THE UNIVERSITY
of ADELAIDE

An Off-lattice Computational Model for the Growth of Saccharomyces Cerevisiae

by

Isaac Nakone

In fulfilment of the requirements for the degree of

Bachelor of Mathematical Sciences - Honours

May 2025

School of Computer and Mathematical Sciences

Faculty of Sciences, Engineering and Technology

The University of Adelaide

Contents

List of Tables

List of Figures

Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Isaac Nakone 12/5/2025

Acknowledgements

Abstract

Introduction

- Biology of two types of yeast!
- Write chapter 2/ 3 and 4 now

The art of mathematical biology is to capture a biological phenomenon with the simplest model possible. Another concern which appears in the connection between applied and pure mathematics, is that of ensuring the model that one selects is amenable to rigorous analysis. This further reinforces the requirement that the model be as simple as possible, but no simpler. Indeed, a model that is sufficiently crystallized so as to capture the essential details of a physical effect, will necessarily not capture everything observable, but has more chance to be useful, since it directs our thinking away from extraneous features.

What is essential in the study of biological systems? One essential aspect is the phenomenon of growth. Growth, when seen on its own, without reference to biology, is simply "change over time". The standard tool for this is calculus, which in its more developed form, is called differential geometry. A cell colony will be modelled here as a subset of $\mathbb{R}^n$ where $n = 1, 2, 3$. In order for that set to "change over time", namely, for the cell to be parametrized by a real number $t \in \mathbb{R}_{\geq 0}$, the geometry of the colony needs to be represented by a finite list of parameters. The most simple way to do this, which comprises one of the models developed here, is to represent each cell as a disk in $\mathbb{R}^2$ with radius $r = r(t) \in \mathbb{R}_{\geq 0}$ centered at the position $(x(t), y(t)) \in \mathbb{R}^2$. The advantage to this approach is that it is more computationally efficient as compared to models which represent each cell as a deformable polygon, for instance.

The question remains: how can cell division be achieved within mathematical growth models? It does not seem obvious except via the continual addition of new parameters, such as new cell radii and positions. As we will see, taking seriously the question of mitosis from a topological standpoint will yield a generalized model for systems in which geometric changes and topological changes can be captured simultaneously. Why is the topology important in the case of biological growth?

As a motivating example, consider the graph of the function $f(x) = x^2$. Now consider the time-dependent set given by $C(t) = f^{-1}(\{t\})$ where $t \in \mathbb{R}_{\geq 0}$ is a time parameter. When $t = 0$, $C(0) = \{0\}$ (only one element), but when $t = 1$, $C(1) = \{-1, +1\}$, which has two elements, and, likewise for $t = 2$ where $C(t) = \{-\sqrt{2}, +\sqrt{2}\}$. This example shows that a catastrophic change (a bifurcation) can occur in $C(t)$ during a smooth change in t. A subtle modification to this model is to consider $f_t(x) = x^2 - t$ and fix $C(t) = f_t^{-1}(\{0\})$. That is, instead of sliding up the query point t, we slide down the whole smooth manifold given by $(x, f_t(x))$ such that $x \in \mathbb{R}$. The set $C(t)$ is actually called a level-0 set. We will just call this a level set of f_t.

We can consider level sets of functions from $\mathbb{R}^n$ to $\mathbb{R}$ for $n = 2, 3$ as well. In these cases, the level sets are, respectively, level curves and level surfaces. A analogous example for $n = 2$, is the set of functions $f_t(x, y) = x^2 + y^2 - t$, the level curves of which look like circles centered at the origin of radius $\sqrt{t}$.

This level of generalization will not be sufficient for our purposes, since the type of manifolds that we will be dealing with will not in general be representable as graphs of functions. To see this, consider the fact that a vertical line in the xt-plane is a perfectly reasonable representation of a stationary cell, and yet there's no function (of x) that has a vertical line graph. What is required is that $C(t) = \{p \in M_t \mid \text{the last component of } p \text{ is } 0\}$ where M_t is a time dependent smooth manifold. The requirement that M_t is smooth for all t is imposed so that the theory of smooth manifolds can be brought to bear on the problem. What we will consider are smooth manifolds $M_t \subset \mathbb{R}^{n+1}$ where n is the dimension in which the colony exists. For instance, a two-dimensional colony would be represented by

$$C(t) = \{p \in M_t \subset \mathbb{R}^3 \mid p_3 = 0\}. \tag{1}$$

A three-dimensional colony would be represented by

$$C(t) = \{p \in M_t \subset \mathbb{R}^4 \mid p_4 = 0\}. \tag{2}$$

In the subdiscipline of abstract algebra called ring theory, it is typical to consider the polynomial ring over the real field in two variables $\mathbb{R}[x, y]$ wherein the typical element looks like

$$p(x, y) = \sum_{n,m} a_{nm} x^n y^m, \tag{3}$$

where n, m are non-negative integers, and a_{nm} is a real coefficient. The set $\mathbb{R}[x, y]$ is closed under multiplication and addition because polynomial multiplication and addition yields another real polynomial. In this thesis, a cell will be modeled as a level set

of the quartic given by

$$f(x, y) = ax^4 + bx^3 + cx^2 + dx + e+ \tag{4}$$

At the beginning, we imagine a colony of yeast cells that is restricted to move along a straight line. The colony is therefore modeled as a set of real numbers $C \subset \mathbb{R}$. Like an archipelago of small islands, the colony as a whole is composed of closed sets, C_j where $j \in \mathbb{N}$ indexes over the cells. Therefore, the colony is a disjoint union of these individual cells,

$$C = \coprod_{j=1}^{N} C_j$$

and N is the total cell count. Closed sets (in the standard topology on $\mathbb{R}$) are chosen to represent the cells for the technical reason that a point (singleton) may also constitute a cell. In fact, we further restrict each cell to a closed interval $C_j = [a_j, b_j]$.

As we shall see in general, all we require is that each cell have no holes. Another way of saying this is that each cell is homeomorphic to a singleton (which works in $\mathbb{R}^2$ and $\mathbb{R}^3$ as well). Finally, the closed interval $[a_j, b_j]$ will be called a parametrization of the cell C_j: this is important to note for the generalization to $\mathbb{R}^2$ and $\mathbb{R}^3$, where parametrizations must also be constructed.

The mechanism of mitosis must account for the fact that several cells can undergo fission at the same time, or more aptly, they undergo mitosis independently. The most general formulation, which is also simple to implement computationally is the addition of a non-intersecting singleton $\{x\}$ to the set C. Note, that this mechanism is chosen principally for the fact that it conserves the Lebesgue measure of the colony,

$$l(C \cup \{x\}) = l(C),$$

where $C \cup \{x\}$ is the colony after mitosis has occurred, since singletons have measure 0.

Of course, some restrictions must be applied to the choice of $x \in \mathbb{R} \setminus C$. Since, we are always working in Euclidean spaces (for practical scenarios), we may as well require that x is close to C in the sense that for all $c \in C$ we have that $d(c, x) < \delta$ for some small positive amount δ where $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$ is the Euclidean metric on $\mathbb{R}$ (which is easily generalized to higher dimensions). This parameter δ represents how the colony spreads out.

A profitable aspect of this model for cell colonies is that it unifies the topological aspects of the colony (it is always homeomorphic to a finite set of points or the empty set), and the measure theoretic properties (one can speak of getting more cells without

changing the total measure). This means we can separate the dynamics of cell division from the dynamics of the geometric growth of the cells, i.e. the change in a_j and b_j. One simple way to do this is to supply an ordinary differential equation for the measure of the colony $V = l(C)$, and equip this with a discrete time update equation for number of cells.

To move us closer to a manageable computer implementation of cell colony growth, we consider the simple growth model given by,

$$\frac{dV(t)}{dt} = g(t),$$
$$N_{n+1} = 2N_n,$$

where $g(t)$ is a growth function. Now we need apply some equation for how each cell grows. For sake of simplicity, say that for all cells j, the cell measure V_j grows as

$$\frac{dV_j(t)}{dt} = g_j(t), \quad t \geq m_j,$$

where $g_j(t)$ is a cellular growth function such that $V_j(t) \to V_f$ is the final cellular volume, and m_j is the birth time of the cell. This actually is not enough to specify the whole colony geometry. So how do we pin down the geometry?

The geometry and how its dynamics evolve of course depends on the chosen parametrization of each cell. For our purposes, we will consider $C_j = \bar{B}(x_j, \varepsilon_j)$ where

$$\bar{B}(x_j, \varepsilon_j) = \{x \in \mathbb{R} : d(x_j, x) \leq \varepsilon_j\},$$

which is called the closed ball centered on x_j of radius ε_j. That means $a_j = x_j - \varepsilon_j$, and $b_j = x_j + \varepsilon_j$. It is more convenient to use this parametrization since closed balls are defined in higher dimensions as well. Another important benefit to the closed ball is that whenever $\varepsilon_j = 0$, $\bar{B}(x_j, 0) = \{x_j\}$. Also, the volume of each cell is simply $V_j = 2\varepsilon_j$ which tells us that the volume is completely independent of the cell center position x_j.

Now we easily obtain a nice formula for $\varepsilon_j(t)$ defined for $t \geq m_j$ as

$$\varepsilon_j(t) = \frac{1}{2}\int_{m_j}^{t} g_j(s)ds.$$

But, recall, since the C_j are each disjoint, V is given by

$$V = l(C) = \sum_{j=1}^{N} l(C_j) = \sum_{j=1}^{N} V_j.$$

This applies to the time derivative too, yielding

$$\frac{dV(t)}{dt} = \sum_{j=1}^{N} \frac{dV_j(t)}{dt}.$$

This means that the colony and cell growth functions must be related by the following,

$$g(t) = \sum_{j=1}^{N} g_j(t).$$

Our functions $g_j(t)$ were defined for $t \in [m_j, +\infty)$ which means they have different domains. To make the analysis easier, we build these functions by taking linear combinations of basis hat functions (defined for $t \in [0, +\infty)$). The basis functions have compact support and are given piecewise by,

$$\varphi_i(t) = \begin{cases} \frac{t-t_{i-1}}{h}, & t \in [t_{i-1}, t_i] \\ \frac{t_{i+1}-t}{h}, & t \in [t_i, t_{i+1}] \\ 0, & \text{otherwise,} \end{cases}$$

where h is the smallest time step, $t_0 = 0$ and $t_i = ih$ for $i \in \mathbb{N}$. Hence, each $g_j(t)$ can be extended to a definition on all of $\mathbb{R}_{\geq 0}$ by

$$\tilde{g}_j(t) = \sum_{i=1}^{\infty} g_j(t_i)\varphi_i(t) = \sum_{i=1}^{\infty} g_{ij}\varphi_i(t).$$

Now that the g_{ij} are defined on the same domain, we can further restrict to $i \leq T$ where T is the total number of time steps. This means that the time dependence is now fixed by a finite number of parameters, g_{ij}. Summing over j, we get that

$$\tilde{g}(t) = \sum_{j=1}^{N} \sum_{i=1}^{T} g_{ij}\varphi_i(t).$$

Chapter 1

Description of the model

1.1 An invitation to algebraic representation of cell boundaries

The use of signed distance fields (SDFs) to model organic sufaces is a time honoured graphical technique used, for example, by Pixar Animation Studios to model hair in *The Incredibles* (see Petrovic et al. (2005)). The idea is to define a function which represents the closest distance from the query point to a point on the surface of the object that is to be represented. If the query point is outside, the SDF is positive, the SDF is zero on the surface and negative inside. SDFs can be rendered within traditional graphics pipelines (such as OpenGL or Vulkan) using raymarching, a method that takes place within shader programs and is therefore meshless. The formulae defining SDFs for common 2D and 3D shapes are easy to find online, see Quilez (2025). In this thesis, we do not use signed distance fields based on the practical reason that their definitions usually involve square roots which are slow to compute. We employ very similar formula without the square roots, which yield the same level sets (in 2D) but are fast to compute.

To motivate the primary mechanism by which cells will undergo mitosis in this thesis, we consider a toy example in which the level sectios of the equations for a pair of ellipses undergo a catastrophic topological change as one real parameter changes, namely the distance d between their centers. We start by considering the equations for two spheres which begin as coincident and move apart as the parameter d becomes larger. The equations that represent the individual cells are

$$f_1(x, y, z) = \left(\frac{x+d}{p}\right)^2 + \left(\frac{y}{q}\right)^2 - 1,$$

$$f_2(x,y,z) = \left(\frac{x-d}{p}\right)^2 + \left(\frac{y}{q}\right)^2 - 1,$$

where $p = 1.0L_0$ and $q = 0.9L_0$ are the semi-major radius and semi-minor radius of the ellispes, respectively, and d is the distance bewteen their centers. The aspect ratio is $q/p = 0.9$ and $L_0 = 7.5\ \mu m$ is the average length of a saccharomyces cerevisiae (Baker's yeast) cell in ideal conditions as per Chavez et al. (2024).

In order to generate the combined algebraic curve for the pair, which, by an abuse of notation, will be called the colony SDF from now on, we build the following. This is done using a what is called a "union" in computer graphics (see FUSEK). This is simply the pointwise minimum,

$$f_{\text{union}}(x,y,z) = \min(f_1(x,y,z), f_2(x,y,z)).$$

To get a smooth transition between the cells as they come apart we could alternatively use the smoothmin which is defined by a smoothness parameter s, as in

$$\text{smoothmin}(f_1(x,y,z), f_2(x,y,z); s) = -s\log\left(e^{-f_1(x,y,z)/s} + e^{-f_2(x,y,z)/s}\right).$$

Figure 1.1 shows the effect of using different values of smoothness s. Ultimately, using smoothmin was abandoned in favour or the computationally quicker `min`, which does not require the costly evaluation of the natural log and exponential functions.

As shown in figure 1.2, we have a smooth splitting of a cell as the parameter d ranges from 0.0 to 9.1 μm. Here s is the smoothing parameter. In order to ensure that only the interior of the SDF level section was plotted, I set positive entries to `nan`, which is MATLAB code for *not a number*. The surface plots in figure 1.2 were produced via the use of MATLAB's `surf` function which ignores `nan` entries in the underlying Z matrix. One additional transformation made before plotting, was to multiply by -1 to reflect the SDF about the level plane, achieving a positive value inside the biomass.

A signed distance field for an ellipse is used to model Baker's yeast cells which are in a pseudo-hyphal growth regime. An ellipse centered at the origin with semi-major dimension p (the x intercept) and semi-minor dimension q (the y intercept) has an SDF given by

$$f(x,y) = \sqrt{\left(\frac{x}{p}\right)^2 + \left(\frac{y}{q}\right)^2} - 1.$$

Recall that for computational efficiency we choose to use,

$$f(x,y) = \left(\frac{x}{p}\right)^2 + \left(\frac{y}{q}\right)^2 - 1.$$

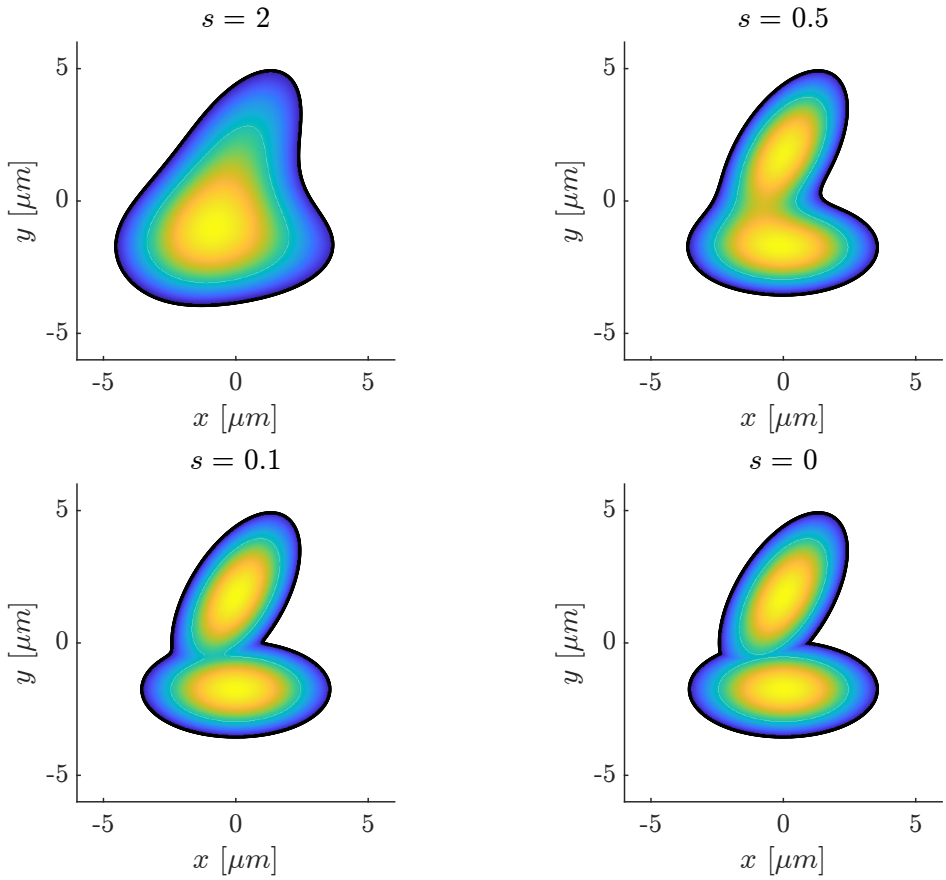

Figure 1.1: Two ellipses blended with various values of smoothness s. The bottom right figure is computed using standard minimum instead of smoothmin. Indeed, as $s \to 0$, the plot looks less smoothed across.

This is not really a *distance* field because it is dimensionless but it will still be called an SDF since it produces the elliptical shape all the same. We can also translate and rotate the ellipse, using

$$\Delta\mathbf{x}' = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \Delta\mathbf{x},$$

where $\Delta\mathbf{x} = (x - x_c)\hat{\mathbf{i}} + (y - y_c)\hat{\mathbf{j}}$ and (x_c, y_c) is the center of the ellipse. We call the components of $\Delta\mathbf{x} = \Delta x\hat{\mathbf{i}} + \Delta y\hat{\mathbf{j}}$. The above trasnformation is a passive rotation because it rotates the whole SDF. The following equation

$$f(x,y) = \left[\frac{(x - x_c)\cos\theta + (y - y_c)\sin\theta}{p}\right]^2 + \left[\frac{-(x - x_c)\sin\theta + (y - y_c)\cos\theta}{q}\right]^2 - 1.$$

is used in the custom MATLAB function `ellipse.m` devloped here. Each individual cell has a unique value of (x_c, y_c, p, q, θ) which we index by k. Bringing them all together

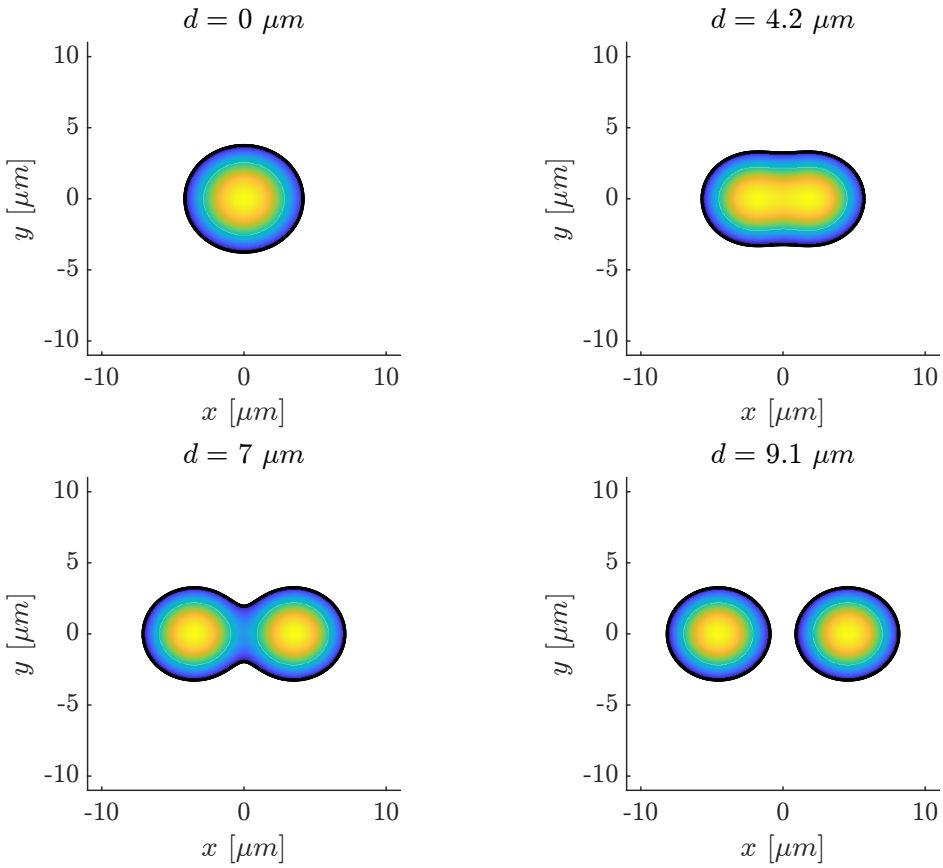

Figure 1.2: The separation d between two ellipse shapes with $s = 0.5$ and an aspect ratio of 0.9

in one function using `min`, we have the colony SDF given by,

$$g(x, y) = \min_{k \in \{1,\ldots,N_{\text{cells}}\}} f_k(x, y),$$

where the individual SDF for each cell is given by

$$f_k(x, y) = \left[\frac{(x - x_{c,k})\cos\theta_k + (y - y_{c,k})\sin\theta_k}{p_k}\right]^2 + \left[\frac{-(x - x_{c,k})\sin\theta_k + (y - y_{c,k})\cos\theta_k}{q_k}\right]^2 - 1.$$

Cell colonies can also be built up by combining the SDFs of the individual cells using a cumulative smoothmin.

Some significant optimsations were made to make sure that the evalution of the colony SDF (evaluated via calling `ellipse.m`) was as fast as possible within the capabilities of MATLAB. These optimisations, which were crucial because `ellipse.m` is called per time step, are commented on in Chapter 2.

1.2 Cell colony dynamics with an underlying discrete network

Now that we have introduced a robust mechanism to represent cell colony shape, the question naturally arises about how to represent the time dependence of this morphology. Baker's yeast grows via budding, as shown in figure 1.5. To represent the

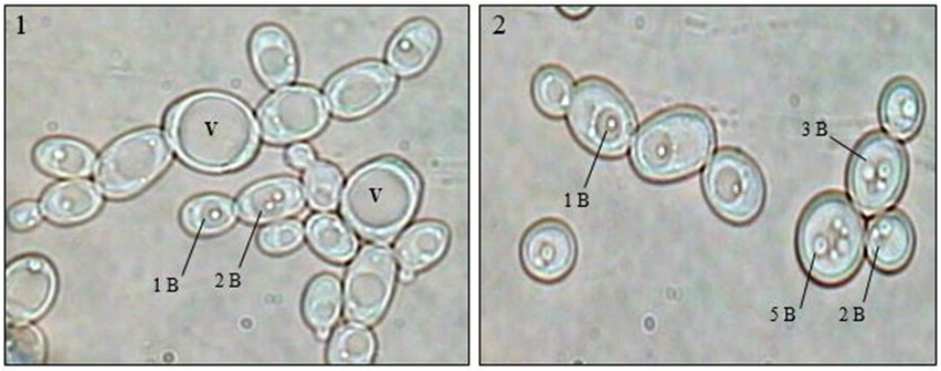

Figure 1.3: Micrographs of a translucent yeast strain studied in Ebrahimi et al. (2020).

branching dynamics of pseudo-hyphal yeast growth, an underlying discrete network which can change node count has been developed. A colony for which the node count N_{nodes} remains fixed is given by a undirected graph with adjacency matrix A_{ij} where the edges are symbolically represented by springs which model the elasticity of the cells. A diagram of this is shon in figure 1.4.

Relating the positions of the nodes in the network is done based on a simple implementation shown in figure 1.5.

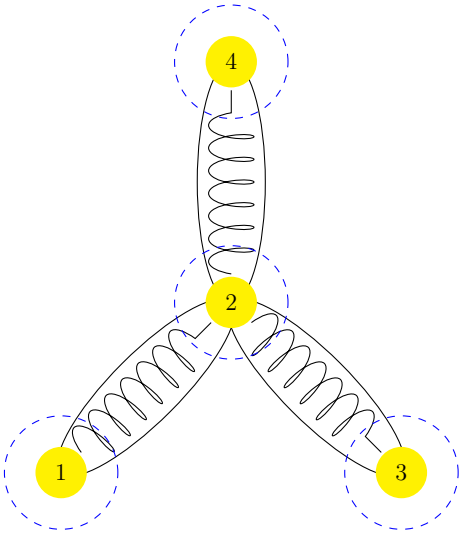

Figure 1.4: A diagram of three elliptical cells with internal springs to represent biomass elasticity ($\lambda_2 = \frac{K}{\eta\mu}$). The nodes are positioned at the ends of the major dimension of each ellipse and there are two nodes per cell. The force acting on node 2 for instance would be due to the forces from nodes 1, 3 and 4. The dashed red circles represent the activation radius ($\lambda_4 = R/L_0$) of the contact force between the nodes.

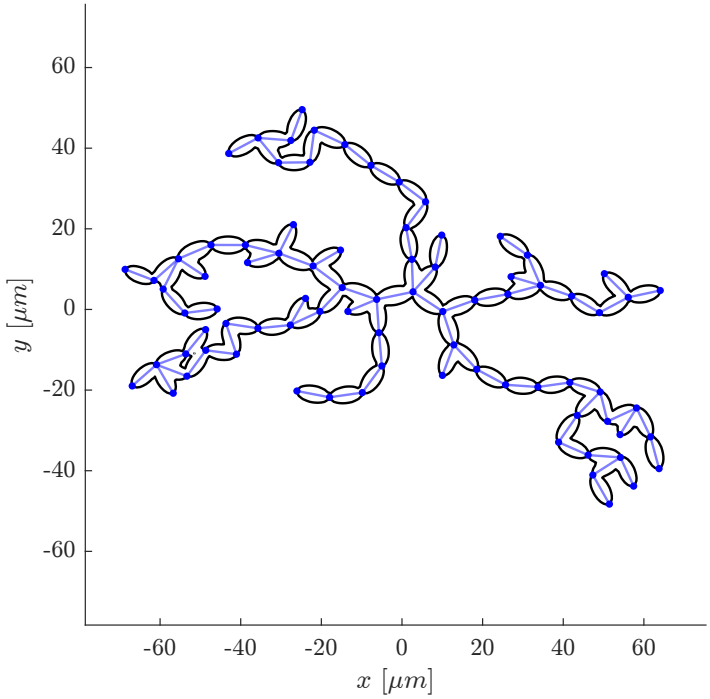

Figure 1.5: A plot showing the underlying network of a simulated yeast colony as well as the level-0 contour of the corresponding colony SDF.

The equations that represent this correspondence between a cell indexed k having parameters $(x_{c,k}, y_{c,k}, p_k, q_k, \theta_k)$ and its constitutive nodes indexed i_1 and i_2 are given by,

$$p_k(t) = \frac{1}{2}||\mathbf{x}_{i_1} - \mathbf{x}_{i_2}||, \text{ where cell } k \text{ is comprised of nodes } i_1, i_2, \tag{1.1}$$

$$q_k(t) = \lambda_7 p_k(t), \tag{1.2}$$

$$\theta_k(t) = \text{atan}(y_{i_1} - y_{i_2}, x_{i_1} - x_{i_2}), \tag{1.3}$$

$$\mathbf{x}_{c,k}(t) = \frac{1}{2}\left(\mathbf{x}_{i_1}(t) + \mathbf{x}_{i_2}(t)\right), \tag{1.4}$$

where atan is the 2 argument inverse tangent.

1.3 New cells from old

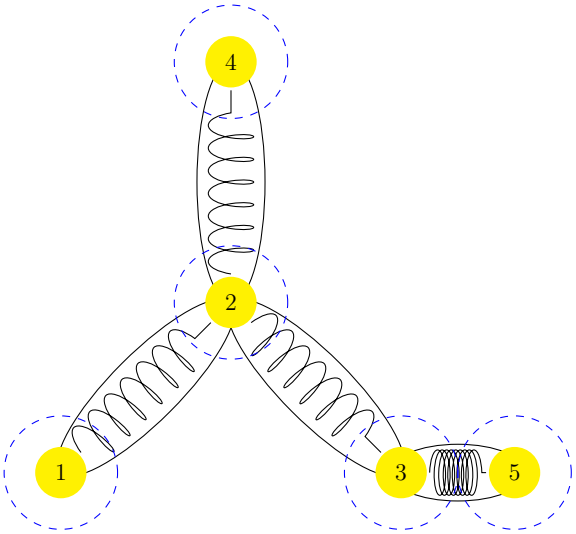

Figure 1.6: A mitosis event occurs via the addition of new nodes connected to old nodes. The nodes are added very close (distance $\delta \ll 1$) by the original nodes (exaggerated here) so that the spring force can be defined. After this point, the initially compressed cell "grows" outwards to achieve its nominal length, under the influence of elasticity, contact and chemotactic forces.

The mechanism of mitosis is a sleight of hand, in the sense that each of the nodes (two nodes per cell) are already there at the simulation outset. The appearance of growth of the colony coincides with the nodes' progressive dislodgement from their equality constraints with one another. Of course, there is no reason why the model could not be modified to add nodes that were not there to begin with, though such a

model would yield equivalent results.

To interpret the model mathematically, we recall that it was desireable to have a time parameter t over which the state of the colony could vary. The main goal of the thesis was to demonstrate that a system, in this case a yeast colony, which appears to acquire more dynamical variables (new coordinates for new cells for example) could be represented by a model with a fixed number of variables which are initially coincident.

Philosophically, however, the model has some limitations. In reality, we know the growth of fungal species such as Baker's yeast, is strongly dependent on factors such as temperature and the environmental conditions that the colony is subjected to. This means that the dynamics of cell growth (the collision and interaction as well as the morphology) in any large colony system cannot be intrinsically defined. To put it succinctly, the dynamics of the colony is strongly coupled to the local environmental conditions.

Does this mean that we can only model biological systems completely if all the conditions are incorporated? As per the discussion in the introduction about mathematical modelling, we aim to consider the simplest viable model. This, in the case of the current work, is a model that has two components: a cell colony as discussed in the preceeding section, and a nutrient field. Indeed, their coupling determines the dynamics.

In the discussion, I will suggest ways to relax these limiting assumptions that are outside the scope of the current work but are nonetheless interesting to consider.

This model, in which the colony appears as the "unfolding" of a network of nodes follows on somewhat naturally from the theory of L-systems, in which the underlying discrete structure of a system is allowed to evolve deterministically based on rules analogous to cellular automata. A criticism which has been leveled (find relevant paper that I read a while ago) at L-systems and other deterministic models of plants for example, is what I call their intrinsic nature. That is, they effectively rely on the assumption that a biological system's rules for developement are somehow contained within it's structure (for instance, DNA). It is more fashionable nowadays, and indeed more scientifically accurate, to think of a biological system as part of a complex network of other organisms and environmental conditions which determine its morphology.

Consider a minimalistic example of modelling the motion of a spherical ball rolling around on a table. Two spatial paramters x and y, together as a tuple are enough to say where it is. However, if the ball unexpectately falls from the table, then you would

suddenly require another parameter to describe the state of the system. The value of this classical example is to demonstrate for N particles and M constraints that may suddenly change, we can derive rich and unexpected dynamics.

The best we can hope for is that our parametrisation is somehow dense in the space of all possible cell colony configurations that we see experimentally. There are even some exotic cases of cellular sytems in which a single cell can have nontrivial topologies (ask Ed about the slime cells which optimise their topology based on nutrient). Therefore, future work should focus on representations of dynamic cell geometries that are as free of assumptions as is conceivable. Once an abstract enough representation of a biological system and its environment becomes available in the theory, one can use it to study and predict novel morphologies that appear in nature. Hopefully it is clear that the present work is just a suggestion of the numerous possibilities in the area of biolgical morphology viewed in the framework of "growing geometry".

1.4 Incorporating the nutrient field

A nutrient medium containing glucose is assumed to be given by a reaction-diffusion partial differential equation. That is the nutrient concentration $c(x, y, t)$ is given by

$$\frac{\partial c(x,y,t)}{\partial t} = D\left(\frac{\partial^2 c(x,y,t)}{\partial x^2} + \frac{\partial^2 c(x,y,t)}{\partial y^2}\right) - rc(x,y,t)b(x,y,t)$$

where $b(x, y, t)$ is the microscopic biomass density of the cell colony, D is a diffusion coefficient and r is a constant measuring the rate at which the cells consume nutrient. The biomass field $b(x, y, t)$ is given as a function of the colony SDF as,

$$b(x,y,t) = \begin{cases} -g(x,y,t), & \text{if } g(x,y,t) \leq 0, \\ 0, & \text{otherwise.} \end{cases}$$

In the case of the plotting of $b(x, y, t)$ we use `nan` instead of 0.

The PDE is subjected to the Dirichlet condition that the nutrient density takes a value of 1.0 at the boundary. In order to simulate the nutrient field numerically a foward time centered space (FTCS) scheme was prototyped on the square grid covering the domain. However, this scheme was found to be numerically unstable for small spatial steps h and large time steps Δt and was discarded in favour of the unconditionally numerically stable Crank-Nicholson scheme detailed in section 1.6. Before this is possible, we carry out the non-dimensionalising of the equations of motion (EOMs).

1.5 Non-dimensionalising the equations of motion

In order to non-dimensionalise the equations of motion (EOMs) we introduce dimensionless parameters $\mathbf{x}_i = X\hat{\mathbf{x}}_i$, $t = T\hat{t}$, $b = B\hat{b}$ and $c = G\hat{c}$, where X, T, B and G are general undetermined scalings for the independent and dependent variables, respectively. At the outset, we fix $X = L_0 = 7.5\ \mu m$ the nominal major cell diameter found by Chavez et al. (2024). The same source gives an aspect ratio of $7.5 \times 5.5\ \mu m$ which we call $\lambda_7 = \frac{q}{p}$. The reason for the numbering will become clear from the non-dimensionalisation of the model. The time scale is chosen as $T = \frac{1}{\mu^*}$ the reciprocal of specific growth rate for yeast, that is the $\mu^* = \mu(0)$ that appears in the formula for the total node count. The value chosen was $\mu^* = 0.46$ hour^{-1} from the value cited at Salari and Salari (2017) which represents ideal conditions for saccharomyces cerevisiae growth on a synthetic culture medium containing glucose. The value given by (Salari and Salari (2017)) is a doubling rate of 90 mins. In the model presented in this thesis, that amounted to $\mu^* = \frac{\log(2)}{1.5 \text{ hours}} = 0.4621$ hour^{-1} which was rounded down to 0.46 hour^{-1} as a conservative estimate.

Symbol	Value	Descriptive name
L_0	$7.5\ \mu m$	Average cell length
μ^*	0.46 hour^{-1}	Ideal growth rate
c_0	$1.0\ \mu m^{-2}$	Nutrient concentration

Table 1.1: A summary of the numerical units

It is necessary to point out that initial concentration, the choice for $G = c_0$, is typically measured in mol$\cdot\mu m^{-2}$ which amounts to Avagadro's number of glucose molecules per μm^2. In our case, we choose a number N_G such that the concentration comes out to $1.0\ N_G \cdot \mu m^{-2}$. We always omit the N_G because it is of no consequence dimensionally. Also useful to recapitulate, is the formula for the total node count,

$$N_{\text{nodes}}(t) = N_{\text{cells},0} e^{t\mu^* \hat{\mu}(t)}, \tag{1.5}$$

where $N_{\text{cells},0} = 1$ because we always begin with a single cell, and $\hat{\mu}(t)$ is the unitless growth rate. Note that $\mu(t) = \mu^* \hat{\mu}(t)$ and that $\hat{\mu}(0) = 1.0$. This equation is used in the computation of μ^* based on the doubling time provided in Salari and Salari (2017). Hopefully it is clear that the values of L_0, μ^*, c_0 can be changed to fit different species of fungus. Those values given here are just representative and further controlled experimental studies would need to be done to verify them.

1.5.1 Non-dimensionalising the nutrient PDE

Substituting these parameters into the reaction-diffusion PDE for nutrient concentration, we obtain

$$\frac{c_0}{(1/\mu^*)}\frac{\partial \hat{c}}{\partial \hat{t}} = \left(\frac{Dc_0}{L_0^2}\right)\left(\frac{\partial^2 \hat{c}}{\partial \hat{x}^2} + \frac{\partial^2 \hat{c}}{\partial \hat{y}^2}\right) - (rc_0 B)\,\hat{c}\hat{b},$$

which results in

$$\frac{\partial \hat{c}}{\partial \hat{t}} = \left(\frac{D}{\mu^* L_0^2}\right)\left(\frac{\partial^2 \hat{c}}{\partial \hat{x}^2} + \frac{\partial^2 \hat{c}}{\partial \hat{y}^2}\right) - \left(\frac{rB}{\mu^*}\right)\hat{c}\hat{b}.$$

Call the dimensionless diffusion constant $\lambda_1 = \frac{D}{\mu^* L_0^2}$ and set $\frac{rB}{\mu^*} = 1$ which are free to do since B is unset to begin with. This means that the scaling for the biomass density is $B = \frac{\mu^*}{r}$. The reaction diffusion equation reduces to

$$\frac{\partial \hat{c}}{\partial \hat{t}} = \lambda_1\left(\frac{\partial^2 \hat{c}}{\partial \hat{x}^2} + \frac{\partial^2 \hat{c}}{\partial \hat{y}^2}\right) - \hat{c}\hat{b}.$$

1.5.2 Non-dimensionalising the nodes ODE

Now, for the EOM for the nodes, we have

$$\begin{aligned}\frac{d\mathbf{x}_i}{dt} = \frac{K}{\eta}\sum_{j=1}^{N}\left[-A_{ij}\left(||\mathbf{x}_i - \mathbf{x}_j|| - L_0\right)\frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||}\right] \\ + \frac{F}{\eta}\sum_{j=1}^{N}\left[H(R - ||\mathbf{x}_i - \mathbf{x}_j||)\frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||}\right] \\ + \frac{\gamma}{\eta}\nabla c(\mathbf{x}_i, t).\end{aligned}$$

Substituting the dimensionless parameters, we acquire

$$\begin{aligned}\frac{L_0}{(1/\mu^*)}\frac{d\hat{\mathbf{x}}_i}{d\hat{t}} = \frac{KL_0}{\eta}\sum_{j=1,j\neq i}^{N}\left[-A_{ij}\left(||\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j|| - 1\right)\frac{\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j}{||\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j||}\right] \\ + \frac{F}{\eta}\sum_{j=1,j\neq i}^{N}\left[H(\hat{R} - ||\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j||)\frac{\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j}{||\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j||}\right] \\ + \frac{\gamma C}{\eta L_0}\hat{\nabla}\hat{c}(\hat{\mathbf{x}}_i, \hat{t}),\end{aligned}$$

which, after rearranging, becomes,

$$\begin{aligned}\frac{d\hat{\mathbf{x}}_i}{d\hat{t}} &= \frac{K}{\eta\mu^*}\sum_{j=1,j\neq i}^{N}\left[-A_{ij}\left(||\hat{\mathbf{x}}_i-\hat{\mathbf{x}}_j||-1\right)\frac{\hat{\mathbf{x}}_i-\hat{\mathbf{x}}_j}{||\hat{\mathbf{x}}_i-\hat{\mathbf{x}}_j||}\right]\\ &+\frac{F}{\eta\mu^* L_0}\sum_{j=1,j\neq i}^{N}\left[H(\hat{R}-||\hat{\mathbf{x}}_i-\hat{\mathbf{x}}_j||)\frac{\hat{\mathbf{x}}_i-\hat{\mathbf{x}}_j}{||\hat{\mathbf{x}}_i-\hat{\mathbf{x}}_j||}\right]\\ &+\frac{\gamma C}{\eta\mu^* L_0^2}\hat{\nabla}\hat{c}(\hat{\mathbf{x}}_i,\hat{t}).\end{aligned}$$

Only one of the coefficients could be set to unity, but instead we choose $C = c_0 = 1$ to be the initial concentration. That leaves us with four additional parameters,

$$\lambda_2 = \frac{K}{\eta\mu^*},$$

$$\lambda_3 = \frac{F}{\eta\mu^* L_0},$$

$$\lambda_4 = \frac{R}{L_0},$$

$$\lambda_5 = \frac{\gamma c_0}{\eta\mu^* L_0^2}.$$

1.5.3 Non-dimensionalising the biomass equation

The field $g(x,y,t) = \hat{g}(\hat{x},\hat{y},\hat{t})$ is unitless so we are left with the following EOM for the biomass field,

$$B\hat{b} = \begin{cases}-\hat{g}(\hat{x},\hat{y},\hat{t}), & \text{if } \hat{g}(\hat{x},\hat{y},\hat{t}) \leq 0,\\ 0, & \text{otherwise.}\end{cases}$$

$$\hat{b} = \begin{cases}-\frac{r}{\mu^*}\hat{g}(\hat{x},\hat{y},\hat{t}), & \text{if } \hat{g}(\hat{x},\hat{y},\hat{t}) \leq 0,\\ 0, & \text{otherwise.}\end{cases}$$

which leaves us with an additional parameter, $\lambda_6 = \frac{r}{\mu^*}$. The final model parameter is the cell aspect ratio, λ_7. The model parameters are summarised in table 1.2.

1.5.4 Non-dimensional model equations of motion (EOMs)

Dropping the hats on variable symbols, we arive at the final system equations of motion which span from equation 1.6 to 1.19.

Symbol	Expression	Descriptive name
λ_1	$\frac{D}{\mu^* L_0^2}$	diffusivity
λ_2	$\frac{K}{\eta\mu^*}$	elasticity
λ_3	$\frac{F}{\eta\mu^* L_0}$	respulsivity
λ_4	$\frac{R}{L_0}$	respulsion radius
λ_5	$\frac{\gamma c_0}{\eta\mu^* L_0^2}$	mobility
λ_6	$\frac{r}{\mu^*}$	metabolic rate
λ_7	$\frac{\text{cell minor axis}}{\text{cell major axis}} = \frac{2q}{2p}$	cell aspect ratio

Table 1.2: A summary of the dimensionless model parameters

$$\boxed{\frac{\partial c}{\partial t} = \lambda_1 \left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} \right) - bc, \text{ for } (x, y) \in \Omega \setminus \partial\Omega,} \tag{1.6}$$

$$\boxed{c(x, y, t) = 1.0, \text{ for } (x, y) \in \partial\Omega,} \tag{1.7}$$

$$\boxed{c(x, y, 0) = 1.0,} \tag{1.8}$$

$$\boxed{\begin{aligned} \frac{d\mathbf{x}_i}{dt} = \lambda_2 \sum_{j=1}^{N_{\text{nodes}}} \left[A_{ij} \left(1 - ||\mathbf{x}_i - \mathbf{x}_j||\right) \frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||} \right] \\ + \lambda_3 \sum_{j=1}^{N_{\text{nodes}}} \left[H(\lambda_4 - ||\mathbf{x}_i - \mathbf{x}_j||) \frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||} \right] \\ + \lambda_5 \nabla c(\mathbf{x}_i, t), \text{ for } i \in \{1, ..., N_{\text{nodes}}\}, \end{aligned}} \tag{1.9}$$

$$\boxed{\mathbf{x}_1(0) = (0.5\cos\Theta_e, 0.5\sin\Theta_e), \ \mathbf{x}_2(0) = (-0.5\cos\Theta_e, -0.5\sin\Theta_e),} \tag{1.10}$$

$$\boxed{N_{\text{nodes}} = N_{0,\text{cells}} e^{t\mu(t)}, \text{where } N_{0,\text{cells}} = 1,} \tag{1.11}$$

$$\boxed{\mu(t) = \frac{1}{N_{\text{nodes}}} \sum_{i=1}^{N_{\text{nodes}}} c(\mathbf{x}_i, t),} \tag{1.12}$$

$$\boxed{b(x, y, t) = \begin{cases} -\lambda_6 g(x, y, t), & \text{if } g(x, y, t) \leq 0, \\ 0, & \text{otherwise,} \end{cases}} \tag{1.13}$$

$$\boxed{g(x, y, t) = \min_{k \in \{1, ..., N_{\text{cells}}\}} f_k(x, y, t),} \tag{1.14}$$

$$f_k(x,y,t) = \left[\frac{(x - x_{c,k}(t))\cos\theta_k + (y - y_{c,k}(t))\sin\theta_k}{p_k(t)}\right]^2 + \left[\frac{-(x - x_{c,k}(t))\sin\theta_k + (y - y_{c,k}(t))\cos\theta_k}{q_k(t)}\right]^2 - 1, \tag{1.15}$$

$$p_k(t) = \frac{1}{2}||\mathbf{x}_{i_1} - \mathbf{x}_{i_2}||, \text{ where cell } k \text{ is comprised of nodes } i_1, i_2, \tag{1.16}$$

$$q_k(t) = \lambda_7 p_k(t), \tag{1.17}$$

$$\theta_k(t) = \text{atan}(y_{i_1} - y_{i_2}, x_{i_1} - x_{i_2}), \tag{1.18}$$

$$\mathbf{x}_{c,k}(t) = \frac{1}{2}\left(\mathbf{x}_{i_1}(t) + \mathbf{x}_{i_2}(t)\right). \tag{1.19}$$

Note that i and j index over node numbers, k over cells, and i_1 and i_2 denote the nodes that comprise cell k. Also note that Θ_e, the orientation of the starting cell is chosen to be different for each ensemble instance e. In fact, this is chosen to be a uniform random number between $[0, 2\pi]$ which is unpdated in the loop over the ensemble in `Master.m` as part of the **CellColonySimulator** MATLAB software. Finally the petri-dish domain is given by

$$\Omega = \left[-\frac{1}{2}L_{\text{petri-dish}}, \frac{1}{2}L_{\text{petri-dish}}\right] \times \left[-\frac{1}{2}L_{\text{petri-dish}}, \frac{1}{2}L_{\text{petri-dish}}\right] \subset \mathbb{R}^2, \tag{1.20}$$

where it is noted that $L_{\text{petri-dish}}$ is unitless.

1.6 Numerically solving for the nutrient field

In order to advance the nutrient field in time, an efficient and stable numerical solver was required. Because of the fact that the nutrient field $c(x, y, t)$ is coupled to the biomass field which is dependent on a discrete network that changes in connectivity and node count, it was necessary to implement the method in-house, as opposed to using a pre-existing MATLAB boundary value solver for instance. The Crank-Nicolson method (Crank and Nicolson (1947)) was chosen for the practical reason that it led to faster simulations overall since a larger time step could be used as opposed to the explicit forward in time central in space (FTCS) scheme. This is a direct consequence of the fact that the (implicit) Crank-Nicholson scheme is unconditionally numerically stable. Let us begin by verifying this fact using the what is known as Von Neumann stability analysis (Charney et al. (1950)).

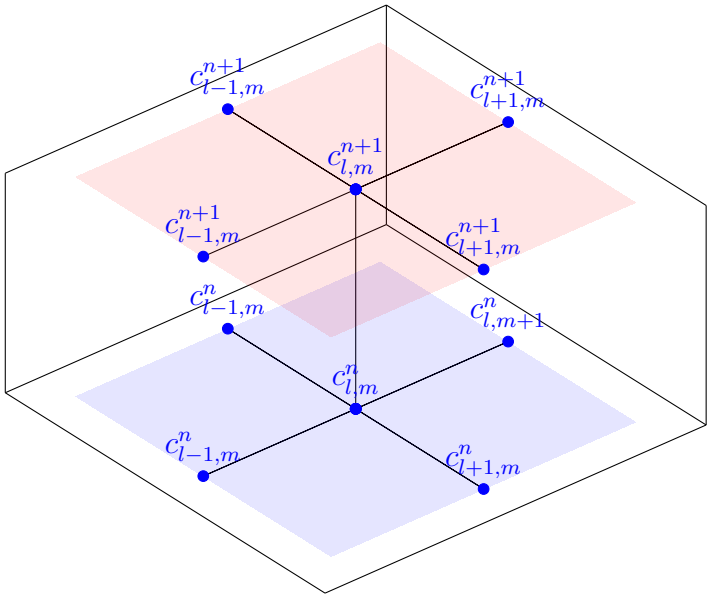

Figure 1.7: The five point stencil for the nutrient field at times $t_n = n\Delta t$ (blue plane) and $t_{n+1} = (n+1)\Delta t$ (red plane) used for the Crank-Nicolson scheme on the domain interior.

Firstly, we define the nutrient field as $c^n_{l,m} = c(x_l, y_m, t_n)$, where $x_l = -\frac{1}{2}L_{\text{petri-dish}} + hl$, $y_m = -\frac{1}{2}L_{\text{petri-dish}} + hm$ and similarly for $b^n_{l,m}$, the biomass. The values $c^n_{l,m}$ precisely satisfy the discretised equations which follow. Let $\bar{c}^n_{l,m}$ be the solution achieved with finite precison arithmetic operations which does not necessarily satisfy the equations, up to some error term,

$$\epsilon^n_{l,m} = \bar{c}^n_{l,m} - c^n_{l,m}.$$

Carrying out the Crank-Nicolson discretisation with the values $c^n_{l,m}$, we derive

$$\frac{c^{n+1}_{l,m} - c^n_{l,m}}{\Delta t} = \frac{\lambda_1}{2}\left(\nabla^2 c^{n+1}_{l,m} + \nabla^2 c^n_{l,m}\right) - \frac{1}{2}b^n_{l,m}(c^{n+1}_{l,m} + c^n_{l,m}).$$

The laplacian terms here are calculated with a five-point stencil at the current and subsequent time steps as shown in figure 1.7. This results in the following discretisation,

$$\begin{aligned}\frac{c^{n+1}_{l,m} - c^n_{l,m}}{\Delta t} = \frac{\lambda_1}{2}&\left(\frac{c^{n+1}_{l+1,m} - 2c^{n+1}_{l,m} + c^{n+1}_{l-1,m}}{h^2} + \frac{c^{n+1}_{l,m+1} - 2c^{n+1}_{l,m} + c^{n+1}_{l,m-1}}{h^2}\right)\\ &+ \frac{\lambda_1}{2}\left(\frac{c^{n}_{l+1,m} - 2c^{n}_{l,m} + c^{n}_{l-1,m}}{h^2} + \frac{c^{n}_{l,m+1} - 2c^{n}_{l,m} + c^{n}_{l,m-1}}{h^2}\right)\\ &- \frac{1}{2}b^n_{l,m}(c^{n+1}_{l,m} + c^n_{l,m}).\end{aligned}$$

To evaluate the numerical stability, we introduce a constant $\alpha = \frac{\lambda_1 \Delta t}{h^2}$. With this constant in place, we continue with

$$\begin{aligned} c_{l,m}^{n+1} - c_{l,m}^{n} &= \frac{\alpha}{2}\left(c_{l+1,m}^{n+1} + c_{l-1,m}^{n+1} + c_{l,m+1}^{n+1} + c_{l,m-1}^{n+1}\right) - 2\alpha c_{j,k}^{n+1} \\ &+ \frac{\alpha}{2}\left(c_{l+1,m}^{n} + c_{l-1,m}^{n} + c_{l,m+1}^{n} + c_{l,m-1}^{n}\right) - 2\alpha c_{l,m}^{n} \\ &- \frac{\Delta t}{2} b_{l,m}^{n}(c_{l,m}^{n+1} + c_{l,m}^{n}). \end{aligned}$$

Since the discretised equations are linear in $c_{l,m}^{n}$ (taking $b_{l,m}^{n}$ to be slowly varying) and since $\bar{c}_{l,m}^{n}$ also form a solution, the $\epsilon_{l,m}^{n}$ are also a solution.

$$\begin{aligned} \epsilon_{l,m}^{n+1} - \epsilon_{l,m}^{n} &= \frac{\alpha}{2}\left(\epsilon_{l+1,m}^{n+1} + \epsilon_{l-1,m}^{n+1} + \epsilon_{l,m+1}^{n+1} + \epsilon_{l,m-1}^{n+1}\right) - 2\alpha \epsilon_{l,m}^{n+1} \\ &+ \frac{\alpha}{2}\left(\epsilon_{l+1,m}^{n} + \epsilon_{l-1,m}^{n} + \epsilon_{l,m+1}^{n} + \epsilon_{l,m-1}^{n}\right) - 2\alpha \epsilon_{l,m}^{n} \\ &- \frac{\Delta t}{2} b_{l,m}^{n}(\epsilon_{l,m}^{n+1} + \epsilon_{l,m}^{n}). \end{aligned}$$

We subsitute in an ansatz for the error term which is standard in Von Neumann stability analysis, namely

$$\epsilon_{l,m}^{n} = \xi^{n} e^{i\nu_x lh} e^{i\nu_y mh},$$

where ν_x and ν_y are wavenumbers and h is the spatial step. Note that $i = \sqrt{-1}$. In order for the numerical method to be stable, we require that $|\xi| \leq 1$ for all values of the wavenumbers ν_x, ν_y. Substituting this expression into the discretised equation for the error, we get,

$$\begin{aligned} \xi^{n+1} e^{i\nu_x lh} e^{i\nu_y mh} - \xi^{n} e^{i\nu_x lh} e^{i\nu_y mh} &= \frac{\alpha}{2}\left(\xi^{n+1} e^{i\nu_x (l+1)h} e^{i\nu_y mh} + \xi^{n+1} e^{i\nu_x (l-1)h} e^{i\nu_y mh}\right) \\ &+ \frac{\alpha}{2}\left(\xi^{n+1} e^{i\nu_x lh} e^{i\nu_y (m+1)h} + \xi^{n+1} e^{i\nu_x lh} e^{i\nu_y (m-1)h}\right) \\ &- 2\alpha \xi^{n+1} e^{i\nu_x lh} e^{i\nu_y mh} \\ &+ \frac{\alpha}{2}\left(\xi^{n} e^{i\nu_x (l+1)h} e^{i\nu_y kh} + \xi^{n} e^{i\nu_x (m-1)h} e^{i\nu_y mh}\right) \\ &+ \frac{\alpha}{2}\left(\xi^{n} e^{i\nu_x lh} e^{i\nu_y (k+1)h} + \xi^{n} e^{i\nu_x mh} e^{i\nu_y (m-1)h}\right) \\ &- 2\alpha \xi^{n} e^{i\nu_x lh} e^{i\nu_y kh} \\ &- \frac{\Delta t}{2} b_{l,m}^{n}(\xi^{n+1} e^{i\nu_x lh} e^{i\nu_y mh} + \xi^{n} e^{i\nu_x lh} e^{i\nu_y mh}). \end{aligned}$$

Dividing through by $\xi^{n} e^{i\nu_x lh} e^{i\nu_y mh}$, we attain,

$$\begin{aligned} \xi - 1 &= \frac{\alpha}{2}\left(\xi e^{i\nu_x h} + \xi e^{-i\nu_x h}\right) + \frac{\alpha}{2}\left(\xi e^{i\nu_y h} + \xi e^{-i\nu_y h}\right) - 2\xi\alpha \\ &+ \frac{\alpha}{2}\left(e^{i\nu_x h} + e^{-i\nu_x h}\right) + \frac{\alpha}{2}\left(e^{i\nu_y h} + e^{-i\nu_y h}\right) - 2\alpha - \frac{\Delta t}{2} b_{l,m}^{n}(\xi + 1). \end{aligned}$$

Using the fact that $\frac{e^{i\nu_x h}+e^{-i\nu_x h}}{2} = \cos(\nu_x h)$ and similarly for the other terms, we have,

$$\begin{aligned}\xi - 1 = \alpha\xi\cos(\nu_x h) + \alpha\xi\cos(\nu_y h) - 2\alpha\xi \\ + \alpha\cos(\nu_x h) + \alpha\cos(\nu_y h) - 2\alpha - \frac{\Delta t}{2} b^n_{l,m}(\xi + 1).\end{aligned}$$

Finally, solving for ξ, and taking its absolute value we have,

$$|\xi| = \left| \frac{1 + \alpha\left[\cos(\nu_x h) + \cos(\nu_y h)\right] - 2\alpha - \frac{1}{2}\Delta t b^n_{l,m}}{1 - \alpha\left[\cos(\nu_x h) + \cos(\nu_y h)\right] + 2\alpha + \frac{1}{2}\Delta t b^n_{l,m}} \right|.$$

In order to complete the analysis, consider when biomass is $b^n_{l,m} = 0$. We have the simplified equation,

$$|\xi| = \left| \frac{1 - \alpha(2 - [\cos(\nu_x h) + \cos(\nu_y h)])}{1 + \alpha(2 - [\cos(\nu_x h) + \cos(\nu_y h)])} \right|,$$

Which has a numerator less than or equal to 1 because $\alpha > 0$ and $0 \leq 2 - [\cos(\nu_x h) + \cos(\nu_y h)] \leq 4$. The denominator is always greater than or equal to 1, which is enough to prove that the Crank-Nicolson scheme for the standard heat equation is numerically stable.

Whether this generalises to non-zero $b^n_{l,m}$ is determined by numerical experiments, and indeed for $\alpha = \frac{\lambda_1 \Delta t}{h^2} = \frac{(0.1)(0.01)}{(0.1)^2} = 0.1$, the scheme was not found to be unstable. However, for higher values of α it was possible to see spurious oscillations in the solution field for c which obviously suggests some form of instability.

Chapter 2

Software Implementation: *CellColonySimulator*

2.1 The structure of the program

The program **CellColonySimulator** is implemented *in-house* with MATLAB as shown in figure 2.1. **CellColonySimulator** is broken up into six seperate scripts as per the programming principle of modularity. These include: `Master.m`, `runSimulation.m`, `ellipse.m`, `updateFood.m`, `addCell.m`, `updateNodes.m`.

2.1.1 Calling the update loop in *runSimulation.m*

The function `runSimulation.m` takes in the model parameter vector, λ, the struct of numerical discretisation parameters, `numericPack`, the times at which statistics are sampled, `sampleTimes`, the colony data including node positions (x, y) and the network adjacency matrix, `connectivity`, and the initial nutrient field given by `food`.

The mesh grid matrices X and Y are taken from `numericPack` and converted to graphics processing unit (GPU) arrays using MATLAB's `gpuArray()`. This is crucial for speed as GPU operations on large matrices are highly optimised and parallelised within the GPU architecture. Besides generating descriptive file names for the output plots and other miscellaneous tasks, `runSimulation.m` then progresses into the primary `for` loop over the time indices, `timeIndex`.

A task to print to the console to let the user know what ensemble instance and `timeIndex` the program is up to, is done first. Secondly, based on the current node positions (x_i, y_i) for $i \in \{1, ..., N_{\text{nodes}}\}$, and the number of edges present in the `connectivity` matrix, the current number of cells N_{cells} is counted up and vectors are associated to

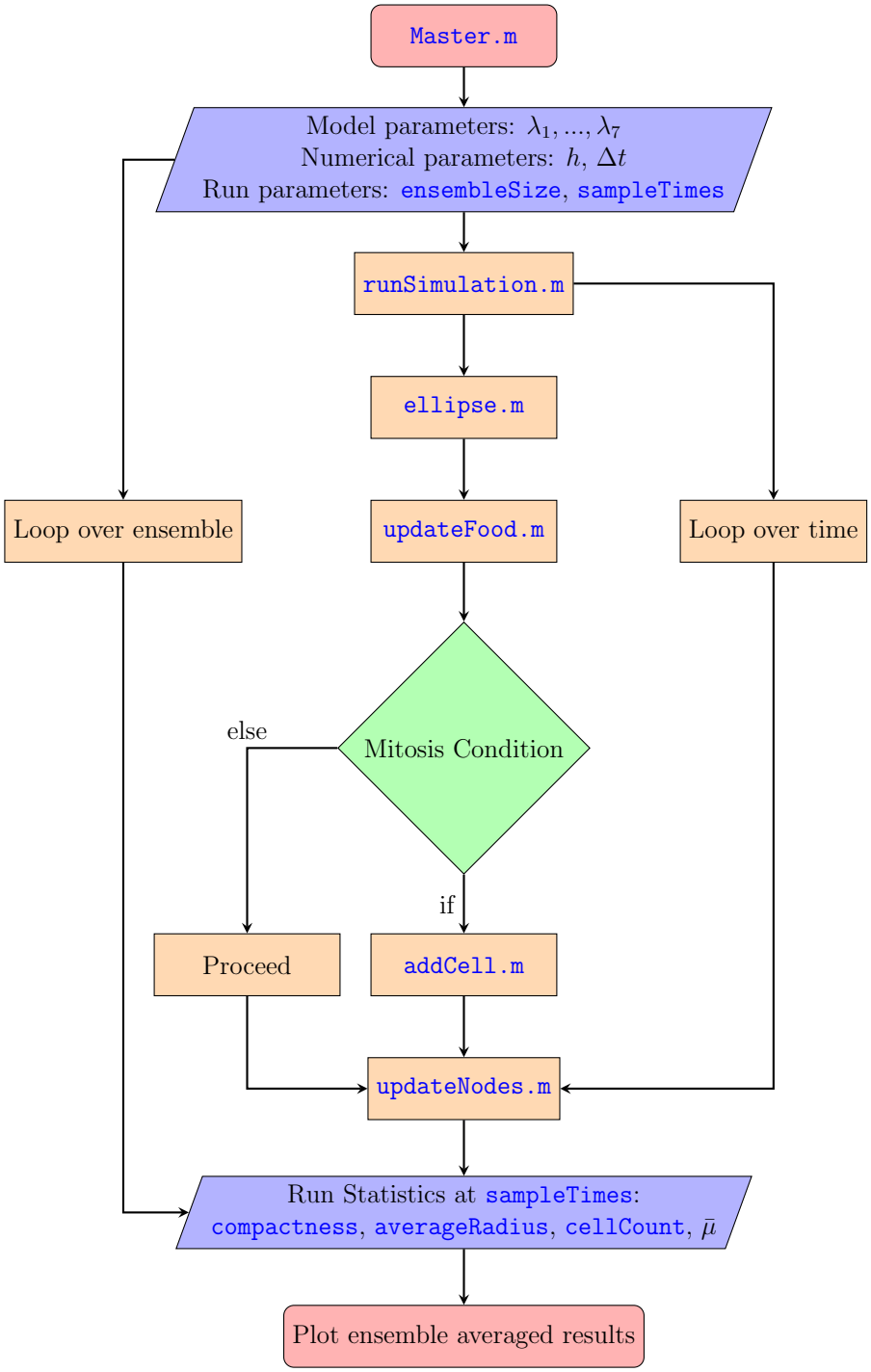

Figure 2.1: A flow chart of **CellColonySimulator** software package in MATLAB.

the variables $x_{c,k}$,$y_{c,k}$, θ_k, p_k and q_k (based on equations 1.16, 1.17, 1.18 and 1.19) where $k \in \{1, ..., N_{\text{cells}}\}$. These vectors are passed to the function `ellipse()` (explained in subsection 2.1.2) which computes the colony-level signed distance field (SDF), which is comprised of the elliptical SDF of each cell indexed k.

At this stage the biomass field, `biomass`, is computed from the colony level SDF, based on equation 1.13. The current value of the nutrient field and the biomass field, are passed to `updateFood()` (explained in subsection 2.1.3) which updates the nutrient field based on the Crank-Nicholson method applied to equation 1.6. The food at the next time step is the output from `updateFood()`.

The value of the growth rate across the colony μ_i is then determined using 2D interpolation by calling `interp2(X, Y, food, x, y)`, where X, Y are the underlying grid matrices, `food` is the updated nutrient field, and (x, y) are the node positions, taken as query points in the interpolation. Recall from equation 1.12, the growth rate is given by the the average of the nutrient field sampled over the nodes.

The updated number of nodes, $N'_{\text{nodes}} = N_{\text{nodes}}(t_{n+\Delta n})$, is found by multiplying the current number of nodes, N_{nodes}, by e^{μ} (as stipulated by the exponential growth in equation 1.11) where μ is the average growth rate over the colony as discussed above. The number of nodes added is then given by

$$\Delta N_{\text{nodes}} = N'_{\text{nodes}} - N_{\text{nodes}} = \lceil e^{\mu} - 1 \rceil N_{\text{nodes}},$$

where μ is given by

$$\mu = \frac{1}{N_{\text{nodes}}} \sum_{i=1}^{N_{\text{nodes}}} \mu_i,$$

where $\mu_i = c(\mathbf{x}_i, t)$ is the value of the nutrient field $c(\mathbf{x}_i, t)$ sampled at the i-th node position (this is just a recapitulation of equation 1.12). The number of time steps per mitosis event, Δn is given by `ceil(1.0/dt)`, and we use an `if` statement to ensure `addCell()` is only called when n is a multiple of Δn.

In `addCell()`, the number of nodes to add, ΔN_{nodes}, is passed as a argument. The mechanism to choose the parent cells to undergo mitosis is explained in subsection 2.1.4. Finally, the new node positions (based on equation 1.9) are determined using `updateNodes()` as detailed in subsection 2.1.5.

The rest of `runSimulation.m` is devoted to plotting the biomass and nutrient fields, aswell as generating the underlying node network using MATLAB's `graph()` where the colony adjacency matrix, `connectivity`, is passed as an argument.

2.1.2 Vectorised computation of colony SDF in *ellipse.m*

The custom function `ellipse.m` has been iterated with high efficiency in mind since `ellipse.m` has to be called every time step. Using MATLAB's widget *run-and-time*, the bottleneck found in `ellipse.m` was eliminated. This significantly improved run times. How was this speedup achieved?

Where previously `ellipse.m` required a `for` loop over the SDFs for each cell, the final iteration of `ellipse.m` uses completely vectorised operations which run fast on the GPU. I have provided the code below.

```
function sdf =                  ... %Colony level SDF
    ellipse(X, Y,               ... %Grid values
            theta,              ... %Orientation angles
            centerX, centerY,   ... %Center coordinates
            semiMajorRadius,    ... %p_k
            semiMinorRadius     ... %q_k
           )

    %Translate the field to the center coordinates:
    Xt = repmat(X, [1,1,size(centerX,2)])-reshape(centerX,
       [1,1,size(centerX,2)]);
    Yt = repmat(Y, [1,1,size(centerX,2)])-reshape(centerY,
       [1,1,size(centerX,2)]);


    %Precompute trigonometric functions for efficiency
    COS = cos(theta);
    SIN = sin(theta);
    cosReshape = reshape(COS, [1,1,size(centerY,2)]);
    sinReshape = reshape(SIN, [1,1,size(centerY,2)]);

    %Rotate translated field by the rotation matrix (
       correctly vectorised):
    rotX =  Xt .* cosReshape + Yt .* sinReshape;
    rotY = -Xt .* sinReshape + Yt .* cosReshape;

    %Precompute reciprocal squared of geometry arrays:
    semiMajorRadiusReciprocalReshape2 = ...
        reshape(1.0 ./ semiMajorRadius.^2, [1,1,size(
           centerX,2)]);
    semiMinorRadiusReciprocalReshape2 = ...
```

```
28          reshape(1.0 ./ semiMinorRadius.^2, [1,1,size(
                centerX,2)]);
29
30      %Instead of using smoothmin (slow) just use min of the
31      %square of the SDF. [This does not require calling sqrt
            ()
32      %which is computationally very costly]
33      sdf = min(rotX .^2 .* semiMajorRadiusReciprocalReshape2
            + ...
34                rotY .^2 .* semiMinorRadiusReciprocalReshape2
                      - ...
35                1.0, [], 3);
36
37      %Normalise to with magnitude 1 in colony region:
38      absMin = abs(min(sdf,[], "all"));
39      sdf = sdf ./ absMin ;
40
41  end
```

Listing 2.1: A code listing for the **ellipse.m** script

One thing to note from the MATLAB code listing is that vectorised division, say called with `./`, is precomputed as reciprocals which are then multiplied through in the compuatation of `sdf` using `.*` which is faster. Precomputing trigonometric functions was also opted for with the aim of reducing repeated compuatations with the same angle. Finally, we recall the discussion in section 1.1, in which it was noted that `min` is significantly faster than evaluating smoothmin. The final optimisation made was to remove the call to `sqrt()` which actually has no effect on the shape of each cell but vastly improves compute time.

2.1.3 A fast Crank-Nicolson scheme in *updateFood.m*

The Crank-Nicolson scheme is an implicit numerical scheme that requires the computation of a (mostly zero) matrix of coefficients `LHM` (which has dimensions $N_{\text{grid}}^2 \times N_{\text{grid}}^2$) a right hand side column vector `RHS` ($N_{\text{grid}}^2 \times 1$), and the solution of the linear system `LHM * x = RHS`.

The most obvious optimisation made in `updateFood()` is to completely vectorise all operations. As evinced in the code listing supplied below, there are no `for` loops.

```
1  function foodNew = updateFood(biomass, food, lambda, dt, h, X)
2
```

```
3     N = size(X,1);
4     V = N .* N;
5     Vi = (N-2) .* (N-2);
6
7     i = 2:(N-1);
8     j = i;
9
10
11    %Generate the five point stencil of indices in row major order.
12    ij   = rmo(i,j,N);
13    im1j = rmo(i-1,j,N);
14    ip1j = rmo(i+1,j,N);
15    ijm1 = rmo(i,j-1,N);
16    ijp1 = rmo(i,j+1,N);
17
18    %Flatten these:
19    fij   = ij(:);
20    fim1j = im1j(:);
21    fip1j = ip1j(:);
22    fijm1 = ijm1(:);
23    fijp1 = ijp1(:);
24
25
26    CFL = dt .* lambda(1) ./ (h .* h);
27    coeff1 = 0.5 .* dt .* biomass(i,j) + 2.0 .* CFL + 1.0;
28    coeff2 = 0.5 .* CFL;
29    coeff3 = 1.0 - 2.0 .* CFL - 0.5 .* dt .* biomass(i,j);
30
31    %Flatten the coefficients:
32    coeff1 = coeff1(:);
33    coeff3 = coeff3(:);
34
35
36
37    c = food(:);
38
39    RHS = coeff3 .* c(fij) + ...
40          coeff2 .* (c(fim1j) + c(fip1j) + c(fijm1) + c(fijp1));
41
42
43
44
45    LHM = spdiags([-coeff2 .* ones(Vi,1),-coeff2 .* ones(Vi,1) ,...
```

```
46                      gather(coeff1),...
47                      -coeff2 .* ones(Vi,1), -coeff2 .* ones(Vi,1)],...
48                     [-N, -1, 0, 1, N], Vi, Vi);
49     yM  = spdiags(-coeff2 .* ones(1,N), 0, N, N);
50
51
52     LHM(1:N,(end - N + 1):end) = LHM(1:N,(end - N + 1):end) + yM;
53     LHM((end - N + 1):end,1:N) = LHM((end - N + 1):end,1:N) + yM;
54
55
56     [cNew, flag] = pcg(LHM,RHS,1e-6);
57
58     foodNew = food;
59
60     foodNew(i,j) = reshape(cNew, [N-2, N-2]);
61
62 end
63
64
65 function index = rmo(i,j,N)
66     index = i' + N .* (j-1);
67 end
```

Listing 2.2: A code listing for the **updateFood.m** script

Another optimisation which was not only efficient but also necessary to avoid memory issues, was to construct `LHM` with sparse matrices, which meant that on the order of N_{grid}^4 redundant zeros did not have to be stored in memory. In particular, the penta-diagonal matrix (5 non-zero diagonals) `LHM` was constructed using `spdiags`. A crucial optimisation from the point of view of perfomance was to replace the direct linear solution, `LMH \ RHS`, with the function, `pcg`, which carries out the preconditioned conjugate gradient numerical method at no appreciable change in accuracy (error bounded by `1e-6`), but significantly faster.

Regarding indexing, note that MATLAB uses column-major-order by default in its vectorised functions (see MathWorks website). To align with this convention, the row-major-order function, `rmo`, transposes the row indices, `i`, and leaves the column indices, `j`, fixed. Other configurations were trialed but this was ultimately the only one that produced the correct nutrient diffusion dynamics.

2.1.4 Introducing new nodes in *addCell.m*

2.1.5 Advancing the node positions using Euler's method in *updateNodes.m*

Chapter 3

Numerical experiments and simulation results

3.1 Statistics from the model

The model developed thus far is a numerical framwork for simulating growing cell colonies. There is an element of randomness in the model: when two nodes are dislodged from the same point there is initially no preferred direction to move in so one must be chosen randomly before other forces can take effect. For this reason, every seperate run of the model starting with identical initial conditions will look very different after time has passed. It is expected however that some "averaged" quantity will stabilise so long as the model is simulated over a fairly large number of runs. Seperate runs of the model belong to a set called an ensemble. We will start with a relatively straightforward metric, the number of cells at time $T = 500$ steps.

A fully grown colony will in general not be perfectly circular in shape. In order to measure the roundness of the colony we use the compactness metric used for roundness in image processing (quote Kai use of this metric)

$$C = \frac{P^2}{4\pi A}, \tag{3.1}$$

where $C \in [1, \infty)$ is 1 for a circle and can get to large numbers for highly branching shapes, A is the colony area, and P is the colony perimeter. Both of these are calculated from the formula for the microscopic cell density which is always given by when $f(x, y, t)$ changes sign. A black and white image is produced at each time step using Matlab's function `bwboundaries` as per Kai's suggestion. The area then is given by summing up the grid squares that are inside the implicit shape givcen by f and multiplying by h^2. The perimeter is got by using `bwboundaries`, which outputs an array of points on

the boundary from which the Euclidean distance between neighbouring points is found and then summed over.

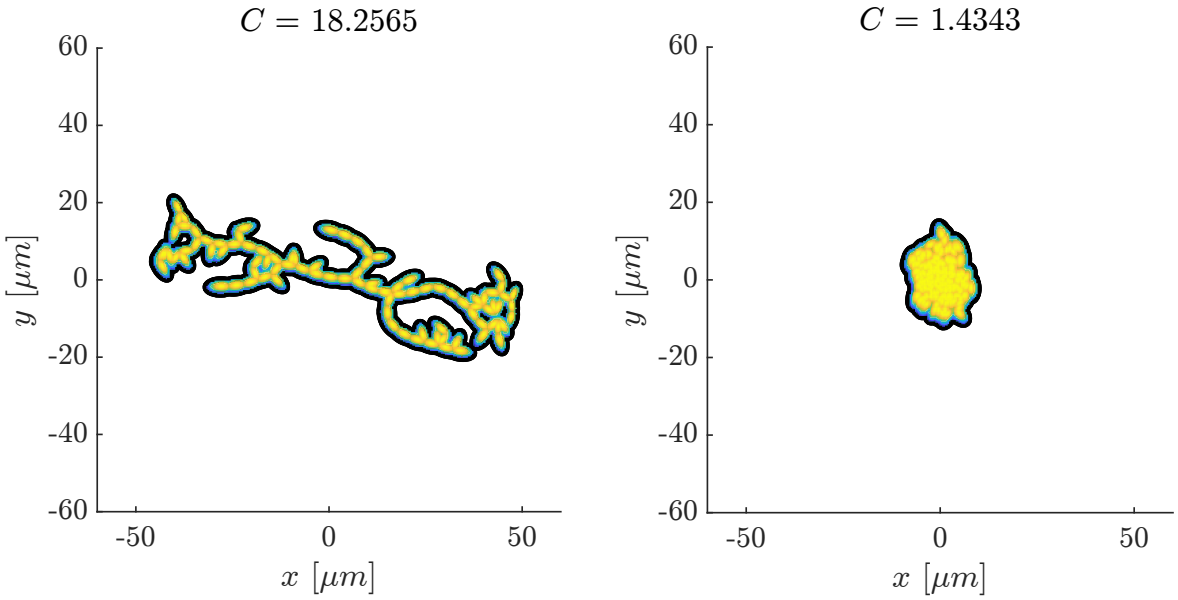

Figure 3.1: A comparsion of high and low compactness

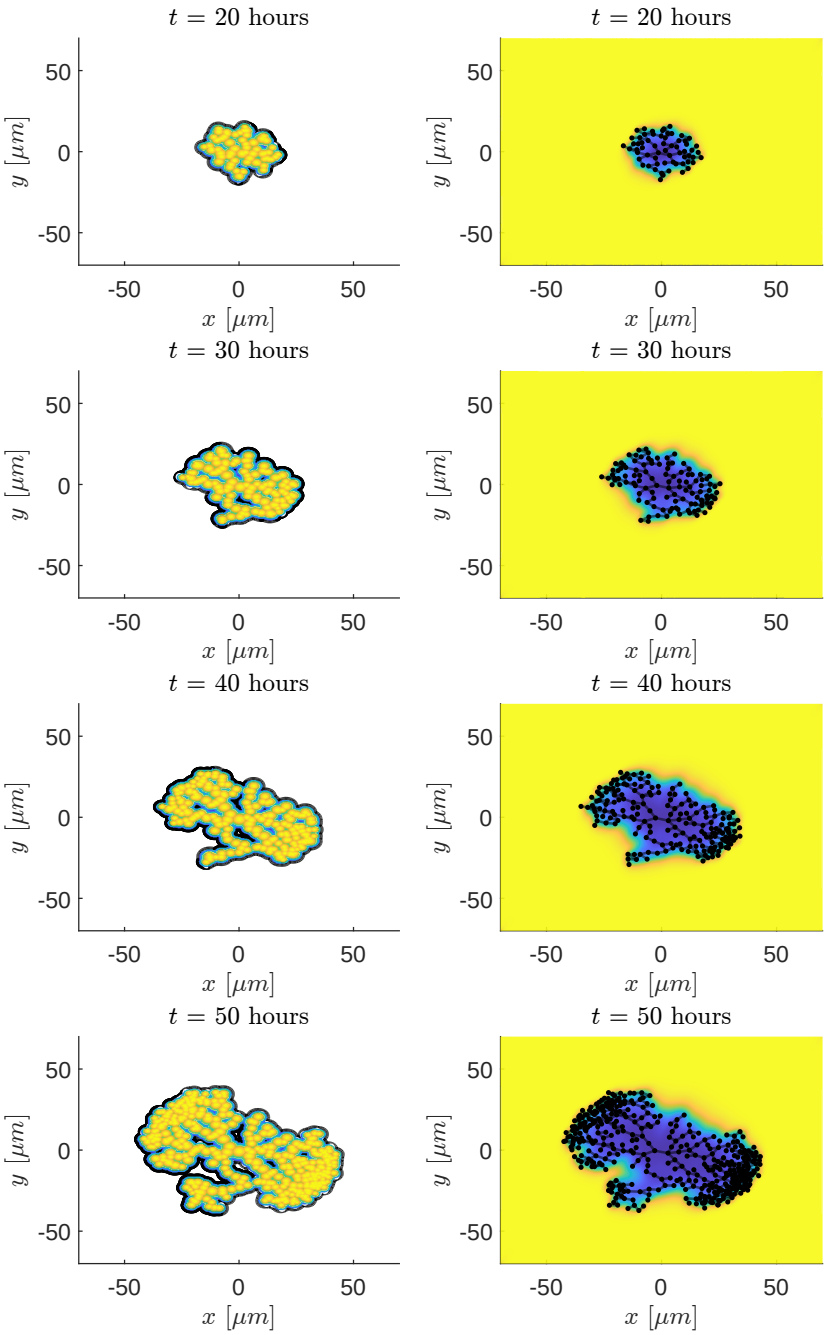

Figure 3.2: A cell colony with parameter values given by $\lambda_1 = 0.1$, $\lambda_2 = 1.0$, $\lambda_3 = 1.0$, $\lambda_4 = 0.5$, $\lambda_5 = 1.0$, $\lambda_6 = 0.5$, $\lambda_7 = 1.0$. On the left we have the biomass field, the nutrient field is on the right.

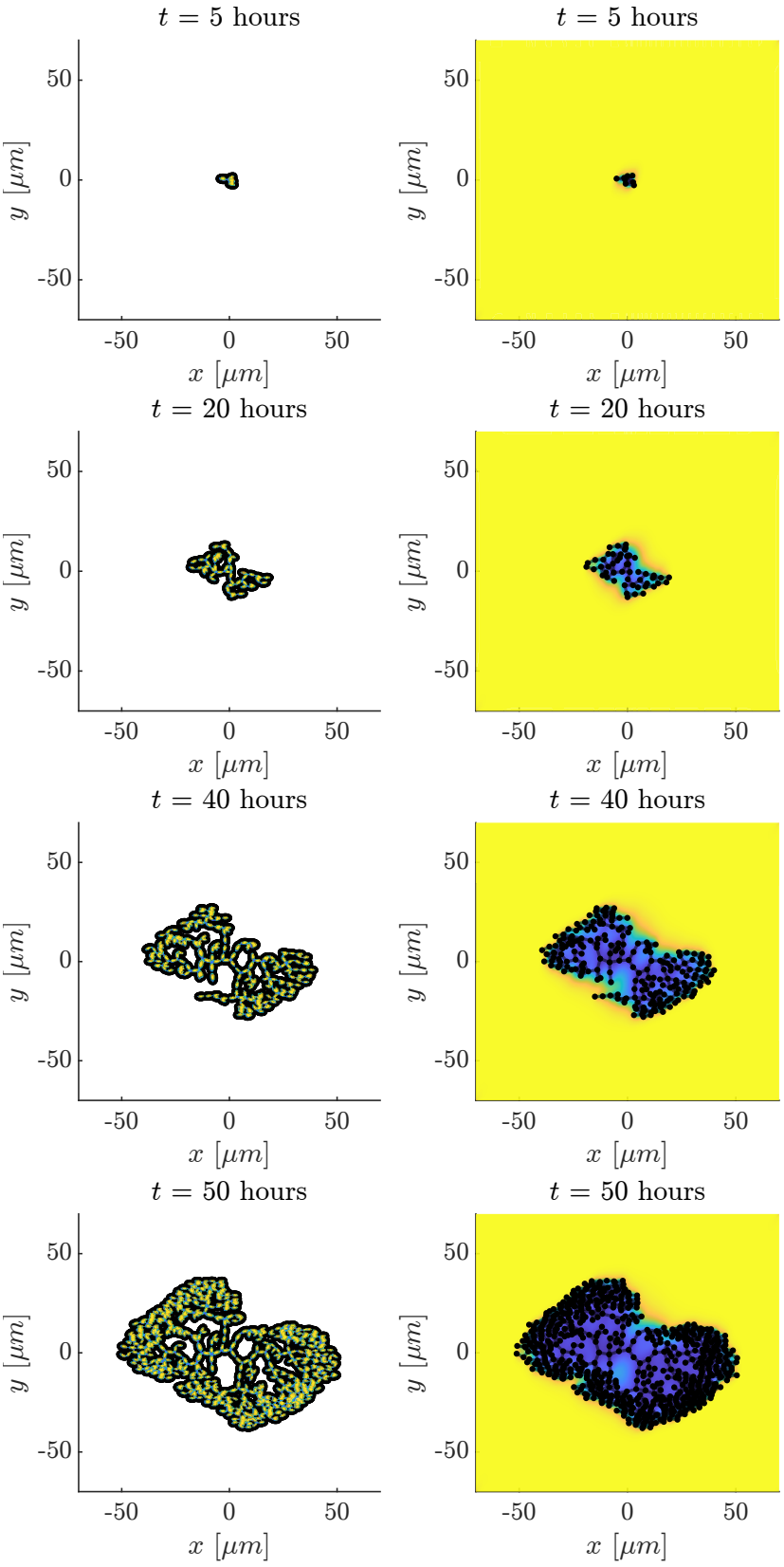

Figure 3.3: A cell colony with parameter values given by $\lambda_1 = 0.1$, $\lambda_2 = 5.0$, $\lambda_3 = 1.0$, $\lambda_4 = 0.5$, $\lambda_5 = 1.0$, $\lambda_6 = 2.0$, $\lambda_7 = 0.5$. Biomass on left and nutrient field on the right.

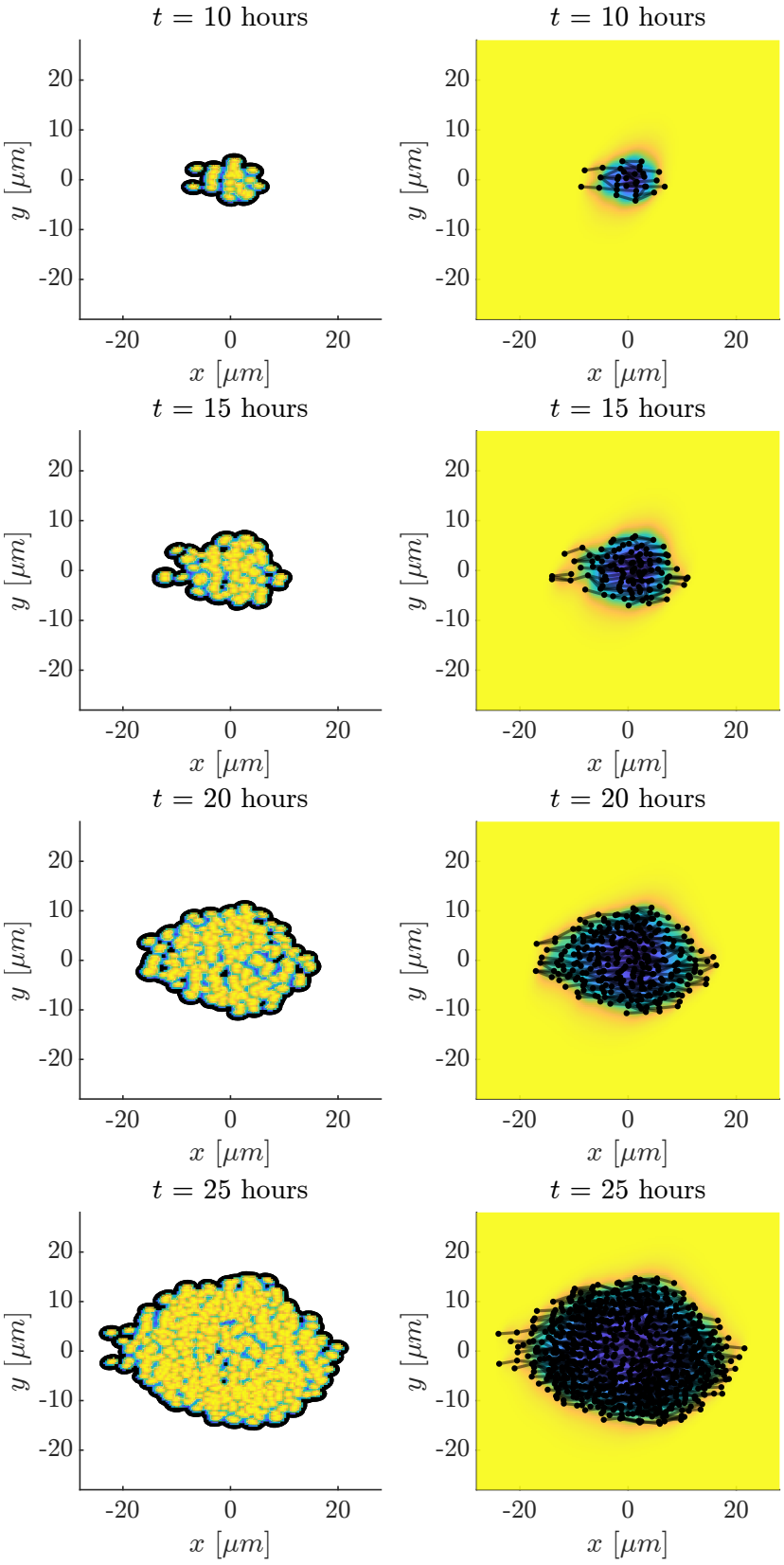

Figure 3.4: A cell colony with parameter values given by $\lambda_1 = 0.1$, $\lambda_2 = 5.0$, $\lambda_3 = 1.0$, $\lambda_4 = 0.5$, $\lambda_5 = 1.0$, $\lambda_6 = 0.4$, $\lambda_7 = 0.6$. Biomass on left and nutrient field on the right.

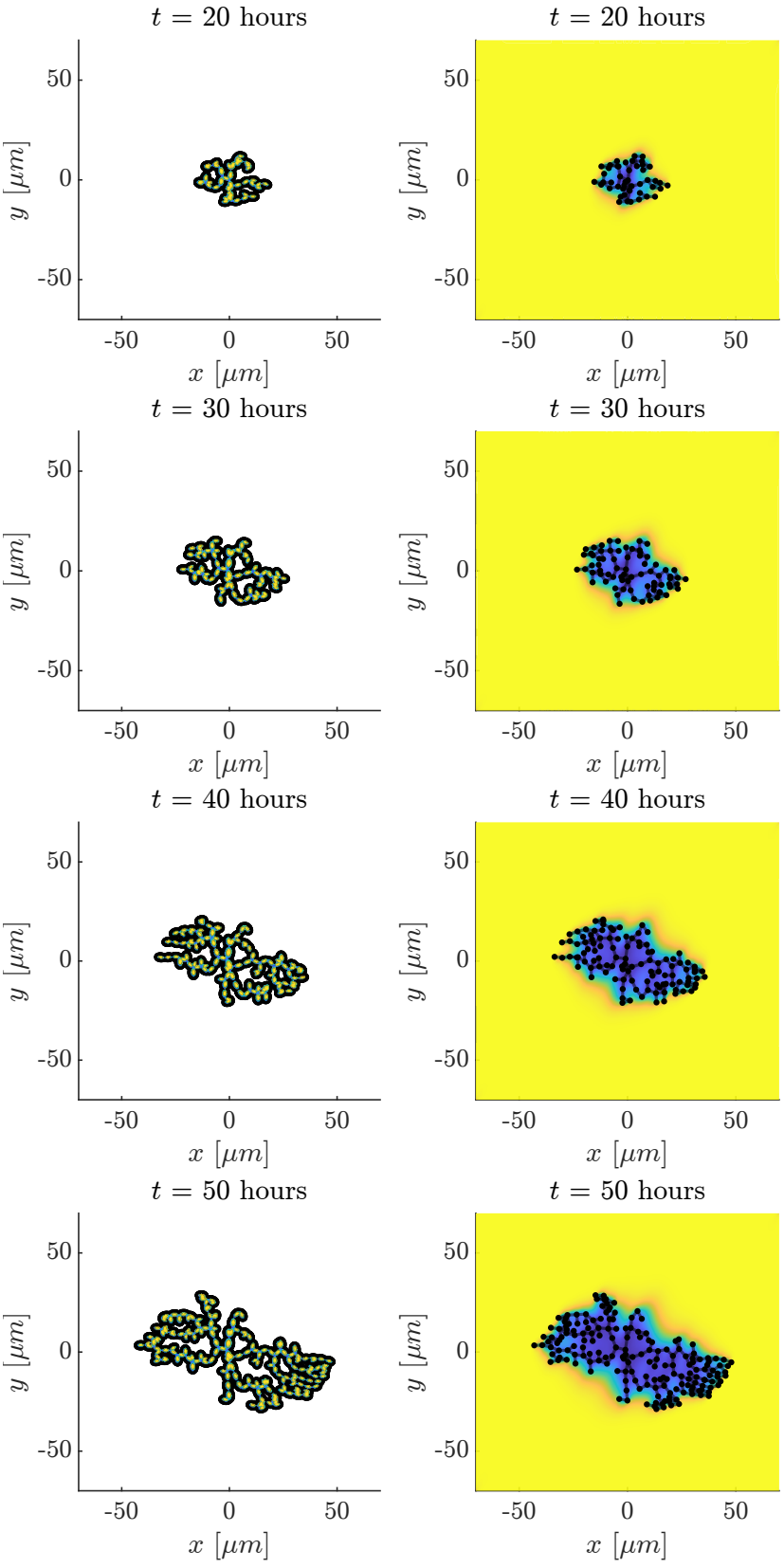

Figure 3.5: A cell colony with parameter values given by $\lambda_1 = 0.1$, $\lambda_2 = 5.0$, $\lambda_3 = 1.0$, $\lambda_4 = 0.5$, $\lambda_5 = 1.0$, $\lambda_6 = 2.0$, $\lambda_7 = 0.6$. Biomass on left and nutrient field on the right.

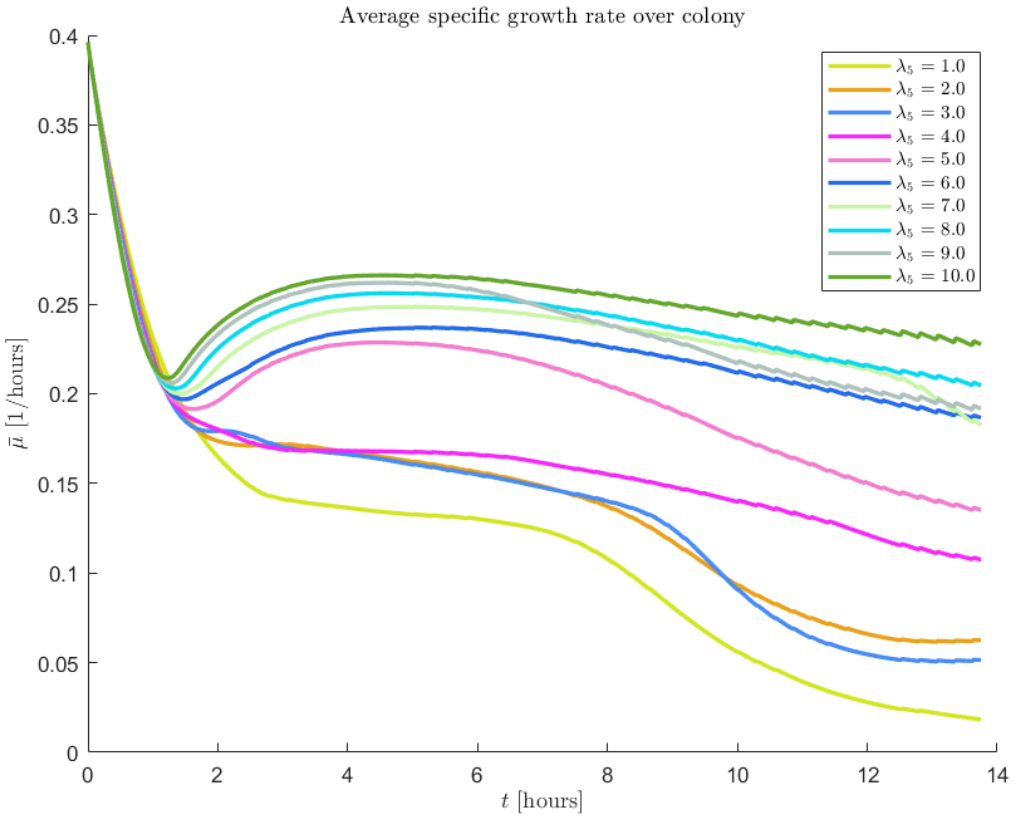

Figure 3.6: The colony average specific growth rate for different values of λ_5 is measured and plotted over time for ensemble size 1. The rest of the parameters took the values: $\lambda_1 = 0.1$, $\lambda_2 = 1.0$, $\lambda_3 = 1.0$, $\lambda_4 = 1.0$, λ_5 (variable), $\lambda_6 = 0.5$, $\lambda_7 = 0.7$. Remarkably, when $\lambda_5 \geq 5.0$ there is an interesting dynamic that occurs based on the compettition between nutrient consumption rate (λ_6) and the mobility (λ_5). For small values of mobility, the cells are not able to move enough into areas where the nutrient has not decayed.

Chapter 4

Discussion and limitations

4.1 Evaluation of the model

The computational model **CellColonySimulator** incorporates design decisions based on a number of concerns. These include

- The requirement for manageable run times on a gaming laptop.
- Modelling both colony scale and cell scale morphology.
- Capturing the process of mitosis in a natural but simple way.
- Ensuring the model is amenable to statisitical analysis.
- Providing a model ready for fitting to experimental micrographs of cell colonies.
- Crystallising the model into a form that could be analysed in a conceptually rigorous setting, say within the context of functional analysis, bifurcation theory or algebraic topology.

Whilst the computational model and conceptual apparatus devloped here take on these concerns, this section will explore to what extent these concerns were not met. In particualar, we comment on the fundamental limitations of the model. In subsection 4.1.1, the commentary begins with an evaluation of the model regarding the primary conceptual goals.

4.1.1 Conceptual limitations

The model was able to deploy the concept of algebraic curves with time dependent coefficients to represent the cells. Indeed the implicit equation for a translated, rotated, and scaled ellispe is a member of the ring of polynomials in two variables $\mathbb{R}[x, y]$ where

coefficients are dependent on the underlying discrete network of nodes. In a previous version of the model, the underlying network of nodes were initially collocated and via the elimination of constraints the network was able to unfold over time. This was implemented with logical "mask" matrices in MATLAB, but brought practical problems. One such problem was that branches could easily deplete their store of nodes, and indeed the number of nodes chosen at the beginning of the simulation had to be arbitrary. This meant that the previous model had prohibitive limits on the number of branches and the size of the branches. Another limitation, was that the ODE solver was slowed down by the need to simulate the evolution of collocated vertices.

CellColonySimulator overcomes this issue by simply adding new nodes to the network, just very close by ($\delta = 0.01L_0$). As it turned out, this was not a consequential fault, since the distance δ can be made so small that the daughter cell appears to bud from the parent cell, as would be observed in the case of pseduo-hyphal growth of Baker's yeast. What I argue is that this is in fact a conceptual fault present in the model because it represents an arbitrary addition of a new variable, the budding angle θ_b not to mention the constant (but small) budding radius δ. What I intended was a "derivation" of cells division within a broader model, as opposed to an imposition of more assumptions and parameters over time to achieve the mitosis.

Unsatisfyingly, the budding angle is chosen to be uniformly distributed on $[0, 2\pi]$ and is the only source of randomness in the model. Of course the budding angle could be chosen to be in line with the nutrient gradient vector and this would eliminate randomness from the model. Future work could seek to reintroduce randomness in the form of a stochastic process, such a langevin term in the the ODEs for the node positions (and even in the nutirent PDE).

A model whaich I have prototyped, is one in which the nutrient field is effectively "converted into nodes". That is, the nutrient field is not simply a scalar field of "stuff", but actually becomes a field of structure. The simplest version of this is to set the nutrient field as a collection of Brownian walkers that can be taken up by a colony and added to the colony's network. In some sense this would solve the problem, since no new nodes are ever added into the simulation, its just that the initially unstructured nodes take on a new behaviour within the cell network.

Another conceptual direction is that of taking seriously the notion of "cell as structured randomness". Biochemistry tells us that cells are reducible to chemical reaction networks (CRNs), but of course to combine CRNs into the workings of a single cell is not trivial. This means effictive models are required, which leave out most of the labrythine details of the cell's internal workings (see proteins, DNA, RNA, mitotic spindles,

golgi apparatus) but allow for efficient computation. Even CRNs are limited by the fact that they usually do not consider the shape of a protein which plays a key role in its functionality. The model presented here brushes over all of these details in favour of an effective mathematical approach, but future work could focus linking nanoscopic molecular scale and microscopic cellular scale.

Regarding the shape of individual cells, the ellipse with variable aspect ratio is chosen to represent eukaryotic fungal cells such as saccharomyces cerevisiae. Beyond changing the cell aspect ratio, there is no mechanism in **CellColonySimulator** to change to plant like cell shapes (for example) which are more blocky. Algebraic modelling of the cell membrane (in the case of plant cells) or a diverse range of cell wall shapes (fungal or even mammal cells) is implied here however. For insance, by chosing more elaborate polynomials, one can derive a taxonomy of cell shapes at the cost of implementation simplicity and perhaps computational performance. An avenue to explore is seeing whether bifurcations can be achieved in a different ring of continuous functions of x and y. Indeed, can the whole colony be represented by (the level set of) one function at each time, such that the growth of the colony can be given by a homotopy between two functions?

4.1.2 Dynamical limitations

A (fixed node count) cell colony undergoes the dynamics of an overdamped spring network capable of internal collisions driven by an external nutrient field. The key concern is whether this dynamical structure is grounded in biological evidence, and whether the model just metaphorical. Firstly, it is worth noting that because we are in the overdamped regime, there are no intrinsic oscillations between connected nodes. This is because the node velocity is proportional to the force as opposed to the case of an underdamped harmonic oscillator, where the acceleration is proportial to force. In answer to the concern posed, the current model is an effective one however it is feasible that the model can be fitted to experimental micrograph data, which is commented on in section 4.2, and thus the seven input parameters could be found from real data.

The springs have a neutral position which is a cell length distant from the parent node. In other words, the spring force encodes a passage between equilibrium positions of the colony. The presence of an (external) chemotactic force coupled into the dynamics complicates this by gradually moving the equilibrium position for a fixed node count colony outwards. One observation of the simulation results, was that internal cells grew beyond the nominal cell length due to the additional force of chemotaxis. In practice, this issue is alleviated by increasing the cell elasticity, λ_2, but this is nonetheless a limitation.

New nodes are added to the network nearby old nodes as stated in Chapter 1. The timing of the addition of new nodes is somewhat adhoc and provisional, but could in theory be elevated to something more sophisticated. Starting with the colony growth rate $\mu(t)$ which is taken to be a fraction of the ideal growth rate $\mu^* = 0.4$ hour^{-1}. Cells are added in at discrete intervals of time, given by

$$\Delta n = \left\lceil \frac{1}{\Delta t} \right\rceil.$$

The value Δn is the number of time steps between two (global) mitosis events. For a time step of $\Delta t = 0.02$, this amounts to $\Delta n = 50$ time steps. The amount of nodes added at this time step, denoted n^* (a multiple of Δn), is given by $(e^{\mu(t_{n^*})}-1)N_{\text{nodes}}(t_{n^*})$ where $\mu(t_{n^*})$ is the average value of the nutrient field (recall that the nutrient field is assumed to be 1.0 at the simulation outset). The assumption that growth rate is defined in terms of the nutrient field makes sense, however the assumption of it being *globally* defined thus is somewhat dubious. A possible impovement is to draw up a coarse grid in which each square can fit a dozen or more cells, and say that the growth rate $\mu = \mu(x, y, t)$ is locally defined by averaging nutrient per grid square. Of course, the reason that a spatially independent growth rate, $\mu = \mu(t)$, was chosen was for implementation simplicity.

The periodic structure of adding cells in at simultaneous snapshots of time, is an artificial choice based on implementation simplicity alone. A higher fidelty model could be constructed in which cells bud off at any time. One way to do this is to calibrate mitosis times per cell based on cell cycle data and allow a cell to undergo mitosis if a certain threshold of nutrient is locally available. This would not be difficult to implement, but comes with a few conceptual caveats. One issue is that a (discrete) logic would need to be introduced per cell, which opens up a fascinating area of study ("cell as computer program"), however this is outside the scope of this work.

Another limitation of the dynamical model is overly simplistic collisions. The collision mechanism is node based, as opposed to edge based or even SDF based. An attempt to improve upon the expedient hard ball collision scheme, is developed in Appendix A.2. Therein, constrained dynamics techniques are employed to ensure no pairwise overlap of the biomass level sections. It is clear that constrained dynamics is the way to go, but my unique implementation is not computationally efficient and therefore was not used in **CellColonySimulator**.

Finally, the use of a reaction-diffusion PDE is traditional in mathematical biology, however in light of the insights developed here, future work could focus on modelling

random fluctuations in nutrient concentration, or even introducing multiple interacting morphogens such as in Turing's morphogenesis paper (Turing (1990)).

4.1.3 Computational limitations

4.2 Readiness for experiemntal validation and fitting

Conclusions

Appendix

A.1 Collocation algorithm with mask matrices

Each cell indexed $j \in \{1, \dots N\}$ has five pieces of data which are enough to define globally the SDF of the cell, namely, the center coordinates (x_j, y_j), the angle of orientation θ_j, and the semi axes dimensions a_j, b_j. Each piece of data will be a function of time t. For the purpose of simplicity, we model each cell as two point masses $m_1 = m_2 = m$ connected by a spring with stiffness K. The point masses are located at $\mathbf{r}_j^{(1)}$ and $\mathbf{r}_j^{(2)}$ inside the ellipse along the major axis and symmetrically about the ellipse center. This means that the center is given by

$$\mathbf{x}_j = \frac{1}{2}\left(\mathbf{r}_j^{(1)} + \mathbf{r}_j^{(2)}\right).$$

Fixing the semi-minor axis b_j, we give the semi-major axis a_j by

$$a_j = a_0||\mathbf{r}_j^{(1)} - \mathbf{r}_j^{(2)}||.$$

We extract the orientation angle using a two argument inverse tangent function,

$$\theta_j = \arctan\left(y_j^{(2)} - y_j^{(1)}, x_j^{(2)} - x_j^{(1)}\right)$$

When the number of cells is fixed, the colony dynamics is modelled using first order EOMs with two primary forces: intracellular spring force and and intercellular contact force to void overlap. We take the assumption that many authors make (include reference here) which is that inertia is negligible due to drag effects (more on this). That is the velocity is directly proportional to the force

$$\mathbf{v}_j^{(i)} = \frac{1}{\eta}\mathbf{F}_j^{(i)},$$

where $i \in \{1, 2\}$ and $j \in \{1, \dots, N\}$ where η is an expression for the drag and $\mathbf{F}_j^{(i)}$ is the sum of the forces acting on the i-th particle of the j-th cell. Overall, the simulation will be begun with V vertices where V is a positive power of 2. We map from local indices (i, j) to a global index n using

$$n = 2(j-1) + i,$$

which is called "row major order". We then flatten the list of x-coordinates and y-coordinates into one state vector $\mathbf{X}(t)$ given by

$$\mathbf{X}(t) = [x_1(t), \dots, x_V(t), y_1(t), \dots, y_V(t)]^T,$$

where (x_n, y_n) is the coordinate of the n-th vertex for $n \in \{1, \dots, V\}$. Note that the change in the number of cells is simulated by removing constraints between the vertices.

At the beginning of the simulation $x_1 = \cdots = x_V$ and $y_1 = \cdots = y_V$. The first order ordinary differential equation is phrased in terms of a mass matrix and a force function,

$$M(t, \mathbf{X}) \frac{d\mathbf{X}(t)}{dt} = f(t, \mathbf{X})$$

We start by considering the spring force acting on the n-th particle due to the m-th particle given n and m are connected by springs. This is given by $\mathbf{F}_{nm}^{\text{spring}}$ as

$$\mathbf{F}_{nm}^{\text{spring}} = K(||\mathbf{x}_m - \mathbf{x}_n|| - L_{nm}) \frac{\mathbf{x}_m - \mathbf{x}_n}{||\mathbf{x}_m - \mathbf{x}_n||},$$

where K is a spring constant meant to represent cell elasticity, and L_{nm} is the nominal length of the spring connecting them. There are three situations regarding edge between vertex n and m: either they are connected by a spring with $L_{nm} > 0$, they are collocated by an equality constraint or they are disconnected. If the two masses are collocated by an equality constraint, then the spring force is undefined so we must omit this. We must also omit the contact force because this has no sense for collocated vertices. The contact force between two disconnected vertices is given as

$$\mathbf{F}_{nm}^{\text{contact}} = \begin{cases} C \frac{\mathbf{x}_m - \mathbf{x}_n}{||\mathbf{x}_m - \mathbf{x}_n||}, \text{ if } ||\mathbf{x}_m - \mathbf{x}_n|| \leq d \\ \mathbf{0}, \text{ if } ||\mathbf{x}_m - \mathbf{x}_n|| > d, \end{cases}$$

which is used to ensure that vertices do not overlap past a threshold distance d. In terms of the connectivity, we can encode the fact that two vertices are connected (by a spring) in an adjacency matrix A_{nm} which is equal to 1 if they are connected and 0 otherwise. We also introduce a second matrix B_{nm} which represents when two vertices are disconnected, i.e., $B_{nm} = 1 - A_{nm}$. A third matrix is introduced for collocation $C_{nm} = 1$ if $n \neq m$ and n and m are collocated and 0 otherwise. This matrix (in fact $\tilde{C}_{nm} =\sim C_{nm}$) will be used as a logical mask to filter out `nan` values from the force matrices.

We can think of the forces (whether elastic or contact) as pairs of matrices $(F_x)_{nm}^{\text{spring}}$ and $(F_y)_{nm}^{\text{spring}}$ and similarly for the contact forces. We construct the x-component of the overall force vector

$$(f_x)_n = \sum_{m=1}^{V} ((F_x)^{\text{spring}}(A \& \tilde{C}))_{nm} + \sum_{m=1}^{V} ((F_x)^{\text{contact}}(B \& \tilde{C}))_{nm},$$

where $A\&\tilde{C}$ are mask matrices that ensure both A and not C are satisfied. Similarly for the y-component,

$$(f_y)_n = \sum_{m=1}^{V} ((F_y)^{\text{spring}}(A \& \tilde{C}))_{nm} + \sum_{m=1}^{V} ((F_y)^{\text{contact}}(B \& \tilde{C}))_{nm}.$$

The question remains: how are the mask matrices A, B and C made to change over time? Here we will do a small illustrative example with $N = 4$ cells and $V = 2N = 8$ vertices.

Initially all the vertices will be collocated at the same position (x_0, y_0). This means that the force vector should come out to zero because we essentially have one particle not interacting with anything. Let us check that $A\&\tilde{C}$ and $B\&\tilde{C}$ both vanish. C is given directly as a matrix of all ones, which says that each vertex is constrained to every other vertex. Thus $\tilde{C} = O$ where O is a 8×8 zero matrix. Initially, say $A = O$ (the zero matrix) because there are no springs. Note that this immediately says that B is a matrix of all ones. In any case, both $A\&\tilde{C} = B\&\tilde{C} = O$.

At some point, the effectively single particle starts growing away from its initial position. This is achieved by parcelling half of the vertices into one set of collocated points, and the other half into another set of collocated points. Programmatically, we select half of the points and translate them to a random nearby position to (x_0, y_0) and modify the C matrix to remove the constraints between the first and second set,

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

At this point, we should also set the nominal length of the springs as L_0 and the corresponding matrix of lengths $L_{mn} = L_0$ (the constant matrix with value L_0). As it turns out, the action of the spring pressing the vertices apart will model the geometric growth of each elliptical cell. The A matrix will be given by

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now let's suppose that the vertices in one set undergo another mitosis event, splitting half of the particles off into a third set. The new C matrix, called C' must reflect

this change. We call $C(\alpha_1, \ldots, \alpha_M) = \text{diag}(\mathbf{1}_{V/2^{\alpha_1}}, \ldots, \mathbf{1}_{V/2^{\alpha_M}})$ where $\mathbf{1}_{V/2^{\alpha_q}}$ is an all 1's matrix of dimension $V/2^{\alpha_q} \times V/2^{\alpha_q}$ where q indexes over the powers of two and represents the accumulation of division events. During a mitosis event in which the α_q-th vertex set splits, $C(\alpha_1, \alpha_2, \ldots, \alpha_M) \to C(\alpha_1, \ldots, \alpha_q + 1, \alpha_q + 1, \ldots \alpha_M)$. In other words, the α_q-th matrix splits into two matrices of half the size. In our 8×8 case, in which we divide the second vertex set, this results in the following

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The new matrix A is got by adding a connections between the left over smaller matrices, which is best understood by visualising the matrix as

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Taking another division event on the second block, we get

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

and

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

A.2 Cell-cell collisions with constrained dynamics

Earlier, it was mentioned that the intersection of cells could be found by taking a smoothmax. To find the overlapping area the sum of the grid points in the intersection can be taken and then multiplied by the grid square area $h_x h_y$. Whilst it is more or less trivial to calculate the overlapping area, ensuring that this area remains zero throughout the simulation is more involved. We employ the technique explained by Witkin (1997) based on constrained dynamics. The constraint in this case is that the area C remains 0 for all times,

$$C(\mathbf{q}) = 0, \tag{1}$$

where $\mathbf{q}$ is the concatenated state vector of the system. One should be aware that, if we have multiple colonies, it will be required to have pairwise constraints forcing no overlap between each of the constitiuent colonies. For the purpose of simplicity our state vector $\mathbf{q}$ will just take into account position and orientation and not size of cells. Later on, this will be generalized. Therefore $\mathbf{q}$ is produced by concatenating over $\mathbf{q}_j$ into one $3N \times 1$ column vector, where

$$\mathbf{q}_j = \begin{bmatrix} x_j \\ y_j \\ \theta_j \end{bmatrix}. \tag{2}$$

Of course, the number of variables per cell can be larger but this comes at a cost in computational time. Recall that the intersecting area is secretly a function of the cell state coordinates $\mathbf{q}_j$ since we are taking a smoothmax of the SDFs of each cell. In other words,

$$f_{\text{intersect}}(x, y, \mathbf{q}) = \text{smoothmax}(f_1(x, y, \mathbf{q}), \ldots, f_N(x, y, \mathbf{q})),$$

for concreteness, we write out the formula for smoothmax which is the negative smoothmin of the negatives of the SDFs or

$$f_{\text{intersect}}(x, y, \mathbf{q}) = -\text{smoothmin}(-f_1(x, y, \mathbf{q}), \ldots, -f_N(x, y, \mathbf{q})).$$

In full this boils down to,

$$f_{\text{intersect}}(x, y, \mathbf{q}) = k \log\left(\sum_{j=1}^{N} e^{f_j(x,y,\mathbf{q})/k}\right).$$

Now we assume the state is given by five variables for full generality:

$$\mathbf{q}_j(t) = \begin{bmatrix} x_j(t) \\ y_j(t) \\ \theta_j(t) \\ b_j(t) \\ R_j(t) \end{bmatrix}, \tag{3}$$

where $b(t)$ is the semi-major axis length and $R_j(t) \in (0, 1]$ is the aspect ratio of the elliptical cell so the semi-minor axis is given by $a_j(t) = R_j(t)b_j(t)$. Why are we going to the effort to write out $C(\mathbf{q})$ in full? Because we need to compute the Jacobian of $C(\mathbf{q})$ with respect to $\mathbf{q}$ in order to carry out the constrained dynamics algorithm. The upshot is that smoothmax is differentiable whereas maximum is not differentiable.

We define the region in $\mathbb{R}^2$ to integrate over by

$$\Omega(\mathbf{q}) = \{(x, y) \in \mathbb{R}^2 \mid f_{\text{intersect}}(x, y, \mathbf{q}) \leq 0\}, \tag{4}$$

so that the area of the overlapping region is simply a two-dimensional integral over $\Omega(\mathbf{q})$ of 1,

$$C(\mathbf{q}) = \int \int_{\Omega(\mathbf{q})} dxdy. \tag{5}$$

Now we break up the integral into a sum over simply connected components (SCC) so that we can safely apply Leibniz's rule

$$C(\mathbf{q}) = \sum_{i \in SCC(\mathbf{q})} \int \int_{\Omega_i(\mathbf{q})} dxdy. \tag{6}$$

Leibniz's rule tells us how to carry out a total derivative of $\int \int_{\Omega_i(\mathbf{q})} dxdy$ with respect to t which determines $\mathbf{q}$. Let's take that total derivative now to attain

$$\frac{d}{dt} \int \int_{\Omega_i(\mathbf{q})} dxdy = \int \int_{\Omega_i(\mathbf{q})} \frac{\partial(1)}{\partial t} dxdy + \int_{\partial\Omega_i(\mathbf{q})} (1)\mathbf{v}_{\partial\Omega_i(\mathbf{q})} \cdot \hat{\mathbf{n}} dl, \tag{7}$$

where $\mathbf{v}_{\partial\Omega_i(\mathbf{q})}$ is the "Eulerian" velocity of the boundary of the i-th SCC. All of this can be avoided if we integrate over a smoothstep which is defined in our region.

$$C(\mathbf{q}) = \int \int_D \left[\frac{1}{2} + \frac{1}{2} \tanh\left(-\frac{1}{K} f_{\text{intersect}}(x, y, \mathbf{q})\right)\right] dxdy, \tag{8}$$

where D is the entire domain of the pertri dish.

$$\frac{\partial f_{\text{intersect}}(x, y, \mathbf{q})}{\partial \mathbf{q}} = \frac{\sum_{j=1}^{N} \frac{\partial f_j(x,y,\mathbf{q})}{\partial \mathbf{q}} e^{f_j(x,y,\mathbf{q})/k}}{\sum_{j=1}^{N} e^{f_j(x,y,\mathbf{q})/k}}.$$

Conveniently, computing the Jacobian, has turned into N problems that are easier to solve individually. Namely computing $\frac{\partial f_j(\mathbf{q})}{\partial \mathbf{q}}$. This requires us to express the SDF for an ellipse in terms of the query point (x, y), the center of the cell (x_j, y_j), and its orientation θ_j. This is given in terms of $l_j^-(\mathbf{q}, x, y)$ and $l_j^+(\mathbf{q}, x, y)$ which are given in the ellipse formula. We use MATLAB's symbolic toolbox to compute the ellipse Jacobian $\frac{\partial f_j(x,y,\mathbf{q})}{\partial \mathbf{q}}$ and return a function that can take numerical inputs using

`matlabFunction(J,"File","ellipseJacobian")`. With that we can compute Jacobian of the constraint using

$$\frac{\partial C(\mathbf{q})}{\partial \mathbf{q}} = -\frac{1}{2K}\int\int_D \left[1-\tanh^2\left(-\frac{f_{\text{intersect}}(x,y,\mathbf{q})}{K}\right)\right]\frac{\partial f_{\text{intersect}}(x,y,\mathbf{q})}{\partial \mathbf{q}}dxdy. \tag{9}$$

Now that all the derivatives have been computed analytically, we can carry out the computation of the integral over x and y using a numerical integral in MATLAB. From now we use f instead of $f_{\text{intersect}}$ for brevity, and we pass to index notation, instead expression the β-th component of J as

$$J_\beta = -\frac{1}{2K}\int\int_D \left[1-\tanh^2\left(-\frac{f(x,y,\mathbf{q})}{K}\right)\right]\frac{\partial f(x,y,\mathbf{q})}{\partial q_\beta}dxdy. \tag{10}$$

For the computation of constraint forces, we need to further compute the total derivative of the Jacobian with respect to time. This turns out to be realted to the Hessian matrix of $C(\mathbf{q})$ (for no explicit time dependence), as

$$\dot{J}_\beta = \sum_{\alpha=1}^{5N}\dot{q}_\alpha\frac{\partial^2 C}{\partial q_\alpha \partial q_\beta} = \sum_{\alpha=1}^{5N}\dot{q}_\alpha H_{\alpha,\beta},$$

We need to compute how the Hessian of C can be written in terms of the Hessian and Jacobian of f.

$$\frac{\partial^2 C}{\partial q_\alpha \partial q_\beta} = \frac{\partial}{\partial q_\alpha}J_\beta \tag{11}$$

Crunching the calculuations we get

$$\frac{\partial}{\partial q_\alpha}J_\beta = -\frac{1}{4K^2}\int\int_D \left[1-\tanh^2\left(-\frac{f}{K}\right)\right]\left[\tanh\left(-\frac{f}{K}\right)\frac{\partial f}{\partial q_\alpha}\frac{\partial f}{\partial q_\beta}+K\frac{\partial^2 f}{\partial q_\alpha \partial q_\beta}\right]dxdy \tag{12}$$

Luckily, the Hessian is built into MATLAB's symbolic toolbox, so we can just call `matlabFunction(H,"File","ellipseHessian")` and the function to compute the Hessian is saved into our working directory. Note that the function `ellipseHessian` computes a $5\times 5\times N_{\text{grid}}\times N_{\text{grid}}$ matrix. We call it for each cell $j\in\{1,\dots,N\}$ but recall that for a given value of $\mathbf{q}$, the Hessian still has (x,y) as free field variables. In order to integrate the field data, we use the cell-wise Jacobian field $J^j_\beta(x,y)$ and the cell-wise Hessian field $H^j_{\alpha,\beta}(x,y)$ to calculate the overall Hessian field for f

$$\frac{\partial^2 f}{\partial q_\alpha \partial q_\beta} = \left(\frac{1}{k}\right)\left[\frac{\left(\sum_{j=1}^N E_j\right)\left(\sum_{j=1}^N E_j(kH^j_{\alpha,\beta}+J^j_\alpha J^j_\beta)\right)-\left(\sum_{j=1}^N E_j J^j_\alpha\right)\left(\sum_{j=1}^N E_j J^j_\beta\right)}{\left(\sum_{j=1}^N E_j\right)^2}\right],$$

where $E_j = e^{f_j/k}$ is used for short. To actually compute the overall integrals for $\frac{\partial C}{\partial q_\beta}$ and $\frac{\partial^2 C}{\partial q_\alpha \partial q_\beta}$ we use MATLAB's `trapz` function the following way

```
trapz(y_lin,trapz(x_lin,integrandJacobian,3),2);
```

or, in the case of the Hessian,

```
trapz(y_lin,trapz(x_lin,integrandHessian,4), 3);
```

where the integrand in the Jacobian cas is given by

$$F_\beta(x,y) = \left(\frac{T^2-1}{2K}\right)\frac{\sum_{j=1}^{N} J_\beta^j E_j}{\sum_{j=1}^{N} E_j},$$

where $T = \tanh\left(-\frac{f(x,y,\mathbf{q})}{K}\right)$ and, in the case of the Hessian

$$G_{\alpha,\beta}(x,y) = \left(\frac{T^2-1}{4K^2}\right)\left(T\left(\frac{\sum_{j=1}^{N} J_\alpha^j E_j}{\sum_{j=1}^{N} E_j}\right)\left(\frac{\sum_{j=1}^{N} J_\beta^j E_j}{\sum_{j=1}^{N} E_j}\right) + K\frac{\partial^2 f}{\partial q_\alpha \partial q_\beta}\right),$$

which, after some serious simplification becomes

$$G_{\alpha,\beta}(x,y) = \frac{T^2-1}{4Kk\left(\sum_{j=1}^{N} E_j\right)^2}\sum_{n=1}^{N}\sum_{m=1}^{N} E_n E_m\left[\left(\frac{kT-K}{K}\right)J_\alpha^n J_\beta^m + J_\alpha^m J_\beta^m + kH_{\alpha,\beta}^m\right]$$

To make things neat we replace $B = \frac{T^2-1}{4K^2}$, $\hat{E} = \sum_{j=1}^{N} E_j$ and we subsitute the ratio of the smoothstep and smoothmax parameters $S = K/k$ to attain,

$$G_{\alpha,\beta}(x,y) = \frac{B}{\hat{E}^2}\sum_{n=1}^{N}\sum_{m=1}^{N} E_n E_m\left[S(kH_{\alpha,\beta}^m + J_\alpha^m J_\beta^m) + (T-S)J_\alpha^n J_\beta^m\right],$$

Thus we can write

$$J_\beta = \int\int_D F_\beta(x,y)dxdy,$$

$$\dot{J}_\beta = \sum_{\alpha=1}^{5N}\dot{q}_\alpha\int\int_D G_{\alpha,\beta}(x,y)dxdy,$$

Now we subsitute to obtain the final integral formulae:

$$J_\beta = \int\int_D \frac{B}{\hat{E}S}\sum_{j=1}^{N} J_\beta^j E_j dxdy,$$

$$\dot{J}_\beta = \int\int_D \frac{B}{\hat{E}^2}\sum_{\alpha=1}^{5N}\sum_{n=1}^{N}\sum_{m=1}^{N}\dot{q}_\alpha E_n E_m\left[S(kH_{\alpha,\beta}^m + J_\alpha^m J_\beta^m) + (T-S)J_\alpha^n J_\beta^m\right]dxdy.$$

We set out to calculate the following scalar $A = JWJ^t$ where $W = M^{-1}$ is the inverse mass matrix of our dynmical system. Note, that since we have only one constraint our goal is to solve for one Lagrange multiplier λ which is given as

$$A\lambda = -\sum_{\beta=1}^{5N} \dot{J}_\beta \dot{q}_\beta - JWQ - \kappa_s C - \kappa_d \dot{C},$$

Recall $C = \frac{1}{2}\int\int_D (1+T)dxdy$ and $\dot{C} = \sum_{\beta=1}^{5N} \dot{q}_\beta J_\beta$. This means we can write out our equation for λ explicitely

$$\left(\sum_{\alpha=1}^{5N}\sum_{\beta=1}^{5N} J_\alpha W_{\alpha,\beta} J_\beta\right)\lambda = -\sum_{\beta=1}^{5N} \dot{J}_\beta\dot{q}_\beta - \sum_{\alpha=1}^{5N}\sum_{\beta=1}^{5N} J_\alpha W_{\alpha,\beta} Q_\beta - \kappa_s \frac{1}{2}\int\int_D (1+T)dxdy - \kappa_d \sum_{\beta=1}^{5N} \dot{q}_\beta J_\beta,$$

Note that its convenient from a computatational point of view to bring the quadriple sum into the integral and then evaluate this using `trapz`:

$$\int\int_D \frac{B}{\hat{E}^2} \sum_{\alpha=1}^{5N}\sum_{\beta=1}^{5N}\sum_{n=1}^{N}\sum_{m=1}^{N} (\dot{q}_\alpha \dot{q}_\beta E_n E_m)\left[S(kH^m_{\alpha,\beta} + J^m_\alpha J^m_\beta) + (T-S)J^n_\alpha J^m_\beta\right]dxdy.$$

where the term $\dot{q}_\alpha\dot{q}_\beta E_n E_m$ is a $(5N)\times(5N)\times N\times N\times N_{\text{grid}}\times N_{\text{grid}}$ array. Now, of course the integral we are after is the following

$$I^{l,m,n}_{\alpha,\beta} = \int\int_D \frac{B_l}{E_l^2}(E_n E_m)\left[S(kH^m_{\alpha,\beta} + J^m_\alpha J^m_\beta) + (T_l - S)J^n_\alpha J^m_\beta\right]dxdy.$$

Observing this formula, we can note that $\hat{E}$ has been replaced by E_l. This is actually an improvement since $\hat{E}$ was only ever part of the smoothmax approximation, we replace it by an arbitrary E_l which must be picked prior to evaluating the integral based on the maximum. I have subscripted B_l and T_l similarly as they depend on E_l. The main reason for this was due to difficulty in evaluating the integral over a reciprical sum when N was arbitrary. Note that if we evaluate this integral symbolically in preprocessing we can essentially divide the amount of computational work by 10^6 for a 1000×1000 grid. Even now, the compute time scales as N^4 for N cells. This is still intractable. A significant optimisation can be made when we realise that, from the point of view of a partial derivative, a cell's SDF is not affected by the coordinates of another cell. Another way to put this is that the Jacobian and Hessian for a given cell j depend only on the attributes of that cell and all the other partial derivatives will vanish. This actually reduces the ammount to compute by a factor of N^2 because we only need to sum over the number of attributes per cell ($N_{\text{attrib}} = 5$ in our case) squared. A modern laptop can perform floating point operations in the order of tens of GFLOPS. Supposing the calculation of $I^{l,m,n}_{\alpha,\beta}$ requires 1000 floating point operations

per (m, n) pair. If we try to push to $N = 1000$ yeast cells, then we will be looking at a second compute time per time step assuming the PC used operates at 1 GFLOP. This is fine for small N but the problem doesn't scale well. In any case, with the current optimisations, we have a tractable simulaton.
It is worth noting that further optimisations can be made if we employ a spatial hash map or a AABB filter which avoids computing matrix elements between cells which are not even nearby each other. For example, if cells 3 and 22 are further than d apart where d is the pair's maximum cell diameter then we can set the matrix element $I_{\alpha,\beta}^{l,3,22} = I_{\alpha,\beta}^{l,22,3} = 0$ straight away (Note sure about this). We carry the full computation in the test phase and add the filter optimisation later.

Bibliography

Row-major and column-major array layouts. `https://au.mathworks.com/help/coder/ug/what-are-column-major-and-row-major-representation-1.html`. Accessed: 2025-05-12.

Agnew, D., Green, J., Brown, T., Simpson, M., and Binder, B. (2014). Distinguishing between mechanisms of cell aggregation using pair-correlation functions. *Journal of Theoretical Biology*, 352:16–23.

Baraff, D. and Studios, P. A. Physically based modeling.

Borzov, S. (2021). *Use of Signed Distance Functions for the Definition of Protein Cartoon Representation*. PhD thesis, Masarykova Univerzit Brno, Czech Republic.

Charney, J. G., Fjörtoft, R., and Neumann, J. v. (1950). Numerical integration of the barotropic vorticity equation. *Tellus*, 2(4):237–254.

Chavez, C. M., Groenewald, M., Hulfachor, A. B., Kpurubu, G., Huerta, R., Hittinger, C. T., and Rokas, A. (2024). The cell morphological diversity of saccharomycotina yeasts. *FEMS Yeast Research*, 24:foad055.

Crank, J. and Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical proceedings of the Cambridge philosophical society*, volume 43, pages 50–67. Cambridge University Press.

Ebrahimi, H., Siavoshi, F., Heydari, S., Sarrafnejad, A., and Saniee, P. (2020). Yeast engineered translucent cell wall to provide its endosymbiont cyanobacteria with light. *Archives of Microbiology*, 202(6):1317–1325.

FUSEK, B. P. Visualization of sdf scenes.

Ghaffarizadeh, A., Heiland, R., Friedman, S. H., Mumenthaler, S. M., and Macklin, P. (2018). Physicell: An open source physics-based cell simulator for 3-d multicellular systems. *PLoS computational biology*, 14(2):e1005991.

Li, K., Green, J. E. F., Tronnolone, H., Tam, A. K., Black, A. J., Gardner, J. M., Sundstrom, J. F., Jiranek, V., and Binder, B. J. (2024). An off-lattice discrete model to characterise filamentous yeast colony morphology. *PLOS Computational Biology*, 20(11):e1012605.

Petrovic, L., Henne, M., and Anderson, J. (2005). Volumetric methods for simulation and rendering of hair. *Pixar Animation Studios*, 2(4):1–6.

Quilez, I. (2025). 2D distance functions. `https://iquilezles.org/articles/distfunctions2d/`.

Salari, R. and Salari, R. (2017). Investigation of the best saccharomyces cerevisiae growth condition. *Electronic physician*, 9(1):3592.

Sauro, H. M., Hucka, M., Finney, A., Wellock, C., Bolouri, H., Doyle, J., and Kitano, H. (2003). Next generation simulation tools: the systems biology workbench and biospice integration. *Omics A Journal of Integrative Biology*, 7(4):355–372.

Turing, A. M. (1990). The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52:153–197.

Witkin, A. (1997). An introduction to physically based modeling: Constrained dynamics. *Robotics Institute*.