,*) dum, dum, pSize
1150      read(19,*) dum, dum, true2D
1151      read(19,*) dum, dum, checkTrav
1152      close(19)
1153  end subroutine getprobSize
1154
1155
1156  !=============================================================================
1157  !   Sets the all the parameter values
1158  !-----------------------------------------------------------------------------
1159  !     Opens the parameter.txt file, which holds all the parameter values and
1160  !   function uses, and sets the variables accordingly.
1161  !     Takes in all the parameters on entry and outputs them with the appropriate
1162  !   value.
1163  !=============================================================================
1164  subroutine paramSet(depth, height, numInnocu, alpha, beta, kappa, &
```

```
1165                    gama, nu, delta, nOuts, tEnd, tDel, e1, e2, eTrav, &
1166                    eSoln, fSelect, dSelect, gSelect, MinitialCond, filename, &
1167                    nameLen)
1168     implicit none
1169     real, intent(out) :: depth, height, kappa, delta, nu
1170     real, intent(out) :: tEnd, tDel, e1, e2, eSoln, eTrav, gama
1171     integer, intent(out) :: alpha, beta,  numInnocu, nOuts
1172     integer, intent(out) :: fSelect, dSelect, gSelect, MinitialCond
1173     integer, intent(in) :: nameLen
1174     character(nameLen), intent(in) :: filename
1175
1176     character :: dum, dum2      ! Dummy variable
1177
1178     open(UNIT = 19, FILE = filename, STATUS = "old", ACTION = "read")
1179     ! Skip first 6 lines
1180     read(19,*); read(19,*); read(19,*); read(19,*); read(19,*); read(19,*)
1181
1182     read(19,*) dum, dum2, depth
1183     read(19,*) dum, dum2, height
1184     read(19,*) dum, dum2, numInnocu
1185     read(19,*) dum, dum2, alpha
1186     read(19,*) dum, dum2, beta
1187     read(19,*) dum, dum2, nu
1188     read(19,*) dum, dum2, kappa
1189     read(19,*) dum, dum2, gama
1190     read(19,*) dum, dum2, delta
1191     read(19,*) dum, dum2, nOuts
1192     read(19,*) dum, dum2, tEnd
1193     read(19,*) dum, dum2, tDel
1194     read(19,*)                          ! Skip xDel
1195     read(19,*) dum, dum2, e1
1196     read(19,*) dum, dum2, e2
1197     read(19,*) dum, dum2, eSoln
1198     read(19,*) dum, dum2, eTrav
1199     read(19,*)                          ! Skip nit
1200     read(19,*); read(19,*); read(19,*)  ! Skip 3 lines
1201     read(19,*) dum, dum2, fSelect
1202     read(19,*) dum, dum2, dSelect
1203     read(19,*) dum, dum2, gSelect
1204     read(19,*) dum, dum2, MinitialCond
1205
1206     close(19)
1207
1208 end subroutine paramSet
1209
1210
1211 !=============================================================================
1212 !   Sets the initial conditions for the system
1213 !-----------------------------------------------------------------------------
1214 !   For C, we have homogenous initial conditions, trivial to do
1215 !   For M, the innoculation point has a smooth curve to avoid sharp numerical
1216 !     artifacts in the system. This is done with a polynomial f(x) = a*x^8+b,
1217 !     this function is computed based on the depth and height parameters,
1218 !     calculating b = height and a = b/(depth)^8
1219 !=============================================================================
```

```
1220 subroutine setInitialConditions(C,M,row,col,n,depth,height,xDel,MinitialCond, &
1221                                 num_innocu_points)
1222     implicit none
1223     integer,intent(in) :: row,col,n,MinitialCond, num_innocu_points
1224     real,intent(in) :: depth, height, xDel
1225     real,dimension(n),intent(out) :: C,M
1226 
1227     integer :: i,j,x,y, xi, yi, p
1228     real :: f,a       ! function for IC curve
1229     real :: innoc     ! Placeholder for new IC value at point
1230     real :: total     ! Counts total innoculation height
1231     real :: mIC, cIC      ! Used to store previous simulation values while reading
1232     real :: xr, yr
1233 
1234     C = 1.; j = 0; M = 0
1235 
1236     ! 2D trav wave smooth curve
1237     if (MinitialCond == 1) then
1238         a = -height/(depth)**4
1239         !$omp parallel do private(f) shared(height,a)
1240         do i = 1, n
1241           x = (i-1)/col
1242           f = a*(x*xDel)**4 + height
1243           M(i) = f
1244           if (f .LE. 0) M(i) = 0
1245         enddo
1246         !$omp end parallel do
1247 
1248     ! homogenous everywhere
1249     else if (MinitialCond == 2) then
1250         M = height
1251 
1252     ! 3D initial conditions
1253     else if (MinitialCond == 3) then
1254         a = -height/(depth)**2
1255         !$omp parallel do private(f,x,y) shared(height,a)
1256         do i = 1, n
1257           x = MOD(i-1, col)
1258           y = i / row
1259           f = a*((x*xDel-0.5)*(x*xDel-0.5) + (y*xDel-0.5)*(y*xDel-0.5)) + height
1260           M(i) = f
1261           if (M(i) .LE. 0) M(i) = 0
1262         enddo
1263         !$omp end parallel do
1264 
1265     !(Random innoclation points over whole domain [3D])
1266     else if (MinitialCond == 4) then
1267       a = -height/(depth*depth)
1268       call init_random_seed()
1269       do i = 1, num_innocu_points
1270         call random_number(xr)
1271         call random_number(yr)
1272         do j = 1, n
1273           if (M(j) .LE. 0) then
1274             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
```

```
1330      else if (MinitialCond == 8) then
1331       ! do i = 1, row
1332       !   do j = 1, col
1333       !     if (i*xDel .LE. depth) then
1334       !       p = j + (i-1)*col
1335       !       M(i) = real(p)/real(n)/4.0
1336       !     endif
1337       !   enddo
1338       ! enddo
1339        do i = 1, n
1340          M(i) = real(i)/real(n)/10
1341        enddo
1342
1343      !(Evenly spaced innoculation points on y=0 side [3D])
1344      else if (MinitialCond == 9) then
1345        a = -height/(depth*depth)
1346        do i = 1, num_innocu_points
1347          xr = depth + (i-1)*2*depth + real((i-1)*(1-num_innocu_points*depth*2))/real(num_inno
1348          do j = 1, n
1349            if (M(j) .LE. 0) then
1350              x = MOD(j-1, col)
1351              y = j / row
1352              M(j) = M(j) + a*((x*xDel-xr)*(x*xDel-xr) + (y*xDel)*(y*xDel)) + height
1353              if (M(j) .LE. 0) M(j) = 0
1354            endif
1355          enddo
1356        enddo
1357
1358      ! Random innoculation points on y=0 and y = 1
1359      else if (MinitialCond == 10) then
1360        a = -height/(depth*depth)
1361        call init_random_seed()
1362        do i = 1, 2*num_innocu_points
1363          call random_number(xr)
1364          call random_number(yr)
1365          yr = yr*0.1
1366          if (i .GT. num_innocu_points) then
1367            yr = yr + 0.9 ! For y = 1 side
1368          endif
1369          do j = 1, n
1370              x = MOD(j-1, col)
1371              y = j / row
1372              innoc = a*((x*xDel-xr)*(x*xDel-xr) + (y*xDel-yr)*(y*xDel-yr)) + height
1373              if (innoc .GE. 0) M(j) = M(j) + innoc
1374          enddo
1375        enddo
1376
1377      ! Read IC from output file... (Continues a previous sim) CANNOT GET WORKING>>>>
1378  !     else if (MinitialCond == 11) then
1379  !       open(UNIT = 119, FILE = "initialCond.dat", STATUS = "old", ACTION = "read")
1380  !       read(119,*) ! Skip first line
1381  !       do i = 1, n/2+1
1382  !         read(119,*) innoc, innoc, innoc, innoc
1383  !       enddo
1384  !       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

```

```
1495      end subroutine solveC
1496
1497
1498  !=====================================================================
1499  !   Generates matrix M in diagonal format for the current timestep
1500  !=====================================================================
1501  subroutine GenMatrixM(M,C,MatrixM,Mioff,Mrhs,row,col,n,ndiag,delta,nu,&
1502                        alpha,beta,kappa,tDel,xDel,dSelect,fSelect)
1503      implicit none
1504      integer,intent(in) :: row,col,n,ndiag,alpha,beta,dSelect,fSelect
1505      real,intent(in) :: delta,kappa,xDel,tDel,nu
1506      real,dimension(n),intent(in) :: M,C
1507
1508      real,dimension(n,ndiag),intent(out) :: MatrixM
1509      real,dimension(n),intent(out) :: Mrhs
1510      integer,dimension(ndiag),intent(out) :: Mioff
1511
1512      real :: xCof
1513      real :: f
1514      real,dimension(n) :: diff
1515      !integer :: x,y,g
1516      integer :: i
1517
1518      real :: tDela
1519      tDela = tDel      ! Used for testing purposes
1520
1521      xCof = 1/(xDel*xDel)
1522
1523      Mioff = (/ -col, -1, 0, 1, col/)
1524      MatrixM(:,:) = 0
1525
1526      ! Compute all the diffusion coefficients
1527      !$omp parallel do shared(M,diff,delta,alpha,beta,dSelect)
1528      do i = 1,n
1529          call dFunc(M(i), diff(i), delta, alpha, beta, dSelect)
1530      enddo
1531      !$omp end parallel do
1532
1533      !!$omp parallel do shared(MatrixM,xCof,diff,Mrhs,tDel,M,C,kappa,nu,fSelect) private(i,
1534      !$omp parallel do private(f)
1535      do i = 1,n
1536          if (i .LE. col) then
1537              MatrixM(i,5) = MatrixM(i,5) - xCof*0.5*(diff(i+col)+diff(i))
1538              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+col)+diff(i))
1539          else
1540              MatrixM(i,1) = MatrixM(i,1) - xCof*0.5*(diff(i-col)+diff(i))
1541              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-col)+diff(i))
1542          endif
1543
1544          if (MOD(i,col) == 1) then
1545              MatrixM(i,4) = MatrixM(i,4) - xCof*0.5*(diff(i+1)+diff(i))
1546              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+1)+diff(i))
1547          else
1548              MatrixM(i,2) = MatrixM(i,2) - xCof*0.5*(diff(i-1)+diff(i))
1549              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-1)+diff(i))
```

```
1550          endif
1551
1552          if (MOD(i,col) == 0) then
1553              MatrixM(i,2) = MatrixM(i,2) - xCof*0.5*(diff(i-1)+diff(i))
1554              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-1)+diff(i))
1555          else
1556              MatrixM(i,4) = MatrixM(i,4) - xCof*0.5*(diff(i+1)+diff(i))
1557              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+1)+diff(i))
1558          endif
1559
1560          if  (i .GE. n-col) then
1561              MatrixM(i,1) = MatrixM(i,1) - xCof*0.5*(diff(i-col)+diff(i))
1562              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i-col)+diff(i))
1563          else
1564              MatrixM(i,5) = MatrixM(i,5) - xCof*0.5*(diff(i+col)+diff(i))
1565              MatrixM(i,3) = MatrixM(i,3) + xCof*0.5*(diff(i+col)+diff(i))
1566          endif
1567
1568          call fFunc(M(i),C(i),f,kappa,nu,fSelect)
1569          MatrixM(i,3) = MatrixM(i,3) - f + (1/tDel)
1570          Mrhs(i) = M(i)/tDel
1571      enddo
1572      !$omp end parallel do
1573
1574  end subroutine GenMatrixM
1575
1576
1577  !=============================================================================
1578  !   Solve Order 2
1579  !-----------------------------------------------------------------------------
1580  !   1. Solves for M_{i+1} using C_i and M_i
1581  !   2. Solves for C_{i+1} using C_i, M_i, and M_{i+1}
1582  !   ... repeat until convergence
1583  !=============================================================================
1584  subroutine solveOrder2(tEnd,nOuts,tDel,n,row,col,M,C,xDel,&
1585      gama,nu,alpha,beta,kappa,delta,ndiag,e1,e2,nit,eSoln,dSelect,fSelect,&
1586      gSelect,avgIters,maxIters,avgNit,maxNit,true2D,checkTrav,eTrav)
1587    implicit none
1588    integer,intent(in) :: nOuts,n,row,col,alpha,beta,ndiag
1589    integer,intent(in) :: dSelect,fSelect,gSelect,true2D,checkTrav
1590    real,intent(in) :: tEnd,tDel,xDel,kappa,delta,nu,gama
1591    real,intent(in) :: eSoln,eTrav
1592    real,intent(inout) :: e1,e2
1593    integer,intent(inout) :: nit
1594    real,dimension(n),intent(out) :: M,C
1595    real,intent(out) :: avgIters,maxIters,avgNit,maxNit
1596
1597    integer :: counter        ! Counts the num. of iter. in system solver
1598    integer :: endLoop        ! endLoop = 1 -> solving loop can stop
1599    integer :: filter         ! Controls frequency of outputs written
1600
1601    real :: stored_e1, stored_e2
1602
1603    real :: totalMassM        ! Total Biomass over region
1604    real :: totalMassC        ! Total Substrait over region
```

```
1605    real :: prevMassC          ! Total Substrait from X timesteps ago
1606
1607    real,dimension(n) :: Mnew
1608    real,dimension(n) :: Cnew
1609    real,dimension(n,ndiag) :: MatrixM
1610    real,dimension(n) :: Mrhs
1611    real,dimension(n) :: Cprev, Mprev
1612    integer,dimension(ndiag) :: Mioff
1613    integer :: stopcritria
1614    integer :: stat
1615    real :: diffC, diffM
1616    integer :: countIters
1617    real :: peak, height, intfac  ! Track Interface and wave peak
1618    integer :: travExist          ! 1 if trav wave exist at this timestep
1619    real :: waveSpeed             ! calculated wavespeed at current timestpe
1620    real,dimension(n) :: Mprev_10 ! M from 10-ish timesteps ago, for travCheck
1621
1622    stored_e1 = e1
1623    stored_e2 = e2
1624
1625    filter = int(tEnd/(nOuts*tDel))
1626    endLoop = 0
1627    counter = 0
1628    countIters = 0
1629    avgIters = 0
1630    avgNit = 0
1631    Mprev_10 = M  ! To initiatize for travCheck
1632    travExist = 0
1633
1634    prevMassC = 1
1635
1636    open(UNIT = 124, IOSTAT = stat, FILE = "total.dat", STATUS = "old")
1637    if (stat .EQ. 0) close(124, STATUS = "delete")
1638    open(UNIT = 120, FILE = "total.dat", POSITION = "append", ACTION = "write")
1639
1640    open(UNIT = 124, IOSTAT = stat, FILE = "COprod.dat", STATUS = "old")
1641    if (stat .EQ. 0) close(124, STATUS = "delete")
1642    open(UNIT = 126, FILE = "COprod.dat", POSITION = "append", ACTION = "write")
1643
1644    if (true2D == 1) then
1645      open(UNIT = 124, IOSTAT = stat, FILE = "peakInfo.dat", STATUS = "old")
1646      if (stat .EQ. 0) close(124, STATUS = "delete")
1647      open(UNIT = 121, FILE = "peakInfo.dat", POSITION = "append", ACTION = "write")
1648    endif
1649
1650    if (checkTrav == 1) then
1651      open(UNIT = 124, IOSTAT = stat, FILE = "travCheck.dat", STATUS = "old")
1652      if (stat .EQ. 0) close(124, STATUS = "delete")
1653      open(UNIT = 138, FILE = "travCheck.dat", POSITION = "append", ACTION = "write")
1654    endif
1655
1656    write(*,*) "   time    avgIter  maxIter      avgNit  maxNit        avgM
1657  avgC"
1658
1659    do while((counter) * tDel <= tEnd)
```

```
1660      ! Output every 100 times more then nOuts for the peak info
1661      if (MOD(counter, int(filter/100)+1) == 0) then
1662        ! Get total M and C
1663        call calcMass(M,totalMassM,n,row,col)
1664        call calcMass(C,totalMassC,n,row,col)
1665        write(120,*) counter*tDel, totalMassM, totalMassC
1666
1667        ! CO_2 production: t, current produced CO2, total produced CO2
1668        write(126,*) counter*tDel, prevMassC - totalMassC, 1 - totalMassC
1669        prevMassC = totalMassC
1670
1671        ! peak-interface, only availbale for 2D graphs
1672        if (true2D == 1) then
1673          call calcPeakInterface(M, row, col, peak, height, intfac)
1674          write (121,*) tDel*counter, peak, height, intfac
1675        end if
1676      endif
1677      if (true2D == 1 .AND. checkTrav == 1 .AND. MOD(counter, int(filter))==0) then
1678        call checkTravWave(M, Mprev_10, row, col, travExist, wavespeed, height, eTrav)
1679        wavespeed = wavespeed/filter ! Makes wavespeed independent of number of outputs
1680        write (138,*) tDel*counter, travExist, wavespeed
1681        Mprev_10 = M
1682      endif
1683
1684      ! Write to file / report Total Mass
1685      if (MOD(counter, filter) == 0) then
1686        if (true2D == 1) then
1687          call printToFile2D(n,row,col,M,C)
1688        else
1689          call printToFile(n,row,col,M,C)
1690        end if
1691
1692        if (counter == 0) then
1693          write(*,'(F8.2,F12.2,I8,F12.2,I8,F12.6,F12.6)') 0.0, 0.0, &
1694            int(maxIters), 0.0, int(maxNit), totalMassM, totalMassC
1695        else
1696          write(*,'(F8.2,F12.2,I8,F12.2,I8,F12.6,F12.6)') tDel*counter, &
1697            real(avgIters/counter), int(maxIters), real(avgNit/avgIters), &
1698            int(maxNit),totalMassM, totalMassC
1699        endif
1700      endif
1701
1702      diffC = 1
1703      diffM = 1
1704      countIters = 0
1705      Cprev = C
1706      Mprev = M
1707
1708      do while(diffC + diffM > eSoln)
1709          nit = 100*n
1710          e1 = stored_e1
1711          e2 = stored_e2
1712
1713          ! Solve M
1714          call GenMatrixM(Mprev,C,MatrixM,Mioff,Mrhs,row,col,n,ndiag,delta,nu,&
```

```
                    alpha,beta,kappa,tDel,xDel,dSelect,fSelect)
        call solveLSDIAG(n,ndiag,Mioff,MatrixM,Mnew,Mrhs,nit,e1,e2,stopcritria)

        ! Solve C
        call solveC(Mprev,Mnew,Cprev,Cnew,n,kappa,gama,tDel,gSelect)

        ! Solve CO_2 production


        if(nit > maxNit) maxNit = nit
        avgNit = avgNit + nit

        call calcDiff(diffC, C, Cnew, row, col)
        call calcDiff(diffM, M, Mnew, row, col)

        C = Cnew
        M = Mnew
        countIters = countIters+1

        if (countIters > 10000) then
          write(*,*) "[!] Over 10000 iterations in one timestep"
          write(*,*) "[!] Solutions not converging. Exit!"
          stop
        end if
!        write(*,*) countIters, diffC, diffM, C(12),M(12)
    enddo
    if(countIters > maxIters) maxIters = countIters
    avgIters = avgIters + countIters

    counter = counter + 1
  enddo
  avgNit = avgNit/(avgIters) ! avgIters right now is the total
  avgIters = avgIters/counter

!  ! Print final solution
!  if (true2D == 1) then
!    call printToFile2D(n,row,col,M,C)
!  else
!    call printToFile(n,row,col,M,C)
!  end if
!  write(*,'(F8.2,F12.2,I8,F12.2,I8,F12.6,F12.6)') tDel*counter, &
!          real(avgIters), int(maxIters), real(avgNit), &
!          int(maxNit),totalMassM, totalMassC

  close(120)
  close(138)
  close(126)
  if (true2D == 1) then
    close(121)
  endif

end subroutine solveOrder2


!==============================================================================
```

```
1770 !   Calculate the Total Mass
1771 !------------------------------------------------------------------------------
1772 !   Uses Riemann Sums kind of concept to check if the program runs correctly
1773 !   since the total mass of the region can be computed and checked.
1774 !==============================================================================
1775 subroutine calcMass(X,totalMass,n,row,col)
1776     implicit none
1777     integer,intent(in) :: n,row,col
1778     real,dimension(n),intent(in) :: X
1779 
1780     real,intent(out) :: totalMass
1781 
1782     integer :: i,j,g
1783 
1784     totalMass = 0
1785 
1786     !$omp parallel do reduction(+:totalMass)
1787     do i=1,n
1788       totalMass = totalMass + X(i)
1789     enddo
1790     !$omp end parallel do
1791 
1792     totalMass = totalMass / n
1793 
1794 end subroutine calcMass
1795 
1796 
1797 !==============================================================================
1798 !   Prints out the solution in a format accepted by gnuplot, used for graphing
1799 !------------------------------------------------------------------------------
1800 !   Runs through the grid, row-by-row. The MOD and filter act to reduce the
1801 !     number of grid points written, useful when comparing different grid
1802 !     sizes.
1803 !------------------------------------------------------------------------------
1804 !   Input:
1805 !     n      =  The problem size
1806 !     row    =  The number of rows
1807 !     col    =  The number of columns
1808 !     M      =  The solution vector for biomass
1809 !     C      =  The solution for substrait
1810 !   Output:
1811 !     none
1812 !==============================================================================
1813 subroutine printToFile(n,row,col,M,C)
1814     implicit none
1815 
1816     integer,intent(in) :: n, row, col
1817     real,dimension(n),intent(in) :: M,C
1818 
1819     integer :: p
1820     integer :: i,j
1821     integer :: stat
1822     integer :: filter
1823 
1824     !----------------------------------------
```

```
1825      ! Deletes the old output file if it exist
1826      !----------------------------------------
1827      open(UNIT = 123, IOSTAT = stat, FILE = "output.dat", STATUS = "old")
1828      if (stat .EQ. 0) close(123, STATUS = "delete")
1829
1830      open(UNIT = 11, FILE = "output.dat", POSITION = "append", ACTION = "write")
1831
1832      filter = 1
1833      if (row .gt. 257) then
1834        filter = (row-1)/256
1835      endif
1836
1837      do i=1,row
1838        if (MOD(i-1, filter) == 0) then
1839          do j=1,col
1840            if (MOD(j-1, filter) == 0) then
1841              p = (j + (i-1)*col)
1842              write(11,*) real(j-1)/real(col-1), &
1843                          real(i-1)/real(row-1), M(p), C(p)
1844            endif
1845          enddo
1846          write(11,*) ' '
1847        endif
1848      enddo
1849      write(11,*)
1850
1851
1852  end subroutine printToFile
1853
1854
1855  !=============================================================================
1856  !   Prints out the solution in a 2D format, used for graphing the Travelling
1857  !     Wave Example.
1858  !-----------------------------------------------------------------------------
1859  !   Runs through the grid, row-by-row. The MOD and filter act to reduce the
1860  !     number of grid points written
1861  !   Unique here is that the average of the x-axis is taken so that the system
1862  !     can be reduced to just y. Also written are the max and min for each y
1863  !     value; this is used for showing that the system can be reduced.
1864  !-----------------------------------------------------------------------------
1865  !   Input:
1866  !     n       =  The problem size
1867  !     row     =  The number of rows
1868  !     col     =  The number of columns
1869  !     M       =  The solution vector for biomass
1870  !     C       =  The solution for substrait
1871  !   Output:
1872  !     none
1873  !=============================================================================
1874  subroutine printToFile2D(n,row,col,M,C)
1875     implicit none
1876
1877     integer,intent(in) :: n, row, col
1878     real,dimension(n),intent(in) :: M,C
1879
```

```
1880      integer :: p
1881      integer :: i,j
1882      integer :: stat
1883      integer :: filter
1884      real :: averageM, averageC
1885      real :: maxM, minM, maxC, minC
1886      real :: y
1887
1888      !-----------------------------------------
1889      ! Deletes the old output file if it exist
1890      !-----------------------------------------
1891      open(UNIT = 124, IOSTAT = stat, FILE = "2D_output.dat", STATUS = "old")
1892      if (stat .EQ. 0) close(124, STATUS = "delete")
1893
1894      open(UNIT = 12, FILE = "2D_output.dat", POSITION = "append", ACTION = "write")
1895
1896      filter = 1
1897      if (row .gt. 257) then
1898        filter = (row-1)/256
1899      endif
1900
1901      do, i=1,row
1902        if (MOD(i-1, filter) == 0) then
1903          averageM = 0
1904          averageC = 0
1905          maxM = 0
1906          maxC = 0
1907          minM = 1
1908          minC = 1
1909          do, j=1,col
1910              p = j + (i-1)*col
1911              averageM = averageM + M(p)
1912              averageC = averageC + C(p)
1913              if(M(p) .ge. maxM) then
1914                  maxM = M(p)
1915              endif
1916              if(M(p) .le. minM) then
1917                  minM = M(p)
1918              endif
1919              if(C(p) .ge. maxC) then
1920                  maxC = C(p)
1921              endif
1922              if(C(p) .le. minC) then
1923                  minC = C(p)
1924              endif
1925          enddo
1926          averageM = averageM/(col)
1927          averageC = averageC/(col)
1928          y = real(i-1)/real(row-1)
1929          write(12,'(f20.12,f20.12,f20.12)') y,averageM,averageC
1930  !wrte(12,'(f14.10,f14.10,f14.10,f14.10,f14.10,f14.10,f14.10)') y, averageM, averageC, minM,
1931        endif
1932      enddo
1933      write(12,*)
1934
```

```
1935
1936 end subroutine printToFile2D
1937
1938
1939 !================================================================================
1940 !   Calculates the difference between each grid point for the solutions at
1941 !       different iterations
1942 !--------------------------------------------------------------------------------
1943 !   Input:
1944 !     row    =  The number of rows
1945 !     col    =  The number of columns
1946 !     C      =  The previous solution for substrait
1947 !     Cnew   =  The current solution for substrait
1948 !   Output:
1949 !     diff   =  The average difference between the two C's
1950 !================================================================================
1951 subroutine calcDiff(diff, C, Cnew, row, col)
1952     integer, intent(in) :: row, col
1953     real, dimension(row*col), intent(in) :: C, Cnew
1954     real, intent(out) :: diff
1955
1956     integer :: i
1957
1958     diff = 0
1959     !$omp parallel do reduction(+:diff)
1960     do i=1,row*col
1961         diff = diff + abs(C(i) - Cnew(i))
1962     enddo
1963     !$omp end parallel do
1964     diff = diff/real(row*col)
1965
1966 end subroutine calcDiff
1967
1968
1969 !================================================================================
1970 !   Calculates the peak and interface info at a single timestep
1971 !--------------------------------------------------------------------------------
1972 !   Input:
1973 !     row    =  The number of rows
1974 !     col    =  The number of columns
1975 !     M      =  The solution for biomass, array size (n)
1976 !   Output:
1977 !     peak   =  Peak location
1978 !     height =  Peak height
1979 !     intfac =  Interface location
1980 !================================================================================
1981 subroutine calcPeakInterface(M, row, col, peak, height, intfac)
1982     implicit none
1983     integer, intent(in):: row,col
1984     real, dimension(row*col), intent(in) :: M
1985     real, intent(out) :: peak, height, intfac
1986
1987     real :: hei
1988     real :: y
1989     integer :: i,j,p
```

```
1990
1991      hei = 0
1992      do, i=1,row
1993        y = real(i-1)/real(row-1)
1994        do, j=1,col
1995            p = (j + (i-1)*col)
1996            if (M(p) >= hei) then
1997                peak = y
1998                hei = M(p)
1999            endif
2000            if (M(p) > 0.1) then
2001                intfac = y
2002            endif
2003        enddo
2004      enddo
2005      height = hei
2006
2007  end subroutine
2008
2009
2010
2011  !=========================================================================
2012  !   Check if there is evidence of a travelling wave solution
2013  !-------------------------------------------------------------------------
2014  !   Makes an educated guess for the wavespeed (which would be incorrect
2015  !   by, at worst, 2*eTrav) and then uses this wavespeed to check if the
2016  !   difference between M and Mprev is consistently wavespeed +- eTrav
2017  !     Reports 1 or 0 for travExists based on if traveilling exists (1)
2018  !   or not (0). Also reports the approximated wavespeed.
2019  !=========================================================================
2020  subroutine checkTravWave(M, Mprev, row, col, travExist, wavespeed, height, eTrav)
2021    implicit none
2022    integer, intent(in) :: row, col
2023    real, intent(in) :: height, eTrav
2024    real, dimension(row * col), intent(in) :: M, Mprev
2025
2026    integer, intent(out) :: travExist
2027    real, intent(out) :: wavespeed
2028
2029    integer :: i, wavePoint1, wavePoint2
2030    real :: diff  ! placeholder for difference between M and Mprev
2031    wavePoint1 = -1
2032    wavePoint2 = -1
2033
2034    do, i=row,1,-1
2035      ! -3 because each column is 4 and I want to start at 1
2036      if (M(i*col-3) > 0.09 - eTrav .AND. M(i*col-3) < 0.09 + eTrav) then
2037        wavePoint1 = i
2038        exit
2039      endif
2040    enddo
2041
2042    do, i=row,1,-1
2043      if (Mprev(i*col-3) > 0.09 - eTrav .AND. Mprev(i*col-3) < 0.09 + eTrav) then
2044        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

```

```
2100   close(128)
2101
2102 end subroutine
2103
2104
2105
2106 subroutine amuxd (n,x,y,diag,idiag,ioff)
2107 !-----------------------------------------------------------------------
2108 !         Mnew times a vector in Diagonal storage format (DIA)
2109 !         f90/f95 version of the sparskit f77 subroutine
2110 !-----------------------------------------------------------------------
2111 ! multiplies a matrix by a vector when the original matrix is stored
2112 ! in the diagonal storage format.
2113 !-----------------------------------------------------------------------
2114 !
2115 ! on entry:
2116 !----------
2117 ! n      = row dimension of Mnew
2118 ! x      = real array of length equal to the column dimension of
2119 !          the Mnew matrix.
2120 ! ndiag  = integer. The first dimension of array adiag as declared in
2121 !          the calling program.
2122 !          (obscolete (=n always)
2123 ! idiag  = integer. The number of diagonals in the matrix.
2124 ! diag   = real array containing the diagonals stored of Mnew.
2125 ! idiag  = number of diagonals in matrix.
2126 ! diag   = real array of size (ndiag x idiag) containing the diagonals
2127 !
2128 ! ioff   = integer array of length idiag, containing the offsets of the
2129 !           diagonals of the matrix:
2130 !           diag(i,j) contains the element a(i,i+ioff(j)) of the matrix.
2131 !
2132 ! on return:
2133 !-----------
2134 ! y      = real array of length n, containing the product y=Mnew*x
2135 !
2136 !-----------------------------------------------------------------------
2137 implicit none
2138     integer, intent(in)::  n, idiag
2139     integer, intent(in),dimension(idiag) :: ioff
2140     real, dimension(n), intent(in) :: x
2141     real, dimension(n,idiag), intent(in) :: diag
2142     real, dimension(n), intent(out) :: y
2143     integer :: j, io, i1, i2, i
2144
2145     !$omp parallel shared(y,diag,x,n) private(j,io,i1,i2)
2146
2147     !!$omp workshare
2148     !$omp do
2149     do i=1,n
2150         y(i)=0.
2151     enddo
2152     !$omp enddo
2153     !!$omp end workshare
2154
```

```
2155      do j=1, idiag
2156          io = ioff(j)
2157          i1 = max0(1,1-io)
2158          i2 = min0(n,n-io)
2159          !$omp do
2160          do i=i1,i2
2161              y(i) = y(i)+diag(i,j)*x(i+io)
2162          enddo
2163          !$omp end do
2164      enddo
2165      !$omp end parallel
2166
2167  end subroutine amuxd
```