universität wien

# MASTERARBEIT / MASTER THESIS

Titel der Masterarbeit / Title of the Master Thesis

„Kinetic Theory Applied to Escherichia Coli“

verfasst von / submitted by

Else Novac, BSc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

| | |
|---|---|
| Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet: | A 066 821 |
| Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet: | Mathematik |
| Betreut von / Supervisor: | Univ.-Prof. Dr. Christian Schmeiser |

# Abstract

The fruitful interaction between kinetic theory and mathematical analysis is illustrated by modelling colonies of *Escherichia Coli*. We start with a system of differential equations, which describe the time-continuous particle model. We later turn this model into a single kinetic equation. We also simulate the grouping of cells based on the microscopic model, as well as on a physical model, when the mechanics between the cells are taken into consideration. By processing microscopic footage and using curve fitting, we estimate the doubling time, the growth speed, and the age of division. Finally, we scale the model and derive the macroscopic equations.



# Zusammenfassung

Die fruchtbare Interaktion zwischen kinetischer Theorie und mathematischer Analysis wird durch Modellierung von *Escherichia Coli* Kolonien dargestellt. Wir starten mit einem System von Differentialgleichungen, das das zeitstetige Partikelmodell beschreibt. Später werden wir dieses Modell in eine einzige kinetische Gleichung verwandeln. Außerdem simulieren wir die Gliederung der Zellen, zuerst unter Anwendung von unserem Modell und dann bezogen auf ein physikalisches Modell, das die Interaktionskräfte zwischen den Bakterien berücksichtigt. Nach der Bearbeitung einer mikroskopischen Aufnahme und mithilfe von Kurvenanpassungen berechnen wir die Verdopplungszeit, die Wachstumsgeschwindigkeit und das Trennungsalter. Schließlich skalieren wir das ursprungliche Modell und leiten wir die makroskopischen Gleichungen her.

# Contents


**1 Introduction** **1**

**2 Microscopic Model** **4**

**3 Simulations** **11**

**4 Statistical estimates** **25**

**5 Macroscopic model** **38**


# Aknowledgements


I would like to express my deep and sincere gratitude to Prof. Christian Schmeiser, for his guidance, patience, and immense knowledge. I would also like to thank Prof. Jim Haseloff from the University of Cambridge for putting me in touch with Dr. Timothy Rudge from Universidad Catolica de Chile. Despite the timezone difference, Tim helped me enormously with the simulation software. Alexandru Capat, MSc, was also kind enough to share his knowledge gained from building racing cars and we had a blast installing dependencies. Last but not least, I am immensely grateful for the love and support shown by Arkadij, my parents, and my closest friends.

# 1 Introduction

Kinetic theory studies problems that involve stochastic interactions between a large number of agents of a system through evolution equations of probability densities. It originates from mathematical modelling of liquids and gases, a simple example being the Boltzmann equation. By looking at the collisions between pairs of molecules, Boltzmann sought to model the behavior of dilute gases. Nowadays, kinetic theory is used in numerous fields, including complex systems, such as schools of fish [1], chemical reactions in solutions [2], and crowd dynamics. Asymptotics for many particles and large time scales are used for deriving the equations for macroscopic quantities (all this being based on the process used by Boltzmann in his kinetic theory of gases, resulting in the gas dynamics equations in the limit).

Mathematical modelling has been used in biology to study processes that encompass a large spectrum of scales, from individual cells to entire populations of organisms. Here, we describe a colony of *Escherichia Coli* using kinetic theory and computer simulations, while relying on the methods and results presented in [3] and [4].

In Section [2], we describe the microscopic time-continuous model. We assume that each bacterium is modelled as a rod, with a fixed width and age-dependent length. The cell grows linearly in time and divides into two identical daughter cells when it reaches the maximum length. When looking at the centers of masses and orientations, we describe the particle dynamics by taking into consideration how they overlap as they grow (and how they can avoid overlapping). This is how we come up with definitions for *badness of center of masses* and *badness of orientations* (and, implicitly, of operators of these *badnesses*). The system of ODEs highlights the repulsion in the normalized direction and the evolution of the direction over time.

In Section [3], we do computer simulations for the aforementioned model, as well as for a physical model where the mechanics between the cells are taken into consideration. In both cases, the cells tend to group themselves in the shape of a circle (see Fig. 3.3 and Fig. 3.5). In the latter case, the lengths of each cell vary as the population grows,

and the density of the bacteria is higher in the middle of the domain, as expected.

In Section [4], we use image processing techniques and the footage found in [5] to study the growth rate and estimate the doubling time of E.Coli. We wish to estimate the growth speed and age of division using curve fitting and minimizing the mean squared error, expecting the growth to slow down as it gets crowded. This is, however, not the case when comparing the two halves of the film, as there are other external factors influencing the exponential growth of the bacteria.

Finally, Section [5] goes back to the system of ODEs presented in Section [2], which is now transformed into a single kinetic equation. This approximation of the particle dynamics is done by using an empirical distribution function and a test function, differentiating with respect to time, and letting the high number of bacteria go to infinity. This model needs to be reformulated, though, because dimensional analysis, or more precisely, scaling, reduces the number of parameters in a system[1]. Computations and the method of generalized collision invariants, which we will present later in this section, lead to the macroscopic equations.

As a warm-up, we will give a short introduction to models for collisions in kinetic theory. A kinetic model for a gas is a non-negative function $f(t, x, v)$ that we define on $[0, T] \times X \times \mathbb{R}^N$, where $[0, T]$ is the time interval, $X$ is the domain in which the gas is contained, and $\mathbb{R}^N$ is the tangent space to $X$. One can regard $f$ as an estimate of the true density of the gas in phase space [6]. As an example, we can take $\rho$ the local density, $u$ the local macroscopic velocity, $\tau$ the local temperature, and at a given point $x$ and time $t$ we have $\rho = \int_{\mathbb{R}^N} f(t, x, v)dv$, $\rho u = \int_{\mathbb{R}^N} f(t, x, v)vdv$, and $\rho|u|^2 + N\rho\tau = \int_{\mathbb{R}^N} f(t, x, v)|v|^2 dv$.

If we wish to model a gas of non-interacting particles, we have constant density along the characteristics $v = dx/dt$ and $dv/dt$ (the latter being 0 by Newton's principle). Therefore, we can write $f(t, x, v) = f(0, x - vt, v)$, which is a weak solution to the equation $\partial f/\partial t + v \cdot \nabla_x f = 0$.

However, when particles do interact, a quadratic collision operator can be derived. Some assumptions are made prior to the derivation, namely that we have binary, elastic, microreversible (i.e. dynamics are reversible) collisions, that the duration of an interac-

[1] In theory, dimensional analysis gives an idea about the scaling relations even if we do not know what the governing equation is. On the other hand, scaling deals with the transformation of the variables only, and the scope is reducing the number of parameters involved, but it does not always lead to dimensionless quantities. We can view scaling as a particular case of the more general dimensional analysis.

tion is small, and that the velocities of two random particles at point $x$ that have not yet interacted are uncorrelated. The operator is defined as:

$$Q(f,f) = \int_{\mathbb{R}^N} \int_{S^{N-1}} [f(v')f(v'_*) - f(v)f(v_*)]B(\sigma, |v - v_*|)d\sigma dv_*, \tag{1.1}$$

with $\sigma \in S^{N-1}, |v - v_*| = |v' - v'_*|, v' = \frac{v + v_*}{2} + \frac{|v - v_*|}{2}\sigma, v'_* = \frac{v + v_*}{2} - \frac{|v - v_*|}{2}\sigma$. $B$ is called the collision kernel. The operator can be split into a loss term (equivalent to the number of collisions in which a particle of velocity $v$ collides with one of velocity $v_*$ and change its velocity) and a gain term (showing how a particle with velocity $v'$ acquires velocity $v$ after the interaction). The full Boltzmann equation is:

$$\partial_t f + v \cdot \nabla_x f = Q(f,f), \tag{1.2}$$

where boundary conditions, i.e. interactions between the particles and the edges of the domain, are not that important when the position space is assumed to be the entire $\mathbb{R}^N$, or the torus.

We can do the dimensional analysis on the Boltzmann equation and obtain:

$$\partial_t f^\varepsilon + \nabla_x \cdot (v f^\varepsilon) = \frac{1}{\varepsilon} Q(f^\varepsilon), \text{with} \tag{1.3}$$

$$Q(f)(v) = \int_{v \in \mathbb{R}^3} \int_{\sigma \in S^2} B(v - v_*, \sigma)(f' f'_* - f f_*)dv_* d\sigma, \tag{1.4}$$

where $\varepsilon$ is the inverse of the collision rate. A *collision invariant* for a collision operator $Q(f)$ is a function $\Psi = \Psi(v)$ such that:

$$\int_{\mathbb{R}^d} Q(f)\Psi dv = 0, \forall f.$$

This means that if $\Psi$ is constant, then it is a collision invariant. In the last section, however, we will need the concept coined as the *generalized collision invariant (GCI)*. A GCI is a function $\Psi_f$, such that:

$$\int_{\mathbb{R}^d} Q(f)\Psi_f dv = 0, \forall f.$$

# 2 Microscopic Model

We begin by describing the time-continuous particle model. The colony dwells in a two-dimensional space and consists of stiff rod-shaped bacteria. We will call into play the observations made in [7] regarding division at midcell. For simplicity, we will also assume that the diameter is fixed, similarly to the model presented in [4]. For a different morphology, where bacteria are not necessarily rod-shaped and their diameter is not constant, see [8].

Each bacterium is, therefore, characterized in the following way: The $i$-th cell ($i = 1, \dots, N(t), \text{with } N \in \mathbb{N}$) has a center position $x_i(t) \in \mathbb{R}^2$, direction angle $\varphi_i(t)$, orientation $\omega_i =: \omega(\varphi_i) = \Big(\cos(\varphi_i) \ \sin(\varphi_i)\Big)$, age $a_i(t)$ and (age-dependent) length $l_i(t)$. The diameter is set to be smaller than half of the maximum length. The bacterium grows equally in both directions, along its major axis, $l_i$ increasing linearly in time. Consequently, the growth rate is proportional to the length if there are no impediments caused by external forces or contacts between cells. When it reaches a certain length $l_d$, which we call the *division length*, the cell splits into two identical bacteria.

If a cell is by itself, it will continue to grow and divide. As stated above, the angle has to be a bit modified in the moment of division, such that we avoid obtaining a long sequence of cells, one after another. If we want to see how much a cell is above another, i.e. what happens is the cells are about to overlap, we have to look at both the position and $\varphi$. One possibility is to keep the direction angle equal to the angle of the mother cell. However, we want to avoid the formation of a long chain of bacteria, and therefore we wish to add some randomness to the angles of the daughter cells.

Let $t_i$ be the time of birth (therefore $a_i$ "starts" at a certain birth time $t_i$). Division will then take place at time $t = t_i + a_d$. Note that $a_i(t) = t - t_i$.
By looking at how age and length are related:

$$l_i = \frac{l_d}{2} + \frac{a_i}{a_d}\frac{l_d}{2}, \tag{2.1}$$

we can deduce the *growth rate* $\frac{l_d}{2a_d}$, with $0 \leq a_i \leq a_d$ and $\frac{l_d}{2} \leq l_i \leq l_d$.

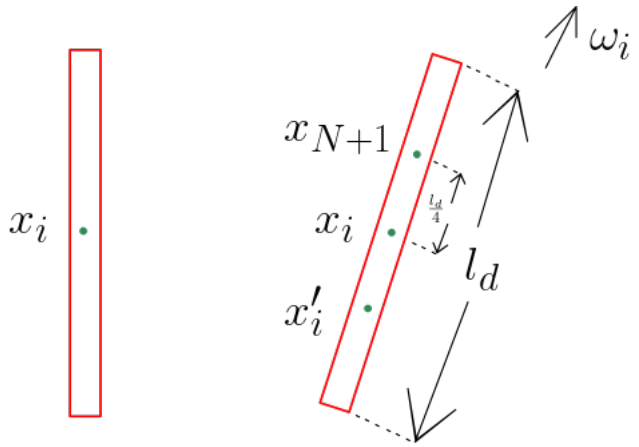



Figure 2.1: Bacterium with its center (left) and before division (right)

As seen in Figure 2.1, and setting $N+1 = N'$, we can write:

$$x_i' = x_i - \frac{l_d}{4}\omega_i, \;\; \varphi_i' = \varphi_i + \Delta\varphi_i \tag{2.2}$$

$$x_{N+1}' = x_i + \frac{l_d}{4}\omega_i, \;\; \varphi_{N+1}' = \varphi_i + \Delta'\varphi_i, \tag{2.3}$$

whereby $\Delta\varphi$ and $\Delta'\varphi$ are random numbers with some probability distribution, for example the symmetric triangular distribution. In the following, we will be disregarding the change of angle at the moment of division.
We therefore need to construct a distribution function $f(x, \varphi, a, t)$ depending on position $x \in \mathbb{R}^2$, direction angle $\varphi \in \mathbb{T}^1$, age $a \in [0, a_d]$ and time $t$. We construct:

$$f(x, \varphi, a, t) = \begin{cases} f_0(x, \varphi, a - t) & t < a \\ f_b(x, \varphi, t - a) & t > a \end{cases}$$

We have:

$$f(x, \varphi, a, 0) = f_0(x, \varphi, a)$$

and

$$f(x, \varphi, 0, t) = f_b(x, \varphi, t)$$

The only thing that is missing is $f_b$:

$$f_b(x,\varphi,t) = f\left(x - \frac{l_d}{4}\omega(\varphi),\varphi,a_d,t\right) + f\left(x + \frac{l_d}{4}\omega(\varphi),\varphi,a_d,t\right).$$

Population will spread in the domain with compact support. The total number of cells is:

$$N(t) = \int f dx d\varphi da.$$

$$\begin{aligned}\dot{N} &= -\int_{\mathbb{R}^2}\int_{\mathbb{T}^1}\int_0^{a_d} \partial_a f da d\varphi dx \\ &= -\int_{\mathbb{R}^2}\int_{\mathbb{T}^1} [f(a=a_d) - f(a=0)] d\varphi dx \\ &= -\int_{\mathbb{R}^2}\int_{\mathbb{T}^1} \left[f(a=a_d) - f\left(x - \frac{l_d}{4}\omega,\varphi,a_d\right) - f\left(x + \frac{l_d}{4}\omega,\varphi,a_d\right)\right] d\varphi dx\end{aligned} \tag{2.4}$$

We can drop the $\frac{l_d}{4}$'s since we're integrating over all $\mathbb{R}^2$ (coordinate transform in the integral). Hence:

$$\dot{N} = \int_{\mathbb{R}^2}\int_{\mathbb{T}^1} [f(a=a_d)] d\varphi dx.$$

We can now write:

$$\partial_t f + \partial_a f = \int [P(a' \to a, x' \to x, \varphi' \to \varphi) f - P(a \to a', x \to x', \varphi \to \varphi') f] = L_f,$$

with

$$P(a \to a', x \to x', \varphi \to \varphi') = \delta(a - a_d)\delta(a')\left[\delta\left(x' + \frac{l_d}{4}\omega\right) + \delta\left(x' - \frac{l_d}{4}\omega\right)\right]\delta(\varphi - \varphi')$$

$$L_f = \delta(a)\left[f\left(x + \frac{l_d}{4}\omega,\varphi,a_d\right) + f\left(x - \frac{l_d}{4}\omega,\varphi,a_d\right)\right] - \delta(a - a_d) f.$$

However, in our case, we have a model with jumps in the direction at cell division:

$$P(a \to a', x \to x', \varphi \to \varphi') = \delta(a - a_d)\delta(a')\left[\delta\left(x' + \frac{l_d}{4}\omega\right) + \delta\left(x' - \frac{l_d}{4}\omega\right)\right] M(\varphi - \varphi'),$$

where

$$L_f = \delta(a)\int \left[f\left(x + \frac{l_d}{4}\omega',\varphi',a_d\right) + f\left(x - \frac{l_d}{4}\omega',\varphi',a_d\right)\right] M(\varphi' - \varphi) d\varphi' - \delta(a - a_d) f$$

and

$$\int M d\varphi = 1.$$

Or with boundary conditions:

$$f(x, \varphi, 0) = \int \left[ f\left( x + \frac{l_d}{4}\omega', \varphi', a_d \right) + f\left( x - \frac{l_d}{4}\omega', \varphi', a_d \right) \right] M(\varphi' - \varphi) d\varphi'.$$

Assuming that we have binary interactions, we wish to find out whether the cells intersect - and if they do, we want to inspect where they overlap, i.e. how much of cell $i$ is on cell $j$. Recall that the width $w$ of the cell is fixed, smaller than half of the maximum length, and that the $l$'s will tend to 0 when we have interaction.

We will proceed in the following way: for the rest of this section, we present a model based on simplifying the shape of the bacterium and parametrizing its center. This will help us find two *badness* operators, which will be useful in Section [5]. In section [3], we will focus on the mechanics between the cells because the biophysical model will be convenient for the simulations.

Therefore, for simplicity, we sketch the bacteria in an elementary way, by just drawing a straight line through the middle point and emphasizing the direction. In this way, we disregard small overlapping areas that are negligible and we pay attention to the overlapping lengths. This will be more noticeable when deriving the kinetic equation and finding a scaling, but it is important to point out that the badness operator does not take widths into consideration. The case in which the corners of the rods meet is indeed not a problem due to the small intersection area, but a more complicated badness operator would have been required if we had wanted to investigate the case shown in the second figure below, where the axes are parallel, but the intersection area is large:

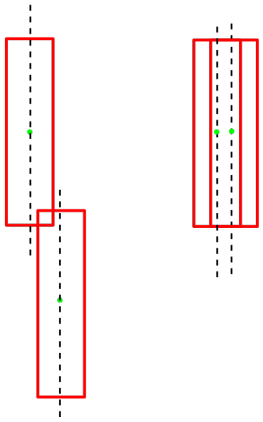


Figure 2.2: Small intersection area (left) and large intersection area (right). Based on our model, both of them will be acceptable.

We start by parametrizing the centers $x_i$ and $x_j$ as $x_i + \lambda_i\omega(\varphi_i)$ and $x_j + \lambda_j\omega(\varphi_j)$, respectively. If the parametrizations are equal, the two bacteria intersect. To determine how much and where the bacteria intersect, we define the following distance that we call *badness*:

$$B_{ij}^{\varphi} = \min\left\{\frac{l}{2} - \lambda_i, \frac{l}{2} + \lambda_i, \frac{l}{2} - \lambda_j, \frac{l}{2} + \lambda_j\right\} \tag{2.5}$$

Consequently, if $\lambda_i$ and $\lambda_j$ are both positive, the bacteria will intersect in the front half of each other. Similarly, if $\lambda_i$ and $\lambda_j$ are both negative, they intersect in the rear half of each other. If one $\lambda$ is positive and the other one is negative, the overlap takes place between the front part of a bacterium and the rear part of the other.

Unless the bacteria are parallel, we can write the $\lambda$'s explicitly in the following way:

$$\lambda_i = \frac{(x_j - x_i)\omega_j^{\perp}}{\omega_i\omega_j^{\perp}},$$

$$\lambda_j = \frac{(x_i - x_j)\omega_i^{\perp}}{\omega_j\omega_i^{\perp}} = \frac{(x_j - x_i)\omega_i^{\perp}}{\omega_i\omega_j^{\perp}}.$$

We now need a function of $B_{ij}^{\varphi}$ that becomes large when the badness is large and goes to 0 when $B_{ij}^{\varphi}$ is negative. However, $B_{ij}^{\varphi}$ is dimensionless, so when we construct this function, we also need to take into consideration how fast the bacterium switches the angle. We have:

$$\psi_{\varphi}(B_{ij}^{\varphi}) = \kappa_1 \max\{0, B_{ij}^{\varphi}\}, \tag{2.6}$$

where $\kappa_1$ is the angular velocity.

Now, we want to avoid the overlapping of centers of masses as well, because if we ever let some $x_i = x_j$, we end up with a "singularity", since the distance is 0. Once again, we need a repulsion function. In the case of centers of masses, the *badness of centers of masses* would be defined as:

$$B_{ij}^x := 1 - \frac{|x_i - x_j|}{w} \tag{2.7}$$

If it is strictly positive, there is interaction. If it is 0, the bacteria are parallel and one next to the other. Moreover, if $w > |x_i - x_j|$, the centers are so close to each other that switching the angle will not avoid overlapping. Consequently, the centers of masses will need to shift. Analogously to $\psi_\varphi(B_{ij})$, we define:

$$\psi_x(B_{ij}^x) = \kappa_2 \max\{0, B_{ij}^x\}, \tag{2.8}$$

where $\kappa_2$ is the maximal velocity.

We can now write down the system of ordinary differential equations which describes the particle dynamics:

$$\dot{x}_i = \sum_{i \neq j} \psi_x B_{ij}^x \frac{x_i - x_j}{|x_i - x_j|} \tag{2.9}$$

$$\dot{\varphi}_i = \sum_{i \neq j} \psi_\varphi B_{ij}^\varphi \operatorname{sgn}(\sin 2(\varphi_i - \varphi_j)) \tag{2.10}$$

$$\dot{a}_i = 1 \tag{2.11}$$

In the first equation there is no active part since the bacteria have no self-propelling speed. However, the interaction term describes the repulsion in the normalized direction and, as mentioned before, there are cases when the centers of masses need to shift in order to avoid overlapping.

The second equation shows the evolution of the direction in time. The factor $k_1$ represents the maximal angular velocity. There is only one thing that we still need in addition to the explanations above regarding the badness, i.e. the direction towards which the badness operates. Hence, we take the difference between two angles, $\varphi_i$ and $\varphi_j$, and we look at the sign of the product between its sine and cosine. If this product is negative, $\varphi_i$ increases, and the bacteria are anti-parallel. If the product is positive,

the bacteria are parallel. Using that $\sin 2(\varphi_i - \varphi_j) = 2\sin(\varphi_i - \varphi_j)\cos(\varphi_i - \varphi_j)$, the expression above follows.

We will come back to this system of ODEs when computing the kinetic equation. The next two sections will mainly consist of simulations, statistical estimates, as well as the relevant algorithms and code.

# 3 Simulations

This section is organized in the following way: firstly, we briefly explain how the program for the simulations is built. We proceed by simulating the conditions of the model proposed in the first section. We then extend this configuration by taking the mechanics between the cells into consideration. The entire code for all the functions and `OpenCL` kernels which do the "backstage" computations will not be provided, since we concentrate on the distinctiveness of the results as we adjust our model.

We used a Python-based framework called CellModeller ( [9], [10]). When modelling large-scale cellular systems, we wish to create a system to simulate these models in populations of growing and dividing cells. There are two important objects: the `CellStates` and the `Simulator`. The first one matches the ID of a bacterium with its properties, such as the size, color, initial position and growth rate. The latter activates the simulation when it is initialized by a model. We will be working with the `CLBacterium` module, but other modules that are focused on chemical dynamics, diffusable signals, and intermixed colonies of various species of cells are available.

Using the Graphic User Interface feature, we can start with a cell that already has the maximum length (i.e, we basically start with two cells), set a growth rate and set the properties of the daughter cells. In our case, we keep the same division length as the mother cell, but in a more realistic experiment it is possible to slightly alter this by adding a sample from a uniform distribution over a small interval. It is also seemingly important to note that, due to the way the graphics are set up, the bacteria appear this time as capsules, just like in You's paper [4]. This will be relevant when simulating the version where forces between the cells play a role.

There are four functions that are necessary for running a simple simulation: `Setup` prepares the modules that are going to be used and adds cells with the chosen initial position and orientation parameters; `Init` establishes the properties of the cell; `Update` loops through the cells and regulates their change over time; finally, `Divide` is used at every cell division and incorporates the properties of the daughter cells.

As the simulation runs, we see how the cells multiply over time, whether they get to touch each other, and whether there is any minor overlapping.

The following is an example of a code for simple cell growth:

```
# Import necessary modules
import random
from CellModeller.Regulation.ModuleRegulator import ModuleRegulator
from CellModeller.Biophysics.BacterialModels.CLBacterium import CLBacterium
from CellModeller.GUI import Renderers
import numpy
import math

def setup(sim):
    # We work with the biophysics module in 2D
    biophys = CLBacterium(sim, jitter_z=False)
    # Other module requirements
    regul = ModuleRegulator(sim, sim.moduleName)
    sim.init(biophys, regul, None, None)
    # Short, green, rapidly-growing cell
    # with an intial position on the x-axis and orientation
    sim.addCell(cellType=0, pos=(0,0,0), dir=(1,0,0))
    # Create visual outputs
    therenderer = Renderers.GLBacteriumRenderer(sim)
    sim.addRenderer(therenderer)
    # Every 10 simulation steps the .pickle file output is saved
    sim.pickleSteps = 10

def init(cell):
    # Maximum cell length
    cell.targetVol = 3
    # Growth rate
    cell.growthRate = 0.7
    cell.color = (0.0,1.0,0.0)
```

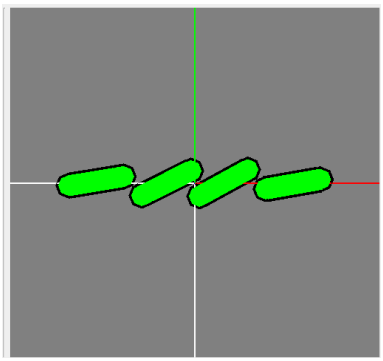
```
def update(cells):
    # Mark the cells that have reached the maximum length
    for (id, cell) in cells.items():
        if cell.volume > cell.targetVol:
            cell.divideFlag = True

def divide(parent, d1, d2):
    # Specify target cell size that triggers cell division
    d1.targetVol = 3
    d2.targetVol = 3
```

The output (in this case, after approximately 30 minutes) looks like this:

```
2 minute(s) or 1911.963745 second(s)
 13830     2 cells       1 cts       0 iterations  residual = 0.000011
   13840           2 cells           1 contacts    0.531429 hour(s) or 31.88573
4 minute(s) or 1913.144043 second(s)
 13840     2 cells       1 cts       0 iterations  residual = 0.000018
   13850           2 cells           1 contacts    0.531758 hour(s) or 31.90547
7 minute(s) or 1914.328613 second(s)
 13850     2 cells       1 cts       0 iterations  residual = 0.000025
   13860           2 cells           1 contacts    0.532096 hour(s) or 31.92576
5 minute(s) or 1915.545898 second(s)
 13860     2 cells       1 cts       0 iterations  residual = 0.000035
   13870           2 cells           1 contacts    0.532426 hour(s) or 31.94558
1 minute(s) or 1916.734863 second(s)
 13870     2 cells       1 cts       0 iterations  residual = 0.000045
   13880           2 cells           1 contacts    0.532813 hour(s) or 31.96880
3 minute(s) or 1918.128174 second(s)
 13880     2 cells       1 cts       0 iterations  residual = 0.000003
   13890           2 cells           1 contacts    0.533233 hour(s) or 31.99396
8 minute(s) or 1919.638062 second(s)
 13890     2 cells       1 cts       0 iterations  residual = 0.000007
   13900           4 cells           3 contacts    0.533663 hour(s) or 32.01976
5 minute(s) or 1921.185913 second(s)
 13900     4 cells       3 cts       0 iterations  residual = 0.000035
   13910           4 cells           3 contacts    0.534128 hour(s) or 32.04767
5 minute(s) or 1922.860474 second(s)
 13910     4 cells       3 cts       0 iterations  residual = 0.000036
```

(a) Batch mode simulation



(b) Graphic vizualization

Figure 3.1: Results after approximately 30 minutes

According to our initial model, the growth rate of a cell is proportional to its length, and it is not influenced by any inter-cellular forces or viscous drag from the environment. If we have friction on small particles and we zoom out, we get viscosity, which means that the higher the friction is, the more viscous the substrate becomes. If we let the drag coefficient go to infinity, the cells will be glued to the outside. Therefore, we will let both the drag coefficient and the coefficient of friction between the cells go to 0. The figure below, created with an extension Ipe module in Python, shows the results of tracking the cells' direction after the first few divisions. We can observe parallel lines in the middle

and cells migrating towards the boundary.

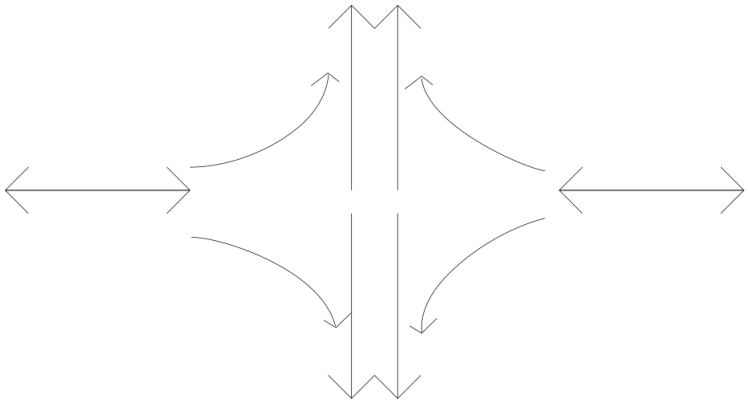


Figure 3.2: Simple pattern of cell directions

However, We expect a larger colony (2500 cells) to be spread circularly in the environment, with a lower density towards the edges. We observe that, although they are not as tidily arranged as in the simple scheme above, the bacteria do form clusters of parallel cells.

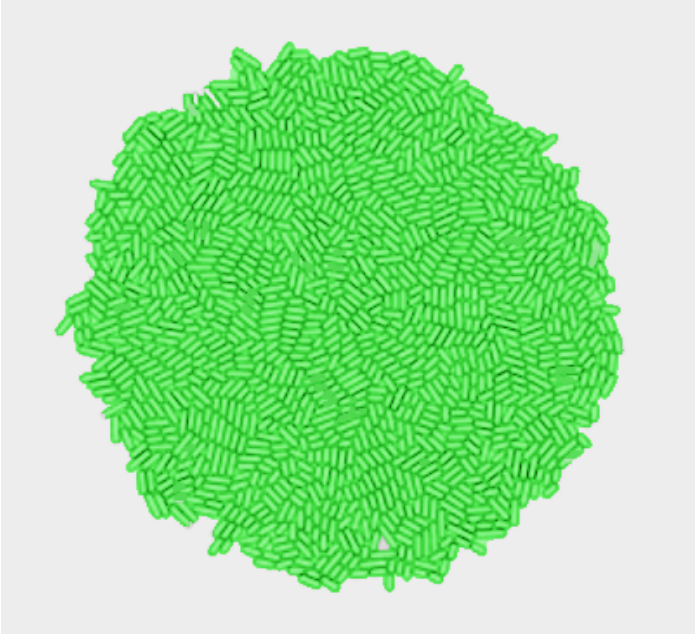


Figure 3.3: Simulation of a larger colony

As mentioned before, we are going to take a look at the situations when the mechanics

do play a role. For this, we need to look at how a point on the surface of a cell moves: the center of mass moves linearly with velocity $\vec{v}$, but rotations can also happen, in which case we introduce the angular velocity $\vec{w}$. Obviously, the growth rate, that we denote by $v_g$, is also a velocity that shifts all the points away from the center. This shifting takes place only along the axis, which we denote by $\vec{x}$. This velocity is "maximal" at the extremities, each of which moving further away from the center as fast as half of the growth rate.

We fix a point on the cell of length $l_i$ and we wish to calculate its velocity, given that the point is placed at a position $r$ relative to the cell's center:

$$\vec{v_r} = \vec{v} + \vec{w} \times r + \frac{1}{2}\frac{\vec{x} \cdot r}{l_i}\vec{x}v_g = \vec{v} - r \times \vec{w} + \frac{1}{2}\frac{\vec{x} \cdot r}{l_i}\vec{x}v_g \tag{3.1}$$

Since we characterize every bacterium by its position of the center $x$, orientation $\omega$ and length $l$, we can construct generalized coordinates by defining the generalized position $\hat{x} = \begin{pmatrix} x & \omega & l \end{pmatrix}^T$. Taking its time derivative, we obtain the generalized velocity $\hat{v} = d\hat{x}/dt = \begin{pmatrix} \vec{v} & \vec{w} & v_g \end{pmatrix}^T$. Analogously, we can define the generalized momentum as $\hat{p} = \begin{pmatrix} \vec{p} & \vec{L} & g \end{pmatrix}^T$ consisting of linear, angular and growth components. If a generalized impulse is applied to the bacterium, the force caused by viscous drag (this time with a non-zero coefficient $\mu$ per unit length) acts on every point of its surface. Note that $\mu$ is much bigger than the mass of the bacterium and we assume no rotational inertia. On a differential length $dy$, the viscous drag force is:

$$dF = -\mu dy \vec{v_r}. \tag{3.2}$$

The rotational force applied at a differential section when the cell is at position $r = y\vec{x}$ is:

$$d\tau = r \times dF. \tag{3.3}$$

The growth impulse is repelled by the drag forces along the axis:

$$F_g = \mu\frac{1}{2}\left[\int_0^{\frac{l_i}{2}} \frac{y}{l_i}v_g dy - \int_{-\frac{l_i}{2}}^0 \frac{y}{l_i}v_g dy\right]. \tag{3.4}$$

To find the net force, we need to firstly find the changes in the generalized position by combining two equations (3.1) and (3.2) and using Newton's laws. Similarly, we apply (3.1) to equation (3.3) and note that the rotational force is not influenced by the drag

force caused by linear velocity, nor by the drag force caused by growth. Finally, for the growth impulse, we take into considerations the cell's own resistance as it grows, in which case it makes sense to add an extra drag term to the $F_g$ in equation (3.4). For detailed computations, see [11]. We find the change in a generalized position:

$$\Delta\hat{x}_i = \begin{bmatrix} \text{diag}(\frac{1}{\mu l_i}) & 0 & 0 \\ 0 & \frac{12P_i}{\mu l_i^3} & 0 \\ 0 & 0 & \frac{1}{\alpha} \end{bmatrix} \begin{bmatrix} \Delta\vec{p_i} \\ \Delta\vec{L_i} \\ \Delta g_i \end{bmatrix}, \tag{3.5}$$

where $P$ is the projection matrix onto the plane orthogonal on the axis of the cell and $\alpha$ is the cell growth drag. This *inverse drag matrix* is applied in the simulations to a vector whose length is equal to the given number of cells and whose only first 7 elements are used (3 center + 3 rotational + 1 length/growth).

We will also be interested in how much the cells managed to overlap. This is given by the minimum and maximum of the residual overlaps between the bacteria. For this, we need the physics solver, which is integrated in the simulator. The theory behind it is the following: two cells overlap and they form a contact. The change in the overlap distance is the difference in the movement of the contact points on each cell, along the contact normal, whereby these distances are adjusted by a set of applied impulses. Ideally, we seek a physically realistic system where cells avoid overlapping, which is equivalent to seeking an impulse applied to every cell that results in a change in the overlap distance, such that this distance is null for every contact. It is possible to form a system of constraints, which might not have a unique solution (for example, consider the situation where the bacteria move straight-up to the normal of their contact). For further details see [11].

We are now ready to analyze some interesting properties of the cells under the above conditions. For this, we switch from the batch mode to an IPython Notebook for better vizualization of the output. Although the original intention was to simulate a model that is similar to the animation used in the next section, with a similar doubling time, it is computationally very expensive to run the simulations for hours, since the calculations are saved every second. Therefore, we did the following: we set the growth rate such that the cells already double after almost only 21 seconds and set the maximum population at 500 cells.

```
import numpy as np
import CellModeller
from CellModeller.Simulator import Simulator
model = '/Users/ElseNovac/cellmodeller/Examples/Sim1.py'
sim = Simulator(model, 0.025, clPlatformNum=0, clDeviceNum=0, saveOutput=True)
# Set the maximum population at 500 cells
while len(sim.cellStates)<500:
    sim.step()
```

```
      10          2 cells          1 contacts    0.002581 hour
(s) or 0.154838 minute(s) or 9.290297 second(s)
   10     2 cells       1 cts       1 iterations  residual = 0.0
00000
      20          2 cells          1 contacts    0.003132 hour
(s) or 0.187913 minute(s) or 11.274765 second(s)
   20     2 cells       1 cts       1 iterations  residual = 0.0
00000
      30          2 cells          1 contacts    0.003581 hour
(s) or 0.214862 minute(s) or 12.891733 second(s)
   30     2 cells       1 cts       1 iterations  residual = 0.0
00000
      40          2 cells          1 contacts    0.003925 hour
(s) or 0.235504 minute(s) or 14.130247 second(s)
   40     2 cells       1 cts       1 iterations  residual = 0.0
00000
```

(a) Initialization

```
00000
      80          2 cells          1 contacts    0.005300 hour
(s) or 0.318003 minute(s) or 19.080198 second(s)
   80     2 cells       1 cts       1 iterations  residual = 0.0
00000
      90          4 cells          3 contacts    0.005806 hour
(s) or 0.348372 minute(s) or 20.902338 second(s)
   90     4 cells       3 cts       1 iterations  residual = 0.0
00070
     100          4 cells          3 contacts    0.006203 hour
(s) or 0.372164 minute(s) or 22.329832 second(s)
  100     4 cells       3 cts       1 iterations  residual = 0.0
00031
     110          4 cells          3 contacts    0.006519 hour
(s) or 0.391120 minute(s) or 23.467199 second(s)
  110     4 cells       3 cts       1 iterations  residual = 0.0
00005
     120          4 cells          3 contacts    0.006859 hour
(s) or 0.411521 minute(s) or 24.691248 second(s)
  120     4 cells       3 cts       1 iterations  residual = 0.0
00006
```

(b) Doubling

Figure 3.4: Results when the maximum cell number is set to 500

We can make some suitable data arrays in order to plot the position and orientation of each cell, after fetching the data from the simulation or from the saved output as

described in the previous code, using the `.pickle` files. We will notice clusters of parallel cells usually have the same orientation, and we expect a lower density closer to the boundary.

The code below, although essential for the graphs we will plot later, is nothing more than the simple syntax for building matrices and then plotting the result. The only, perhaps, interesting element is adding a "2D field" of arrows using `matplotlib.pyplot.quiver`, with arrow locations and arrow directions as "arguments". Moreover, some of the plots we will present in this chapter will be made bigger in size, so that the relevant pixels are more noticeable.

```
# Extract the data
print sim.phys.n_cells
print sim.stepNum
cs = sim.cellStates
import cPickle
data = cPickle.load(open('/step-00500.pickle','rb'))
cs = data['cellStates']
sim.loadFromPickle(data)

# Build matrices from cell properties
lengths = np.array([cell.length for (id,cell) in cs.iteritems()])
pos = np.array([cell.pos for (id,cell) in cs.iteritems()])
norm = np.array([cell.dir for (id,cell) in cs.iteritems()])

# Plot the dots with arrows
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
plt.figure(figsize=(8,8))
plt.plot(pos[:,0], pos[:,1], 'o')
plt.quiver(pos[:,0], pos[:,1], norm[:,0], norm[:,1])
```

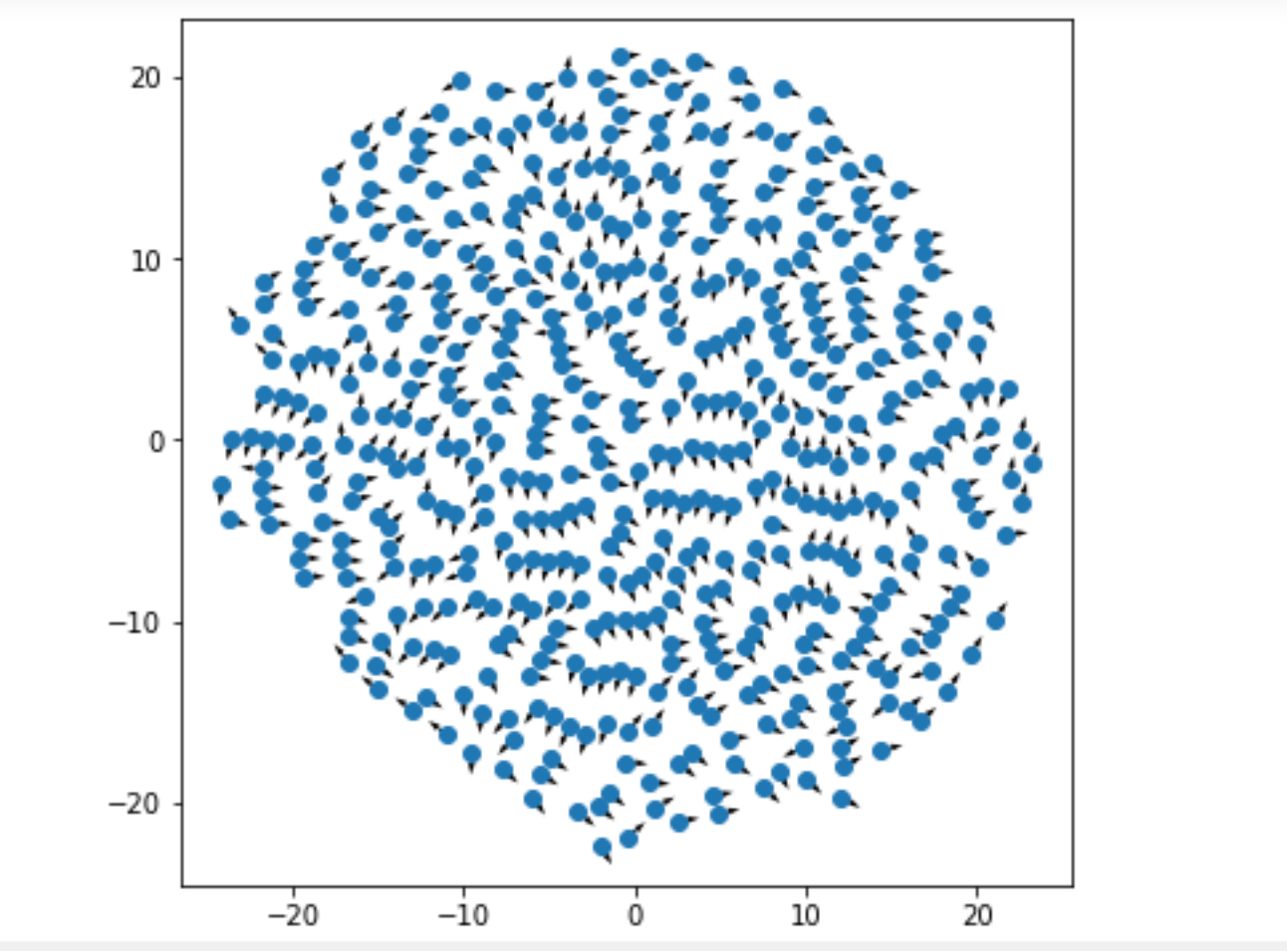



Figure 3.5: Cell positions and orientations

We can now plot histograms for some cell properties, such as length and distance from the origin:

```
plt.figure(figsize=(5,3.5))
plt.title('Cell length')
plt.hist(lengths, edgecolor='black', color='gray')
plt.xlabel('um')
plt.ylabel('Count (N=%d)'%(len(cs)))
plt.figure()
plt.title('Position')
plt.hist(np.sqrt(pos[:,0]**2+pos[:,1]**2))
```

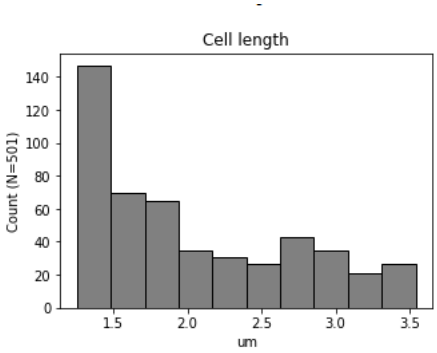



Figure 3.6: Histogram for cell lengths

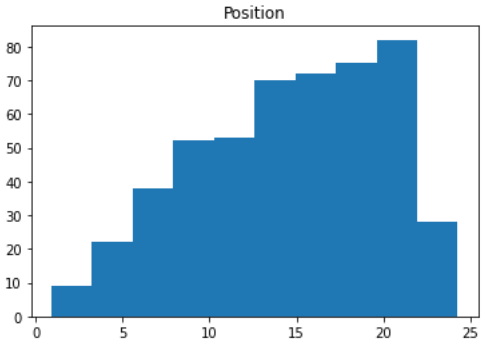



Figure 3.7: Histogram for positions

We can now create a heat map for the colony density. We need the above graphs and the rest of the code is just rendering the color map and the axes. The output is a picture whose pixels are shown as squares of multiple pixels, due to using the default `interpolation='nearest'`.

```
fig = plt.figure(figsize=(4,4))
plt.title('Colony density')
H, xedges, yedges = np.histogram2d(pos[:,0], pos[:,1],
                    weights=lengths, bins=16)
plt.imshow(H, interpolation='nearest', origin='low',
           extent=[xedges[0], xedges[-1],
           yedges[0], yedges[-1]], cmap='jet')
plt.colorbar()
```

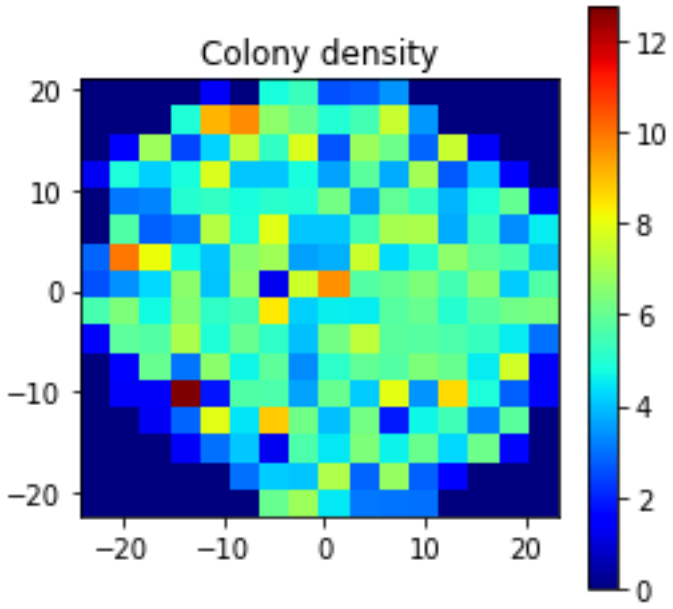



Figure 3.8: Density of population on the plane

```
phys = sim.phys
phys.find_contacts()
# Overlap per cell
d = 0.5*phys.ct_dists_dev[0:phys.n_cells,:].get()
print np.min(d.ravel()), np.max(d.ravel())
      -0.0899241 0.017062306
```

As in the simpler example at the beginning of the section, we see clusters of parallel cells, as well as a lower density towards the boundary. The lengths are not equal as a result of the external and inter-cellular forces.

We applied the same procedure to the case when cells supposedly grow as fast as before, but the maximum population is 100 cells. Although this time it took 35 seconds to get from 2 cells to 4 cells, the idea is that we expect to have a higher population density in the center of the domain, since the cells no longer need to migrate to the edges because of lack of space. Although smaller clusters of parallel cells are still observable, the pattern appears more chaotic and not symmetric like in the simple tracking sketch we presented earlier in this section. Below there are the output plots:

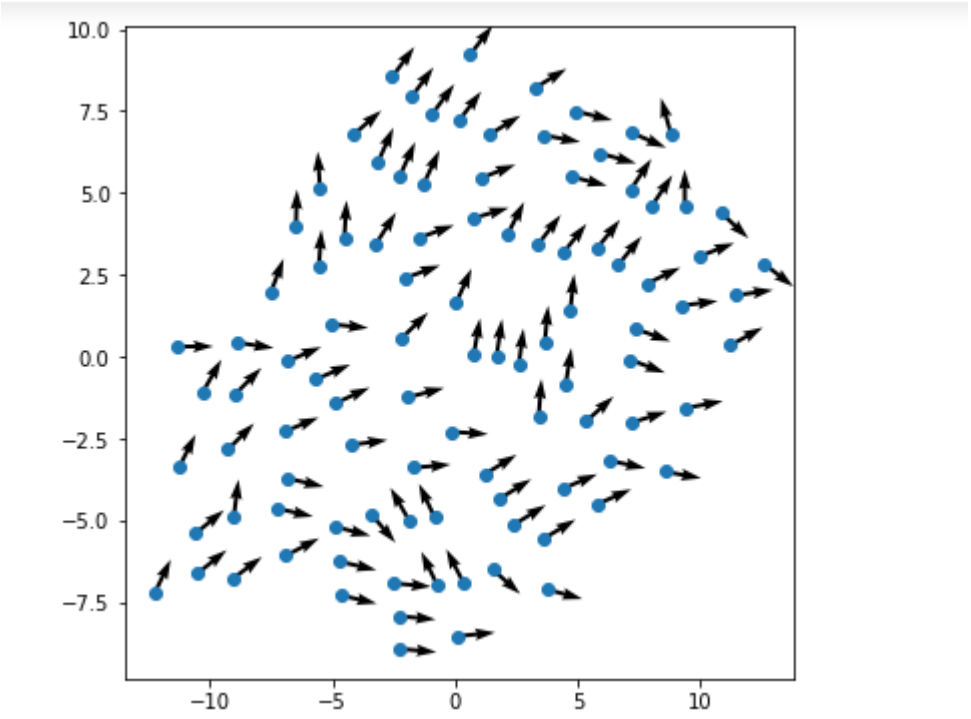



Figure 3.9: Cell positions and orientations

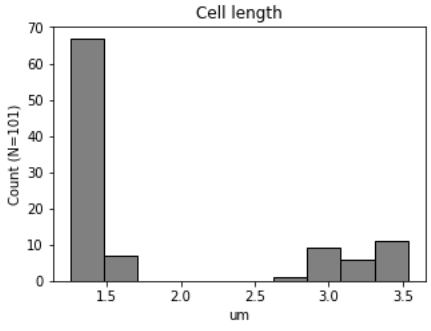



Figure 3.10: Histogram for length

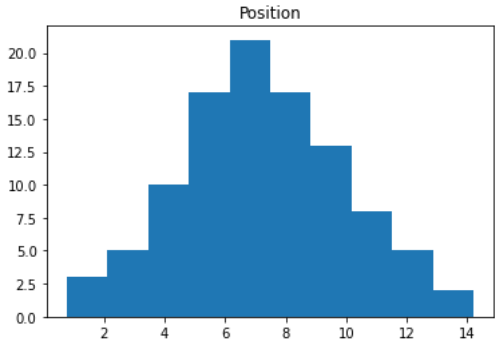



Figure 3.11: Histogram for positions

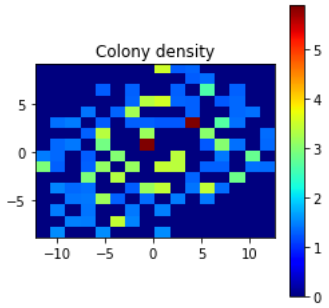



Figure 3.12: Density population on the plane

Finally, we simulate a system where the maximum number of cells is 2500. Although it is hard to distinguish the orientations, we will use this experiment to count the number of neighbors that are sisters. The output will be given by 2500 lists, each of which containing the IDs of the neighbors that are sisters (if this is not the case, then the list is empty). We give a short sample of the output, an improvement of this feature being a way of coloring the related cells and tracking how far away from each other they can travel as the population grows.

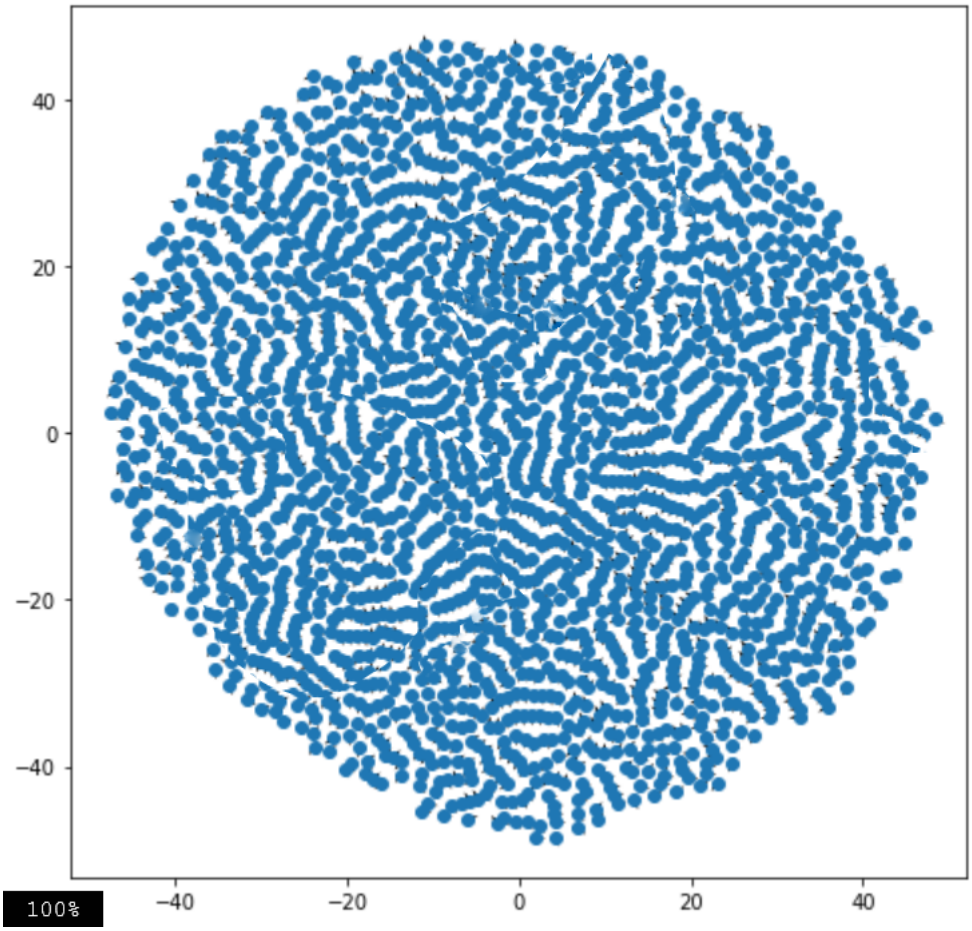


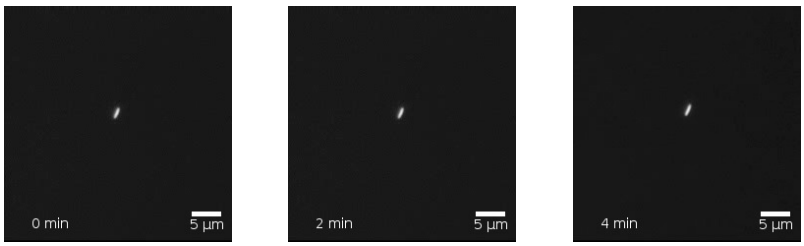

Figure 3.13: Cell positions and orientations

```
count = 0
phys.computeNeighbours = True
for (id,cell) in cs.iteritems():
    sim.phys.updateCellState(cell)
    for i in cell.neighbours:
        nbr = cs[i]
        if lineage[id]==lineage[i]:
            count += 1
[], [2345, 4296, 2986], [], [] ... [], [1730, 2738, 2902, 5556, 4484, 2864, 2134] ...
```

# 4 Statistical estimates

In this section, we bring out some image processing techniques, which are practical if we want to examine the growth of a microcolony. We start with a time-lapse video of an E.coli colony that grows on a microscope sheet of glass[1] [5]. We wish to estimate the doubling time of E.Coli by analyzing the objects in each frame and fitting a parameter to experimental data.

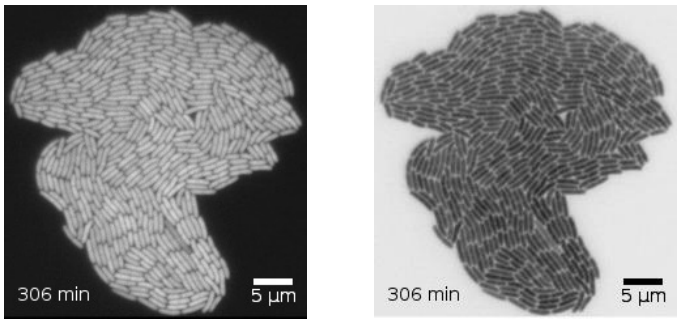
We divide the movie into 154 frames, spanning 306 minutes, with a 2-minute-step. Although the images are already grayscale, we observed that by inverting the colors we would identify the objects more easily. This was done with Python Imaging Library (PIL) using the ImageOps module.




(a) Frame 1 (b) Frame 2 (c) Frame 3

Figure 4.1: The first three frames

[1]The animation used in this work is not the one in the cited paper. The original one was initially divided into only 114 frames, with images taken either every 4 minutes or 2 minutes, which was too inconvenient when transforming it to a homogeneous data set.




(a) Raw (b) Negative

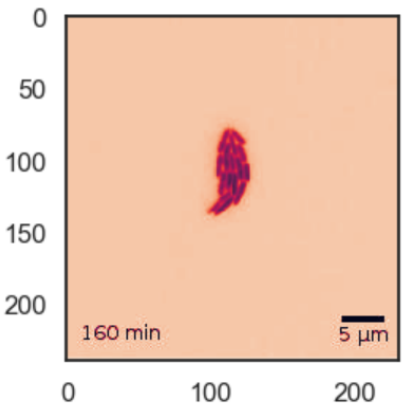
Figure 4.2: The last frame: original and inverted

Now, we are going to start our analysis from the 80th frame instead of the first one, since we have a large amount of data and the population of cells is more visible and easier to segment after some time passed. We will load the first image: it appears as a matrix whose entries are positive integers and represent the pixels. Each pixel ranges between 0 and $2^8 - 1$. The image is then printed and the next steps consist of identifying the cluster of pixels in the middle.

```
import numpy as np # numerics library
import glob # parsing the files
import skimage.io # image processing
import skimage.filters
import matplotlib.pyplot as plt # plotting
import seaborn as sns # statistical data visualization
from PIL import Image, ImageOps # module for image processing operations
sns.set(style="darkgrid") # setting up figure design
sns.set_context('talk')
files = glob.glob('C:\\ElseNovac\\frame_*_delay-0.1s-n.jpg')
files2=files[80:] # start from the 80th frame
im = skimage.io.imread(files2[0]) # the first image as an array
```

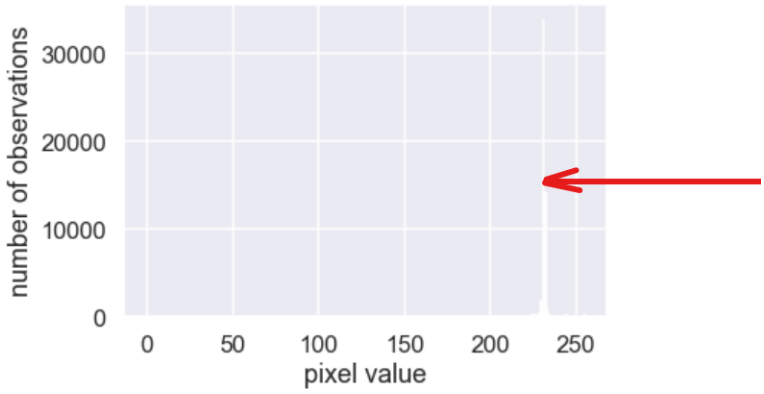
```
array([[231, 231, 231, ..., 232, 232, 232],
       [231, 231, 231, ..., 232, 232, 232],
       [231, 231, 231, ..., 232, 232, 232],
       ...,
       [234, 234, 234, ..., 234, 234, 234],
       [245, 245, 245, ..., 247, 247, 247],
       [255, 255, 255, ..., 255, 255, 255]], dtype=uint8)
```

```
with sns.axes_style('white'):
    plt.imshow(im) # plot the first image
```

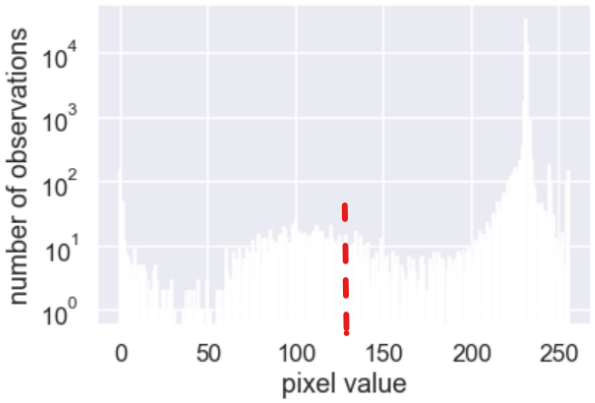



We notice that most of the pixels in our image are given by the background. Therefore, we are going to plot the histogram in log scale as well. The reason why the plot bars are white and harder to tell apart is the contrast level of the picture. This can also be changed in Python, but for our purposes it is enough to just observe the peaks of the histograms.

```
counts, bins = skimage.exposure.histogram(im)
plt.bar(bins, counts) # bar plot
plt.xlabel('pixel value') # adding labels
plt.ylabel('number of observations')
```

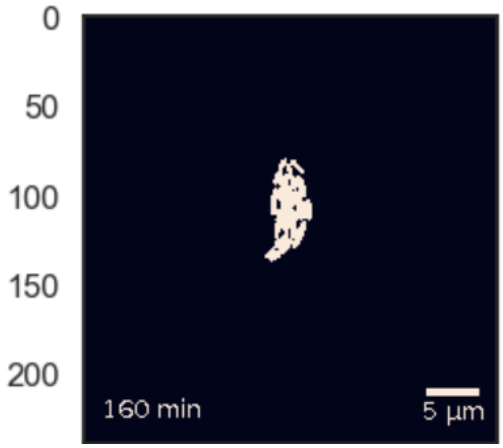



```
plt.bar(bins, counts)
plt.yscale('log') # histogram in log scale
plt.xlabel('pixel value') # adding labels
plt.ylabel('number of observations')
```




Intuitively, we can set a threshold around 130, so that we separate the cells from the background. The output is going to be a new picture, whose pixels greater than 130 are marked as $False$, and $True$ otherwise (boolean).

```
im_thresh = im <= 130
with sns.axes_style('white'):
    plt.imshow(im_thresh) # plot the thresholded image
```



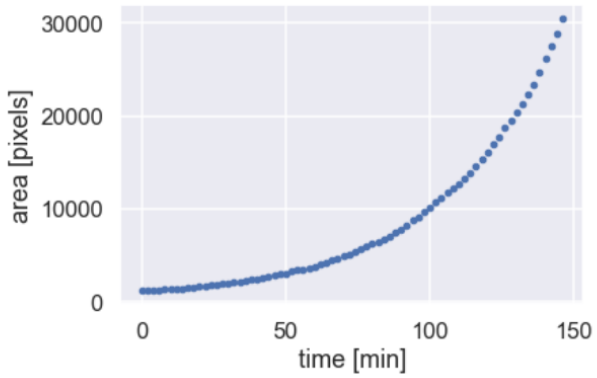

It is now fairly easy to measure the white sector of the image that resembles the original one. Next, we want to loop through the entire data and store the cell area for each phase. For this, we need a function that automatically computes the area (in pixels).

```
area = np.sum(im_thresh)
print('The bacterium is ' + str(area) + ' pixels.')
      The bacterium is 1154 pixels.
def find_area(image, threshold_value):
    im_thresh = image <= threshold_value # apply the threshold
    cell_area = np.sum(im_thresh) # compute the area
    return cell_area
cell_area = find_area(im, 130) # test out the function
print(cell_area)
```

We wish to plot the cell area against time. For that we need an empty vector where we could store all the computed areas as we loop through the images, and a time vector. The latter has length equal to the length of the data list we are looping through. Multiplication by 2 is necessary since the time step between the frames is 2 minutes.

```
# Time vector
time_range = np.arange(0, len(files2)) * 2
# Plot the area vs time
plt.plot(time_range, cell_area, '.')
#plt.yscale('log')
```

```
# Labelling
plt.xlabel('time [min]')
plt.ylabel('area [pixels]')
```



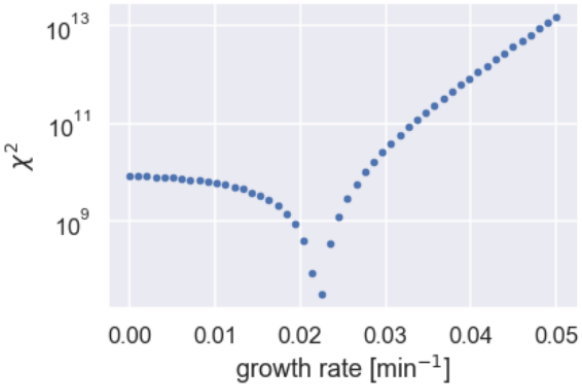

If we were to draw a trend line across these data points, we would be interested in minimizing the residuals, i.e. we would want to find out if there is any statistically significant difference between the observed and the expected bacteria areas at time $t$. We write the $\chi^2$-test as the sum over the residuals:

$$\chi^2 = \sum_{k=0}^{N} (S_k(t) - S_E(t))^2 \tag{4.1}$$

where $S_E$ is the expected area and $N$ is the number of data points. We choose to plot this $\chi^2$-test as a function of the growth rate, whose range we need to set. As above, we build an empty vector where we collect the $\chi^2$ values and just use the definition (4.1) while looping through all the possible growth rates.

```
# Range of growth rates
g_range = np.linspace(0, 0.05, 50)
# Empty vector to store the chi squared value
chi_sq = np.zeros(len(g_range))
# Iteration through growth rates
for i in range(len(g_range)):
    # Compute the expected value at each time point.
    expect = cell_area[0] * np.exp(g_range[i] * time_range)
```

```
    # Compute chi squared
    chi_sq[i] = np.sum((cell_area - expect)**2)
# Plot chi squared test
plt.plot(g_range, chi_sq, '.')
plt.yscale('log')
# Labelling
plt.xlabel('growth rate [min$^{-1}$]')
_ = plt.ylabel('$\chi^2$')
```



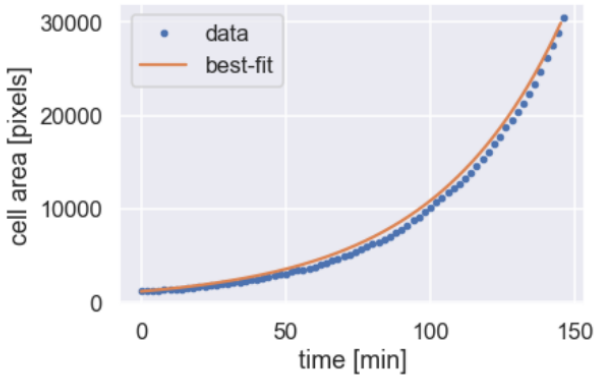

We can see that the minimum is around 0.02, and we can even determine it precisely. All we need to do now is to plot the best-fit line and compute the doubling time. We observe that the growth is exponential and that the growth rate $g$ is given by the minimum in the plot above.

Therefore, we can compute the doubling time as:

$$t = \frac{\ln 2}{g} \approx 30 \text{ mins}$$

```
# Find the minimum of chi squared
min_ind = np.argmin(chi_sq)
best_g = g_range[min_ind]
print('The index at the minimum is ' + str(best_g) + ' min^-1.')
    The index at the minimum is 0.022448979591836737 min^-1.
```

```
# Computing the best fit
t = np.linspace(0, 145, 500)
S_t = cell_area[0] * np.exp(best_g * t)
# Plotting
plt.plot(time_range, cell_area, '.', label='data')
plt.plot(t, S_t, '-', label='best-fit')
# Labelling
plt.xlabel('time [min]')
plt.ylabel('cell area [pixels]')
_ = plt.legend()
```




According to different papers ( [12], [13]), the generation time for E.Coli varies depending on temperature, nutrient levels, pH, and in real life every population of bacteria is heterogeneous and we can only estimate a mean doubling time. In general, the doubling time for laboratory-grown E.Coli is 20 minutes, but based on [13], anything below 30 minutes can be insufficient even in an enhanced culture medium that contains all the growth requirements.

Now, suppose we start at time t = 0 with one cell of length $l_d/2$ and use the assumptions from the microscopic model. Then the total length $L(t)$ of all cells satisfies:

$$L(t) = 2^{k-1} l_d \left(1 - k + \frac{t}{a_d}\right) \text{ for } k a_d \leq t \leq (k+1) a_d. \tag{4.2}$$

This piecewise linear curve interpolates the values

$$L(k a_d) = 2^{k-1} l_d, \tag{4.3}$$

and therefore, for $a_d \ll t$, i.e. after many divisions,

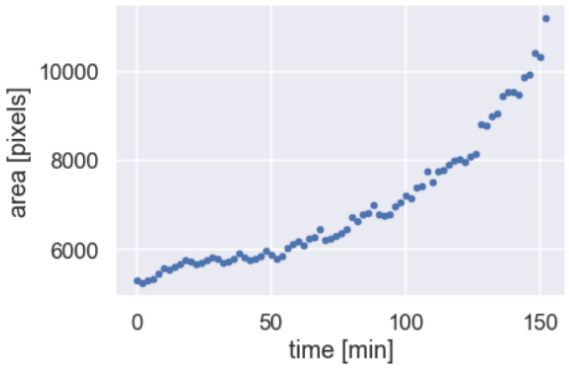
$$L(t) \approx \frac{l_d}{2} 2^{t/a_d} = s a_d 2^{t/a_d}, \tag{4.4}$$

where we have used the growth speed $s := \frac{l_d}{2a_d}$ and $\approx$ means *small relative error*. In the image analysis that has been done above, pixels are counted, which means that the total area covered by bacteria is measured. With the width $w$, it is given by:

$$A(t) = w s a_d 2^{t/a_d}. \tag{4.5}$$

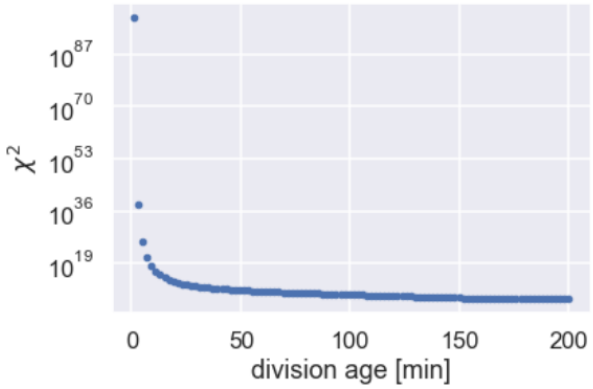
Assuming $w$ to be given, estimates for $s$ and $a_d$ can be obtained by fitting this formula to the measured results. This can be done by minimizing the mean square error. Therefore, we would estimate the parameters using the first half of the movie, and then check if the obtained curve is also a good fit for the second part of the movie. We expect the growth of the bacteria to slow down when it gets crowded.

However, when we take the first half of the frames and apply the same algorithms as above, we observe that the growth is not exactly exponential and the rest of the computations also look peculiar:

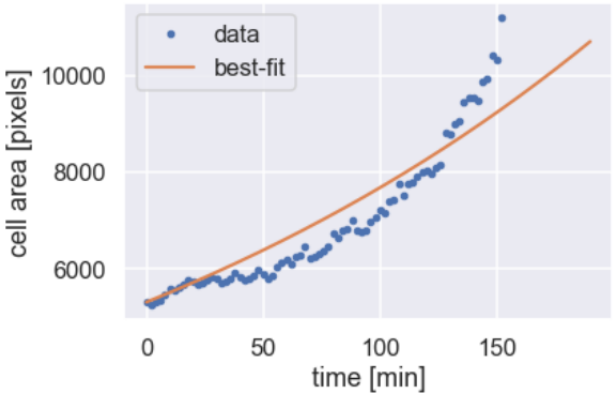



```
a_range = np.linspace(1, 200, 100)
chi_sq = np.zeros(len(a_range))
# Iteration through division ages
for i in range(len(a_range)):
    # Compute the expected value at each time point.
    expect = cell_area[0] * (2**(time_range/a_range[i]))
    chi_sq[i] = np.sum((cell_area - expect)**2)
```

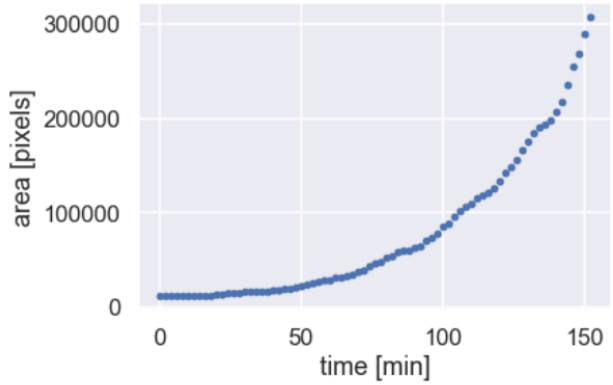
```
plt.plot(a_range, chi_sq, '.')
plt.yscale('log')
plt.xlabel('division age [min]')
_ = plt.ylabel('$\chi^2$')
```



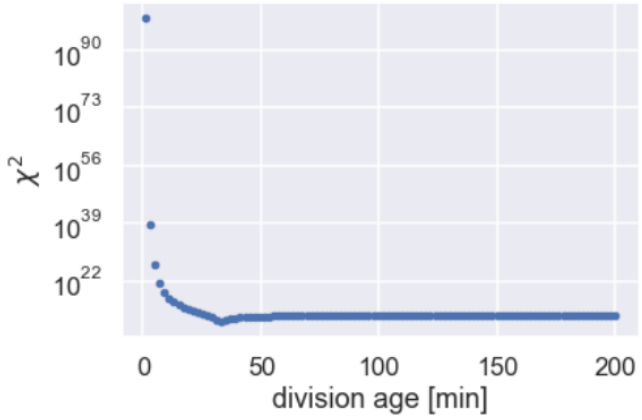

```
min_ind = np.argmin(chi_sq)
best_a = a_range[min_ind]
print('The index at the minimum is ' + str(best_a) + ' min.')
    The index at the minimum is 187.93939393939394 min.
t = np.linspace(0, 190, 500)
S_t = cell_area[0] * (2**(t/best_a))
plt.plot(time_range, cell_area, '.', label='data')
plt.plot(t, S_t, '-', label='best-fit')
plt.xlabel('time [min]')
plt.ylabel('cell area [pixels]')
_ = plt.legend()
```



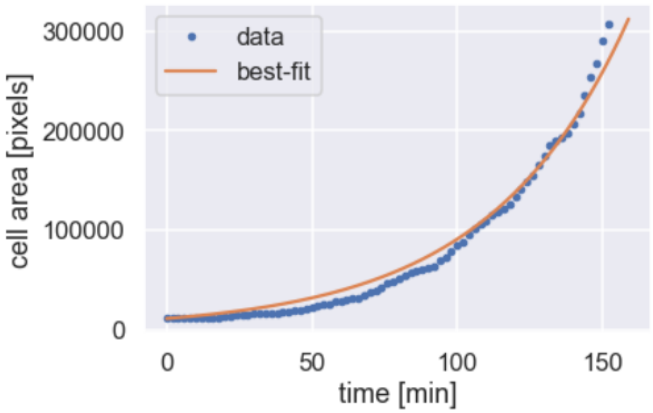

Similarly, for the second half of frames, the output is:




```
a_range = np.linspace(1, 200, 100)
chi_sq = np.zeros(len(a_range))
# Iteration through division ages
for i in range(len(a_range)):
    # Compute the expected value at each time point.
    expect = cell_area[0] * (2**(time_range/a_range[i]))
    chi_sq[i] = np.sum((cell_area - expect)**2)
plt.plot(a_range, chi_sq, '.')
plt.yscale('log')
plt.xlabel('division age [min]')
_ = plt.ylabel('$\chi^2$')
```




```
min_ind = np.argmin(chi_sq)
best_a = a_range[min_ind]
print('The index at the minimum is ' + str(best_a) + ' min.')
    The index at the minimum is 33.16161616161616 min.
t = np.linspace(0, 190, 500)
S_t = cell_area[0] * (2**(t/best_a))
plt.plot(time_range, cell_area, '.', label='data')
plt.plot(t, S_t, '-', label='best-fit')
plt.xlabel('time [min]')
plt.ylabel('cell area [pixels]')
_ = plt.legend()
```




The reason why this unexpected event happens is, supposedly, the environment in

which the bacteria is multiplying. According to [5], the microscope being placed in an incubator lead to a surplus of nutriments. Apart from this, light spread in the optics is believed to have played a role when measuring the size of the border cells.

# 5 Macroscopic model

In this section, we go back to our original particle model. The kinetic equation will replace our original system of ODEs, but it is not sufficient for a macroscopic formulation. Moreover, the kinetic equation will contain units, and therefore we will need to find a dimensionless form that indicates the dominant, more relevant terms.

Before writing the mean field kinetic model, which will simplify our analysis, let us rewrite the ODE system which corresponds to the microscopic model:

$$\dot{x}_i = \sum_{i \neq j} \psi_x B_{ij}^x \frac{x_i - x_j}{|x_i - x_j|} \tag{2.9}$$

$$\dot{\varphi}_i = \sum_{i \neq j} \psi_\varphi B_{ij}^\varphi \operatorname{sgn}(\sin 2(\varphi_i - \varphi_j)) \tag{2.10}$$

$$\dot{a}_i = 1, \tag{2.11}$$

with

$$B_{ij}^x := 1 - \frac{|x_i - x_j|}{w}$$

and

$$B_{ij}^\varphi = \min\left\{ \frac{l(a_i)}{2} - |\lambda_i|, \frac{l(a_j)}{2} - |\lambda_j| \right\},$$

where

$$l(a) = \frac{l_d}{2}\left(1 + \frac{a}{a_d}\right).$$

Since there is a high number of bacteria, the particle dynamics could be instead approximated by a single equation. We start with an empirical distribution function $f^N(t, x, \varphi, a)$ and a test function $g$. We have:

$$\langle g, f^n \rangle = \frac{1}{N} \sum_{i=1}^{N} g(x_i(t), \varphi_i(t), a_i(t)). \tag{5.1}$$

We can differentiate this with respect to time by applying the definition of $g$ and the previous ODEs:

$$\begin{aligned}
&\frac{d}{dt}\langle g, f^n\rangle = \frac{d}{dt}\frac{1}{N}\sum_{i=1}^{N} g(x_i(t),\varphi_i(t),a_i(t)) \\
&= \frac{1}{N}\sum_{i=1}^{N}[(\nabla_x g)(x_i(t),\varphi_i(t),a_i(t))\frac{dx_i}{dt} \\
&+ (\partial_\varphi g)(x_i(t),\varphi_i(t),a_i(t))\frac{d\varphi_i}{dt} \\
&+ (\partial_a g)(x_i(t),\varphi_i(t),a_i(t))] \\
&= \frac{1}{N}\sum_{i=1}^{N}[(\nabla_x g\cdot v_f)(x_i(t),\varphi_i(t),a_i(t)) \\
&+ (\partial_\varphi g\cdot F_f)(x_i(t),\varphi_i(t),a_i(t)) \\
&+ (\partial_a g)(x_i(t),\varphi_i(t),a_i(t))] \\
&= \langle \nabla_x g\cdot v_f, f^N\rangle + \langle \partial_\varphi g\cdot F_f, f^N\rangle + \langle \partial_a g, f^N\rangle \\
&= -\langle g, \nabla_x\cdot(v_f f^N)\rangle - \langle g, \partial_\varphi\cdot(F_f f^N)\rangle - \langle \partial_a g, f^N\rangle \\
&= \langle g, -\nabla_x\cdot(v_f f^N) - \partial_\varphi\cdot(F_f f^N) - \partial_a(f^N)\rangle,
\end{aligned} \tag{5.2}$$

where:

$$v_f(x,\varphi,a) = \int \psi_x\left(1-\frac{|x-x'|}{w}\right)\frac{x-x'}{|x-x'|}f(x',\varphi',a')dx'd\varphi'da' \tag{5.3}$$

and

$$F_f(x,\varphi,a) = \int \psi_\varphi(B^\varphi(x,x',\varphi,\varphi',a,a'))\,\mathrm{sgn}(\sin 2(\varphi-\varphi'))f(x',\varphi',a')dx'd\varphi'da'. \tag{5.4}$$

We have:

$$\frac{d}{dt}\langle g, f^N\rangle = \langle g, \partial_t f^N\rangle = \langle g, -\nabla_x\cdot(v_f f^N) - \partial_\varphi(F_f f^N) - \partial_a(f^N)\rangle$$

Letting $N\to\infty$, we introduce a distribution function $f^N(t,x,\varphi,a)\to f(t,x,\varphi,a)$. The kinetic equation is:

$$\partial_t f + \nabla_x\cdot(v_f f) + \partial_\varphi(F_f f) + \partial_a f = 0. \tag{5.5}$$

For notation simplicity, we can define the macroscopic density as:

$$\rho(x) = \int f dad\varphi$$

and write:

$$v_f = \int \psi_x \left(1 - \frac{|x - x'|}{w}\right)_+ \frac{x - x'}{|x - x'|} \rho(x) dx', \tag{5.6}$$

where $\psi_x(B^x) = \kappa_2 \cdot \max\{0, B^x\}$.

We recall that $x + \lambda\omega(\varphi) = x' + \lambda'\omega(\varphi')$ and $l(a) = \frac{l_d}{2}\left(1 + \frac{a}{a_d}\right)$, therefore we can express the badness of orientations in the following way (note that the denominator will cancel after scaling):

$$B^\varphi(x, x', \varphi, \varphi', a, a') = \frac{1}{l_d} \min\left\{\frac{l(a)}{2} - |\lambda|, \frac{l(a')}{2} - |\lambda'|\right\}$$

We have:

$$F_f = \int \psi_\varphi (B^\varphi)_+ \operatorname{sgn}(\sin(2(\varphi - \varphi'))) f dx' da' d\varphi', \tag{5.7}$$

where $\psi_x(B^\varphi) = \kappa_1 \cdot \max\{0, B^\varphi\}$.

The next step is to find a scaling such that we could the kinetic equation takes a dimensionless form. We need the following substitutions:

- $x \to x_0 x$, whereby $x_0$ has a dimension and the $x$ on the right-hand side is dimensionless (for simplicity, we will use similar notation for the rest of the scaled variables);
- $f \to f_0 f$, with $f_0 x_0 a_d = 1$;
- $t \to t_0 t$;
- $\lambda \to l_d \lambda$;
- $l \to l_d l$;
- $a \to a_d a$;

After scaling, we have:

$$x + \frac{l_d}{x_0}\lambda\omega(\varphi) = x' + \frac{l_d}{x_0}\lambda'\omega(\varphi'), \tag{5.8}$$

and the kinetic equation becomes:

$$\frac{1}{t_0}\partial_t f + \frac{1}{x_0}\nabla_x(v_f f) + \partial_\varphi(F_f f) + \frac{1}{a_d}\partial_a(f) = 0. \tag{5.9}$$

The first and last terms are linear in $f$, whereas the middle ones are quadratic in $f$. Still, we need to replace the velocity $v_f$ and angular velocity $F_f$ by the following:

- $v_f \to \kappa_2 v_f$;
- $F_f \to \kappa_1 F_f$,
  since $f_0 x_0 a_d = 1$.

Now, if we multiply the kinetic equation by $t_0$, the parameters will become dimensionless:

$$\partial_t f + \kappa_2 \frac{t_0}{x_0}\nabla_x(v_f f) + \kappa_1 t_0 \partial_\varphi(F_f f) + \frac{t_0}{a_d}\partial_a(f) = 0. \tag{5.10}$$

Next, we will introduce some notation, so that we can write the kinetic equation in a simpler form and see what the dominating term is:

- $\frac{a_d}{t_0} =: 1$
- $\kappa_1 f_0 x_0 a_d t_0 =: \frac{\mu}{\varepsilon}$ (big parameter)
- $\frac{l_d}{x_0} =: \delta(\varepsilon)$ (small parameter representing the rescaled length)
- $\kappa_2 f_0 a_d t_0 =: \frac{1}{\varepsilon}$ (big parameter)

We need to check whether this scaling works by writing $\varepsilon, \mu, \delta, t_0$, and $x_0$ in terms of the original parameters. We have that $\varepsilon = \frac{x_0}{\kappa_2 a_d}$, so $\varepsilon = \left(\frac{w}{\kappa_2 a_d}\right)^{4/5}$ and $x_0 = w^{4/5}\kappa_2^{1/5} a_d^{1/5}$. Since $\frac{\mu}{\varepsilon} = \kappa_1 t_0$, we get $\mu = \kappa_1 \left(\frac{w}{\kappa_2}\right)^{4/5} a_d^{1/5}$.

Therefore, we can now write:

$$\partial_t f + \frac{1}{\varepsilon}\nabla_x(v_f \cdot f) + \frac{\mu}{\varepsilon}\partial_\varphi(F_f \cdot f) + \partial_a f = 0, \tag{5.11}$$

where $\frac{\mu}{\varepsilon}\partial_\varphi(F_f \cdot f)$ is the dominating term.

Obviously, equation (5.8) becomes, by using the rescaled width $\delta w$:

$$x + \delta\lambda\omega(\varphi) = x' + \delta\lambda'\omega(\varphi'). \tag{5.12}$$

Recalling that the density is $\rho(x') = \int f(x', \varphi, a')da'd\varphi$, let us focus on the integral derived from (5.6):

$$\int_{x' \in B_{\delta w}(x)} \left(1 - \frac{|x - x'|}{\delta w}\right) \frac{x - x'}{|x - x'|}\rho(x')dx'.$$

Note that $B_{\delta w}(x)$ is a small ball and $\psi_x = 0$ outside of it. Keep in mind that the "badnesses" are approximately 1 and everything that concerns dimension lies in the $\kappa$'s.

We introduce a directional vector $\nu$, with $|\nu| \leq 1$, such that $x - x' = \delta w\nu$. After change of variables (and not omitting the Jacobian, since this is in 2D), the integral becomes:

$$\delta^2 w^2 \int_{\nu \in B_1} (1 - |\nu|)\frac{\nu}{|\nu|}\rho(x - \delta w\nu)d\nu.$$

We now want to Taylor expand this. We observe that $\dfrac{\nu}{|\nu|}$ is an odd function and $\delta w = 0$, therefore the first approximation of the Taylor expansion is 0. For the second approximation:

$$\delta^2 w^2 \int_{\nu \in B_1} (1 - |\nu|)\frac{\nu}{|\nu|}\rho(x - \delta w\nu)d\nu \approx -\delta^3 w^3 \int_{\nu \in B_1} \frac{(1 - |\nu|)}{|\nu|}(\nu \otimes \nu)\nabla_x\rho(x),$$

where $\nu = (\nu_1, \nu_2)^T$, and $\displaystyle\int_{\nu \in B_1} \frac{(1 - |\nu|)}{|\nu|}(\nu \otimes \nu)$ is a matrix:

$$A := \int_{\sqrt{\nu_1^2 + \nu_2^2} < 1} \frac{1 - \sqrt{\nu_1^2 + \nu_2^2}}{\sqrt{\nu_1^2 + \nu_2^2}} \begin{pmatrix} \nu_1^2 & \nu_1\nu_2 \\ \nu_2\nu_1 & \nu_2^2 \end{pmatrix} d\nu_1\nu_2 \tag{5.13}$$

The values on the diagonal are easy to compute using polar coordinates, and we obtain $\dfrac{\pi}{12}$. An interesting exercise is if we use a machine to compute

$$\int_{\nu_1=-1}^{\nu_1=1} \int_{\nu_2=-\sqrt{1-\nu_1^2}}^{\nu_2=\sqrt{1-\nu_1^2}} \frac{1 - \sqrt{\nu_1^2 + \nu_2^2}}{\sqrt{\nu_1^2 + \nu_2^2}}\nu_1^2 d\nu_2\nu_1 :$$

```
from scipy import sqrt
from scipy.integrate import dblquad
func = lambda x, y:
```

```
            0.0 if x == y == 0
            else ((1-sqrt(x**2+y**2))/(sqrt(x**2+y**2)))*(x**2)
print(dblquad(func, -1, 1,
            lambda x: -sqrt(1-x**2), lambda x: sqrt(1-x**2)))
```

The integrand is not defined at the point $(0,0)$, so computationally `func(0.0,0.0)` gives a `NaN`, and if `dblquad` happens to evaluate the integrand at that point, we get `NaN` as a result. Therefore, we extend the definition of the integrand to the whole unit disk by setting its value at $(0,0)$ to be 0 (the integrand is continuous near $(0,0)$, and has a limit of 0 as $(\nu_1, \nu_2)$ approaches $(0,0)$ from any direction). The output is `(0.26179938777665224, 1.4367001171400878e-08)`, so the machine is giving a result which is accurate within the desired error bounds.

The values corresponding to the antidiagonal are equal to 0, making $A$ a multiple of the identity matrix. The second approximation is therefore:

$$\int_{x' \in B_{\delta w}} \left(1 - \frac{|x-x'|}{\delta w}\right) \frac{x-x'}{|x-x'|} \rho(x')dx' \approx -\delta^3 w^3 A \nabla_x \rho(x),$$

where $A = \dfrac{\pi}{12}$.

We can take $\varepsilon = \delta^3 w^3$, since both $\varepsilon$ and the rescaled width are dimensionless, and we obtain:

$$\nabla_x (v_f \cdot f) \approx \nabla_x (-Af \nabla_x \rho(x)). \tag{5.14}$$

Recall that the rescaled length is denoted by $\delta(\varepsilon)$. We will follow the same steps, but for the interaction part of the orientations, namely for:

$$\int \psi_\varphi \left(\frac{B^\varphi}{\delta(\varepsilon)}\right)_+ \operatorname{sgn}(\sin(2(\varphi - \varphi'))) f dx' da' d\varphi', \tag{5.15}$$

with

$$\frac{B^\varphi}{\delta(\varepsilon)} = \min\left\{\frac{l(a)}{2\delta(\varepsilon)} - \frac{|\lambda|}{\delta(\varepsilon)}, \frac{l(a')}{2\delta(\varepsilon)} - \frac{|\lambda'|}{\delta(\varepsilon)}\right\}. \tag{5.16}$$

Analogously, we have that $x - x' = \delta(\varepsilon)\nu$, which leads to the formula:

$$\delta(\varepsilon)^2 \int \left(\frac{B^\varphi}{\delta(\varepsilon)}\right)_+ \operatorname{sgn}(\sin(2(\varphi - \varphi'))) \rho(x - \delta(\varepsilon)\nu) d\nu, \tag{5.17}$$

which implies, for $\delta(\varepsilon)$ small, that we have the following approximation:

$$\partial_\varphi(F_f f) \approx \partial_\varphi\left(\frac{\mu}{\varepsilon} f\left(\delta(\varepsilon)^2 \int \left(\frac{B^\varphi}{\delta(\varepsilon)}\right)_+ \operatorname{sgn}(\sin(2(\varphi-\varphi'))\rho(x-\delta(\varepsilon)\nu)d\nu\right)\right). \quad (5.18)$$

Since the divergence of the $\varphi$ term is dominating, $\delta(\varepsilon)^2\frac{\mu}{\varepsilon}$ has to dominate over $\frac{\delta^3 w^3}{\varepsilon}$. Let us define the ratio between the rescaled width and rescaled length as:

$$r = \frac{\delta w}{\delta(\varepsilon)},$$

with $\varepsilon = \delta^3 w^3$. $r$ is small because the rescaled length is bigger than the rescaled width. Therefore:

$$\delta(\varepsilon)^2\frac{\mu}{\varepsilon} = \frac{\delta^2 w^2 \mu}{r^2\delta^3 w^3} = \frac{\mu}{r^2\delta w}.$$

The denominator is small, so we can assume $\mu = \frac{1}{\delta(\varepsilon)^2}$.

Finally, (5.11) becomes:

$$\partial_t f + \nabla_x(-Af\nabla_x\rho(x)) + \partial_\varphi\left(f\left(\frac{\mu}{r^2\delta w}\mathcal{F}(f)\right)\right) + \partial_a f = 0, \quad (5.19)$$

with

$$\mathcal{F}(f) := \int \frac{B^\varphi}{\delta(\varepsilon)}_+ \operatorname{sgn}(\sin(2(\varphi-\varphi'))\rho(x-\delta(\varepsilon)\nu)d\nu. \quad (5.20)$$

Although we use the macroscopic density for simplicity, we want to find a formula for $f$ such that (5.20) is 0 because we want to see what happens at equilibrium (when $\varepsilon$ goes to 0). At equilibrium, we have two groups of cells (of volume $\rho_+$ and $\rho_-$) which have opposite orientations ($\varphi_+$ and $\varphi_+ + \pi$ respectively). Writing $\delta_{\varphi-\varphi_+(x,a)}$ the Dirac delta in $\varphi - \varphi_+(x,a)$, we find:

$$f(x,\varphi,a) = \rho_+\delta_{\varphi-\varphi_+(x,a)} + \rho_-\delta_{\varphi-\varphi_+(x,a)-\pi}. \quad (5.21)$$

As in [3], we take $\Psi = 1$ and $\Psi = \varphi$ for the case of collision invariants for the collision operator $\delta_\varphi(f\mathcal{F}(f))$ and we obtain:

$$\partial_t\int \Psi f d\varphi + \nabla_x\int \Psi f v_f d\varphi + \int \Psi\partial_\varphi f F_f d\varphi + \partial_a\int \Psi f d\varphi = 0 \quad (5.22)$$

The reason why we introduced the GCI in the Introduction is because the simpler concept of collision invariants only spans a two-dimensional space. Integration by parts shows that $\Psi_f = 1_{\{\varphi_{0,f},\varphi_{1,f}\}}$ is a GCI, where $(f\mathcal{F}(f))(\varphi_{i,f}) = 0, i = 0, 1$ because $f\mathcal{F}(f)$ has at least two zeros.

Letting $\varepsilon \to 0$ in (5.22), we obtain the macroscopic equation:

$$\partial_t \int \Psi_f f d\varphi + \nabla_x \int \Psi_f (f(-A\nabla_x \rho_f)) d\varphi + \int (\partial_\varphi \Psi_f) f F_f d\varphi + \partial_a \int \Psi_f f d\varphi = 0, \quad (5.23)$$

with

$$f(x, \varphi, a) = \rho_+ \delta_{\varphi - \varphi_+(x,a)} + \rho_- \delta_{\varphi - \varphi_+(x,a) - \pi} \quad (5.24)$$

$$\rho_f = \rho_+ + \rho_-, \quad (5.25)$$

where $\rho_+$ and $\rho_-$ are independent of $\varphi$.

Plugging in the GCI into (5.23), we obtain:

$$\partial_t \int_{\varphi_{0,f}}^{\varphi_{1,f}} f d\varphi + \nabla_x \dot{\int}_{\varphi_{0,f}}^{\varphi_{1,f}} (-A\nabla_x \rho_f)) d\varphi + (fF_f)(\varphi_{0,f}) - (fF_f)(\varphi_{1,f}) + \partial_a \int_{\varphi_{0,f}}^{\varphi_{1,f}} f d\varphi \quad (5.26)$$

We need to introduce the value of $f$ at the equilibrium. The first and the last terms are equal to $\partial_t \rho_+$ and $\partial_a \rho_+$, respectively. Similarly, the second term can be written as $\nabla_x \dot{(}\rho_+(-A\nabla_x \rho_f))$. For the remaining terms, we just rewrite them as $\rho_+ \delta(\varphi_{0,f} - \varphi_+) F_f(\varphi_{0,f}$ and $\rho_+ \delta(\varphi_{1,f} - \varphi_+) F_f(\varphi_{1,f}$, respectively.

# Bibliography


[1] Hiro-Sato Niwa (1993) *Kinetic Theory of Fish Schooling*, National Research Institute of Fisheries Engineering, Hasaki, Kashima, Ibaraki 314-04, Japan.

[2] C.H. Bamford, C.F.H. Tipper, R.G. Compton (1985) *Chapter 12 The Kinetic Theory Applied to Chemical Reactions in Solutions*, Comprehensive Chemical Kinetics, Elsevier, Volume 25, Pages 339-359.

[3] Dottoressa Sarah Benedetto (2019) *Kinetic Theory Applied to Skiers*, University of Vienna.

[4] Zhihong You, Daniel J. G. Pearce, Anupam Sengupta, Luca Giomi (2018) *Geometry and mechanics of microdomains in growing bacterial colonies*, Physical Review X, Volume 8, American Physical Society (APS).

[5] Eric J. Stewart, Richard Madden, Gregory Paul, François Taddei (2005) *Aging and Death in an Organism That Reproduces by Morphologically Symmetric Division*, PLOS Biology 3(2): e45.

[6] Cédric Villani, (2002) *A Review of Mathematical Topics in Collisional Kinetic Theory*, Handbook of Mathematical Fluid Dynamics. 1. 10.1016/S1874-5792(02)80004-0.

[7] Imrich Barák, Anthony Wilkinson (2007) *Division site recognition in Escherichia coli and Bacillus subtilis*, FEMS microbiology reviews. 31. 311-26. 10.1111/j.1574-6976.2007.00067.x.

[8] Arthur L. Koch (1982) *On the Growth and Form of Escherichia coli*, Journal of General Microbiology, 128, 2527-2539.

[9] Timothy J. Rudge, Paul J. Steiner, Andrew Phillips, Jim Haseloff (2012) *Computational Modeling of Synthetic Microbial Biofilms*, ACS Synth. Biol. 1, 8, 345–352.

[10] Timothy J. Rudge, Fernán Federici, Paul J. Steiner, Anton Kan, Jim Haseloff (2013) *Cell Polarity-Driven Instability Generates Self-Organized, Fractal Patterning of Cell Layers*, ACS Synth. Biol. 2, 12, 705–714.

[11] Timothy J. Rudge, Paul J. Steiner, Jim Haseloff (2012-2013) *CellModeller4*, University of Cambridge, github.com/HaseloffLab/CellModeller.

[12] Beth Gibson, Daniel J. Wilson, Edward Feil, Adam Eyre-Walker (2018) *The distribution of bacterial doubling times in the wild*, Proc Biol Sci. 2018;285(1880):20180789.

[13] Lindsay Plank, J.D. Harvey (1979) *Generation Time Statistics of Escherichia coli B Measured by Synchronous Culture Techniques*, Journal of general microbiology. 115. 69-77.

[14] Pierre Degond, Sébastien Motsch (2008) *Continuum limit of self-driven particles with orientation interaction*, Math. Models Methods Appl. Sci., 18(Suppl.):1193-1215.

[15] Pierre Degond, Angelika Manhart, Huy Yu (2017) *A continuum model for nematic alignment of self-propelled particles*, Discrete and Continuous Dynamical Systems - B. , 22 (4) : 1295-1327. doi: 10.3934/dcdsb.2017063.