# Simulated Accelerated Evolution by Modeling the Rapid Fixation of Bacteria Along an Antibiotic Gradient

By Michael Frood

A thesis submitted to the Faculty of Graduate Studies in partial fulfillment of the requirements for the degree of Masters of Science in Physics

Lakehead University

December 2019

## Abstract

The work in this thesis makes connections between statistical mechanics and genotype frequencies in population sizes, and how mutation of the bacteria *E. coli* helps to increase its resistance to the antibiotic ciprofloxacin. The objective of this thesis is to use existing models in statistical mechanics as a basis in order to create a new program to simulate how an antibiotic gradient affects the rate of mutation of a bacterial population, and what influence the shape of the gradient has on the evolutionary rate. In addition, several other factors affecting the evolutionary rate are identified as well. Overall, a steeper antibiotic gradient will result in accelerated evolution as long as the initial growth of bacteria is not in the presence of antibiotic. It was observed that by allowing for initial bacterial growth in the absence of antibiotic, followed by migration towards an increasing gradient of antibiotic results in accelerated mutation.

Acknowledgments

To mom and dad with acknowledgment to Dr. Linhananta as well.

Table of Content

List of Tables

List of Figures

List of Abbreviations

# Chapter 1

# Introduction

## 1.1 Introduction and Theory

The use of antibiotics in combating infections has transformed medical sciences, and while there existed a golden era of antibiotic development in the 1930s to 1960s, has been followed by a comparative failure to develop and discover new antibiotics to combat the emergence of antibiotic resistant bacteria [1]. While the emergence of antimicrobial resistance (AMR) is a problem [1, 2], genetic analysis of bacterial mutation usually fails to expand on physical evolutionary dynamics due to the environment which can enhance the rate of drug resistance of a bacterial population [2].

There are two basic concepts that drive evolution: All living organisms reproduce themselves, and living organisms reproduce themselves with imperfections in their genome. The imperfections are changes in nucleotides which produce mutations in the genome. A mutant's genetic variation creates selective differences and it's possible that it may have a different reproduction rate than their parents. These original (wild type) and mutated offspring compete with each other via natural selection which in turn changes gene frequencies in a population [3, 4]. The thesis observes previous studies of wild type *E. coli*, and how they form mutant *E. coli* to resist the effects of the antibiotic Ciprofloxacin in order to create two simulations modeling these studies [2, 5]. While there is no obvious analogy to connect to physical systems, there have been several

attempts to connect statistical mechanics to evolutionary biology using qualitative analogies [3, 5-9].

Statistical Mechanics is useful for predicting global properties of a system by accounting for all possible configurations and progressions of the microscopic components inside [10]. It uses statistical tools in order to average all possible configurations so that the most probable result can be obtained; the microscopic dynamics help to come up with deterministic macroscopic laws. Traditionally, this methodology is usually applied to systems in equilibrium, however for reasons which will soon be discussed; evolutionary dynamics may be classified as a non-equilibrium process [7, 10].

## 1.2 Objective

The objective of this thesis is to create two computational models using existing models in statistical mechanics and reaction-diffusion as a basis in order to simulate certain dynamics of antimicrobial resistance to antibiotics. What is of particular interest is how environmental and other evolutionary conditions can enhance the rate of mutation then subsequent fixation, and how this can be simulated. Fixation means that all of the individuals in a population have a particular mutation or genotype [5]. The main environmental factor of interest is how an antibiotic gradient affects the fixation of a mutation in a bacterial population, and what influence the shape of the gradient has on evolutionary rate.

# Chapter 2

# Background

## 2.1 Statistical Mechanics and the Boltzmann Distribution

In statistical mechanics the Boltzmann distribution is a probability distribution that gives the probability of selecting a state as a function of that state's energy (or in this case fitness function). If the temperature $T$ is well defined, the probability that a certain energy at level *i*, ($E_i$), is occupied is known as the Boltzmann factor ($P_i \propto e^{-\beta E_i}$), where $\beta = {}^{1}/_{k_B T}$ and $k_B$ is the Boltzmann constant. A Hamiltonian is an energy function which accounts for the energy of a particular state ($H(i) = E_i$), where $E_i$ is the energy of the state ($i$). A summation over all the possible energy levels is known as the partition function ($Z$), and each state is represented in $Z$ by its Boltzmann factor $Z = \sum_i e^{-\beta E_i}$ [10-12]. The probability that an $i^{th}$ energy level is occupied is $P_i = \frac{e^{-\beta E_i}}{Z}$. The area under the Boltzmann distribution curve is the total number of states and certain energies which are given by their Boltzmann factor are more probable because there are a greater number of states for that particular energy level, (increasing the probability of being in that energy state). This can be seen in Figure 1.

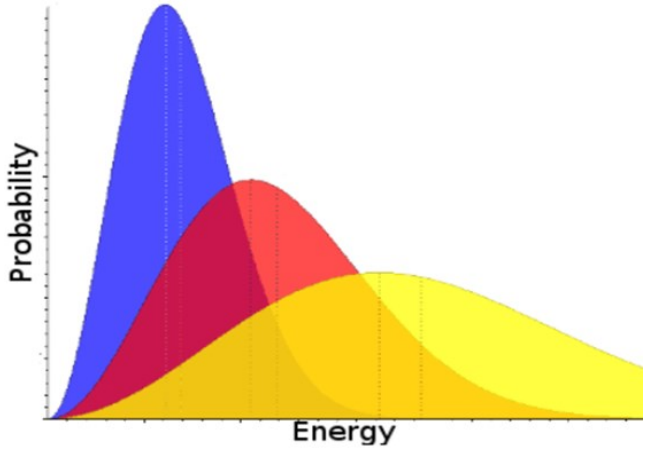

Figure 1: The Boltzmann Probability Distribution for various temperatures. The larger the temperature the wider the distribution. The area under the curve is the percentage distribution for each energy level [11].

## 2.2 Markov Chain

A Markov Chain (MCh) is a model of a collection of random variables describing a sequence of possible states in which the probability of the next configuration depends on the current state of the system and not the previous history [10, 12]. Using a MCh, one can obtain a sample of the desired distribution by observing the chain after a number of steps. The more steps there are, the more closely the distribution of the sample matches the actual desired distribution. Further information will be discussed in the following sections 2.6 and 2.7, but the desired distribution is for the system to converge to equilibrium. If "*i*" and "*j*" are two discreet states of the system, the transitions (from *i* to *j*) are stochastic (i.e. the transitions are random and controlled by transition probabilities):

$$P(i \to j) = P_{ij} \tag{1}$$

An *N x N* matrix ($P$) is called stochastic if all of its elements are non-negative ($P_{ij} \geq 0$), and the sums of all its columns are equal to 1 (the normalization condition):

$$\sum_{i=1}^{N} P_{ij} = 1 \tag{2}$$

A stochastic matrix is not symmetric, where $P_{ij} \neq P_{ji}$ in the matrix $P$. An Eigenvector which describes a stationary state of the system is associated with an Eigenvalue of 1 for a stochastic matrix [12]. This means that the probability of finding the system in a possible configuration does not evolve with time ($t$):

$$\omega(t + \Delta t) = P \cdot \omega(t) = 1 \cdot \omega(t) = \omega(t) \tag{3}$$

here $\omega(t)$ is the Eigenvector at a particular time $t$, and $P$ is the probability matrix of equation 2. In the case of a system in thermodynamic equilibrium, the probability for finding a system in energy state "$i$" is then proportional to the Boltzmann distribution:

$$\omega_i = \frac{e^{-\beta E_i}}{Z} \tag{4}$$

### 2.3 The Master Equation for Three States

The master equation is a differential equation which measures the change in probability of being in a particular microstate [10, 12]. In certain systems like the Ising model each site or location can be in a certain state, (up or down), but since different eigenvalues for that site depend on the states of neighbouring sites, that site's Hamiltonian is a function of several different microstates [36,37]. The microstates in the Ising model are a configuration of the selected state with the states of its nearest neighbours[36, 37]. This will be discussed further in section 2.20.

A system, (like a finite Ising model) can be composed of a large number of microstates, however for this example let us assume there is a system with three discreet states ($i, j, k$). The stochastic probability matrix of this system can be seen in Table 1.

Table 1: Stochastic Matrix of a Three State System. The matrix is shaded in grey [10,12].

| | $\boldsymbol{P(i)} \rightarrow$ | $\boldsymbol{P(j)} \rightarrow$ | $\boldsymbol{P(k)} \rightarrow$ |
|---|---|---|---|
| $\boldsymbol{P(i)} \leftarrow$ | $P(i \rightarrow i)$ | $P(j \rightarrow i)$ | $P(k \rightarrow i)$ |
| $\boldsymbol{P(j)} \leftarrow$ | $P(i \rightarrow j)$ | $P(j \rightarrow j)$ | $P(k \rightarrow j)$ |
| $\boldsymbol{P(k)} \leftarrow$ | $P(k \rightarrow i)$ | $P(j \rightarrow k)$ | $P(k \rightarrow k)$ |
| **Sum of Columns** | 1 | 1 | 1 |

The probability of being at state $k$ at time $t$ is represented by a vector $p(k,t)$. (For this example the eigenvector $\omega(k,t)$ will not be used). The probability of staying at $k$ until time $\Delta t$ is also dependent on the probability of being at a different state (either $i$ or $j$) and making the transition to $k$ during time $t + \Delta t$ [10, 12]:

$$p(k,t+\Delta t) = p(k,t)P(k \rightarrow k) + p(i,t)P(i \rightarrow k) + p(j,t)P(j \rightarrow k) \qquad (5.1)$$

For this equation, the transition probabilities can be written as $P(m \rightarrow n) = T_{mn}\Delta t$ where $T_{mn}$is the transition rate per unit time.

Using Table 1 the transition $P(k \rightarrow k) = 1 - (P(k \rightarrow i) + P(k \rightarrow j))$. Also the limit $\Delta t \rightarrow 0$ can be taken and the equation can be rewritten as:

$$\lim_{\Delta t \rightarrow 0} \; p(k,t+\Delta t) = p(k,t)\Big(1 - \big(P(k \rightarrow i) + P(k \rightarrow j)\big)\Big) + p(i,t)P(i \rightarrow k) + p(j,t)P(j \rightarrow k) \qquad (5.2)$$

Rearranging and adding the new transition probabilities:

$$\lim_{\Delta t \to 0} \quad p(k,t+\Delta t) - p(k,t) = -p(k,t)\big(T_{ki} + T_{kj}\big)\Delta t + p(i,t)T_{ik}\Delta t + p(j,t)T_{jk}\Delta t \tag{5.3}$$

By rearranging and taking the limit $\Delta t \to 0$, equation 5.3 becomes:

$$\frac{\partial p(k,t)}{\partial t} = p(i,t)T_{ik} + p(j,t)T_{jk} - p(k,t)\big(T_{ki} + T_{kj}\big) \tag{5.4}$$

The differential equation 5.4 is the master equation for three discreet states which measures the change in probability of being at state $k$.

## 2.4 Ergodicity

If a system is ergodic, then this means that any state of a system is reachable from any other state. Even if the transition rates between two states are zero, a Markov chain is ergodic if any state can be reached from any other state through a path of transition with non-zero rates connecting the steps [12].

## 2.5 Detailed Balance

Detailed balance states that for a system to change from *i* to *j*, the probability of selecting a state in $i$, $(\omega_i)$, and moving towards a state $j$, $(P(i \to j))$, is the same as the probability to select a state in *j* and to change from *j* to *i*. Detailed balance states that the transition probabilities between the two microstates are symmetric, in equilibrium each elementary process is equilibrated by its reverse process and requires reversibility of all elementary processes [12]. The master equation 5.4 is a global equation where steady state is reached for three states, but as discussed in section 2.3, a system can be composed

of several different localized microstates. Given that the master equation is global, when applied to a single microstate, certain transition probabilities may not be possible.

If a certain microstate in a $i$ state has a probability of transferring to a $j$ state then the master equation 5.4 can look like:

$$\frac{\partial p(i,t)}{\partial t} = p(j,t)T_{ji} + p(k,t)T_{ki} - p(i,t)\big(T_{ij} + T_{ik}\big) \tag{6.1}$$

but in particular cases for multiple reasons the transition between two states can be zero. If this is the case, the transition between $i$ and $k$ is zero and there is no probability of selecting a microstate $k$, so $T_{ki} = T_{ik} = p(k,t) = 0$. Now the master equation looks like:

$$\frac{\partial p(i,t)}{\partial t} = p(j,t)T_{ji} - p(i,t)T_{ij} \tag{6.2}$$

If this microstate obeys detailed balance then $\frac{\partial p(i,t)}{\partial t} = 0$, so for every couple of states that are selected [10, 12]:

$$\omega_i T_{ij} = \omega_j T_{ji} \tag{6.3}$$

The Boltzmann distribution is used for when a model is closer to equilibrium where detailed balance can apply [38]. Using the Boltzmann distribution eigenvectors of the selected states from equation 4, and the transition probabilities of equation 1, this can be expressed as a Boltzmann factor:

$$\frac{P(i \to j)}{P(j \to i)} = e^{-\beta(E_j - E_i)} \tag{7}$$

## 2.6 Metropolis Algorithm

Detailed balance is an important factor that a Markov chain has to fulfill for it to move towards equilibrium. The larger the probability ratio in equation 7, the faster the system can converge to equilibrium. As shown in Table 1, these rates must also satisfy the normalization condition. In this case the Metropolis Algorithm can be used to satisfy these conditions [12]. Assume that the energy of state $j$ is greater than the energy of state $i$, $(E_j > E_i,\ \Delta E = E_j - E_i > 0)$, then the system will spontaneously transit from a higher energy level to a lower one, $P(j \to i) = 1$. Therefore equation 7 becomes $P(i \to j) = e^{-\beta \Delta E}$. The Metropolis Algorithm can be derived from the detailed balance equation in equation 7. The probability rates can be split as follows:

$$P(i \to j) = g(i \to j)A(i \to j) \tag{8}$$

Where $g(i \to j)$ is the selection probability (or proposal distribution) and $A(i \to j)$ is the acceptance ratio. The Metropolis algorithm starts by selecting a "candidate" sample while the selection probability describes which states can be generated from the selected initial state. A symmetric selection probability satisfies detailed balance [12, 13].

The acceptance ratio is the conditions required for the actual transition to take place and is shown below:

$$A(i \to j) = \begin{cases} 1 & if\ (E_j - E_i) < 0 \\ e^{-\beta(E_j - E_i)} & if\ (E_j - E_i) > 0 \end{cases} \tag{9}$$

If the transition leads to a lowering of energy then it is always accepted, otherwise there is a probability of $e^{-\beta(E_j - E_i)}$ of the transition being accepted [12]. This algorithm is used in the computational model further discussed in chapter 3.

## 2.7 Equilibrium

For a system as a whole, Ergodicity and Detailed Balance are useful factors to help the system move towards equilibrium [10, 12]. However, evolutionary dynamics may be classified as a non-equilibrium process in that the genetics in populations do not settle down to a stable point but are continuously changing [7]. When a system is far from equilibrium the actual statistical distribution will vary and may be different from the Boltzmann distribution. While biological processes may not be in equilibrium, attempts can be made to analyze population evolutionary dynamics using statistical mechanics [3,5-9]. This will be discussed further in section 2.8. Part of this involves isolated systems which allow gene frequencies to settle towards an equilibrium point so long as there is a reference point to compare them to. This will be discussed in more detail with S. G. Wright's Fitness Landscape in section 2.10.

## 2.8 Connecting Statistical Mechanics and Evolutionary Dynamics

One of the examples used to connect the evolutionary dynamics of a population and statistical mechanics can be seen in Table 2.

Table 2: State variable, Additive Fitness and Energy, Population Size and Temperature, and the Boltzmann Factor [8].

| **Object** | **Evolutionary Dynamics** | **Statistical Mechanics** |
|---|---|---|
| State Variable | $\vec{g}$ $= \{(A, T, G, or\ C), \dots, (A, \dots)\}$ | $\vec{s} = \{(\vec{q}, \vec{p})\}$ |
| Additive Fitness and Energy | $x_i = \ln(f(\vec{g}))$ | $E_i = H(\vec{s})$ |
| Population Size and Temperature | $\nu_{Moran} = N - 1$ $\nu_{WF}^{h} = 2(N - 1)$ $\nu_{WF}^{d} = 2N - 1$ | $\beta = {}^{1}/_{k_B T}$ |
| Boltzmann Factor | $P_i \propto e^{\nu x_i}$ | $P_i \propto e^{-\beta E_i}$ |

For this analogy the genotype can be regarded as a state analogous to the degrees of freedom in statistical physics ($\vec{g}$), that is, the single-nucleotide polymorphism (SNP) sites in a gene can be in one of four states (Adenine, Guanine, Cytosine, or Thymine). SNPs are the building blocks on DNA and configurations of these sites on the gene represent the state [8]. While $\vec{g}$ is the state variable, $f(\vec{g})$ describes the fitness of that genotype. Fitness is how well the bacteria can survive and reproduce, so the function describes the likelihood of finding a particular genotype in a bacterial population [5, 9]. In this analogy fitness takes the opposite sign of energy in statistical mechanics, so the gene frequencies of a population will want to move towards a higher fitness peak as opposed to how a system will want to move towards a lower energy level in statistical mechanics. Here the population size ($\nu_{Moran,}\ \nu_{WF}^{h}, \nu_{WF}^{d}$), vary depending on the type of

cell being analyzed. The important point of the variables just described is that there are different birth-death processes various cell types which affect the population size [8]. This is analogous to the inverse of the temperature ($T$) in statistical mechanics which was previously discussed, with $k_B$ being the Boltzmann constant and $\beta = {}^{1}/_{k_B T}$. In this case the temperature remaining constant will be similar to a constant population. As mentioned in section 2.7, there may be problems applying the statistical distribution equilibrium method to non-equilibrium biological systems due to the genetics in populations not settling to a stable point but continuously changing [7]. However when compared to a reference point, population genetics of an isolated system can settle down to a stable point. This will be discussed in section 2.10.

### 2.9 Genetic Epistasis

In the previous statistical mechanics example, the model proposed an additive view of genetic (or phenotypic) fitness, whereas the vast majority of real genetic systems are more complex due to pervasive epistasis [9]. Epistasis generally means the interaction between different genes, and can be further defined as a situation whereby the allele of one locus is affected by the presence or absence of another (a genetic background whereby one form of a gene can affect another); this can be further complicated by the complex pathways of several interacting loci [14]. There are several problems with this general definition, particularly when applied to binary phenotypic traits, (a gene being expressed or not), as well further definitions about the "effect" of a locus need to be defined [14]. In this case epistasis refers to any type of genetic interaction which leads to a dependence of mutational effects on the modifier genes, (i.e.

the genetic background) [9]. In addition, epistasis for this thesis is investigated by looking at the genotypic frequencies for a particular phenotypic trait in a population (e.g. *E. coli* resistance to the antibiotic ciprofloxacin) [14].

### 2.10 S. G. Wright's Fitness Landscape

There have been models other than the one in section 2.8 made to identify genetic fitness. Notable geneticist S. G. Wright proposed a three phase process in which evolution can occur [7]. He first proposed that large populations can be split up into semi-isolated subpopulations called demes [7]. Within each deme there is a genotypic fitness landscape describing multiple gene combinations each with their own fitness peak. The fitness peaks have varying levels on the complex landscape and are separated by "valleys" of low-fitness genotypes whereby gene frequencies tend to move toward local peaks [7, 9]. Another name for the bacterial fitness is the Wright evolutionary potential. This approach is contrasted by Wright's colleague Ronald Fischer, who assumed that there is a single optimal genotype in a constant environment and that subsequent mutations are additive [4]. However, epistasis and complex fitness landscapes have shown more experimental evidence towards Wright's model [3, 7, 9, 14].

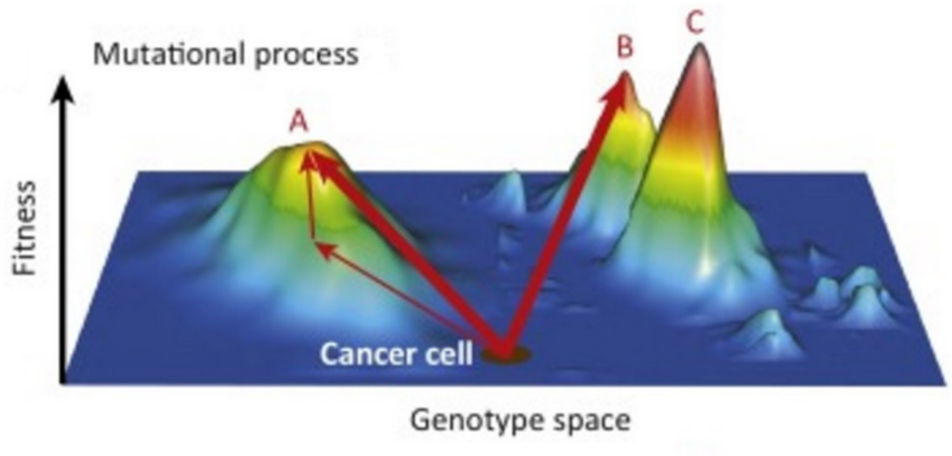


Figure 2: Evolutionary outcomes of cancer mutation of a fitness landscape with multiple peaks on the z axis. The x-y plane represents multiple genotypic combinations [15].

It has been argued that there is no perceived potential function underlying evolution and this is demonstrated by how gene frequencies in a population do not settle down to a stable point but are changing forever (though this could be due to a continuously changing environment) [7]. Wright's evolutionary potential is a relative concept, in so far as the fitness of a bacterial genotype is compared to the fitness of another genotype or even another species; research has not been able to produce a meaningful potential landscape that describes the fitness of a specific genotypic in a population [7]. It is seen in thermodynamics that entropy changes lead to a progressive disorganization of the physical world, while evolutionary changes are seen as producing a higher organization of the organic world. Evolutionary dynamics may be classified as a non-equilibrium process and the flow of energy and matter may be used to build and maintain order [7]. The Wright evolutionary potential is also a local and individual concept, as opposed to entropy which has an absolute value and is global in nature. The Wright evolutionary potential for a genotype of bacteria needs a reference point for each fitness surface (or deme) relative to the fitness of another genotype. Therefore, correct

interpretations of evolution should be dynamical in nature, not thermodynamic, because biological systems at nearly every level are heterogeneous [3].

The probability that a population will have a particular genotype on the fitness landscape (independent of time) leads to the Boltzmann-Gibbs probability distribution:

$$P(\vec{g}, t = \infty) = \frac{1}{Z} exp\left(\frac{\psi(\vec{g})}{\varepsilon}\right) \tag{10}$$

where $Z$ is the partition function [8]. For this model $\psi(\vec{g})$ is the Wright evolutionary potential (which would be equivalent to the *Epistatic fitness* as opposed to the *Additive fitness*). As described before, this is a nonlinear function for a particular genotypic state $\vec{g}$. The role of $\psi$ is similar to potential energy but with opposite sign, while $\varepsilon$ is analogous to temperature in statistical physics [7]. Here $\varepsilon$ is a parameter associated with the stochastic drive of shifting gene frequencies and can be proportional to the inverse of the total population size [7, 8]. The mean squared displacement of the stochastic drive is proportional to $\varepsilon$, so therefore a larger value will mean that it is more likely for genotype frequencies to vary on a fitness landscape. Therefore, the larger the value of $\varepsilon$, the wider the equilibrium distribution will be because of the greater genetic variation, or in other words, a larger value of $\varepsilon$ will reduce the probability of a population having a particular genotype which may increase the probability of other genotype frequencies. In terms of population density, a larger population will most likely have a lower stochastic drive ($\varepsilon \propto {}^{1}/_{\nu}$), making it more difficult for a genotype frequency to move away from certain fitness peaks due to it being harder for a new genotype to fix in that population. As previously mentioned in the introduction, fixation means that all of the individuals in a population have a particular mutation or genotype [5]. Because a larger population is

proportional to a smaller value of $\varepsilon$, the population will have less genetic variation and consequently a narrower probability distribution. Accompanying Wright's theorem is Ronald Fischer's fundamental theorem of Natural Selection, (or F-Theorem), which states that the rate of increase in fitness of any organism at any time is equal to its genetic variance in fitness [7].

New developments in fitness landscapes introduced functional intermediates in evolutionary experiments which differ by single gene changes (epistatic gene combinations with their own fitness values on the landscape). As gene frequencies shift toward local peaks through singular gene changes, all intermediate epistatic states must be functional for the pathway to be accessible on the landscape [9].

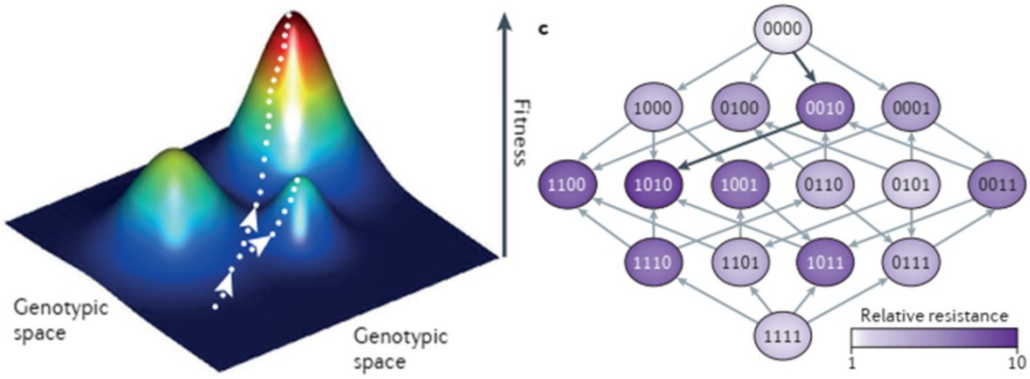


Figure 3: A fitness landscape, (left), with the genotype on the x-y plane while the fitness is on the z axis. An example of an empirical fitness landscape, (right), based on the 3D fitness landscape. This involves four mutations while the relative resistance (fitness against antibiotic) is measured for each genotype. Functional pathway arrows are shown between genotypes [9].

The distance between travelling from one genotype to another is measured by the number of genotypic substitutions needed (the Hamming distance), or in the case of this thesis, the number of mutations. Random drift from the stochastic drive can help move

gene frequencies across landscapes to a possibly higher fitness peak [7, 9]. The model shows that a smaller population size in a deme enhances a shift in genotype frequencies because a fluctuation in frequency will have a larger impact on the smaller population. This property greatly increases the chances of fixing rare outliers; as well more fit genomes also have a greater probability of fixing in a smaller population [3, 7]. The thesis constructs a computational model whereby the energy is analogous to the bacterial fitness described in this section. The bacterial fitness discussed in 2.8 and 2.10 can be used by the metropolis algorithm in section 2.6 to determine the probability of moving to a more fit genotype. More information about the model is described in chapter 3.

## 2.11 Zhang *et al*'s Experiment and the Microfluidic Array

While fitness landscapes help to understand how resistance spreads within a deme, more work can be done to understand how spatially heterogeneous environments affect the spread of antibiotic resistant mutants within populations of bacteria. Starting in 2006 Zhang *et al*, including noted biophysicist Robert H. Austin, performed several ingenious experiments which induced accelerated antibiotic resistance in bacteria using a microfluidic array device [2, 3, 5].

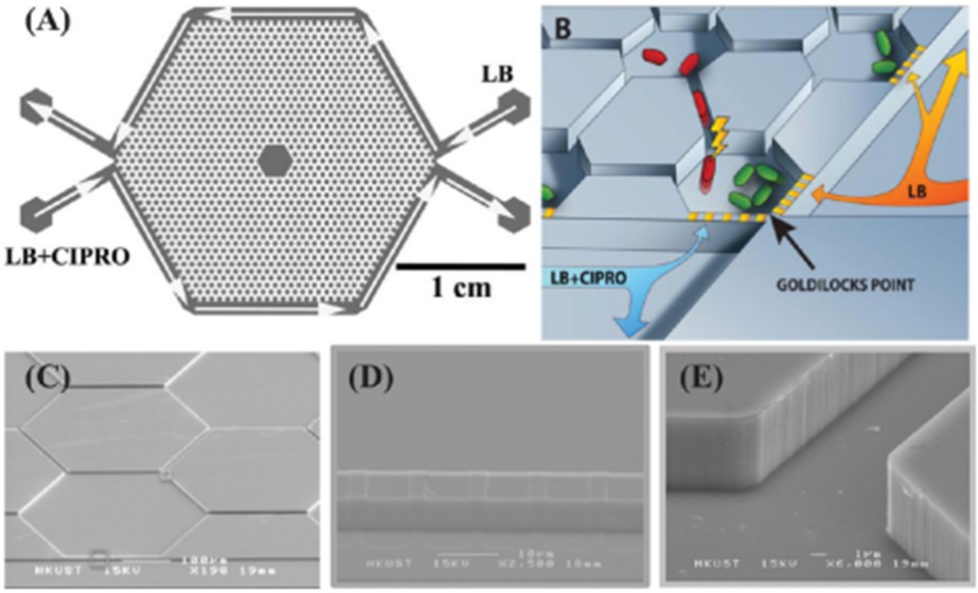


Figure 4: An image of the microfluidic array used in the experiment by Zhang *et al*. Food broth forms a gradient on the upper half of the device, while Food + Cipro form a gradient on the lower half [5].

The experiment had an *E. coli* culture inoculated into the center of the 2D microfluidic array. The device was 2 cm in diameter and housed 1200 hexagonal microhabitats each with a 200 μm diameter. The microhabitats are interconnected by 10 μm wide channels which allow for the motion of bacteria between different microhabitats. In order to simulate a non-homogenous environment, a food broth (LB) circulates around the upper half of the device, while LB + Ciprofloxacin circulate around the other half. The continuous flow of broth or antibiotic around the perimeter generates gradients of food and antibiotic within the array of microhabitats with channels connecting them as they flow towards the center of the device [2, 5]. For this array, the microhabitats resemble demes while the channels help the transfer the frequency of genotypes from one population's fitness landscape to another's.

## 2.12 Accelerated Fixation

The experiments suggest this non-uniform distribution of antibiotic and food can greatly accelerate the evolution of antibiotic resistant bacterial populations [2, 5]. In a spatially non-uniform environment an enhanced rate of mutation may occur for two reasons: first, if a bacterium mutates to resist local stress the relative fitness of the mutant is increased if it moves to join a population with lesser resistance exposed to even higher stress [2, 5, 6]. The second is because fitness landscape models have shown that fluctuations in fitness are enhanced in smaller population sizes (demes) [3, 7]; a region with higher stress will reduce the population of bacteria with lesser resistance. A rare mutation that gives resistance to a bacterium may not be fixable if the native wild type population is very high, so if a mutation occurs which increases the fitness, it can be fixed more rapidly if it moves into regions of lower population density [2, 5, 6, 16]. The hypothesis is that "This stepwise movement of motile mutant bacteria via successive mutations into regions of higher stress is accelerated if the mutation rate is very high and the population density gradient … is very steep" [5].

## 2.13 The Goldilocks Point

Chemotaxis is the movement of bacteria along a concentration gradient for a particular substance due to stimulus, (in this case the food gradient). The experiments by Robert Austin with Zhang *et al* have shown that Chemotaxis due to the consumption of nutrients drives bacteria along the gradient to where it reaches a maximum (the perimeter of the device) [5].

The emergence of resistant mutant bacterial populations occur at locations where the combined stress gradient (antibiotic gradient) and food gradient is at a maximum. The

non-homogenous environment caused by the nutrient and antibiotic gradients generate a steeper population density gradient where fixation is most likely to occur, thereby accelerating evolution in lower population densities, so long as the organisms are motile [2, 5]. This region is called the "Goldilocks point" [2, 3], and can be seen in Figure 5. The figure shows mutant bacteria fixing at two points (green points with white arrows pointing to them in Figure 5 part A). In parts B to D the mutant bacteria are shown to spread out from the Goldilocks points and even invade back towards the center of the device where the bacteria were inoculated. The reason given for the invasion is because of the fitness advantage the mutant *E. coli* have in an environment with a food reservoir and where no other sensitive competitors can live [5].

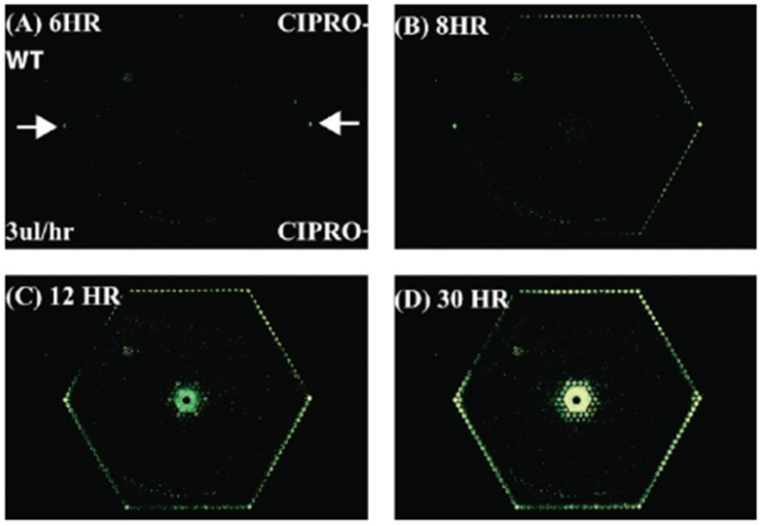


Figure 5: Snapshots of mutant bacteria population densities of the device taken over a 30h timespan. The white arrows point to where the Goldilocks points are. After initial mutation, the mutant type bacteria spread around the perimeter and invade back towards the higher concentration of bacteria in the center [5].

The term "point" may be a slight misnomer because accelerated evolution takes place in a small *region* inside one of the device's demes at the perimeter where the gradients are steepest. The name works because from a distance, a mutant population which fixes in that region will first look like a point on the device.

## 2.14 The Four Single-Nucleotide Polymorphisms Measured

After mutant fixation took place in the experiment, Zhang *et al* examined the genome sequences to understand where mutation took place in the DNA of *E. coli* [2]. The mutated DNA was compared to a baseline sequence of the wild type *E. coli*. Four single-nucleotide polymorphisms (SNPs) mutations in the DNA of *E. coli* were counted in the experiments. The SNPs were likely to be functional and fixed in the population within 10 hours of exposure to ciprofloxacin. A Thymine to Cytosine SNP mutation occurred at the genome DNA gyrase subunit A (gyrA), where ciprofloxacin inhibits the gene function [17]. Another mutation was an Adenine to Thymine mutation for the rbsA gene. This is a component of the ribose ABC transporter complex, and has been reported to export other antibiotics [2]. The third and fourth were Cytosine to Guanine and Adenine to Cytosine at a neighbouring nucleotide sequence in the multiple antibiotic resistant (mar) gene, (marR). The function of marR is to repress the mar operon, and can confer multiple drug resistance by modulating efflux pumps in bacteria (transport proteins involved in removing toxic substances) [2,18]. While the exact order with which the mutation took place is unknown, it is unlikely that the mutations occurred simultaneously [2, 17, 19]. However due to experiments by W. Schröder *et al*, it is possible that gyrA mutation occurred first, followed by rbsA [17]; and then work by H.H. Lee *et al* shows that it is possible that the two marR mutations occurred after that [19].

The program in Chapter 4 makes use of this data to include a series of increasingly fit mutations in the computer model. More information on the mutation rate used in the computer model is described in section 2.17 and chapter 4. The information described here is also used in the following section 2.15.

## 2.15 The SOS Response and Adaptive Mutation

The Bacterial SOS response is a global response to DNA damage, whereby the cell cycle is stopped and DNA repair and mutagenesis are induced [20]. Ciprofloxacin interferes with the gyrA subunit and induces double-stranded breaks in DNA, which in turn activates the SOS response [17], which could explain why this may be the first mutation to occur, as previously mentioned. Adaptive mutation is a collection of phenomena which allows mutations to form in bacteria or cells due to stress and some of these mutations allow for growth to continue to occur [17]. Like the SOS mutagenesis, this implies that evolution can be hastened when the need arises. The effective mutagenic rate of *E. coli* due to the SOS response of Ciprofloxacin is 10000 times greater than the base rate [5].

Recently, the inheritance of acquired traits (IAT) was shown to exist in some species due to the study of epigenetics [21]. A defense mechanism termed Clustered Regularly Interspaced Short Palindromic Repeats (CRISPR) was discovered, which genomes use to protect themselves from phages and other foreign DNA, and that a mutation acquired from CRISPR during a cell's lifetime is non-random and inherited due to IAT [21]. In this sense the mutation is adaptive. However, mutations will cause antibiotic resistance for an individual bacteria, the main mechanism used for acquiring

antibiotic resistance is horizontal gene transfer (HGT) [21,22], whereby genetic information is shared between cellular organisms. In this sense the difference between IAT and random mutation can be difficult to distinguish in the bacterial world. It should also be noted that while CRISPR can have an epigenetic effect on the cell during a stress response, and the CRISPR system as a whole can increase the fitness of the host bacteria, CRISPR has been identified in less than half of prokaryotic, (bacteria lacking membrane bound organelles), genomes studied so far [21]. The cellular process to determine whether or not CRISPR will help increase the fitness of a cell through adaptive mutation can be random. An argument can be put in place that while adaptive mutation is an acute response, the effectiveness and actions of CRISPR as a whole in how it increases fitness for the bacteria can be random and subject to Darwinian Natural Selection.

### 2.16 Reversibility of Antibiotic Resistance

There are several factors which affect the fitness of resistance of a bacterium: growth rate, transmission capacity, transmission dynamics in relation to selection, persistence despite lack of stress factors, adherence factors to the environment, etc. [22]. It is important to note that an acquired mutation in response to antibiotics may impose a fitness cost on a bacterium (measured by a decreased growth rate). This cost can be deleterious to the bacterial strain unless it continues to be exposed to an environment with antibiotic, as resistance to one antibiotic is inclined to be resistant to one or more other antibiotics (called associated resistance) [22]. The fitness cost can also be mitigated by a rapid development of compensatory mutations which can compensate for a decrease in fitness [22]. The nationwide restriction of ciprofloxacin in Israel showed a major decrease in resistant *E. coli* from 12% to 9% in the population in a 6-month time period

[23]. However, the reduction in population may be unlikely due to recent data indicating the fitness cost may be rapidly compensated for due to compensatory mutations thereby increasing the survivability of resistant *E. coli* [24].

### 2.17 Mathematics of Chemotaxis and Diffusion

A model is made as part of this thesis in order to simulate the expansion of bacteria at the Goldilocks point. The model uses the following equations discussed in order to simulate the bacterial dynamics through computation. Further details of the model are described in sections 2.18 and 4.1.

Diffusion is the random migration of particles (or in this case bacteria) due to thermal energy [28,30]. The flux of migration can be expressed using Fick's first law of diffusion [30]:

$$\vec{J}_d = -D\vec{\nabla}N_i \tag{11}$$

where $\vec{J}_d$ is the flux vector to describe the direction bacteria travel, $D$ is the diffusion constant, and $\nabla N_i$ is the population gradient for the number of bacteria. Fick's law states that the flux of particles crossing an area per unit time in a given direction is proportional to the gradient of concentration in that direction [28]:

$$\frac{\partial N_i}{\partial t} = -\vec{\nabla}\cdot\vec{J} = D\nabla^2 N_i \tag{12}$$

here $N_i$ is the number of bacteria of type $i$, which will be explained below under equation 14. An algorithm can be applied to Fick's law to measure a random walk of a particle starting at an origin and travelling in one dimension along an x-axis, and then can be extended to a population whereby it applies to each cell.

In the one dimensional system along an x-axis, for each time step, $\Delta t$, each particle will move in a direction $+\Delta X$ or $-\Delta X$ in equal probability. The distance migrated is proportional to the square root of the diffusion constant, $D$, which depends on the bacteria in question. The distance the bacteria or particle travels in one dimension is what varies and is given by the value, $\Delta X = \sqrt{2D\Delta t} * \lambda$, where λ is a random number generated between 0 and 1, and $\Delta t$ is the constant time step. There is a 50/50 probability of the particle, (bacteria), moving to a neighbouring section or staying in place and the value of λ determines what action occurs. After several time steps (a total time of "$t$") the population of cells along x approaches a Gaussian distribution with a zero mean and a standard deviation of $\sigma = \sqrt{2Dt}$, which also has a mean square [28]:

$$\langle(\Delta X - X_0)^2\rangle = 2D\Delta t = \Delta X^2 \tag{13}$$

Here $X_0 = 0$ due to the zero mean. This equation shows that the diffusion satisfies, $D = \Delta X^2/2\Delta t$.

In addition to diffusion, cells for the simulation in chapter 4 representing bacteria can reproduce and even die. The Fisher-Kolmogorov equation in one dimension is a reaction-diffusion equation which combines the diffusion term with a non-linear population growth factor:

$$\frac{\partial N_i}{\partial t} = D\frac{\partial^2 N_i}{\partial x^2} + g_i N_i\left(1 - \frac{N_i}{K}\right) - dN_i \tag{14}$$

where $D$ is again the diffusion constant, with $g_i$ being the reproduction rate, $d$ is the death rate, and $K$ is the carrying capacity of each position the bacteria are located on [6,31,32]. The variable $N_i$ is the number of $i^{th}$ type E. coli at the $x$ position. Here, the

subscript $i$ describes the type of bacteria (either wild type or mutant), where $i = 0$ is the wild-type, $i = 1, 2, 3, \ldots$ are the first, second, third, etc. mutant types. During replication, there is a probability μ that the offspring of the bacteria will be mutated derived from the work of Allen and coworkers [6]. The part with $\left(1 - \frac{N_i}{K}\right)$ describes how the carrying capacity, $(K)$, restricts the reproduction of bacteria so that the population does not overpopulate the position $x$. The equation describes the reaction-diffusion process for each mutant and wild type bacteria.

Equation 14 predicts the expansion of a population of bacteria with a travelling wave solution with a speed [6,31]:

$$v = 2\sqrt{D(g - d)} \tag{15}$$

which shows that the wave speed $v$ is dependent on the diffusion constant, reproduction rate, and the death rate.

The mobility of bacteria is also influenced by an attractant. As mentioned in section 2.13, chemotaxis can chemically direct the movement of bacteria along a concentration gradient of an attractant [5,32]. In the case of the experiment by Zhang *et al* the attractant is a food gradient [5]. An attractant gradient can increase the flux of cells entering a region [32]:

$$\vec{J}_a = N_i \chi_o \vec{\nabla} a \tag{16}$$

where $\vec{J}_a$ is the flux vector of bacteria entering due to chemotaxis and $\chi_o$ is the coefficient describing the effect the gradient of the concentration for attractant, a, has on bacterial migration. The flux is now a combination of chemotaxis and diffusion, $\vec{J} = \vec{J}_d + \vec{J}_a$.

After combining equations 11 and 16 to give the new flux term, it is then inserted into the Fisher-Kolmogorov equation 14:

$$\begin{aligned} \frac{\partial N_i}{\partial t} &= \vec{\nabla} \cdot \vec{J} + gN_i\left(1 - \frac{N_i}{K}\right) - dN_i \\ &= D\nabla^2 N_i - \chi_o \vec{\nabla} \cdot \left(N_i \vec{\nabla}\mathrm{a}\right) + gN_i\left(1 - \frac{N_i}{K}\right) - dN_i \end{aligned} \tag{17}$$

this equation is called the Keller-Segel-Fisher equation [32,33,34].

Equation 17 can be rewritten into a one-dimensional form given below [34]:

$$\frac{\partial N_i}{\partial t} = D\frac{\partial^2 N_i}{\partial x^2} - \frac{\partial}{\partial \mathrm{x}} \cdot \left( N_i U \left( \frac{\partial \left( \log(a)/\delta \right)}{\partial \mathrm{x}} \right) \right) + gN_i\left(1 - \frac{N_i}{K}\right) - dN_i \tag{18}$$

where $U(z) = \frac{2\chi_o}{\pi}\arctan(z)$, and $\delta$ is a chemotactic response variable describing how effective the concentration of attractant, *a*, is at displacing the bacteria.

The attractant is a chemical and diffuses; it can also be produced by the bacteria:

$$\frac{\partial a}{\partial t} = D_a \nabla^2 a + f(a, n) \tag{19}$$

which is Fick's law for the diffusion of attractant [32]. Here $D_a$ is the diffusion coefficient of the attractant, and $f(a,n)$ is the kinetics/source term which describes the rate attractant is produced by bacteria or naturally decays over time.

The wave speed of bacteria due to the Fisher wave speed combined with the chemotaxis equation of an attractant gradient is then:

$$v = 2\sqrt{D(g-d)} \pm \frac{2\chi_o}{\pi}\arctan\left(\frac{1}{\delta\sqrt{D_a}}\right) \tag{20}$$

As previously mentioned, the chemotactic response is a measure of how effective the attractant is at displacing the bacteria, so if the chemotactic response becomes negligible, then $\delta \to \infty$ and the effect the attractant has at stimulating bacteria to travel goes to zero; the propagation speed reverts back to the speed in equation 15. However if $\delta \to 0$ the chemotactic response is non-negligible and equation 20 better represents the wave speed.

### 2.18 Stochastic Simulation Algorithms (SSA)

Some other ways to simulate bacterial population dynamics is through the use of Gillespie algorithms as well as Stochastic Simulation Algorithms (SSA) [25]. A lower abundance of bacteria allows for the simulation to track all of them as individual particles, while a high abundance would require the Gillespie algorithm [25-27]. The Gillespie algorithm was originally designed to simulate coupled (bio)chemical reactions within a thermal bath in a well-mixed environment [27]. Stochastic models (including the Gillespie algorithm) provide a more detailed understanding of reaction-diffusion processes compared to more deterministic models, and are necessary when biological phenomena depend on stochastic fluctuations [28]. The Gillespie Algorithm generates a

time step based on a continuous distribution $\theta(\tau) = N\lambda e^{-N\lambda\tau}$, where $N$ is the total number of particles, and $\lambda$ is the population mean rate of the set of processes. A computer program goes through cycles called iterations whereby it uses an algorithm to determine the behavior of the particle selected during that iteration. For the Gillespie algorithm a particular event can modify the number of events that can occur for the next iteration [26,27]. A SSA model was made as part of this thesis in order to simulate the expansion of bacteria at the Goldilocks point using equation 14 described in the previous section, (section 2.17). There are additional relations added to the growth rate derived from the work of Allen and coworkers which will be further discussed in the following section, (section 2.19) [6,29]. Models based on a low abundance of bacteria are computationally less demanding and an existing SSA used for measuring the Brownian/Smoluchowski dynamics can be used [25, 28].

The model made for the thesis has a lower abundance of bacteria and so is computationally less demanding. Thus, a SSA used for measuring the Brownian/Smoluchowski dynamics is used instead of a Gillespie algorithm. A more detailed description on how the model's algorithm functions is discussed in the later part of section 4.1.

### 2.19 Goldilocks Point Modeling

Recent modeling by Allen and coworkers has shown that in the presence of drug gradients waves of increasingly fit mutants advance and colonize towards regions with higher drug concentrations in a stepwise manner, as seen in Figure 6 [6]. Fixation can be

accelerated if the mutation rate is high enough, the organisms are more motile, and the population density gradient is steep (i.e. a steeper antibiotic gradient) [5, 6].

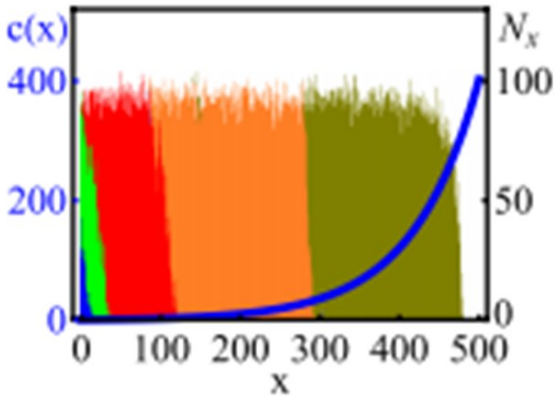


Figure 6: A 1D model of mutant bacteria progressing into an antibiotic region. The Blue line shows the concentration of antibiotic. The different colours represent the waves of increasingly fit mutants colonizing toward regions with higher drug concentration [6].

While the model didn't have a food gradient, the reproduction rate of bacteria was proportional to the concentration of antibiotic, and motility was kept to a constant rate. The function used to model how the population of bacteria travels is the Fisher wave in equation 15. There is a constant mutation rate represented by the symbol μ, which describes the probability of mutation every time a bacteria replicates. The expansion of waves of mutants are described by the Fisher-Kolmogorov equation 14 except some of the variables are specific to that mutation, $(N_i, g_i)$, as described in section 2.17. The model shows that the antibiotic spatial gradient generates local "niches" where there is lower drug concentration to allow less resistant bacteria to survive. This allows waves of increasingly fit mutant to colonize regions of higher drug concentration; however, the model does not analyze what impact a food gradient has on fixation and motility.

The calculations by Allen and coworkers showed that using a sequence of increasingly fit mutants is one factor involved in accelerated fixation, but these results were contrasted with a fitness valley, as previously described in S. G. Wright's fitness landscape [6, 9]. Instead of a sequence of increasingly fit mutants, one of the intermediate genotypes had a lower minimum inhibitory concentration (MIC) than its neighbouring genotype, (the concentration of antibiotic needed to inhibit growth of the bacteria) [6]. The model showed that a non-uniform drug distribution does not speed up resistance, and may even slow down the emergence of resistant bacterial fixation if one of the mutants had a lower fitness value than the previous mutant it evolved from. This may better represent the fitness landscape of streptomycin, which has a fitness valley, and so an antibiotic gradient may actually slow down resistance [35]. The computer model discussed in chapter 4 is derived from this work by Allen and coworkers [6].

## 2.20 The Ising Model

As part of this thesis, a model was constructed to simulate the microfluidic array. The model discussed in Chapter 3 and uses the Ising model. The Ising model consists of a lattice of identical particles which exchange electron spins between individual sites and particles neighbouring themselves. This is a square model so there are four neighbours arranged horizontal and vertical to a center element. Different configurations of the selected element and its neighbours make up different microstates. The original purpose of this model was to study ferro- and paramagnetic phenomena. The electron spins can have either a spin up value ($\sigma = +1$) or a spin down value ($\sigma = -1$) [36, 37].

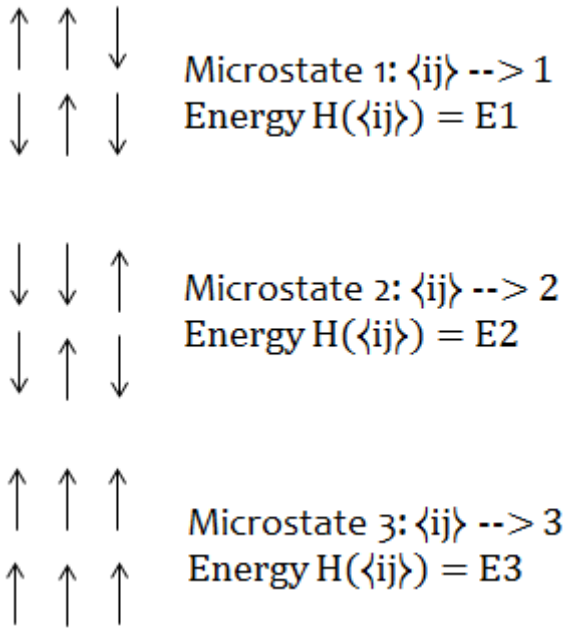


Figure 7: Different spin microstates with their associated Hamiltonian Eigenvalues. The example is meant to show how different combinations of spins on the lattice can form different microstates. The example does not show four neighbours arranged horizontal and vertical to a central atom, as is the case of the Ising model.

Different configurations of nearest neighbour pairs (with values $ij$) make up different microstates, denoted with the symbol $\langle ij \rangle$. The Hamiltonian of different microstate configurations have their associated Eigenvalues ($H(\langle ij \rangle) = E_{\langle ij \rangle}$), which denote the energy for that microstate. For a selected spin state ($\sigma_i$) the Eigenvalue of the Ising model is a product from the spin and its nearest four neighbours, which is given by the Hamiltonian function [36, 37]:

$$H(\sigma_i) = -\frac{J}{2}\sum_{\langle ij \rangle} \sigma_i \sigma_j = E_i \tag{21}$$

where $J$ is the exchange constant which describes the energy of interaction between the two elements. If $J$ is positive then the exchange energy favours parallel spins because a

lower energy state is more favourable, and so $E_i < 0$. This type of exchange interaction when the spin moments are aligned is called ferromagnetic [36].

## 2.21 The Markov Chain Monte Carlo Technique

This simulation of the Ising model uses the Monte Carlo (MCa) method to generate pseudorandom numbers to directly simulate representative probability distributions of the system. The MCa method generates states of a finite system with a weight proportional to the probability density of the Hamiltonian, ($P_i \propto e^{-\beta E_i}$) [37, 38]. There are no dynamics; instead the program goes through cycles called iterations whereby it uses an algorithm to generate a new configuration for the next iteration.

In this case a Markov Chain Monte Carlo (MCMC) technique uses the Metropolis algorithm discussed in section 2.6 to randomly select a spin on the array [12]. It then calculates the change in energy of shifting from a trial state compared to the initial state selected. The Metropolis algorithm follows the acceptance ratio shown in equation 9, so if $\Delta E < 0$ the spin automatically shifts to a new state, otherwise there is a probability of $P = e^{-\beta \Delta E}$ of the spin shifting anyway. The Metropolis does this for every iteration and calculates the acceptance ratio with the trial states available in order to shift towards the most probable state [37, 38]. Using the MCMC technique one can obtain a sample of the desired distribution by observing the chain after a number of steps. The more steps there are, the more closely the distribution of the sample should match the actual desired distribution. As was mentioned in section 2.2, the desired distribution is for the system to converge to equilibrium. In this case of the Ising model, a shift in energy is associated with a spin flip:

$$\Delta E = E_{final} - E_{initial} = \left(-J(-s_i)\sum_{j=1}^{q} s_j\right) - \left(-Js_i\sum_{j=1}^{q} s_j\right) = 2\,J\,s_i\sum_{j=1}^{q} s_j \qquad (22)$$

Here $\Delta E$ is associated with the shifting of energy from one spin configuration (either +1 or -1) to the opposite, and $q$ is the number of neighbours. Since $J > 0$ it is more energetically favourable to select parallel spins, (because $\Delta E < 0$). The computer simulation in chapter 3 uses the MCMC with the Ising model.

## 2.22 Shannon Entropy

Shannon entropy is a measure of the uncertainty of the final message (or outcome of a process in general). A larger Shannon entropy value means that a process has more uncertainty. The expression of the Shannon entropy is connected to the thermodynamic entropy, and has its same expression in the context of statistical mechanics:

$$\Delta S = -\sum_{E} P(E) ln P(E) \qquad (23)$$

Here the probability of being at a particular energy state is given by $P(E)$. Shannon entropy reaches a maximum when the uncertainty is maximum (in this case a greater shift in energies). The larger the Shannon entropy, the more equiprobable the possible outcomes in an experiment and the greater the spread of the probability distribution. For an energy probability distribution Shannon entropy has its largest value when the energy values are equally probable and there are a greater number of energy states with appreciable probability. Shannon entropy can allow for the most appropriate probability distribution for certain information to be calculated [10, 43]. Some of the results discussed in chapter 3 use Shannon entropy of equation 23.

## Chapter 3

## Ising Type Bacterial Evolution Goldilocks Model and Results

### 3.1 The Hamiltonian and Program

To simulate the experiment performed by Zhang *et al* [2,5] the Ising model system described in section 2.20 is used. Unlike the Hamiltonian function in equation 21, a modified Hamiltonian function is used to describe the energy (or possible fitness) of the bacteria. One thing of importance to note is that the sign of the energy used is the same used in statistical physics to describe energy. Therefore the sign of a more favourable energy level will be negative, (which may mean a higher fitness peak in certain microstates).

The lattice model is not infinite, but took the form of a finite 100 x 100 square grid array. Unlike the hexagonal device used by Zhang *et al* [2,5], the computational model uses a square grid based off of the Ising model. Like the Ising model described in section 2.20, a selected spin on the array has four neighbours with two neighbours arranged horizontally and two neighbours arranged vertically. The combinations between the central spin and its neighbours make up different microstates. The Hamiltonian equation of the model contained spins represented by the sigma symbol $\sigma$. The modified Hamiltonian function contained the Ising term in equation 21 as well as four others terms to describe the energy of a particular microstate:

$$E = -\frac{J}{2}\sum_{ij}\sigma_i\sigma_j + J_d\sum_{i=1}\sigma_i{}^2 - \sum_{ij}\frac{J_f}{2} * \sigma_i{}^2\sigma_j{}^2 + \sum_{i=1}J_c * \sigma_i$$
$$- A * \left[\sum_{ij} e^{-\frac{J_{fmax}}{kT}+\frac{J_f}{kT}+0.095}(1-\sigma_i)(1-\sigma_j)\right] * \left[\prod_i(\sigma_i{}^2)\right],\ A < J \tag{24}$$

where the Eigenvalues between the central spin, $\sigma_i$, and its nearest neighbours, $\sigma_j$, describe the energy for that particular microstate. The equation contains the Ising term, a death term, a term describing growth from food, an antibiotic term, and a mutant food affinity term. The terms will be described more in detail in the following sections. In this equation $A$, $J$,and $J_d$ are constants, while $J_f$ and $J_c$ are variables, whose meanings will be given later when describing the physical meaning of each of the various terms of the equation in subsequent subsections. For this model the Hamiltonian and spin microstates selected represent the state of a central deme and the influence by its neighbouring demes, whereby the relationship between demes is what influences the energy values of the Hamiltonian.

Unlike the classic Ising model, the spin values (or demes) for this system can have three different values. When $\sigma = +1$ this represents a deme which is fixed with a population of wild type *E. coli*. If $\sigma = 0$ this represents a system which is absent of bacteria. A final value of $\sigma = -1$ represents a system which has been fixed by mutant bacteria. These integer values don't take into account population, but taking into account the association between statistical mechanics and evolutionary biology, the value of the Boltzmann constant and temperature would be equivalent to the stochastic drive of the system. This relation is described in sections 2.8 and 2.10, with the value of the

stochastic drive having the value $kT = \varepsilon = 0.2$. As well, the value can be proportional to the inverse of the deme's population size, proportional to 5 bacteria.

The program was written using MATLAB and the model is a Markov Chain Monte Carlo (MCMC) simulation using a Metropolis algorithm described in sections 2.6 and 2.21. The program containes a 100 x 100 array. Initially all the spins are zero except for four neighbouring +1 spins at the center of the model, to represent wild type bacteria inoculated into the center of the device. For each iteration one spin is selected at random on the array. The acceptance ratio of equation 9 calculates the change in energy of the Hamiltonian of equation 24 by flipping the central spin state to another value, where $\Delta E = E\left(\sigma_i^{final}\right) - E(\sigma_i^{initial})$. If $\Delta E < 0$ the spin automatically shifts to a new state; otherwise, there is a probability of $P = e^{-\beta \Delta E}$ of the spin shifting anyway. The terms in equation 24 control whether a selected element, (deme), on the array behaves. An empty element can switch to wild type or mutant, or just stay the same. An element with a wild type/mutant can flip to a mutant/wild type, or also stay the same. How the terms of the Hamiltonian in equation 24 dictates the behavior of an element using the Metropolis algorithm is described in the following sections. When selecting a spin on the array, the row and column are recorded for simplifying calculations. A function called "circshift" is used to shift the neighbouring spins towards the same location as the central spin but on a separate array. This can be seen in Figure 8 below.

```
% Find its nearest neighbours
% A = to the right [0 1]
A = circshift(spin, [0 1]);
% B = to the left [0 -1]
B = circshift(spin, [0 -1]);
% C = Shift down
C = circshift(spin, [1 0]);
% D = shift up
D = circshift(spin, [-1 0]);

    neighbours = A + B + C + D;
    neighbours_sqrd = A.*A + B.*B + C.*C + D.*D;
    OneMinusNeighbour = (1 - A) + (1 - B) + (1 - C) + (1 - D);
```

Figure 8: Circshift used to move the neighbouring spins. Here "neighbours" is a summation of the four neighbours used in the Ising term described in section 3.2. As well, "neighbours_sqrd" is a summation of the neighbours squared, used by the food term discussed in section 3.4. The last term "OneMinusNeighbour" is used by the food affinity term discussed in section 3.11.

A product of the neighbouring spins is calculated for use in the energy shift calculation on a separate array, $\Delta E$. The positions of the central spin on the main array and the product of neighbouring spins on the separate array are the same using the recorded row and column. The shift in energy, $\Delta E$, is calculated using the spin selected on the main array and the product calculated on the other array. For the death and antibiotic terms discussed in subsequent sections, only the shift in spin is needed. Because the row and column were recorded initially, the shift in energy of only that location is calculated thereby reducing the time. There are a large number of energies for the microstates due to the different combinations between the selected spin and its neighbours. Therefore, identical microstates may not have the same energy depending on where the states are located on the array. The food, ($J_f$), and antibiotic, ($J_c$), symbols are variables which also change depending on the position on the array. More will be discussed about $J_f$ and $J_c$ in sections 3.6 and 3.8.

The purpose of this simulation is to bring an initial system to an equilibrium state using the Hamiltonian of equation 24. The Hamiltonian is designed so that in the process of shifting the system to the desired state, the simulation resembles what was observed in the experiment by Zhang *et al* which showed a shift in the genetics of the population by accelerated mutation at the Goldilocks point [2,5]. Therefore, the values of $A, J, J_d, J_f,$ $J_c$, and $kT$ which are discussed in subsequent sections are selected to resemble the experiment as the program uses the Metropolis algorithm to bring the system to equilibrium. More information of the parameters are discussed in the conclusion of section 3.12.4.

### 3.2 The Ising Term: Wild-Type/Mutant Produces Wild-Type/Mutant

The first term discussed is the Ising term, as previously seen in equation 21. The exchange value for a selected element is constant and the term can be rearranged:

$$E_{Ising} = -\frac{J}{2}\sum_{ij}\sigma_i\sigma_j = -J\sigma_i\sum_{j=1}\sigma_j \tag{25}$$

Here, $ij$, represents a sum over nearest neighbour pairs, and if the selected elements have the same interaction strength the Hamiltonian is symmetric. This symmetry is what allows the spin of the central element, $\sigma_i$, to be taken out of the summation along with the ½ factor which is meant to prevent double counting. The exchange constant, $J$, is positive and has a value of 1.86, in order for the simulation to resemble what was observed in the experiment by Zhang *et al*, as discussed at the end of section 3.1 [2,5]. The purpose of the term is to support "like equals like" by shifting the energy to be positive or negative depending on the neighbours present. If a majority of the neighbours

have the same sign, then a flip in the spin of the central element will lead to a positive energy value and so this shift will not be as favourable. The selected spin (or deme) is more likely to flip if it is surrounded by demes with the opposite sign, with $\Delta E < 0$. This can be seen in Figure 9.

$$\begin{matrix} -1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 0 \end{matrix} \quad \rightarrow \quad \begin{matrix} -1 & 1 & 1 \\ 1 & +1 & -1 \\ -1 & 1 & 0 \end{matrix}$$

$$E_{initial} = -(-1) \cdot J \cdot (1 + 1 - 1 + 1) = +2J$$

$$E_{final} = -(+1) \cdot J \cdot (1 + 1 - 1 + 1) = -2J$$

$$\Delta E = -2J - 2J = -4J$$

Figure 9: Shift in energy from flipping the selected spin from -1 to +1. The change in energy between microstates is negative so the shift is favourable and likely to occur.

In the case of no neighbours or an empty element selected then the product will be zero, ($\sigma_i \sigma_j = 0$). The purpose of "like equals like" is to simulate bacterial competition. A wild type population will try to fight off a neighbouring mutant population and vice versa to try and flip the opposing deme.

### 3.3 The Bacterial Death Term

The second term in equation 24 is the death term. This term deals with the selected central element and neighbours individually and is not a product between the two:

$$E_{death} = +J_d \sum_{i=1} {\sigma_i}^2 \tag{26}$$

where $J_d$ is the death constant. The constant is positive and has a value of 5.95 which is larger than the other constants and variables in equation 24. The value is set to help resemble the experiment by Zhang *et al* [2,5], and to help the death term counteract other terms in the equation which are designed to help with replication or to keep bacteria alive. Because the term does not change in relation to the neighbours present, it gives each deme selected a constant value of $J_d$. The spin selected is squared so that the sign is always positive to increase the energy and make death more favourable for both types of spins, as can been seen in Figure 10.

$$\sigma_i = 1 \rightarrow \sigma_i = 0$$
$$\Delta E = -J_d$$
$$\sigma_i = -1 \rightarrow \sigma_i = 0$$
$$\Delta E = -J_d$$

Figure 10: The shift in energy from the cell death term of a deme shifting from occupied to vacant. The change in energy is negative making the shift more favourable. Because the neighbours are unchanging, their energy values cancel out when the central element shifts in energy. The shift in energy affects both type of spins equally.

The shift in energy from an occupied deme to a vacant one is negative so that the element will flip to zero. Making the shift more energetically favourable makes death more probable, so if there are no like neighbours or food present the occupied element will flip to zero.

### 3.4 The Food Term

The third term in equation 24 is the food term which helps increase the probability that bacteria will grow and a deme remain occupied. The term functions similar to the Ising term except that the central element and its neighbours are squared:

$$E_{food} = -\sum_{ij} \frac{J_f}{2} * {\sigma_i}^2 {\sigma_j}^2 = -J_f {\sigma_i}^2 \sum_j {\sigma_j}^2 \tag{27}$$

where $J_f$ represents the concentration of food and varies depending on the location on the array. The equation is symmetric and can be rearranged like in equation 25. More information will be discussed about the food exchange variable in section 3.6. The section will also explain why the exchange variable may be moved out of the summation in equation 27.

The spins of the food term are squared. This is so that bacterial growth is supported for both types of bacteria regardless which type of bacterial population is fixed in a neighbouring deme. In a neighbouring deme there should still be identical bacteria present even if they have not fixed in that deme's population. The food term works against the death term so long as there are neighbours present. Because there is a continuous supply of food provided, the term is larger when there are more neighbours present due to a larger supply of bacteria to colonize the selected deme because of migration from neighbouring sites. If the system is closed and there is a limited food supply then the rate of growth would eventually decrease with an increasing bacteria population consuming the limited food supply. This would make any neighbours present decrease the probability of growth. The dependency of neighbours also helps to prevent spontaneous growth, because there needs to be a supply of bacteria from a neighbouring

deme for the food term to work. If there are no neighbours then the selected deme remains zero or uncolonized.

### 3.5 Food and Ising Term Relation

The food and Ising terms are made so that they can either work with or compete against one another. This depends on the sign of the neighbouring demes present as can be seen in Figure 11. If it is the case that half of the neighbours are like and half are unlike, then the Ising term will go to zero.

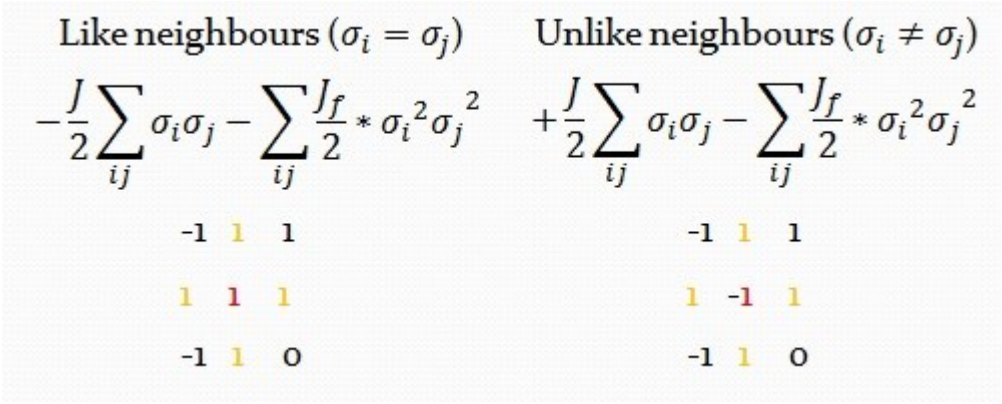


Figure 11: The food and Ising terms can work with or compete with one another depending on the sign of the majority of neighbouring demes. The left side shows the Ising term as negative because the neighbours are the same, and the Ising term on the right side is positive because neighbours are opposite.

If there are like neighbours present the Ising term works with food, but if there are unlike neighbours present the Ising term now counteracts the food term. This is because neighbouring demes with unlike bacteria present will provide the central selected deme with opposite bacteria, thereby making it more likely to shift the energy upwards due to bacteria competition. So the food term is less likely to counteract the death term.

### 3.6 Food Exchange Value Gradient

The food term contains an exchange value in front which increases linearly from the center towards the perimeter of the model. As mentioned before, $J_f$ represents the food concentration of the device by Zhang *et al* [2,5], and so the linear increase represents the food gradient.

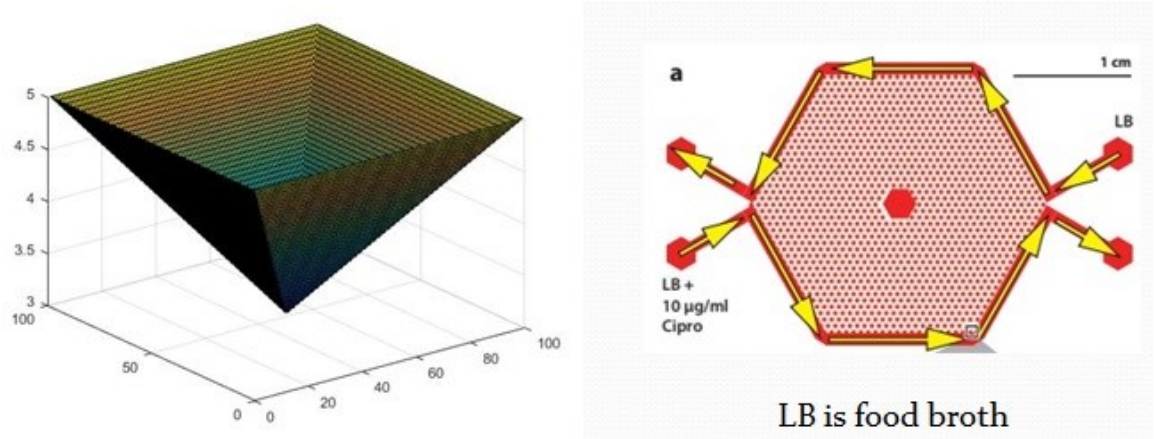


Figure 12: The image on the left shows the food gradient using the value of $J_f$ on the array. The food exchange value is lowest in the center and linearly increases toward the perimeter where it reaches a maximum. The figure to the right is an image of the device used by Zhang *et al* [3]. A food broth is circled around the perimeter of the device whereby it flows towards the center to form a gradient of food concentration. The simulation on the left is a square grid model in order to represent the hexagonal array on the right.

As can be seen in Figure 12, the value of the food exchange symbol varies ($3.0 \leq J_f \leq 5.0$). The value is chosen for the same reason mentioned in section 3.1, whereby $J_f$ is selected to resemble the experiment by Zhang *et al* as the program uses the Metropolis algorithm to bring the system to equilibrium [2,5]. The parameter is chosen also to allow growth for the bacteria inoculated in the center and then so the food gradient will encourage more rapid growth towards the perimeters of the device. The experiment by Zhang *et al* states that chemotaxis due to the consumption of nutrients at low flow

rates encourages bacteria to migrate toward the perimeter of the device [5]. This is simulated in the model with the increasing food gradient. Near the center, bacterial growth is slower due to a lower $J_f$ value; however, as the bacteria begin to grow and migrate, the food gradient increases the rate at which bacteria move toward the perimeter of the device.

One thing important to note in equation 27 is that even though the food exchange value is a variable and depends on where in the array it is located, it can still be moved out of the summation. This is true even when the value of $J_f$ changes between the central spin and its nearest neighbour due to the food value being negligibly small or there is no change at all ($\Delta J_f/2 = \left|J_{f_{\sigma_j}} - J_{f_{\sigma_i}}\right|/2 = 0, 0.02$). In the case when there is a shift in the food exchange value along an increasing gradient, because opposing neighbours are arranged parallel to one another, the shift in food terms balance each other out and equal $J_f$ of the central deme selected. In the rare case at the boundary when there is no opposing neighbour, then the total shift in $J_f$ for the system can be 0.02, where $\sum_{ij} \frac{J_f}{2} {\sigma_i}^2 {\sigma_j}^2 \cong J_f {\sigma_i}^2 \sum_j {\sigma_j}^2$. In the vast majority of cases on the array, equation 27 holds true. In the case of a model with an exponentially increasing gradient this may not hold true depending on how steep the gradient is.

### 3.7 The Antibiotic Term

The fourth term is the antibiotic term. It is similar to how the death term functions in equation 26, but unlike the death term the energy depends on the type of bacteria present:

$$E_{Cipro} = +J_c \sum_{i=1} \sigma_i \tag{28}$$

where $J_c$ is the antibiotic term's variable to describe the concentration of antibiotic. The energy changes in relation to the type of bacteria present (sign of the spin) to describe the effect the antibiotic has in the selected bacteria's fitness.

$$E_i = +J_c\sigma_i$$

$$\sigma_i = 0 \rightarrow \sigma_i = 1 \quad \Delta E = +J_c \qquad \sigma_i = 0 \rightarrow \sigma_i = -1 \quad \Delta E = -J_c$$

$$\sigma_i = 1 \rightarrow \sigma_i = 0 \quad \Delta E = -J_c \qquad \sigma_i = -1 \rightarrow \sigma_i = 0 \quad \Delta E = +J_c$$

Figure 13: The shift in energy of the antibiotic term. The energy of the selected deme depends on the sign of the spin. When spin = 1, the probability for wild type bacteria to die is encouraged and growth is discouraged. For mutant bacteria with spin = -1, growth is encouraged and death is discouraged.

If the wild type bacteria has fixed in the selected deme's population, (if the spin of the element is 1), then the antibiotic term is positive. The positive term increases the energy thereby making it more likely for the bacteria to die and shift the element to zero. The increase in energy also makes it less likely that the bacteria will grow in that region with antibiotic. If mutant bacteria have fixed in the deme's population, the term will be negative and the shift to lower energy will discourage the bacteria from dying. This would also encourage the mutant bacteria to grow in the selected deme. Like the food exchange value, the antibiotic value, ($J_c$), is a variable and the concentration of antibiotic depends on the location on the array.

### 3.8 The Antibiotic Gradient

The antibiotic value represents the concentration of antibiotic in the device. The value increases linearly from the center towards the perimeter of the model, just like the food exchange value. Unlike the food variable however, the antibiotic variable is only present on the lower right diagonal half of the model.

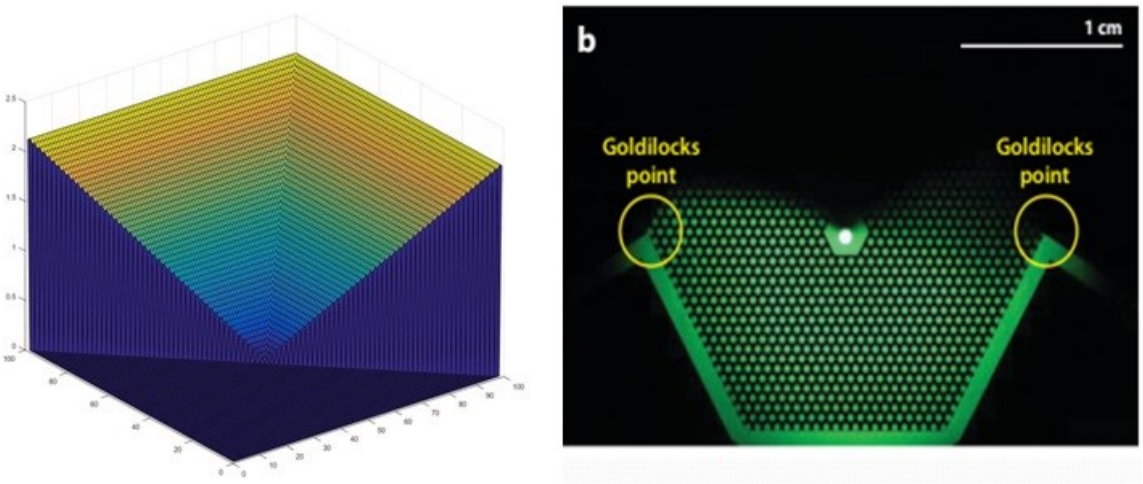


Figure 14: The antibiotic concentration of Ciprofloxacin in the model is shown on the left. The gradient increases linearly from the center to the bottom and right perimeters but the antibiotic value is not located on the upper left diagonal. The blue colour represents a share discontinuity where the concentration goes to zero, and the black region is top left diagonal where the concentration is zero. The antibiotic gradient in the device used by Zhang *et al* [3] is on the left. The gradient is only on one half of the device, just like the Ising type model. In this case the concentration of antibiotic is on the bottom half of the hexagonal array, like it is on the bottom right diagonal on the square model.

The antibiotic value increases linearly as can be seen in the figure and is given by the range: $0 \leq J_C \leq 2.115$. This is for the same reason mentioned in section 3.1, whereby $J_C$ is selected to resemble the experiment by Zhang *et al* as the program uses the Metropolis algorithm to bring the system to equilibrium [2,5]. Due to the antibiotic term in equation 28, the concentration of antibiotic near the center of the device is not enough to kill off the wild type bacteria, but it does help to prevent the wild type bacteria from

migrating closer to the bottom and right perimeters where the antibiotic concentration becomes greater. There is a sharp discontinuity of antibiotic along the diagonal. While this is not realistic, the gradient of antibiotic used in the device by Zhang *et al* showed a rapid drop shift in antibiotic around the center half of the device [3].

### 3.9 Bacterial Migration

The food term helps drive bacteria towards the perimeter of the device, although this model has indirect migration. The model is binary in so far as each deme can either be occupied or unoccupied and does not indicate the population size for each element on the array. As a result, the diffusion of bacteria into a neighbouring deme is an indirect result of the Hamiltonian flipping a spin to occupied. Whether or not an empty deme is populated depends on if a neighbouring deme is populated by bacteria. If during an iteration an empty spin is selected then the terms of the Hamiltonian are constructed in a way to increase the probability that the deme will flip to an occupied state depending on whether or not there is a neighbouring occupied deme present.

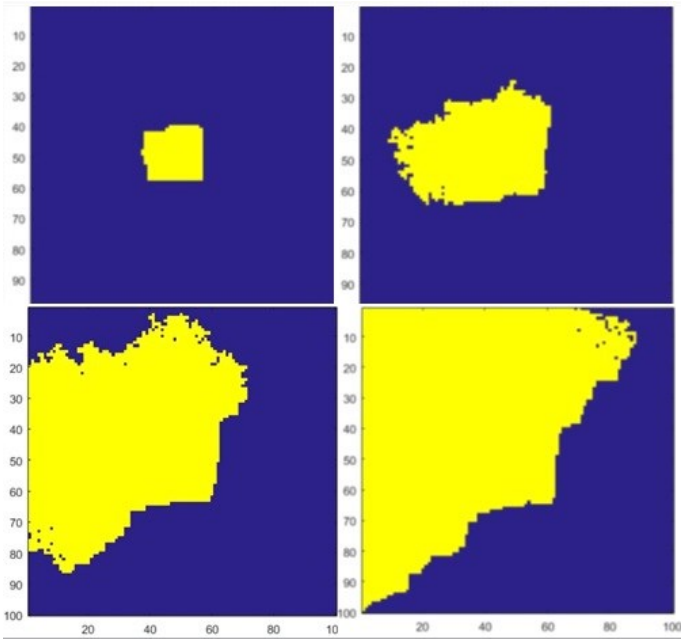

Figure 15: Stages of bacterial migration during the simulation. The empty regions are blue and the wild type occupied regions are yellow. The top left image shows bacteria beginning to grow at the center of the device at around $4.40 \times 10^6$ iterations. The top right image next shows the bacteria beginning to grow and migrate at a more rapid pace as they move their way along the food gradient at around $4.75 \times 10^6$. The bottom left image shows that as bacteria migrate the antibiotic gradient prevents growth along the bottom left diagonal at around $4.90 \times 10^6$. The bottom right image shows bacteria moving towards two corners of the device at around $4.98 \times 10^6$. $J = 1.86$, $A = J/2$, $J_d = 5.95$, $3.0 \leq J_f \leq 5.0$, $0 \leq J_C \leq 2.115$, and $kT = 0.2$. The colour scheme is automatically set by MATLAB.

Initially the bacteria begin to grow out from the center at a slow pace. The antibiotic term and gradient helps prevent the bacteria from migrating towards the bottom and right perimeters. As the bacteria are drawn towards the top and left perimeters the concentration of food increases. As the concentration of food increases the rate at which the bacteria will move and grow towards the perimeters increases. The rapid growth drives bacteria to the bottom left and top right corners of the model where the antibiotic term is at its highest. The lack of antibiotic in the upper left diagonal gives the wild type

bacteria a pathway towards the two corners, which will be the Goldilocks points of the model.

### 3.10 Mutant Growth

The food term does allow for the bacteria to migrate slowly into regions of high antibiotic if $J_f$ is high enough. This can simulate how the presence of drug gradients allows the population to evolve drug resistance in a sequence of waves of increasingly better adapted mutants to expand into regions of antibiotic in a step by step manner [2]. However in this case the model is binary when it comes to the type of bacteria, (+1 or -1, not including 0), and there is only one type of mutant. The food and antibiotic variables function like an applied field in the Ising model and can influence whether or not a spin will flip, or in this case whether or not a wild type bacteria will evolve (spin = 1 to spin = -1). While mutation is binary (spin = 1 to spin = -1), the simulation also allows for an empty term to become populated with wild type bacteria (spin = 0 to spin = -1). This is what is seen at the Goldilocks point, whereby neighbouring wild type demes allow for mutant bacteria to advance and colonize the Goldilocks point.

$$E_i = -J\sigma_i \sum_j \sigma_j + J_d {\sigma_i}^2 - J_f {\sigma_i}^2 \sum_j {\sigma_j}^2 + J_c \sigma_i$$

$$\sigma_i = 0 \rightarrow \sigma_i = -1, \quad \sigma_j = 1$$

$$\Delta E_i = +J \sum_j \sigma_j + J_d(1) - J_f \sum_j {\sigma_j}^2 + J_c(-1)$$

Figure 16: The terms used to determine whether or not bacterial mutation takes place to flip the spin from 0 to -1. At the Goldilocks point flipping the spin to a mutant deme is most probable so the migration of bacteria from neighbouring demes will be mutant as opposed to wild type, instead of wild type migrating and flipping the deme to +1 with subsequent mutation to -1. The food term works with the antibiotic term to flip the spin.

The Ising term opposes mutation because a mutant spin is unlike the wild type. The death term helps to prevent growth of any bacteria.

At the perimeter the values of $J_f$ and $J_c$ are at their greatest, and due to this it is most probable for a deme selected at a Goldilocks point to flip from 0 to -1. As can be seen in Figure 16, the Ising term and death term work against bacterial mutation, however there are fewer opposing neighbours at the corners. Because of this the Ising term will not be as large at the corner compared to a location with multiple neighbours present. This is what encourages mutants to fix at the Goldilocks points located at the top right and bottom left corners.

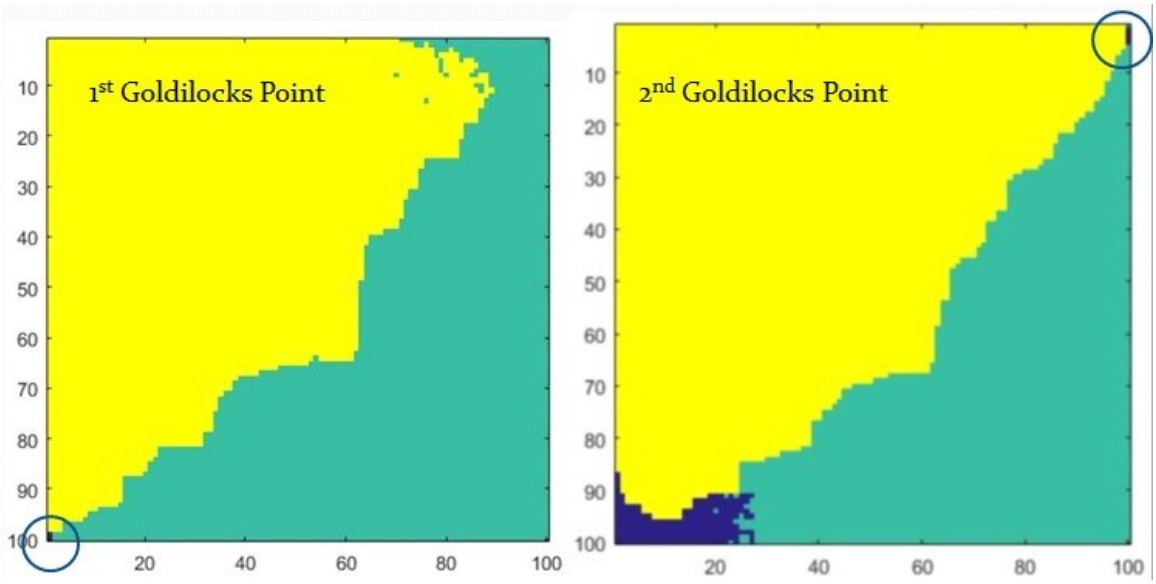

Figure 17: The two Goldilocks point mutations taking place in the corners of the model. After the spin flip the mutant bacteria spread into regions with a higher concentration of antibiotic. The wild type bacteria are yellow, the empty spaces are green, and the mutant bacteria are blue. $J = 1.86, A = J/2, J_d = 5.95, 3.0 \leq J_f \leq 5.0, 0 \leq J_C \leq 2.115$, and $kT = 0.2$. The colour scheme is automatically set by MATLAB.

Because the antibiotic term gives a fitness advantage to the mutant bacteria, after mutation takes place at the Goldilocks points the mutant bacteria spread into regions with a higher concentration of antibiotic.

### 3.11 The Food Affinity Term

The experiment by Zhang *et al* observed that the mutant type bacteria may have a food fitness advantage where they state "The basic reason for … [the] invasion of resistance is the huge fitness advantage for mutant resistant E. coli in a microenvironment where the food reservoir is nearby and no other sensitive competitors can live" [5]. The invasion of mutant bacteria along the perimeter of the device and into wild type populations can be seen in Figure 5. This may be a compensatory mutation as described in section 2.16. The huge fitness advantage helps mutant *E. coli* to populate and spread along the regions on the perimeter of the device where the concentration of food is highest [5]. This allows the mutant bacteria to fix into regions previously with wild type bacteria even if it is in a region with a low concentration of antibiotic. A final term is made to represent the food fitness advantage the mutant bacteria have at regions with a high food concentration:

$$E_{Affinity} = -A * \left[ \sum_{ij} e^{-\frac{J_{fmax}}{kT} + \frac{J_f}{kT} + 0.095} (1 - \sigma_i)(1 - \sigma_j) \right] * \left[ \prod_i (\sigma_i{}^2) \right] \qquad (29)$$

where the constant $A$ is given a value of $J/2$ so that the food affinity of the mutant *E. coli* can allow it to compete against the Ising term of an unlike wild type E coli deme. While the final part of the term is based on the Adair model for the cooperative binding of haemoglobin [45], the other parts of this term are made for this model.

The exponential section of the term allows for it to become more intense the closer the selected deme is towards the perimeter of the model. $J_{fmax}$ is the maximum value of the food gradient on the perimeter, while $J_f$ is the food variable which is

dependent on its position in the model. At the perimeter $J_{fmax} = J_f$, and so then $e^{0.095} \approx 1.1$, which allows the affinity term to have an effect in the Hamiltonian. If a deme is not selected at the perimeter then the exponential section will make the affinity term negligibly small.

The Ising part of the term helps favour the mutant bacteria. Like the Ising term in equation 25, the Ising part of the affinity term is a product of the selected deme and its neighbours, $A\sum_{ij}(1-\sigma_i)(1-\sigma_j)$. Unlike the Ising term this part will be zero if the bacteria are wild type, ($\sigma = 1$). A negative mutant, ($\sigma = -1$), will be favoured because then $1 - \sigma = 2$. The term however favours the growth of the mutant type even when there are not bacteria present ($\sigma = 0$). This major flaw would favour spontaneous growth of mutant at the perimeter even without neighbouring occupied demes present. To counteract this problem a final part of the term was put in to check to see if all the neighbouring demes have bacteria present. This check is based off of the system used in the Adair model for cooperative binding of oxygen to haemoglobin [45].

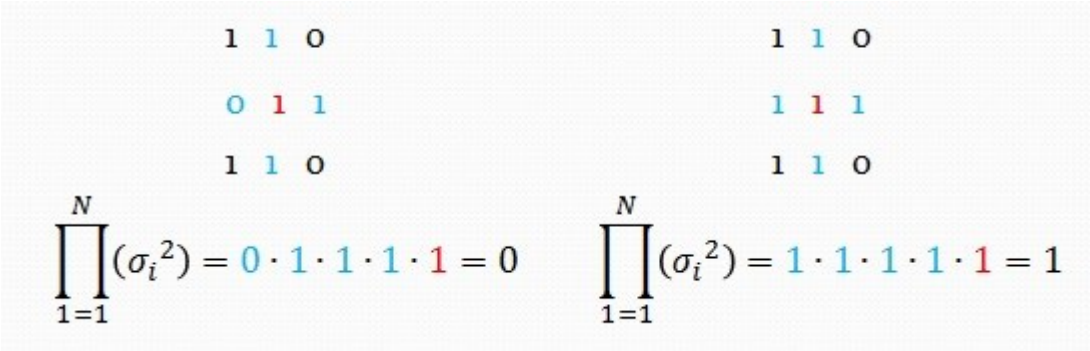


Figure 18: Part of term to check to see if neighbouring demes are occupied. An empty deme makes the term go to zero.

The section goes through every combination of central spin and neighbours to check to see if occupied demes are present, $\frac{1}{5!}\sum_{\{i,j,k,l,m\}}^{120}\sigma_i^2\sigma_j^2\sigma_k^2\sigma_l^2\sigma_m^2 = \prod_{i=1}^{5}\sigma_i^2$.

This simplification is correct if $\sigma^2$ is either 0 or 1. If there is an empty deme the term goes to zero. If this is the case then the previous four terms of the Hamiltonian will take precedence. This will encourage colonizing an empty neighbouring deme instead of competition between wild type and mutant bacteria. The deme sites will not flip from a wild type value to a mutant type (-1) until the neighbouring demes are occupied.

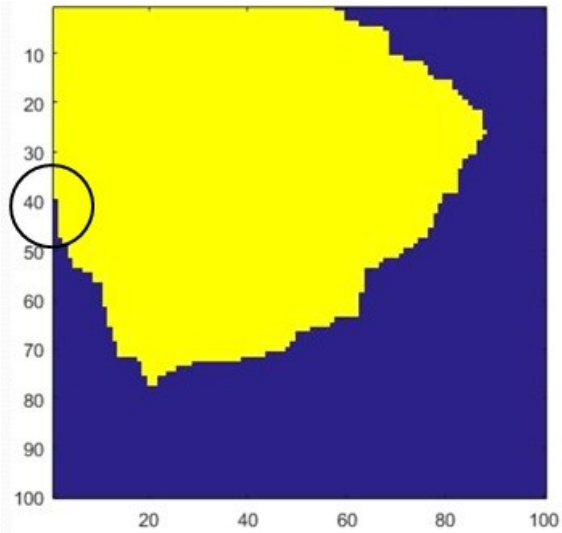


Figure 19: Food affinity migration of mutant bacteria. The circled region is where the food affinity term is supporting migration of the mutant bacteria in blue along the perimeter of the model. The wild type bacteria are yellow. $J = 1.86$, $A = J/2$, $J_d = 5.95$, $3.0 \leq J_f \leq 5.0$, $0 \leq J_C \leq 2.115$, and $kT = 0.2$. The colour scheme is automatically set by MATLAB and this image is taken at around $5.70\text{x}10^6$

Just like in the experiment by Zhang *et al* the mutant bacteria now have an added fitness advantage compared to the wild type bacteria and will invade regions with the greatest concentration of food even if no antibiotic is present. Usually mutations impose a fitness cost reducing the growth rate compared to wild type bacteria unless the mutants develop compensatory mutations [22]. Compensatory mutations could be what gave the mutant *E. coli* a fitness advantage with a food reservoir.

### 3.12 Results

As mentioned in section 3.1, the lattice is a 100 x 100 finite square grid array. Repeating what was mentioned in section 2.21, the program goes through cycles called iterations whereby it uses the Metropolis algorithm to generate a new configuration for use in the next iteration. For each iteration the program uses the Hamiltonian of equation 24 to move the system to equilibrium with the Metropolis algorithm. How the Metropolis algorithm moves the system to equilibrium is also discussed in sections 2.6 and 2.21. It should be restated that the values of $A, J, J_d, J_f, J_c$, and $kT$ are selected to resemble the experiment as the program brings the system to equilibrium, which is discussed in section 3.1. This includes the Goldilocks point occurring at the two corners of the model. The program ran through six million iterations in general to move the system to equilibrium. At equilibrium the lattice sites are all mutant bacteria because that is the most favourable type. While there was no consistent CPU time for a given run of the program, in general it took around 10 minutes to execute completely. However, calculations described further in the results section, involving energy, Shannon entropy, and results shown in Figures 25 and 26 would take one to two days. The program was performed using MATLAB and was done using a 16GB ram laptop PC.

Various factors are measured to compare how the model related to experiments on bacteria in vitro. To observe how the model measures qualitative factors such as bacterial fitness, subsequent measurements on the model are discussed in this section.

### 3.12.1 Bacterial Rate of Growth

There are four phases of bacterial growth in a liquid nutrient medium; the lag phase, the log (exponential) phase, stationary phase, and the death phase [46]. During the lag phase bacterial biomass begins to increase as the bacteria begin to replicate. Then during the exponential phase the biomass increases linearly with time as the population doubles per unit of time. The stationary phase is when a steady state is reached between growth rate and death rate. After, bacterial growth is inhibited due to the essential nutrients required becoming exhausted. This is when the death phase is reached and populations of bacteria begin to die off.

The growth phases of bacteria are replicated in the model. When bacteria started in the center of the model, growth is virtually absent due to a very low probability of replication. This is because the population is at its smallest so the probability during an iteration of one empty deme being selected next to a neighbouring occupied one is smallest, and replication with a reduced food value also minimizes the probability. When bacteria begin to replicate they enter the log phase, so the probability of selection and replication begins to exponentially increase. These phases are show in Figure 20. The log phase for wild type bacteria in the model begins at around iteration $2x10^6$; however, the mutation at the Goldilocks point prevents the wild type bacteria from entering a stationary phase as they become overrun by the invading mutant species. Mutant bacteria begin to appear at around iteration $3.5x10^6$.

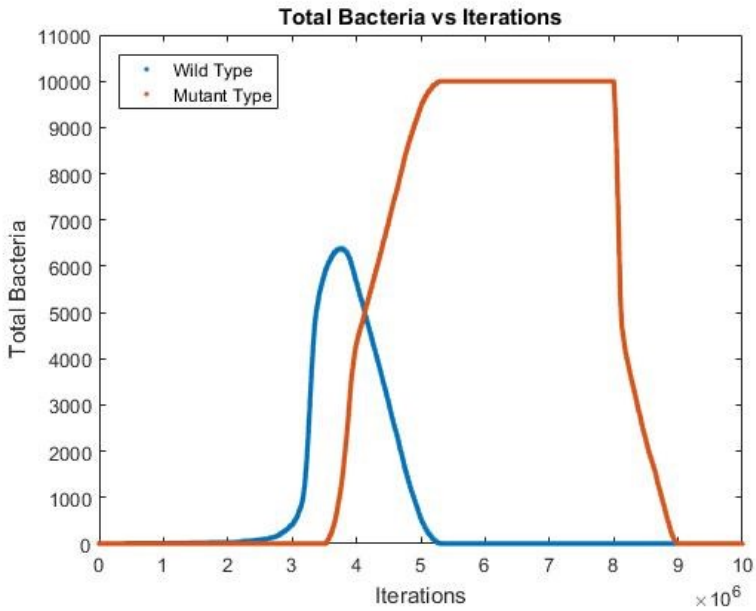


Figure 20: Bacteria growth phases of the model. The wild type bacteria reach the exponential phase then the death phase as mutant bacteria grow and take over. The stationary phase results as the mutant bacteria flip all the demes in the model. When food is instantaneously removed bacterial death results.

One factor to control the growth rate of bacteria during the stationary phase is the carrying capacity of the deme, so when the model reaches a steady state and all the finite demes contain the mutant bacteria, steady state is reached. Steady state in the model is reached when all 10000 demes have flipped to mutant bacteria.

The antibiotic and food gradients act as fields like in the Ising model to flip the spins a certain way until a steady state is reached. In this case the applied fields represent the food and antibiotic. When the food field is removed, the system is no longer in a steady state and the bacterial death phase results. This is what occurred at iteration $8 \times 10^6$. In experiments the death phase is reached as food eventually becomes exhausted [46]. However, in Figure 20 the food is instantaneously removed. This is why cell death is so rapid, but the instantaneous removal is not physical. In practice there is a gradual decrease in food and increase in inhibitory byproducts [46], which would result in the

death phase having a gradual curve instead of an instantaneous one. Without a field the model reaches a new equilibrium with the death of all bacteria, (flipping all spins to zero). To replicate the mutant steady state, the model requires at least four neighbouring wild type spins in the very center.

### 3.12.2 Antibiotic Gradients

One of the main objectives of the thesis is to further understand how an antibiotic gradient can affect bacterial evolution. To test this there are four types of antibiotic gradients constructed which are shown in Figure 22. The first gradient model is the original linear gradient used by the model previously described. The total volume of antibiotic is integrated using the array size as the surface area (an array 100 units x 100 units in area), and the value of antibiotic as height, (with $0 \leq J_C \leq 2.115$). The function $J_c(x)$ is the linear increase of the antibiotic value in the y-axis moving away from the center of the array to the perimeter, where $x$ is the position from the center moving straight towards the perimeter:

$$J_c(x) = \frac{2.115}{50} x \tag{30}$$

with the $x$ value $0 \leq x \leq 50$, (half the 100 unit distance on the array). Integrating is performed from the center to the perimeter for a quarter triangle section of the array (half the antibiotic), where a quarter triangle can be seen in Figure 21 and 22. The cross sectional area has $J_c(x)$ as the height and the horizontal width is $2x$:

$\int_0^{50} Area\, dx = \int_0^{50} (2x)(\frac{2.115}{50}x)\, dx = 3525$ units$^3$. Since this is half the antibiotic concentration the total volume of antibiotic is then 7050 units$^3$. The gradient of equation 30 is:

$$\vec{\nabla} J_c(x) = \frac{2.115}{50}[\hat{x}] \tag{31}$$

which gives the slope of gradient 1 a magnitude of $2.115/50 = 0.0423$ moving along $\hat{x}$ to the perimeter. All the gradient values are measured in (concentration of antibiotic)/(distance from center). Because the array represents deme microhabitats, each deme = (array unit)$^2$. Making the volume calculated 7050 units$^3$ = 7050 (concentration of antibiotic)*deme.

A second gradient model is constructed to increase the slope of the linear gradient. An overhead image of the second gradient is seen in Figure 21. The antibiotic gradient is shifted back 25 units from the center thereby creating an empty triangular space of 50 by 50 demes from the model's diagonal. This removes 1250 units$^2$ of area on the array compared to what was in linear gradient 1. To preserve the quantity of antibiotic used in gradient model 1, the slope of gradient 2 is calculated to be $3.384/25$ so that integration yields the same volume originally calculated. However a maximum value of $J_c = 3.384$ is too large for the parameters of the model because spontaneous growth occurs for mutant bacteria, which will be discussed in section 3.12.4. A new slope is used to give a maximum value of $J_c = 2.5$. The function used to describe the antibiotic concentration in the y-axis for a quarter of the model is:

$$J_c(x) = \frac{2.5}{25}x \tag{32}$$

but here the $x$ value $0 \leq x \leq 25$, as shown in Figure 21. Integrating with the new slope parameter for half the antibiotic volume, the cross sectional area has $J_c(x)$ from equation 32 as the height and the horizontal width is $(2x + 50)$: $\int_0^{25} Area\, dx = \int_0^{25} (2x + 50)(\frac{2.5}{25}x)\, dx \cong 2604$ units$^3$. This gives the antibiotic gradient model 2 a total volume of 5208 units$^3$. The gradient of equation 31 is then:

$$\vec{\nabla} J_c(x) = \frac{2.5}{25}[\hat{x}] \tag{33}$$

which gives the slope of gradient model 2 a magnitude of $2.5/25 = 0.10$ along $\hat{x}$ to the perimeter.

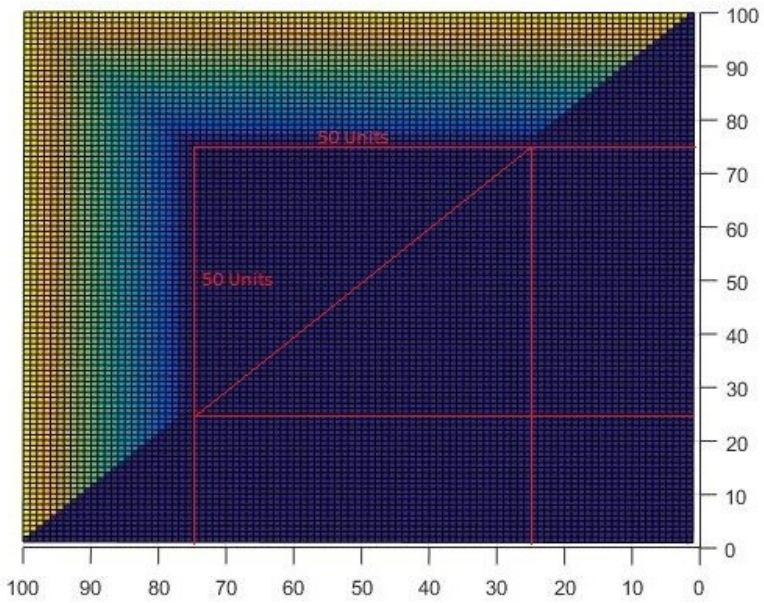


Figure 21: Overhead view of antibiotic gradient model 2. The total area removed for gradient 2 from the center compared to gradient 1 is $\frac{50 \times 50}{2} = 1250$ units$^2$, or 1250 demes. The range of the antibiotic value is now $0 \leq J_c \leq 2.5$. The image of the antibiotic gradient is the bottom right diagonal of the model but is shown in the upper right quadrant for the figure.

Gradient model 3 has the same volume as gradient model 1 with 7050 units$^3$; however the function for antibiotic concentration along the y-axis is now exponential.

The surface area on the array covered by gradient 3 is also along the bottom left diagonal like in gradient model 1. To preserve the volume of antibiotic like in the previous models, the function of the antibiotic along the y-axis is:

$$J_c(x) = e^{\frac{2.52}{50}x} - 1 \tag{34}$$

where the $x$ value is the position from the center to the perimeter of the array, ($0 \leq x \leq 50$). This gives the antibiotic value a range of $0 \leq J_c \leq 2.52$, where $J_c$ increases exponentially. The value of the gradient for the antibiotic equation 34 is:

$$\vec{\nabla} J_c(x) = \frac{2.52}{50} e^{\frac{2.52}{50}x} [\hat{x}] \tag{35}$$

which now gives the gradient a range depending on the value of $x$. The range of the magnitude for gradient 3 is $0.053 \leq |\nabla J_c| \leq 0.6264$. The approximation in section 3.6 holds true because the gradient is still not steep enough to be significant, (where the shift is largest at the perimeter and around 0.06 instead of 0.02).

The fourth gradient model is similar to gradient model 1 except the maximum value reached by $J_c$ is now 2.7 instead of $J_c = 2.115$ for model 1. A value of 2.7 pushes the parameter of the antibiotic term to the limit which, as said before, will be discussed in section 3.12.4. Using equation 31, but replacing the height of the slope to 2.7, the magnitude of the gradient is calculated to be $2.7/50 = 0.054$. The volume for gradient model 4 is also larger than the previous models at 9000 units$^3$. An image of the four gradients tested can be seen in Figure 22.

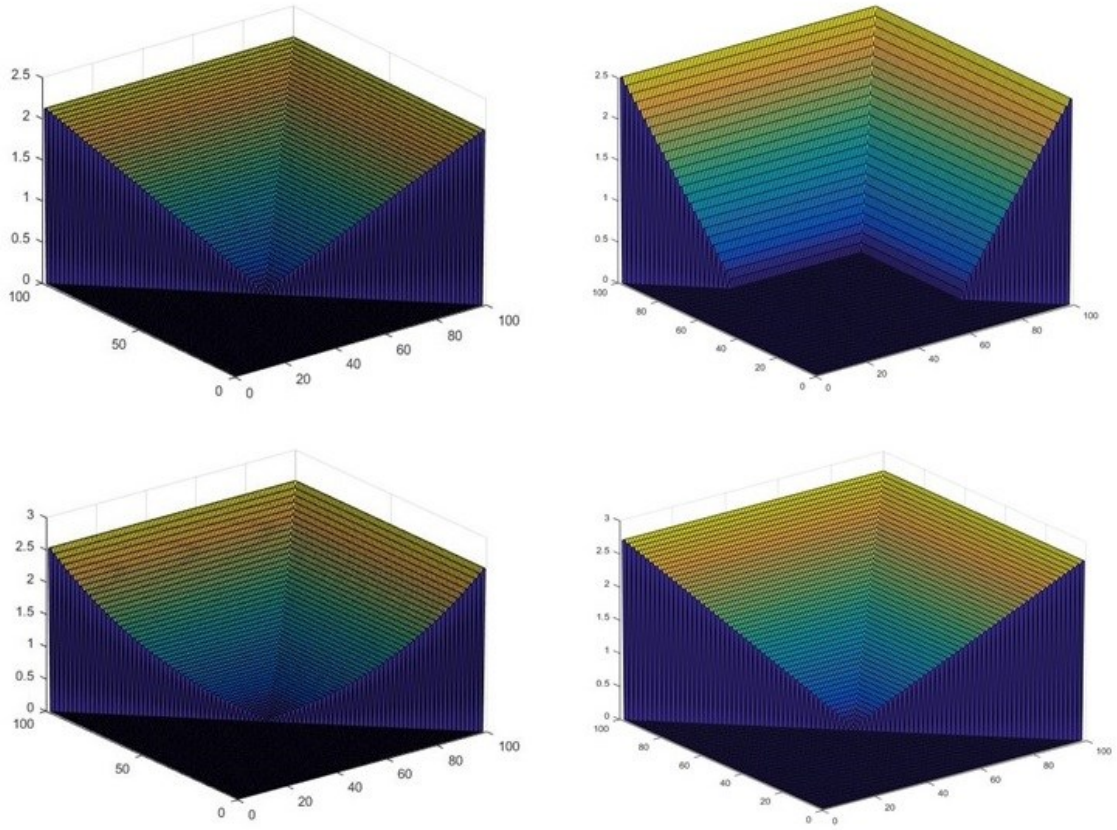

Figure 22: The four antibiotic gradient models tested. The top left is gradient 1, the top right is gradient 2, the bottom left is gradient 3, and the bottom right is gradient 4, where each gradient model is described in the text.

The number of iterations it took for a mutant spin to fix in the Goldilocks point is recorded for each gradient tested. The various amounts of time required are recorded 50 times and the average time required for each gradient can be seen in table 3.

Table 3: The magnitude of the slope of the gradients with the range of magnitude for gradient 3. The mean time in iterations recorded for a mutation to occur at the Goldilocks point. The standard deviation and median in iterations for each mean time is also recorded.

| | **Standard Gradient Model 1** | **Gradient Model 2** | **Gradient Model 3** | **Gradient Model 4** |
|---|---|---|---|---|
| **Slope or Slope Range of Gradient (antibiotic/ deme)** | 0.0423 | 0.1000 | 0.0530 - 0.6264 | 0.0540 |
| **Mean Time (Iteration)** | $3.14 \times 10^6$ | $2.70 \times 10^6$ | $3.01 \times 10^6$ | $3.04 \times 10^6$ |
| **Median (Iteration)** | $3.02 \times 10^6$ | $2.68 \times 10^6$ | $2.96 \times 10^6$ | $3.04 \times 10^6$ |
| **Standard Deviation (Iteration)** | $6.61 \times 10^5$ | $5.28 \times 10^5$ | $5.06 \times 10^5$ | $5.63 \times 10^5$ |
| **Standard Error (Iteration)** | $9.35 \times 10^4$ | $7.47 \times 10^4$ | $7.16 \times 10^4$ | $7.96 \times 10^4$ |

The average time to mutation for the standard gradient 1 is $3.14 \times 10^6$ iterations, compared to that of gradient 2 with an average time of $2.70 \times 10^6$ iterations. Even though the quantity of antibiotic for gradient 2, (5208 units$^3$), is less than that of gradient model 1, (7050 units$^3$), the higher magnitude of the gradient slope for model 2 decreased the mean time for mutation. The time to mutation for gradient model 2 is $2.70 \times 10^6$ iterations which is a reduction of 14% compared to gradient 1 with an average time of $3.14 \times 10^6$ iterations. The variation for the average time is also reduced with the standard deviation of gradient 2 being $5.28 \times 10^5$ iterations, 20% below the standard deviation of gradient 1 with $6.61 \times 10^5$ iterations. Even though the gradient became steeper in gradient model 3 compared to gradient model 2, the average time to mutation took longer. The average time for model 3 is $3.01 \times 10^6$ iterations.

The lowest standard deviation is for exponential gradient model 3 which is $5.06x10^5$. This is also the steepest gradient, followed by gradient 2 which had the second lowest standard deviation. There is less variation in the time for mutation if the gradient is steeper. Gradient model 2 has a larger standard deviation than gradient 3, however the separation between median and mode is smaller for model 2 than for gradient model 3. Half the trials for model 3 are under the median $2.96x10^6$ iterations but that half would still be closer to the mean relative to model 2 due to the reduced standard deviation. While the spread between the median and mode for model 2 is less than for model 3, the standard deviation is greater, which could mean the deviation of half the values under the median of $2.68x10^6$ iterations could be relatively lower than for model 3. Therefore while there is less deviation with a steeper gradient, model 2 is still most likely to have the lowest outliers from the mean, so more likely to have the lowest time to mutation. The spread between the median and mode for gradient model 1 is the greatest while also having the highest standard deviation of $6.61x10^5$ iterations. Gradient model 1 is most likely to have outliers with the highest value of iterations shifting the mean to a higher value than the median. The spread between the mean and median for gradient model 4 is zero with the mean and median equal to $3.04x10^6$ iterations. The standard deviation is the second highest at $5.63x10^5$ iterations. A steeper gradient with a higher volume than model 1 had a lower mean time with a lower standard deviation.

The average time recorded for each gradient shows that a steeper gradient can actually reduce the time to mutation under certain conditions. This can be seen by comparing the mean values for mutation from gradient models 2 to 3 to the value for gradient model 1. This is also the case when the volume of antibiotic is greater or

reduced, as long as the gradient is steeper. The discrepancy is mostly seen between gradient models 2 and 3. While the gradient for model 3 is greater than for model 2, mutation at the Goldilocks point is accelerated for model 2. This reason for this is due to the presence of antibiotic near the bacteria during the initial log phase of growth. As was briefly touched on in section 3.10, the food term does allow for the bacteria to migrate slowly into regions of antibiotic if $J_f$ is high enough. If there is more antibiotic in the center half, (along the diagonal), of the device where bacteria are placed then the time to mutation is greater. This is because a higher concentration will reduce the initial growth rate thereby increasing the total time for the population to reach the exponential log phase and increase the time to mutation overall. This is why gradients 1, 3, and 4 have a higher time to mutation compared to gradient 2. Gradient 2 has no antibiotic at the center of the model thereby reducing the time to reach exponential growth. Therefore it is found that a greater initial population of wild type bacteria during the initial phase of the model will greatly reduce the time to migration towards the Goldilocks point. Comparing only models 1, 3, and 4, a lower concentration of antibiotic in the center with a steeper gradient near the perimeter will reduce the time to mutate at the Goldilocks point.

### 3.12.3 Total Energy and Shannon Entropy

The energy for every deme is calculated using equation 24, and then summed in order to calculate the total energy of the system. The model runs for $7x10^6$ iterations and a histogram is constructed to measure the distribution of total energy over the entire runtime of the program.

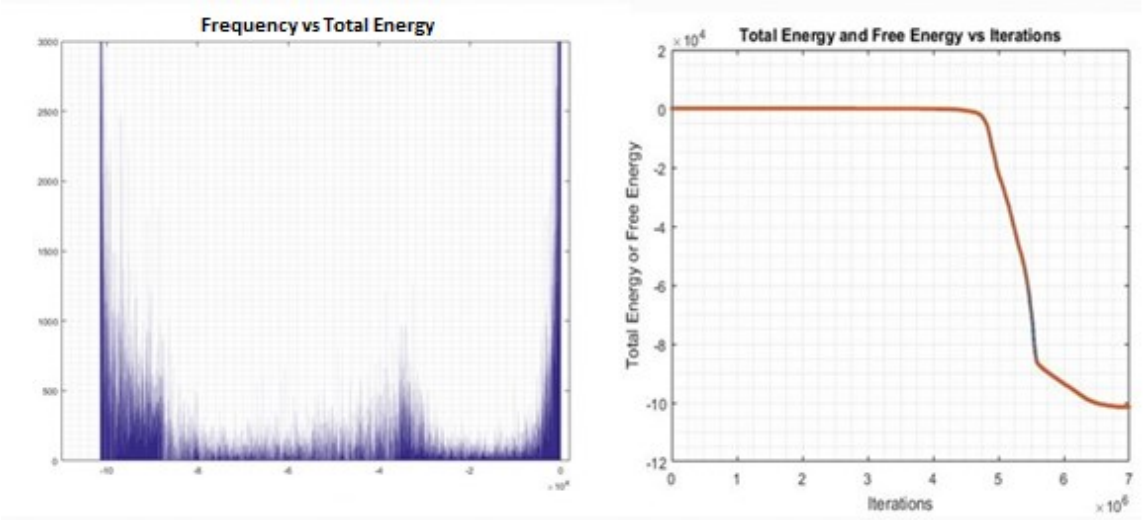


Figure 23: Energy histogram and total energy vs. iterations. The histogram is on the left and plot of total energy vs. iterations on the right. The runtime of the program is $7x10^6$ iterations. The drop in energy taking place at around $4.75x10^6$ on the right of the figure is due to the log phase of growth of wild type bacteria.

As figure 23 shows, the energy reaches a more stable state at equilibrium around a value of $-10^5$. There is a minor spike in the histogram around when the energy reaches $-3.5x10^4$ where mutation takes place.

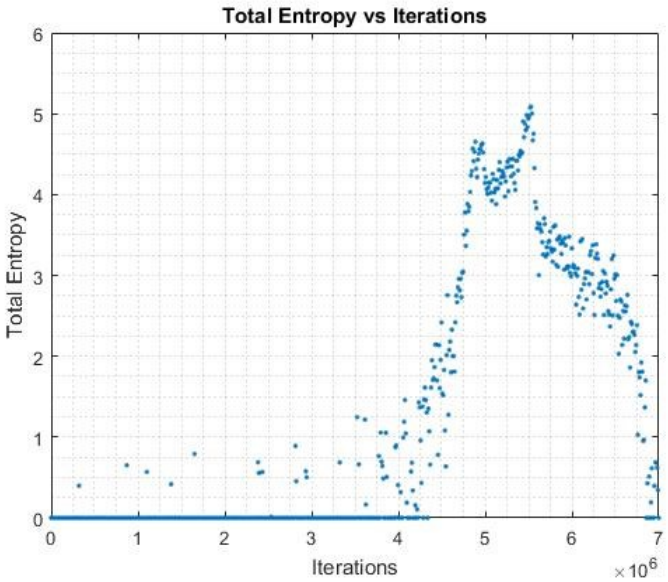


Figure 24: Total Shannon entropy per 10000 iterations of the model. The runtime of the program is $7x10^6$ iterations. The two spikes occur due to the mutant growth at the Goldilocks points of the model.

Figure 24 shows the average Shannon entropy for the total energy of the model per 10000 iterations using the recorded data from the energy histogram. Shannon entropy was calculated using equation 23 in section 2.22. There are two peaks in the graph. The peaks occur due to the mutations occurring at the goldilocks points. Introducing the mutant type bacteria into the population of wild types introduces more uncertainty to the system due to a rapid shift in energy. As the mutant type begin to spread and take over populations of wild type the entropy begins to decrease until the second Goldilocks point mutation occurs. The decrease in entropy is due to the system reaching equilibrium as it becomes saturated with mutant bacteria.

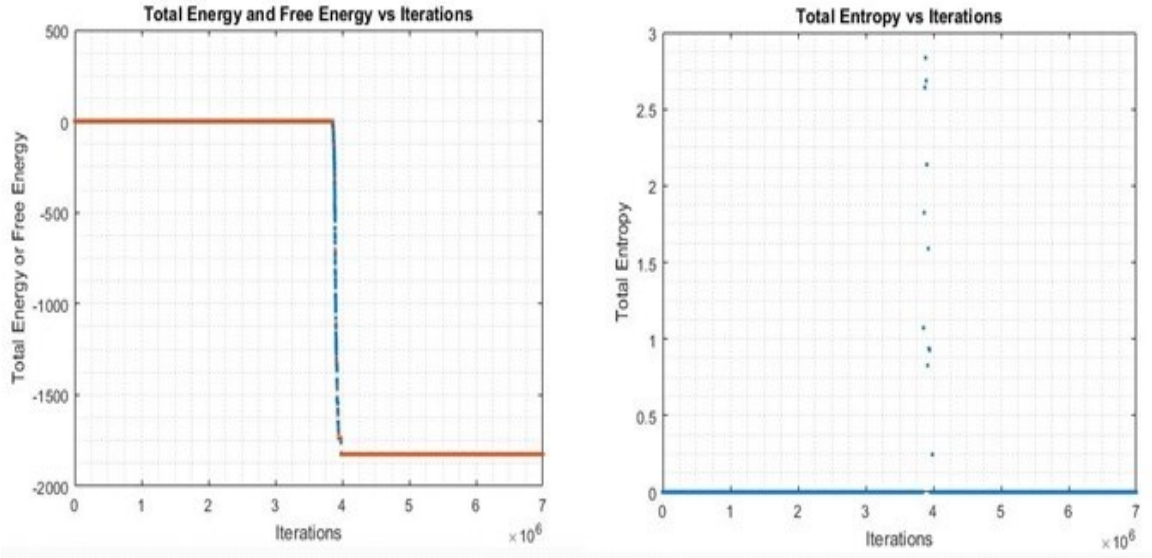


Figure 25: Total energy and entropy vs. iterations of the Goldilocks point. Total energy is on the left and total Shannon entropy is on the right. The values are taken in a 10x10 region in the corner of the model. The runtime of the program is $7x10^6$ iterations.

The shift between bacteria is more binary, (from +1 to -1, not including 0), so the energy shift is more rapid as mutation occurs. This is seen in Figure 25, which shows the energy and Shannon entropy in a 10x10 corner of the model where the Goldilocks point takes place. The average Shannon entropy of the Goldilocks point per 10000 iterations

was calculated using another histogram and equation 23. The graph in the right of Figure 25 also shows a spike where the rapid binary shift in energy occurs. The entropy goes back to zero when the mutant bacteria take over.

### 3.12.4 Fitness Parameters Landscape for Food and Antibiotic Values

The methods used to calculate total energy and Shannon entropy values in Figure 25 are recorded for various values of food and antibiotic values. The final total energy and average Shannon energy is recorded for $1.5 \times 10^6$ iterations.

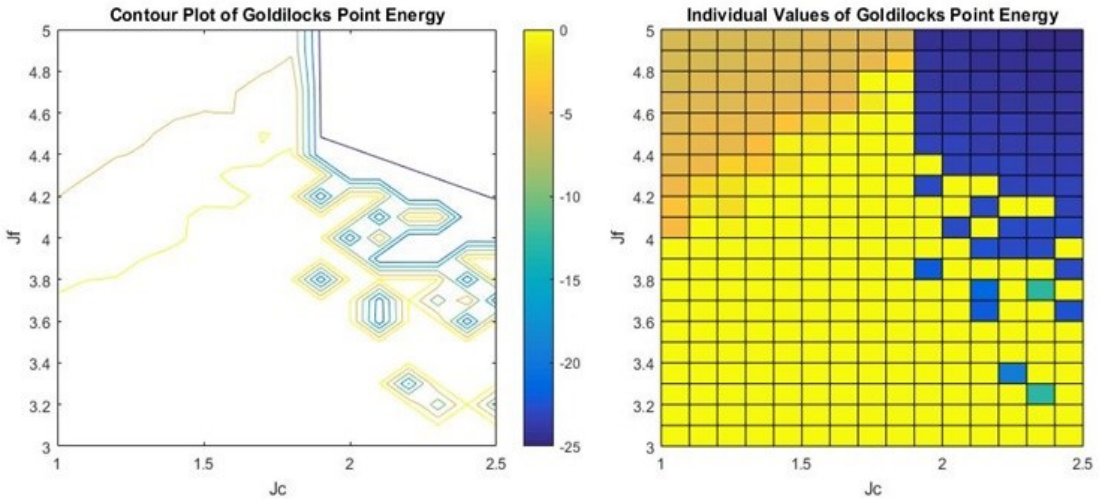


Figure 26: Contour and individual value plot of energy for the Goldilocks point for various $J_f$ and $J_c$ parameters. The values in this plot are, $3.0 \leq J_f \leq 5.0$, and $1.0 \leq J_c \leq 2.5$. The lighter yellow colour shows a higher energy (a lower fitness) and is not as favourable, while a darker blue colour is lower in energy (more fit) and more favourable.

As can be seen in figure 26, there are regions where there is an energy value of 0 (yellow colour), representing no growth of bacteria, either because the food parameter is too low or the antibiotic parameter is too high. As the value of the food parameter increases, the probability of wild type bacterial growth increases, shown by a darker shade of orange. The diagonal of wild type bacterial growth on the left hand side of the

plot, (the orange colour), shows that the food parameter must overcome an increasing antibiotic parameter for wild type growth to occur. When the value of $J_c$ is higher than 1.9, the shift in energy from the Hamiltonian is enough to allow a deme to flip to a mutant bacteria, so long as $J_f$ is high enough, (which is shown in the blue region of the figure). This forms a sort of fitness valley separating wild type bacteria from the mutant bacteria (mutant being more fit due to antibiotic).

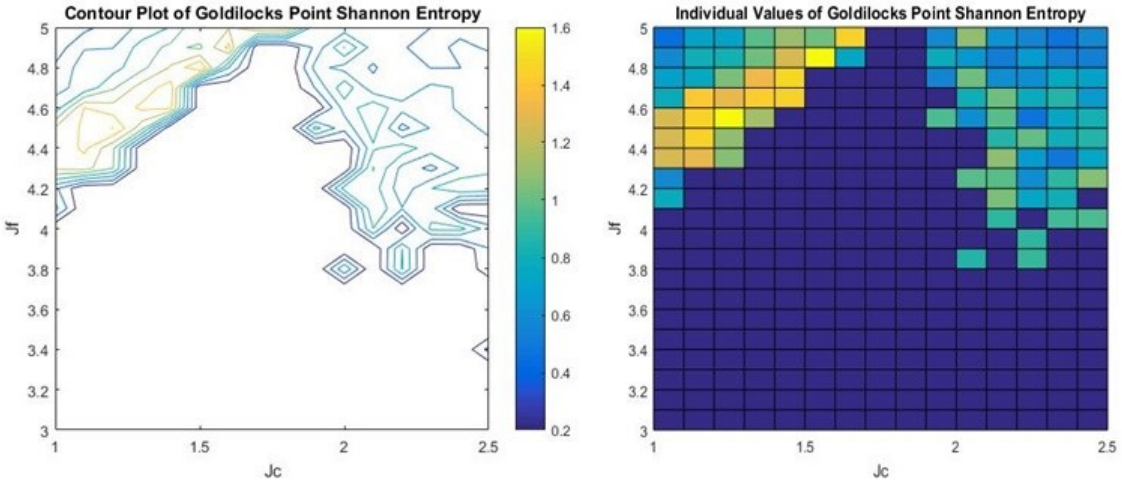


Figure 27: Contour and Individual value plot of Shannon entropy for the Goldilocks point for various $J_f$ and $J_c$ parameters. The values in this plot are, $3.0 \leq J_f \leq 5.0$, and $1.0 \leq J_c \leq 2.5$. The lighter yellow colour shows higher Shannon entropy, while a blue colour is lower in entropy, and a dark blue shows no entropy.

The Shannon entropy results of Figure 27 are calculated using the values of an energy histogram for the Goldilocks point just like in Figure 23 and Figure 24. A greater spread of the frequency of energy values in the histogram results in a greater Shannon entropy. If bacterial growth occurs, Shannon entropy is larger when no mutation occurs as opposed to when there is a mutation. The energy histogram with no mutation has a larger spread of occupied values because the shift in energy is not as great for when there is mutant growth, so the growth rate is not as rapid. With a larger shift in energy, fixation

and growth is more rapid and Shannon entropy will be lower for mutant bacteria due to a reduction in occupied energy levels. The system shifting to increased fitness (lower stable energy) helps lower the average Shannon entropy with the system moving towards a single bacterial genotype. The decrease in entropy as the system moves to a more stable energy may be better associated with a non-equilibrium system.

One flaw in the system is the antibiotic term, where if $J_c$ is too large spontaneous growth of mutant bacteria has a probability of occurring even if there are no neighbours present. Too large a value of $J_c$ stresses the model to function improperly because equation 28 shows the antibiotic term does not depend on neighbouring bacteria. As Figures 13 and 26 shows, a shift in energy with a high enough $J_c$ value will lower the energy enough to allow for spontaneous growth to occur. Figure 26 gives an optimal value for spontaneous growth of mutant bacteria to occur at the goldilocks point when food and antibiotic are around $4.8 \leq J_f \leq 5.0$, and $1.9 \leq J_c \leq 2.5$. The food parameter for the model had to have a minimum value of 3.0 near the center in order for both the food and Ising term to counteract the death term. The death term was selected to be large enough to counteract spontaneous growth due to the antibiotic term. The maximum values of the food and antibiotic term were 5.0 and 2.115 in order for the system to simulate the Goldilocks points at the two corners of the device. A higher antibiotic term shifts the Goldilocks points closer towards the center and away from the corners. As can be seen in Figure 24 another limitation in the model is that it only deals with two types of bacteria, (seen in the rapid shift in energy between two bacteria), instead of several waves of increasingly fit mutants. The energy valley between the bacteria in Figure 26 could resemble a fitness valley such as the valley seen for the antibiotic streptomycin. The

fitness landscape of various genetics of *E. coli* susceptibility to streptomycin show different adaptive peaks separated by a valley [35]. The neighbouring demes present also have an effect on the fitness of bacteria. Previous fitness valleys measured fitness as a function of the bacterial genotype [9], whereas the fitness of this system is measured with the Hamiltonian of each deme.

# Chapter 4

# The SSA Accelerated Fixation Goldilocks Point Model

## 4.1 Model

One of the goals of the thesis is to derive a new model using added parameters to include in a Monte Carlo simulation based on the work of Zhang *et al*, and Allen and coworkers [2,5,6,29]. To better understand accelerated fixation of a population of *E. coli*, a second computer model separate from the Ising computer simulation is constructed. Unlike the Ising model in chapter 3, the model described in this chapter is meant to simulate one of the Goldilocks points in the experiments by Zhang *et al* [2,5]. As mentioned before, a lower abundance of bacteria allows for the simulation to track all of them as individual particles, while a high abundance would require the Gillespie algorithm [25-27]. With this method the bacteria are allowed to overlap into the same position due to the low abundance. As the program selects each position in sequence, the motion of every individual particle can be calculated. As mentioned in section 2.18, a more crowded model with non-overlapping particles is computationally more demanding [25]. The model is based on an existing stochastic simulation algorithm (SSA) used for measuring the Brownian/Smoluchowski dynamics, and part of the model is derived from the one discussed in section 2.19 [6,25,28]. The algorithm is similar to the Gillespie algorithm; however, the distribution of times is discreet and in constant increments, not a continuous exponential function due to non-overlapping particles being computationally more demanding as previously described [25,28]. More detail about how the algorithm functions will be discussed further into the section.

Just like what is mentioned in chapter 3, the following parameters discussed in this section are set so that the simulation matches what was observed in the experiments by Zhang *et al* [2,5].

To reproduce the food and antibiotic gradient used in the experiment by Zhang *et al* and by Allen and coworkers [2,5,6], a non-uniform gradient of both antibiotic and food is reconstructed in the simulation. Unlike the gradient used by Allen and coworkers [6], this model's antibiotic gradient has been reduced several magnitudes lower. The impact the concentration has on reducing growth rate on bacteria is relatively comparable however due to the minimum inhibitory concentration (MIC) being reduced as well. The MIC equation used for this model is based on the one used in reference [29], which will be discussed further in this section. One other important difference that must be mentioned is that unlike the model used by Allen and coworkers [6], there has been a food gradient constructed in addition to the antibiotic gradient to more closely simulate the Goldilocks point experiment.

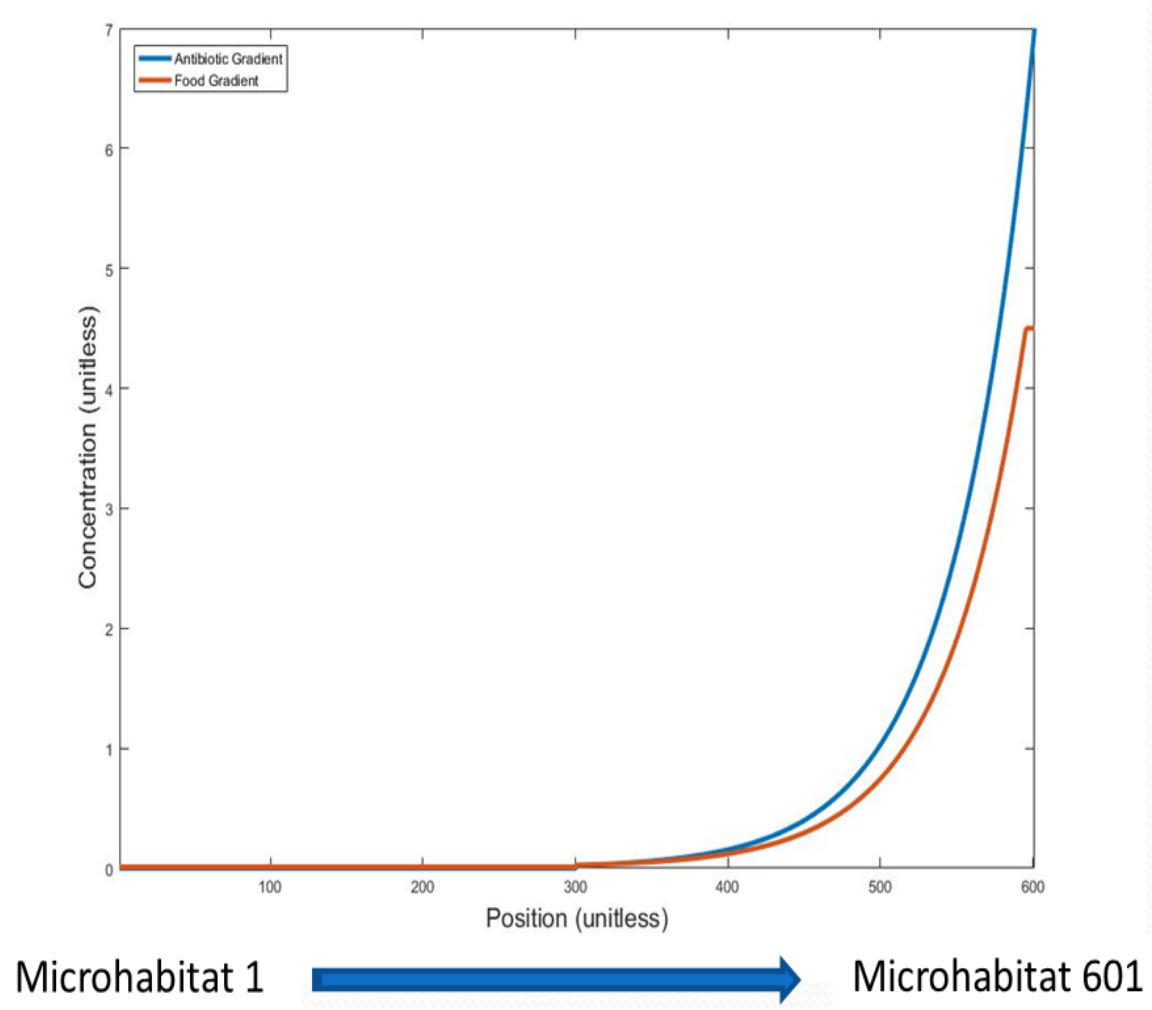


Figure 28: The food and antibiotic gradients used in the model. The food gradient from the center to the right hand side has a value from 0.01 to 4.5. The antibiotic gradient from the center to the right hand side has a value from 0 to 7.

The curved shape of model's food gradient in Figure 28 remains constant during the runtime of the simulation, (there is no consumption or degradation decreasing food concentration), thereby implying that the incoming food flux matches the bacterial metabolite consumption rate all the time [5]. The antibiotic is not consumed as well, (as stated by Zhang *et al* [5]), so the antibiotic curve in Figure 28 maintains its shape, implying that the concentration remains the same with respect to time. Each position on the model is represented by an integer "$x$" and can represent a microhabitat on the device.

A deme can be composed of several microhabitats thereby representing the "region" where accelerated mutation takes place. The model contains 601 microhabitats, where bacteria are deposited into the center located at microhabitat 301.

Part of shaping the population density gradient is the growth rate, and the growth rate is dependent on three factors. The first factor to be discussed is the pharmacodynamics function. The pharmacodynamics function is based on the Hill function. The function describes the death rate of a bacterial population exposed to an antibiotic of concentration in reference [39]:

$$\mu(c) = E_{max} \frac{(c/\beta)^{\kappa}}{1 + (c/\beta)^{\kappa}} \tag{36}$$

where μ is the death rate of the bacteria due to the concentration of antibiotic, $c$. Here β is the minimum inhibitory concentration (MIC) needed to inhibit growth of the selected bacteria, and κ is the Hill coefficient between the range 0.5 to 2.5. In the paper the coefficient is determined to be around 1.1 to match experimental antibiotic curves it was based off of [39]. The growth rate in the paper is assumed to have the relationship $\psi(c) = \psi_{max} - \mu(c)$, where $\psi_{max}$ is the maximum growth rate. However, $\mu(c)$ used in equation 36 is not used, but the pharmacodynamic function used to approximate the Hill function in equation 36 is used instead:

$$\phi_i(c) = 1 - (c/\beta)^{\kappa} \tag{37}$$

where $\phi_i(c)$ is the pharmacodynamic function used for location $i$ in the model. The coefficient κ is set to 2, which helps the function fit the threshold-like growth curves of E coli in Ciprofloxacin as seen in Allen's work [29,40]. This function no longer produces the death rate of the bacteria because the drug ciprofloxacin does not kill the cell directly

but instead the antibiotic traps the gyrase-DNA complex where DNA is cut in order to inhibit DNA replication [5,6]. This equation is a product of the growth rate, and now the growth rate is proportional to the pharmacodynamic function of equation 37 instead of subtracting the maximum growth rate by the death rate of equation 36.

The second factor is the effect which food has on encouraging bacteria to grow, and this is proportional to the Monod Equation [29,41,42]:

$$f_i(s) = f_{max} * \left(\frac{s}{K + s}\right) \tag{38}$$

where $f_{max}$ is the maximum growth rate of the bacteria, $s$ is the concentration of food at the location $x$ in the microhabitat, and $K$ is the Monod constant, (which is the concentration of food when $\frac{f_i(s)}{f_{max}} = 0.5$) [29,41]. The equation relates to how a food source affects microbial growth in a liquid medium [41]. The value of K is set to $s_{max}/4$ and the concentration of food has a maximum value of 4.5 on the gradient. The value of $f_{max}$ is 1.25 so that when the concentration of the food gradient reaches 4.5, the maximum value of the Monod equation is $f_i(s) = 1.0$.

Based on the growth curves produced by Allen and coworkers, how susceptible bacteria are to the antibiotic depends on the bacteria's growth rate [29,40]. This means that the MIC in equation 37 is nutrient-dependent and is described using the function:

$$\beta(f) = F_i * \left[c_{max} - c_j \left(\frac{f}{f_{Mon}}\right)\right] \tag{39}$$

where $c_{max}$ is the MIC needed if there is no bacterial growth, (thereby requiring a maximum MIC), while $c_j$ is the concentration subtracted due to a higher relative growth

rate, given by $\left(\frac{f}{f_{Mon}}\right)$, (the subscripts "*max*" and "*j*" are just for labeling). For this model $f_{Mon}$ describes the maximum growth rate due to the parameters set on the food gradient and Monod equation; in this case the maximum value is 1. Ciprofloxacin is a fast growth targeting antibiotic (FGTA) so this means that fast-growing *E. coli* due to higher nutrients are more susceptible than slow-growing *E. coli* [29]. For this reason the value used for $c_{max}$ is 0.03 while the value for $c_j$ is 0.027 thereby making the MIC needed to eliminate bacteria decrease linearly with growth rate. The value $F_i$ is the fitness function used to measure the relative fitness of bacteria *i*. For this model the fitness of the wild type bacteria is 1, while the fitness of the first mutant is 20, followed by 100 for the second mutant, and 200 for the third and final one. This value is chosen because Zhang *et al* measured four SNPs in the mutant species which they believed occurred sequentially and which had increased the resistance of the species by around 200 times the required MIC [2]. The model contains three types of mutants because two of the mutations occurred at neighbouring nucleotides and so will be counted as one mutation. Mutation has to occur at a rate that allows computation time to be feasible. Every time a bacterium reproduces, the probability of a mutation forming is $\mu = 0.001$ which may be slightly high due to computation time. This also allows for the model to show that fixation is more rapid than turnover for reasons which will be discussed further in the results section for this model.

Because of the combined food and Ciprofloxacin gradient in the model, an increased growth rate due to the Monod equation will actually decrease the MIC needed to reduce growth rate. So while diffusion may draw the bacteria along the length of the model and the food gradient may increase the rate of growth of a population, a competing

FGTA gradient will decrease the MIC needed to prevent growth, and this may slow down the ability of a bacterium to invade further down the food gradient.

The final factor is the population factor which prevents the bacteria from growing beyond the location's carrying capacity. Each location represents a microhabitat with a limited amount of space to survive. This is represented by the following equation:

$$P(N) = 1 - \frac{N_i}{CC} \tag{40}$$

where $P$ is the population factor at that microhabitat, $N_i$ is the population of bacteria $i$, and $CC$ is the carrying capacity for each microhabitat. The carrying capacity of the model is 400 in order to reduce computation time. As the population increases, the growth rate for bacteria in the microhabitat should decrease linearly in proportion to the habitat's population. While the rate of growth is slowed for each bacterium, a higher population size should lead to an increased amount of bacteria reproducing due to there being a greater sample size to draw from.

Using equations 37, 38, and 40, the total growth rate is a product of all three factors:

$$g_i(c, s, N) = \phi_i(c) * f_i(s) * P_i(N) \tag{41}$$

this shows that the growth rate is dependent on the concentration of antibiotic, food, and the total number of bacteria. The subheading $i$ denotes the type of mutant the equation selects, (or the genotype selected).

A constant death rate is added in order to prevent bacteria from filling each microhabitat completely. This allows the death rate to counteract the growth rate in order

to create a steady state which prevents the population remaining static. The death rate for this model is set to $d = 0.001$. Continuous reproduction allows a mutation to occur and possibly fix in that population however, the death rate has to be low enough to not allow the turnover rate towards a mutant majority to occur faster than bacteria can migrate. This would represent an environment with greater stress than the antibiotic stress gradient. The high stress environment would therefore not show accelerated fixation and not represent the experiment by Zhang *et al.*

The algorithm functions by drawing a random number in order to choose a $n^{th}$ cell from the $N_{total}$ cells in the system, ($n = 1, \ldots, N_{total}$) on the first iteration. For each iteration the algorithm cycles through every cell at random until all the cells of the current cycle is selected. Then the rates at which the cell may replicate, migrate to a neighbouring microhabitat, or die are calculated. First the replication rate is calculated: $R_{growth} = g_i(c, s, N)$. The rate of migration is then $R_{migration} = D$; $D$ being the diffusion rate. The death rate of the cell is then: $R_{death} = d$. Three random numbers are drawn, $r \in [0 \ldots 1)$, where each $r$ is calculated for each value in the order $R_{growth}$, $R_{migration}$, $R_{death}$. If the first $r < R_{growth}$, the algorithm executes replication. Death is replicated if if the first $r \geq R_{growth}$ and the second $r < R_{death}$, and migration is executed if the first and second $r \geq R_{growth}$ and $R_{death}$, and the third $r < R_{migration}$. As mentioned before the type of mutant, (or the genotype selected), is denoted by $i$, where i = 1, …, 4; the 4$^{th}$ being the most fit mutant and 1$^{st}$ being the least. If the replication step is executed, an additional number is drawn, $r_{mut} \in [0...1)$. If $r_{mut} < \mu/2$, then add a new cell of genotype $i - 1$, but if $\mu/2 < r_{mut} < \mu$, instead add to the system one new cell of genotype i + 1. Otherwise, add a new

cell of genotype i. At the end of the cycle the simulation is updated and the time is progressed t → t+Δt.

The simulation modeled the diffusion of bacteria using the equations discussed in section 2.17. The diffusion coefficient used for the simulation is given a value of $D = 5$ which helps set the time step. Therefore, rearranging equation 13 the time step used would then equal $\Delta t = \Delta X^2/2D = 0.1$ to give 10 cycles per unit of time. This helps make calculating the parameters of the model easier. The bacteria then have a probability of moving with a distance $\Delta X = \sqrt{2D\Delta t} * \lambda = 1 * \lambda$, where λ is a random number between 0 and 1. Chemotaxis is not significant enough to have an impact on the migration so the chemotactic response of equation 20 goes to infinity ($\delta \to \infty$), and the speed with which a wave of bacteria travel reverts back to equation 15, which is derived from the Fisher-KPP wave equation 14, $v = 2\sqrt{(D/2)(g-d)}$. The diffusion constant is given a value $D/2$ because there is an equal probability of the bacteria diffusing in either direction.

While chemotaxis is negligible, the wave speed is still dependent on the growth rate of bacteria, $g$, which is proportional to the Monod equation 38, $f_i(s)$. Therefore food does have an impact on increasing the speed with which the wave travels even if there is no chemotactic response. The increased growth rate of a higher concentration of food will increase the wave speed of a bacterial population because there will be more bacteria to diffuse away from their initial location. As well as the diffusion constant, combining equations 15 and 41 shows that the wave speed to travel from a location $x$ is dependent on the Monod equation, the pharmacodynamic function, and the death rate,

$v_i \propto \sqrt{\left[f_{max} * \left(\frac{s}{K+s}\right) * (1-(c/\beta)^2)\right] - d}$. These factors will be addressed further when discussing the wave speeds for the various types of bacteria and their location in the SSA model.

In the Smoluchowski Simulation the boundary conditions for diffusion are reflective or called zero flux boundary conditions [28]. However certain models of bacterial migration show that boundary conditions can be non-reflective and that bacteria can build up on the boundary [44]. Therefore if bacteria migrate to one of this simulation's boundaries, the bacteria stop migrating instead of an elastic reflection back towards the opposite direction. In the next iteration there is a probability for the bacteria to migrate backwards based on the diffusion of equation 13.

## 4.2 Results

The SSA model is run to determine what effect an antibiotic and food gradient had on bacterial migration and accelerated fixation of mutant bacteria. Population density is also analyzed to determine if fixation of migrant mutant bacteria is more rapid in a smaller population compared to a bacterial turnover of a mutant species in a large population.

### 4.2.1 Diffusion and Population Density

Initially bacteria are placed in the center of the model at position 301. The food term allows bacteria to replicate until they reach the carrying capacity of their position. As growth rate is decreased the probability of migration increases. Due to the diffusion of equation 13 the bacteria spread out to form a narrow peak with a finite width.

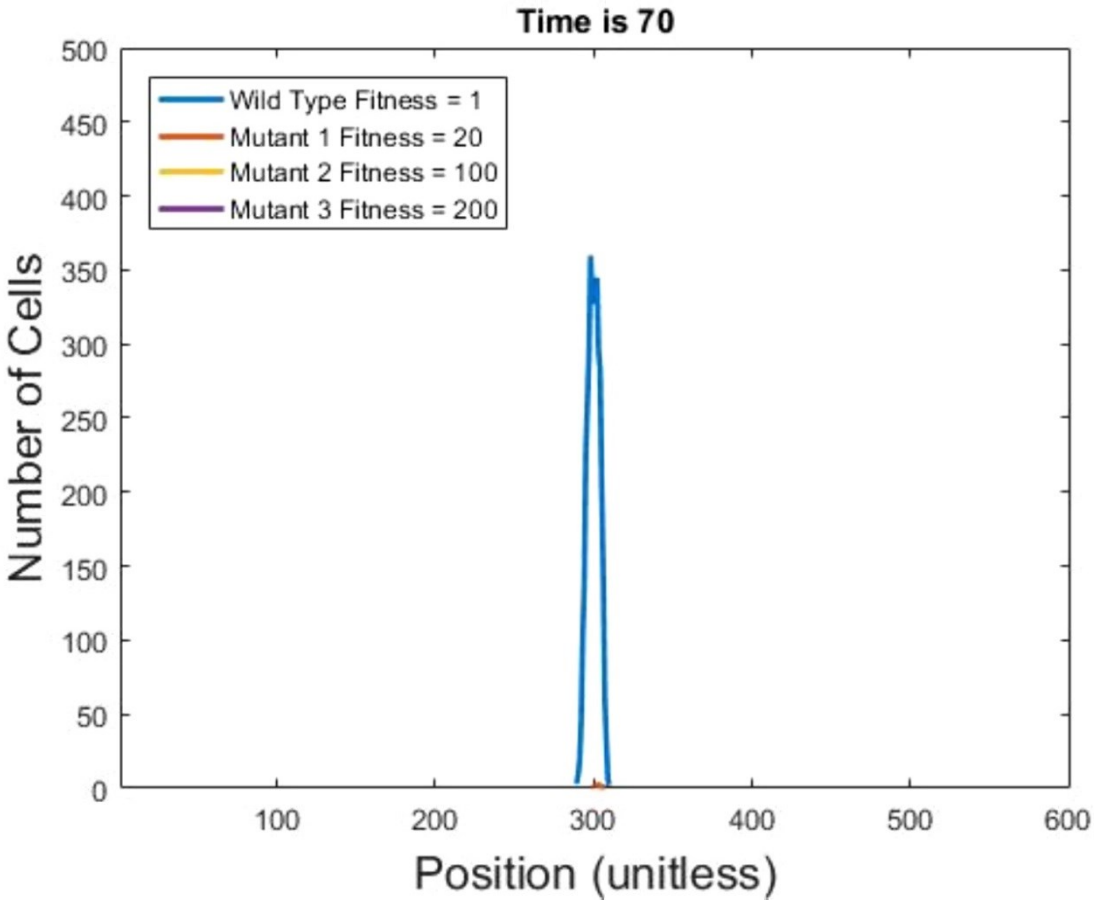

Figure 29: Initial growth of wild type bacteria. As the carrying capacity is reached diffusion becomes more probable and a quasi-delta function is formed. There is also a minor growth of mutant 1 bacteria starting.

Because diffusion is directionless, chemotaxis does not play a part in shaping the growth of bacteria; rather, only wave equation 15 determines the speed with which a population of bacteria travel. The food concentration seen in Figure 28 needs to be large enough for bacterial growth to occur. If the concentration is zero, the bacteria die due to the constant death rate. The slope of the quasi-delta function shape forms a steep population density gradient on the function's boundary.

### 4.2.2 Wave Speed of Bacteria and Accelerated Fixation

As wild type bacteria replicate, mutant bacteria begin to migrate outwards as seen in Figure 30.

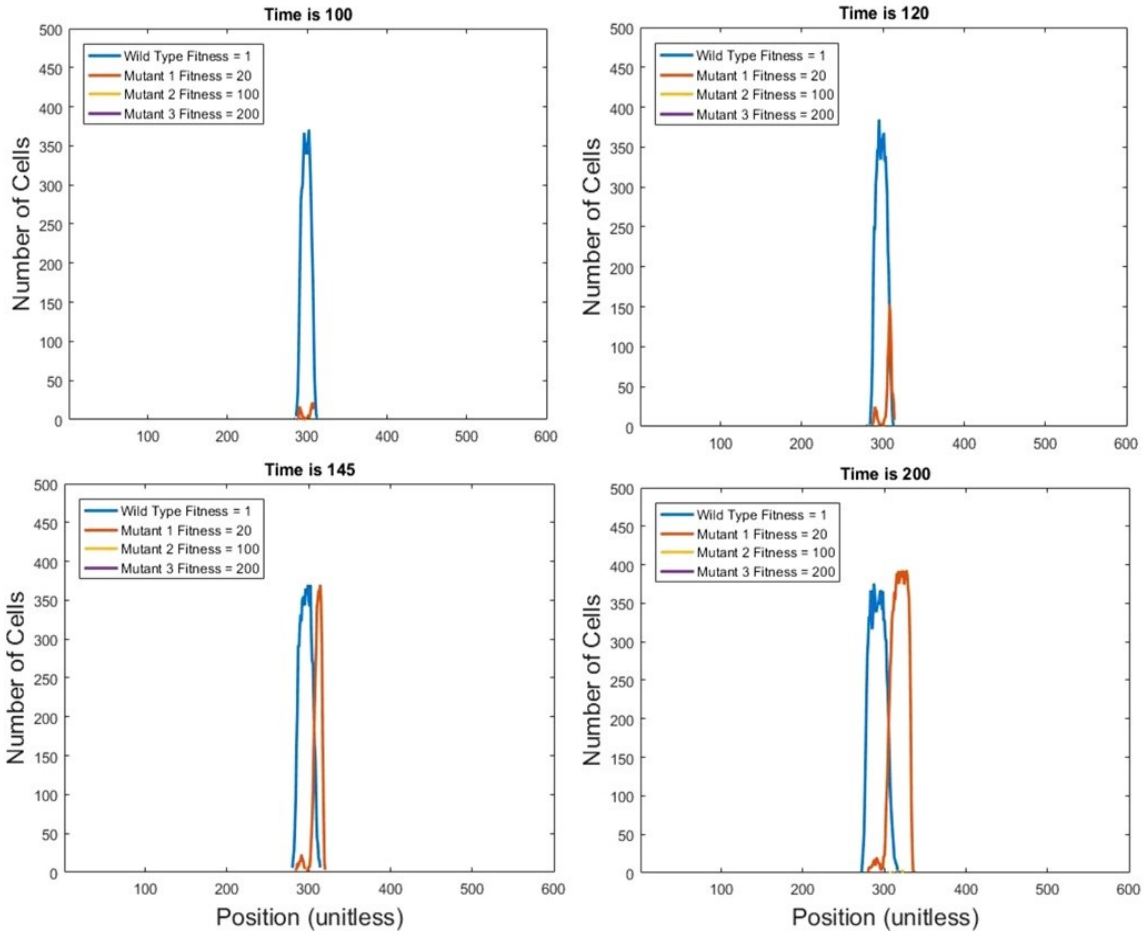


Figure 30: Mutant 1 accelerated fixation from time 100 to 200. Top left is time 100 where initial growth of mutant bacteria occur. Top right is time 120 where mutant 1 bacteria begin to fix in a low wild type population. Time 145 on the bottom left shows complete fixation of mutant 1 bacteria. Time 200 on the bottom right is when mutant 1 bacteria migrate along the food gradient.

As the wild type bacteria evolve to mutant 1 bacteria, the mutant 1 bacteria migrate towards a region with a lower population density of wild type bacteria if the migration rate is high enough. In Figure 30 this can be seen as the mutant bacteria migrate towards the right along the antibiotic gradient. The antibiotic gradient helps to

slow the progression of the wild type bacteria wave due to equation 37, allowing the mutant 1 bacteria to more quickly migrate towards regions with lower wild type population. This is why fixation is seen on the right side. As mutant bacteria migrate outwards, the bacteria can fix more rapidly in a region with a lower population of wild type bacteria, as seen in Figure 30. On the left the antibiotic gradient is zero; as well the food is at a minimum. Migration is lower so that the diffusion rate of the mutant bacteria cannot keep up with the wave speed of the wild type bacteria. Therefore the region with no antibiotic shows no fixation of mutant 1 bacteria on the left side of the model.

Because the MIC in equation 39 required is larger for mutant 1 bacteria, the mutant population wave speed can progress along the increasing antibiotic concentration, while also progressing along the food gradient of Figure 28. The bacterial fitness of mutant 1 is 20 compared to the fitness of 1 for the wild type bacteria.

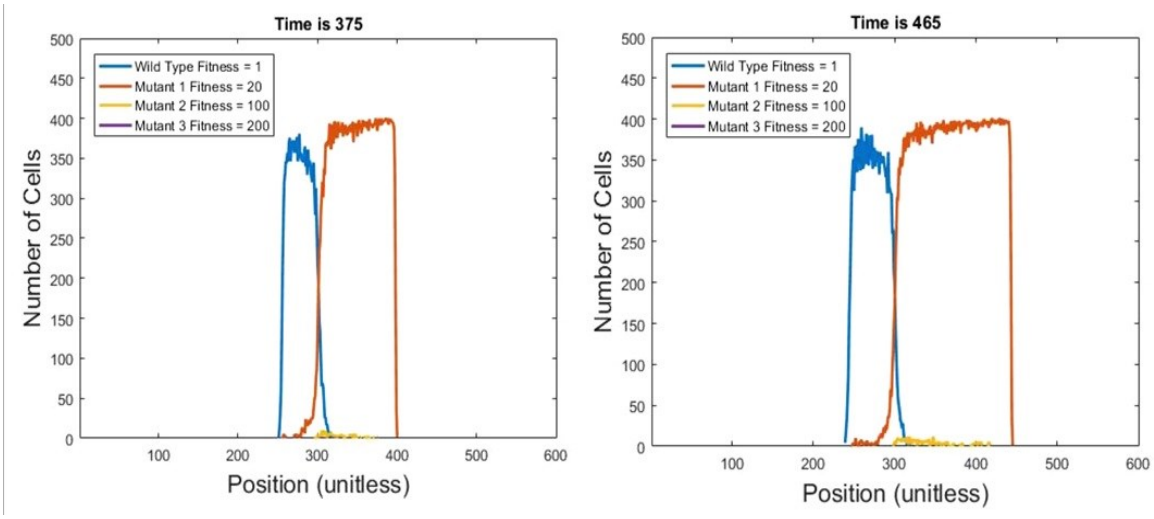


Figure 31: Progression of mutant 1 bacteria wave from time 375 to 465. The mutant bacteria wave travels from position 400 to 450 in the time span.

As the mutant 1 bacteria progress the wave speed increases. Figure 31 shows the mutant 1 bacteria wave travels from position 400 to 450 in a time span of 90 units of time. This gives the travelling wave an average speed of 0.556 positions per unit of time. As the mutant 1 wave travels along an increasing concentration of antibiotic, the speed of the wave will decrease, just as the speed of the wild type bacteria wave did before.

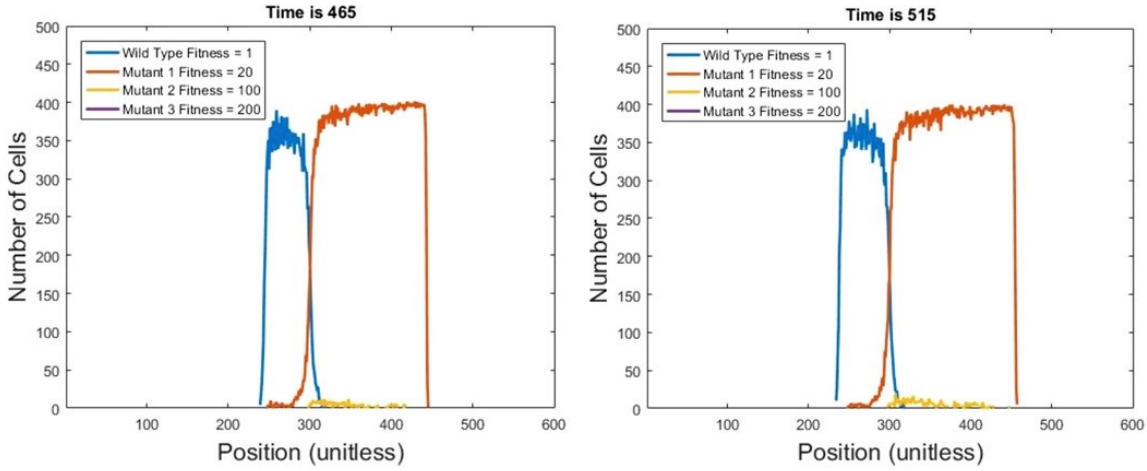


Figure 32: Progression of mutant 1 bacteria wave from time 465 to 515. The mutant bacteria wave travels from position 450 to 460 in the time span.

Figure 32 shows the mutant 1 bacteria wave travels from position 450 to 460 in a time span of 50 units of time. This gives the travelling wave an average speed of 0.2 positions per unit of time. This is 0.356 positions/unit time lower than the time to progress from position 400 to 450. The antibiotic gradient slows down the progression of the bacteria because the wave equation 15 is proportional to the pharmacodynamic equation 37. Eventually the MIC needed to prevent bacterial growth is reached and the wave of mutant 1 bacteria stops progressing along the antibiotic gradient.

The MIC equation 39 describes the susceptibility of bacteria to a FGTA ciprofloxacin. The increasing concentration of food along the gradient will have an

impact due to the Monod equation 38. Therefore a higher replication rate will make bacteria more susceptible to the antibiotic because it will lower the MIC required to prevent growth due to equation 39. The combination of increasing food concentration with the antibiotic gradient may actually more dramatically reduce the replication rate at the wave edge compared to a lower concentration of food with the same antibiotic gradient. The increased fitness of mutant 1 compared to wild type bacteria still however increased the MIC required to prevent growth, allowing the mutant bacteria to expand into the antibiotic regardless of the FGTA susceptibility.

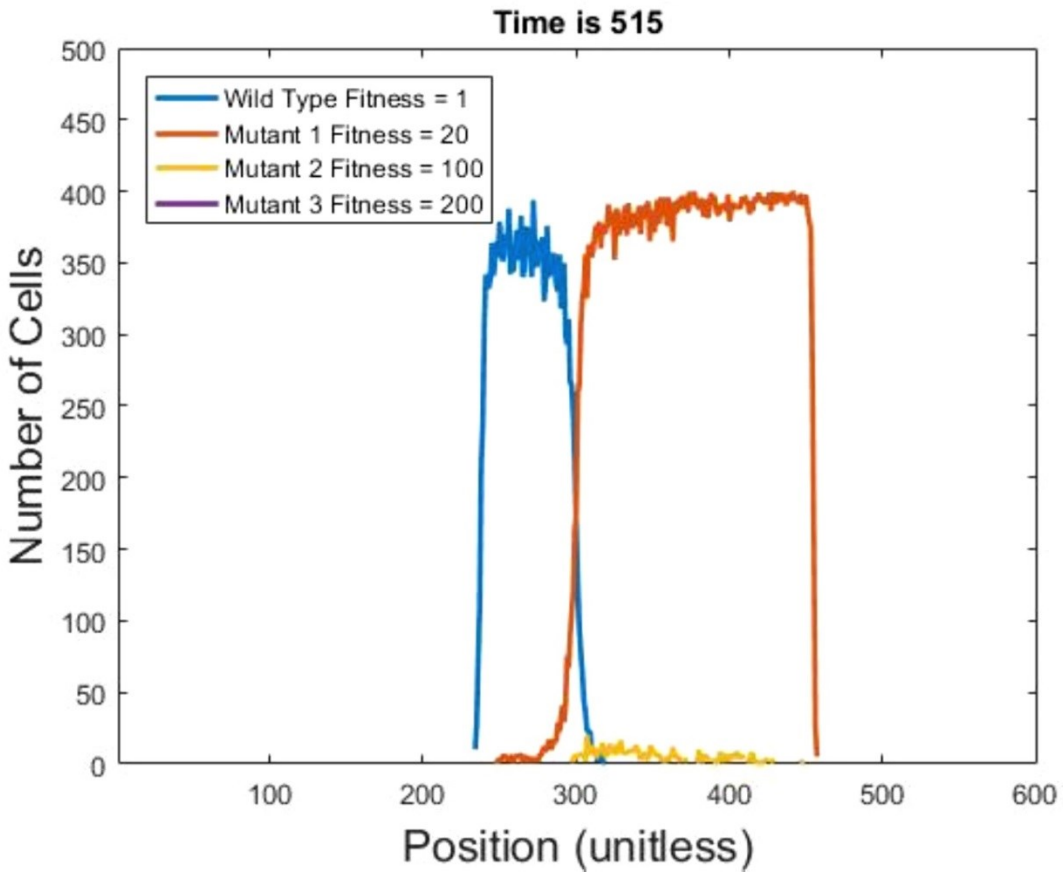


Figure 33: The distance travelled by mutant 1 bacteria compared to wild type bacteria.

Figure 33 above shows the progression of mutant 1 bacteria compared to the distance travelled by the wild type bacteria. This shows that due to the wave speed being proportional to the Monod equation, the rate of migration is faster when there is a food gradient.

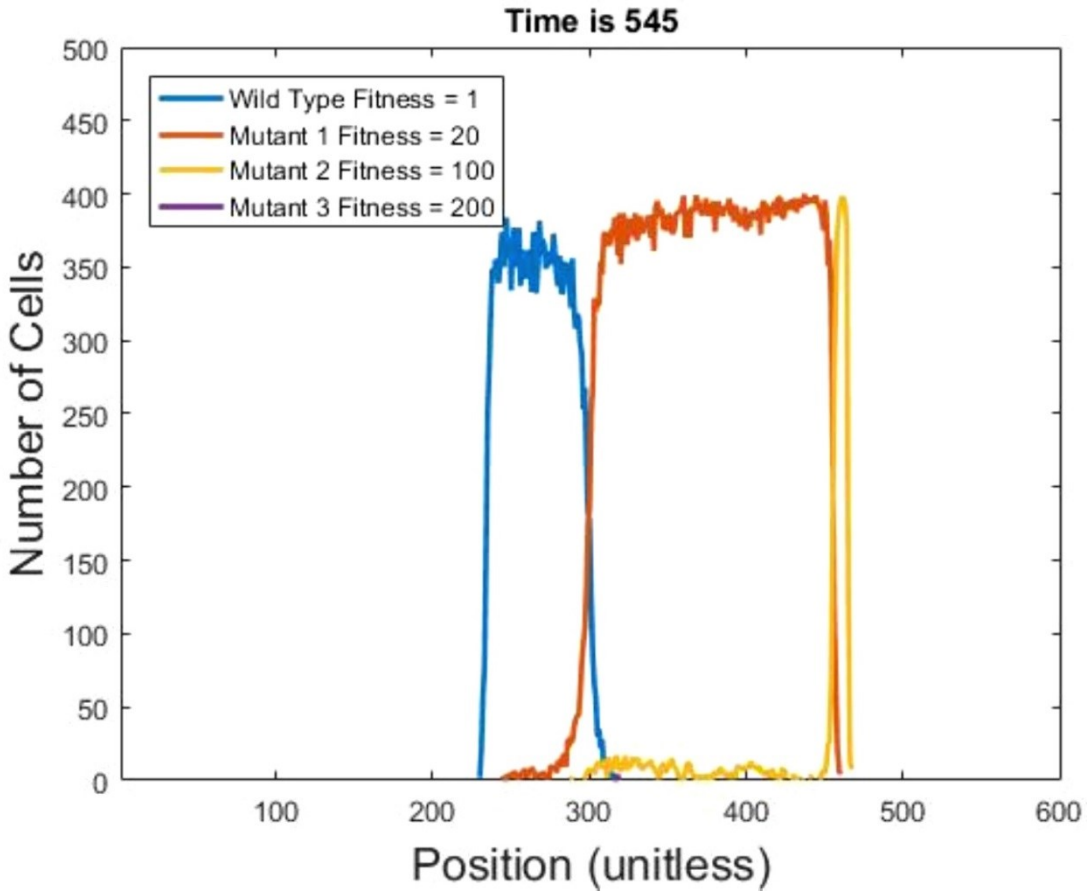


Figure 34: The accelerated fixation of mutant 2 bacteria.

Figures 33 and 34 show that mutation of mutant bacteria 2 occurs within populations of mutant bacteria 1. The bacteria do not fix in the populations of mutant bacteria 1 due to the higher population density. Migration of mutant bacteria 2 is not enough to keep up with the wave edge for mutant bacteria 1, therefore fixation does not occur. When the wave edge of mutant bacteria 1 stops progressing, migration of mutant

bacteria 2 can travel to the front with a lower population density. Fixation of mutant bacteria 2 occurs due to an increased fitness advantage of 100 compared to 20 for mutant bacteria 1, and because of the lower population density. This is what is observed in Figure 34.

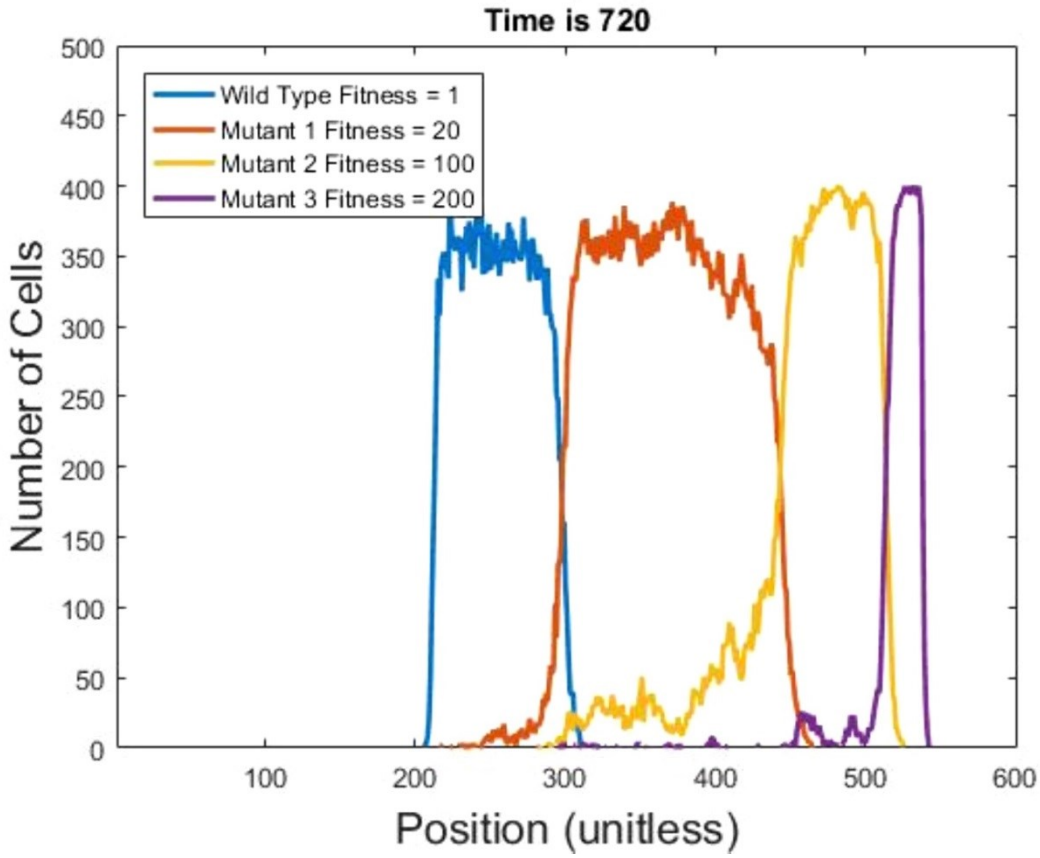


Figure 35: The accelerated fixation of mutant 3 bacteria.

The same process discussed occurs for the migration of mutant bacteria 2 along the antibiotic gradient. When the wave edge for mutant bacteria 2 decreases, accelerated fixation occurs for mutant bacteria 3.

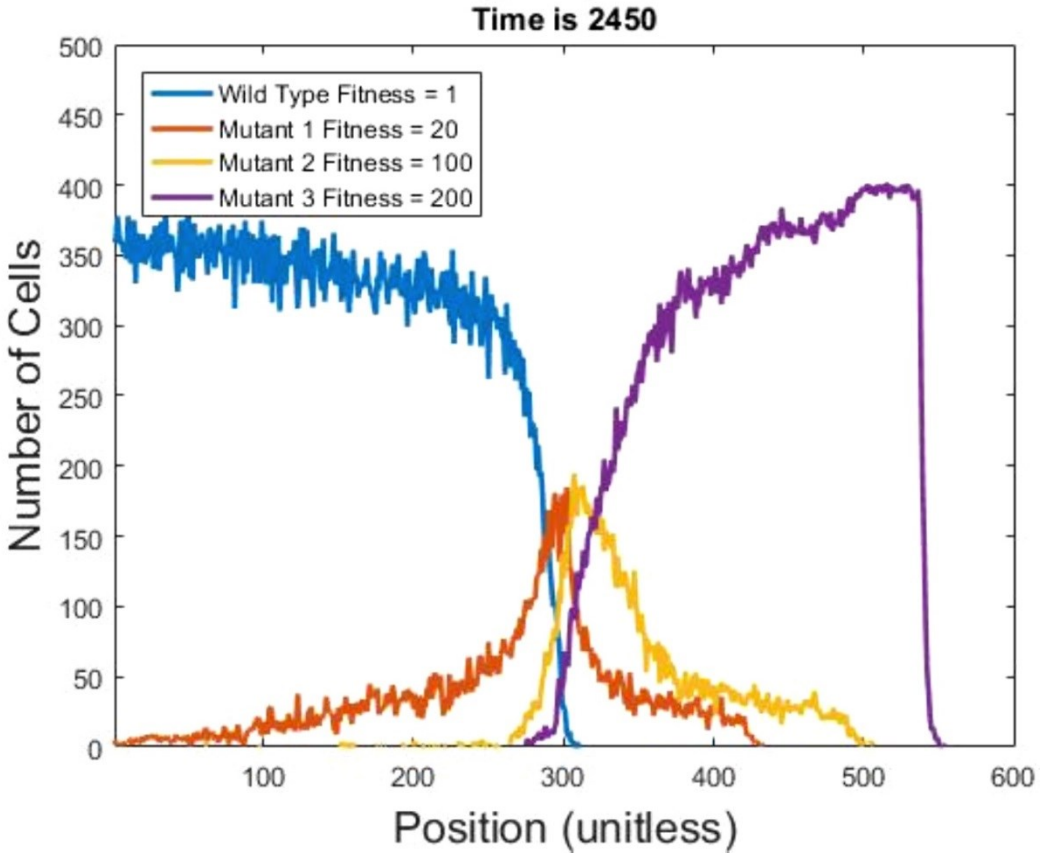

Figure 36: Mutant bacteria 3 fixes on the right hand side of the model. The region on the left side without antibiotic prevents mutant bacteria from fixing in the region.

As the model continues to progress, the mutant bacteria 3 fixes on the right hand side. This can be seen in Figure 36 where the time has progressed to 2450 units of time. The diffusion of mutant bacteria back towards the left side with the wild type population could explain what is observed in the experiment by Zhang *et al* where there was an invasion of mutant bacteria back towards the center of the device [5].

### 4.2.3 Discussion

The model is meant to replicate how a food and antibiotic gradient can accelerate evolution for antibiotic resistance in bacterial populations. The model describes how *E. coli* populations mutate and fix to increase the fitness to resist the FGTA ciprofloxacin. The hypothesis by Zhang *et al* which is tested in the model states "the stepwise movement of motile mutant bacteria via successive mutations into regions of higher stress is accelerated if the mutation rate is very high and the population density gradient … is very steep" [5]. Figures 30 to 35 display how an antibiotic gradient can help accelerate mutation under certain conditions.

The sharp distribution curve at the wave edge of a bacterial population formes a population density gradient. The steep decrease in population is directed towards a higher concentration of antibiotic. The probability of fixation increased in agreement with equation 10, the Wright's evolutionary potential. A larger fitness potential will increase the probability that a bacterial population will move towards a more fit genotype. This is also represented by the stochastic drive, ε, discussed in section 2.10. A larger value of ε will reduce the probability of a population having a particular genotype which may increase the probability of other genotype frequencies. As F-theorem in section 2.10 states, a more diverse genotypic frequency will allow for the potential of more fit mutants [7]. However, in terms of population density, a larger population will likely have a lower stochastic drive because of a greater frequency of one particular genotype. A lower population density will be more susceptible to a fluctuation in genotypic frequencies, so a more fit mutant will fix more rapidly if the population is

lower [3,7]. This is what is observed in the model. It is also shown is that a very steep population density gradient would result in a rapid fixation of mutant bacteria.

The mutation rate needs to be high enough to show fixation. If mutation is too low the probability of a mutant bacteria migrating towards the edge of a population gradient would be too low. This is also the case if the migration rate is too low. Then a mutant bacterium would not be able to migrate towards the wave edge of a less fit bacterial population; especially if the wave speed of the edge is too fast. The replication and death rate have to be balanced with the mutation and migration rate. If the turnover between replication and death rate is too fast then fixation would be fast enough without having the need for mutant bacteria to migrate towards a lower population density. This also helped confirm the hypothesis by Zhang *et al* that the bacteria needed to be motile and the mutation rate needed to be very high [5]. A slower mutation rate is also difficult because then the computation time became too long. The current parameters show how fixation is more rapid in a lower population compared to bacterial turnover in a large population.

The time to fixation is measured as in section 3.12.2. Fifty trials are performed to measure the time it takes for mutant 3 to fix and reach a population of the carrying capacity of 400 cells. Three exponential antibiotic gradients are measured given by the function:

$$c(x) = e^{\frac{\ln(m+1)}{300}x} - 1 \tag{42}$$

where $x$ is the position on the right half of the model, $(0 \leq x \leq 300)$. The value of $m$ is selected so that at the far right boundary of the model, $c(300) = m$. This is to determine

the maximum concentration of the exponential antibiotic function. The time to fixation can be seen on Table 4.

Table 4: Time for the population of mutant bacteria 3 to fix and reach carrying capacity. The average time, standard deviation, and median is calculated for 50 trials.

| | **Gradient m = 0.25** | **Gradient m = 1.00** | **Gradient m = 7.00** |
|---|---|---|---|
| **Average Time (unitless)** | 427.33 | 331.46 | 331.89 |
| **Standard Deviation (unitless)** | 48.48 | 14.56 | 29.79 |
| **Median (unitless)** | 438.70 | 328.85 | 329.10 |
| **Standard Error (unitless)** | 6.86 | 2.06 | 4.21 |

Unlike section 3.12.2, the volume of antibiotic increases with increasing $m$ value. The lowest time to fix is the antibiotic gradient with $m = 1.00$, with an average time of 331.46 units. The standard deviation and median is also lowest at 14.56 and 328.85 units of time. The highest average time is for the gradient with $m = 0.25$, at 427.33 units. It also had the highest standard deviation at 48.48 units. Average time to fix and fluctuation is greatest for the lower concentration because the MIC needed to prevent mutation 2 with a fitness of 100 to grow is 0.3. Setting the antibiotic to be lower than the MIC allowed for mutant 2 to migrate to the boundary and prevent mutant 3 from fixing. This pattern is seen in Figure 36 where the wild type bacteria prevent mutant 1 from fixing on the left hand side of the model. The higher concentration of antibiotic for the gradient with $m = 7.00$ showed an average time of 331.89 units, slightly greater than $m = 1.00$. The standard deviation and median is also greater at 29.79 and 329.10 units of time. While the higher concentration of antibiotic may slow down the time until fixation, the greater standard deviation and higher median may mean more outliers further away from the average times of $m = 1.00$ and $m = 7.00$. This is similar to what

was observed in section 3.12.2. The greater concentration of antibiotic slows down the initial log phase of growth for wild type bacteria. A greater initial population of wild type bacteria during the log phase of growth will greatly reduce the time to migration and increase the probability of mutation due to a larger selection of bacteria to mutate. Even though the total volume of antibiotic for $m = 7.00$ is greater than $m = 1.00$, the mean time to fixation is virtually the same, which could be due to the steeper gradient. If the model is run whereby the concentration of antibiotic is greater than the MIC of the wild type bacteria, no evolution occurs and the wild type bacteria die.

There are some limitations of the model which need to be discussed. The major factor involved in spreading antibiotic resistance is horizontal gene transfer [22]; however, this is not included in the model. Mutation in the model only occurs randomly through replication. Chemotaxis is also not included, even though the food gradient still played a part in increased migration, (as discussed at the end of section 4.1). Based on what is observed, a greater direction and speed to migration for mutant bacteria through chemotaxis may actually enhance accelerated mutation. The fitness of mutants also plays a part when there is no antibiotic. When a bacterium mutates there is a fitness penalty in replication unless it continues to be exposed to the antibiotic or it develops some compensatory mutation [22]. If a mutant bacterium is in a region with no concentration of antibiotic then there is no fitness penalty compared to a wild type one. A reverse of antibiotic resistance in *E. coli* of three percent to ciprofloxacin has been observed over a six month period [22]. However, compensatory mutations could be what gave mutant *E. coli* a fitness advantage in the experiment by Zhang *et al* [5], as discussed in section 3.11. The model is built so that mutant fixation to a fitness of 200 is inevitable; however, if the

MIC is lower than necessary a more fit mutation may not have an advantage over less fit one, especially if there is a fitness penalty in replication.

## Chapter 5

## Conclusion

The models constructed identified several evolutionary factors required for accelerated evolution of antibiotic resistance. The required factors identified by Zhang *et al* for accelerated mutation were tested [5]. These include whether the bacteria are more motile, the mutation rate is high enough, and how steep the population density gradient is. The main factor investigated is how antibiotic gradients enhance the rate of mutation. It is found overall that the rate of mutation is enhanced the steeper the antibiotic gradient under certain conditions. The initial population growth of bacteria plays an important part in the time to fixation. If the concentration of antibiotic slows the initial exponential growth of bacteria then the time to fixation is reduced. An initial concentration of antibiotic greater than the MIC of wild type bacteria will result in the death of the wild type population. Allowing for initial bacterial growth in the absence of antibiotic followed by migration towards an increasing gradient of antibiotic will result in accelerated mutation.

The SSA model looked at a linear additive fitness increase but different epistatic genotypic combinations have varying degrees of fitness and the pathways to the most fit genotype may not be additive, as discussed in sections 2.09 and 2.10. The model discussed in section 2.19 showed that a non-uniform drug distribution may slow down the emergence of resistant bacterial fixation if one of the epistatic genotypes had a lower fitness value than the previous genotype it evolved from, which resembles a fitness valley discussed at the end of the section [6]. One way to improve the Ising and SSA models is

to look to see how an epistatic pathway of intermediate genotypes with varying degrees of fitness affect accelerated resistance if utilization of food also plays a role. As was stated in section 2.16, an acquired mutation in response to antibiotics may impose a fitness cost on a bacterium (measured by a decreased growth rate); this cost can be deleterious to the bacterial strain unless it continues to be exposed to an environment with antibiotic or also can be mitigated by a rapid development of compensatory mutations [22]. How well bacteria can utilize food as a compensatory mutation and the effect mutation has on bacterial growth would be noteworthy additions to improve the model. As mentioned in section 2.15, the main mechanism used for acquiring antibiotic resistance is horizontal gene transfer (HGT) [21,22]. Transferring mutations between two particles, (bacteria), would be an important mechanism which should be added to future models to better represent accelerated mutation.

Combining the two models may serve to provide new and interesting systems to analyze. The Ising model in chapter 3 is limited due to the demes represented by discreet states and not by populations of bacteria. Currently the Ising model does not represent population density gradients. The SSA model in chapter 4 is limited to a one dimensional model. By combining the models of chapters 3 and 4, each model's strength may compensate for the other model's shortcomings. For instance, a new combined model may be a two dimensional Ising model like in chapter 3, with a population of bacteria per deme as there are in the SSA model of chapter 4. A future theoretical program would greatly improve how bacterial evolution is modeled and could be applied for a wider use of systems.

**References**


[1] B. Aslam , W. Wang, M. I. Arshad, M. Khurshid, S. Muzammil, M. H. Rasool, M. A. Nisar, R. F. Alvi, M. A. Aslam, M. U. Qamar, M. K. F. Salamat, Z. Baloch. *Antibiotic resistance: a rundown of a global crisis*. Infection and Drug Resistance **11** (2018) 1645–1658

[2] Q. Zhang, C. Tung, K. Robin, N. Pourmand, D. Liao, G. Lambert, H. Kim, & R. Austin. *Acceleration of Emergence of Bacterial Antibiotic Resistance in Connected Microenvironments*. Science **333** (2011) 1764–1767

[3] Q. Zhang, R. H. Austin. *Physics of Cancer: The Impact of Heterogeneity*. Annu. Rev. Condens. Matter Phys. **3** (2012) 363–382

[4] D. E. Dykhuizen. *Experimental Studies of Natural Selection in Bacteria*. Annu. Rev. Ecol. Syst. **21** (1990) 373–398

[5] Q. Zhang, K. Robin, D. Liao, G. Lambert, & R. Austin. *The Goldilocks Principle and Antibiotic Resistance in Bacteria*. Molecular Pharmacology **8** (2011) 2063–2068

[6] P. Greulich, B. Waclaw, & R. J. Allen. *Mutational Pathway Determines Whether Drug Gradients Accelerate Evolution of Drug-Resistant Cells*. Physical Review Letters **109(8)** (2012) 088101-1–088101-5

[7] P Ao. *Laws in Darwinian Evolutionary Theory*. Physics of Life Reviews **2(2)** (2005) 117–156

[8] G. Sella, A. E. Hirsh. *The application of statistical physics to evolutionary biology*. PNAS **102(27)** (2005) 9541–9546

[9] J.A.G.M. de Visser, J. Krug. *Empirical fitness landscapes and the predictability of evolution*. Nature Reviews Genetics **15** (2014) 480–490

[10] J. Venegas-Ortiz. (2013). *Statistical Mechanics of gene competition*. (Doctoral Dissertation). The University of Edinburgh.

[11] A. McQuarrie. (2000) *Statistical Mechanics*. University Science Books, California

[12] E. Carlon. *Computational Physics*. (2012) Flanders, Belgium: KU Leuven

[13] I. Yildirim. *Bayesian Inference: Metropolis-Hastings Sampling*. (2012) Rochester: University of Rochester

[14] H.J. Cordell. *Epistasis: what it means, what it doesn't mean, and statistical methods to detect it in humans*. Human Molecular Genetics, **11(20)** (2002) 2463–2468

[15] K. Lipinski, L. Barber, M. Davies, M. Ashenden, A. Sottoriva, M. Gerlinger. *Evolution and the Limits of Predictability in Precision Cancer Medicine*. Trends in Cancer **2** (2016) 10.1016/j.trecan.2015.11.003

[16] R. Hermsen, T. Hwa. *Sources and Sinks: A Stochastic Model of Evolution in Heterogeneous Environments*. Phys. Rev. Lett. (2010) **105** 248104

[17] W. Schröder, J. Bernhardt, G. Marincola, L. Klein-Hitpass, A. Herbig, G. Krupp, K. Nieselt, C. Wolz. *Altering gene expression by aminocoumarins: the role of DNA supercoiling in Staphylococcus aureus*. BMC Genomics **15(291)** (2014) doi:10.1186/1471-2164-15-291M.

[18] P. Sharma, J. R. J. Haycocks, A. D. Middlemiss, R. A. Kettles, L. E. Sellars, V. Ricci, L. J. V. Piddock, D. C. Grainger. *The multiple antibiotic resistance operon*

*of enteric bacteria controls DNA repair and outer membrane integrity*. Nature Communications **8(1444)** (2017) doi:10.1038/s41467-017-01405-7

[19] H. H. Lee, M. N. Molla, C. R. Cantor, J. J. Collins. Bacterial charity work leads to population-wide resistance. Nature **467(7311)** (2010) 82–85. doi:10.1038/nature09354

[20] G. J. McKenzie, R. S. Harris, P. L. Lee, S. M. Rosenberg. *The SOS response regulates adaptive mutation*. PNAS **97(12)** (2000) 6646 – 6651

[21] A. Weiss. *Lamarckian Illusions*. Trends in Ecology & Evolution **30(10)** (2015) 566–568

[22] M. Sundqvist. *Reversibility of antibiotic resistance*. Upsala Journal of Medical sciences **119** (2014) 142– 48

[23] B. S. Gottesman, Y. Carmeli, P. Shitrit, M. Chowers. *Impact of quinolone restriction on resistance patterns of Escherichia coli isolated from urine by culture in a community setting*. Clin Infect Dis. **49** (2009) 869–75

[24] L. L. Marcusson, N. Frimodt-Moller, D. Hughes. *Interplay in the selection of fluoroquinolone resistance and bacterial fitness*. PLoS Pathog. **5** (2009) e1000541

[25] M. Klann, A. Ganguly, H. Koepp. *Hybrid spatial Gillespie and particle tracking simulation. Bioinformatics* (Oxford University Press) **28** (2012) doi:10.1093/bioinformatics/bts384

[26] C. L. Vestergaard, M. Genois. *Temporal Gillespie Algorithm: Fast Simulation of Contagion Processes on Time Varying Networks*. PLoS Comput Biol **11(10)** (2015) doi:10.1371/journal.pcbi.1004579

[27] M. Boguna, L. F. Lafuerza, R. Toral, M.A. Serrano. *Simulating non-Markovian stochastic processes*. Cornell University (2014) arXiv:1310.0926v3 [cond-mat.dis-nn]

[28] R. Erban, S. J. Chapman, P.K. Maini. *A Practical Guide to Stochastic Simulations of Reaction-Diffusion Processes*. Cornell University (2007) arXiv:0704.1908 [q-bio.SC]

[29] P. Sinclair, M. Carballo-Pacheco, R. J. Allen. *Growth-dependent drug susceptibility can prevent or enhance spatial expansion of a bacterial population*. Phys. Biol. **16** (2019) 046001

[30] D. Emerson, R. M. Worden, J. A. Breznak. *A Diffusion Gradient Chamber for Studying Microbial Behavior and Separating Microorganisms*. Applied and Environmental Microbiology **60(4)** (1994) 1269 – 1278

[31] D. G. Aronson, H. F. Weinberger. *Multidimensional nonlinear diffusion arising in population genetics*. Adv. Math. **30** (1978) 33 –76. doi:10.1016/0001-8708(78)90130-5

[32] J. D. Murray. (2000) *Mathematical Biology: I. An Introduction, Third Edition*, Springer, New York City

[33] G. Nadin, B. Perthame, L. Ryzhik. *Traveling waves for the Keller-Segel system with Fisher birth terms*. Interfaces and Free Boundaries (2007) doi: 10.4171/IFB/200

[34] V. Calvez, B. Perthame, S. Yasuda. *TRAVELING WAVE AND AGGREGATION IN A FLUX-LIMITED KELLER-SEGEL MODEL.* Kinetic and Related Models , AIMS, **11 (4)** (2018) 891–909. ff10.3934/krm.2018035ff. ffhal-01591490ff

[35] S. J. Schrag, V. Perrot, and B. R. Levin. *Adaptation to the fitness costs of antibiotic resistance in Escherichia coli*. Proc. R. Soc. Lond. B **264** (1997) 1287–1291

[36] D. Tong. *Statistical Physics: University of Cambridge Part II Mathematical Tripos*. (2012) University of Cambridge, UK. [PDF] http://www.damtp.cam.ac.uk/user/tong/statphys.html

[37] M. Plischke, B. Bergersen. (2006) *Equilibrium Statistical Physics: 3rd Edition*. Singapore: World Scientific Publishing Co.

[38] R.K. Pathria, P.D. Beale. (2011) *Statistical Mechanics: Third Edition.* Oxford: Elsevier Ltd.

[39] R. R. Regoes, C. Wiuff, R. M. Zappala, K. N. Garner, F. Baquero, B. R. Levin. *Pharmacodynamic Functions: a Multiparameter Approach to the Design of Antibiotic Treatment Regimens*. Antimicrob Agents Chemother **48(10)** (2004) 3670–3676

[40] P. Greulich, M. Scott, M. R. Evans, R. J. Allen. *Growth-dependent bacterial susceptibility to ribosome-targeting antibiotics*. Mol. Syst. Biol. (2015) 11: 796

[41] K. Kovarova, A.J.B. Zehnder, T. Egli. *Temperature-Dependent Growth Kinetics of Escherichia coli ML 30 in Glucose-Limited Continuous Culture*. Journal of Bacteriology **178(15)** (1996) 4530 – 4539

[42] D. E. Dykhuizen. *Experimental Studies of Natural Selection in Bacteria*. Annu. Rev. Ecol. Syst. **21** (1990) 373 – 398

[43] T. D. Schneider, G. D. Stormo, L. Gold, and A. Ehrenfeucht. Information content of binding sites on nucleotide sequences. *J. Mol. Biol.* **188** (1986) 415–431

[44] F. Matthaus, M. Jagodic, J. Dobnikar. *E. Coli Superdiffusion and Chemotaxis–Search Strategy, Precision, and Motility*. Biophysical Journal **97** (2009) 946–957

[45] R. Phillips, J. Kondev, J. Theriot, H. Garcia. (2012) *Physical Biology of the cell. 2nd Edition*. New York: Garland Science, Taylor and Francis Group LLC ISBN: (Paperback)

[46] H. M. Al-Qadiri, N. Al-Alami, M. Lin, M. Al-Holy, A. G. Cavinato, B. A. Rasco. *Studying of the Bacterial Growth Phases Using Fourier Transform Infrared Spectroscopy and Multivariate Analysis*. Journal of Rapid Methods & Automation in Microbiology **16** (2008) 73–89

## Appendix

## A

## Ising Type Bacterial Evolution Model

```
% Mutation Model

kT = 0.2;
N = 100;

%Check to make sure the N dimension of array is even.
Even = mod(N,2);
if Even == 1 % Odd number
    disp('Cannot use this value. N has to be even.');
    return;
end
probSpinUp = 0.5;

% Set up the random spin grid "spin"
% 4 spins in the center with sigma = +1
spin = zeros(N);
spin(N/2,N/2) = 1;
spin(N/2+1, N/2) = 1;
spin(N/2, N/2+1) = 1;
spin(N/2+1, N/2+1) = 1;

%%
% %FOOD GRID

Food_Function = zeros(N);
Food_ElRow = [];
Food_ElCol = [];
ColMax = N/2;
RowMax = N/2;
ColMin = 0;
RowMin = 0;

for Food_ElRow = 1:N
    for Food_ElCol = 1:N
        if Food_ElCol <= N/2 && Food_ElRow <= N/2
            Food_Function(Food_ElRow,Food_ElCol) = max((ColMax+1 -
Food_ElCol),(RowMax+1 - Food_ElRow));
        elseif Food_ElCol > N/2 && Food_ElRow <= N/2
            Food_Function(Food_ElRow,Food_ElCol) = max((Food_ElCol-
ColMax),(RowMax+1 - Food_ElRow));
        elseif Food_ElCol <= N/2 && Food_ElRow > N/2
            Food_Function(Food_ElRow,Food_ElCol) = max((ColMax+1 -
Food_ElCol),(Food_ElRow- RowMax));
        elseif Food_ElCol > N/2  && Food_ElRow > N/2
            Food_Function(Food_ElRow,Food_ElCol) = max((Food_ElCol-
ColMax),(Food_ElRow- RowMax));
```

```
        end
    end
end

Food_Function = 3.0 + 2.0*(Food_Function/(N/2));
surf(Food_Function);

%% CIPRO GRID

Cipro_Function = zeros(N);
Cipro_ElRow = [];
Cipro_ElCol = [];
ColMax = N/2;
RowMax = N/2;
ColMin = 0;
RowMin = 0;
El_Col_Sum = 0;
for Cipro_ElRow = 1:N
    for Cipro_ElCol = 1:N
        El_Col_Sum = Cipro_ElRow + Cipro_ElCol;
        % Top Right Square
        if El_Col_Sum >= N + 1 && Cipro_ElCol > N/2 && Cipro_ElRow <=
N/2
            Cipro_Function(Cipro_ElRow,Cipro_ElCol) = max((Cipro_ElCol-
ColMax),(RowMax+1 - Cipro_ElRow));
        % Lower Left Square
        elseif El_Col_Sum >= N + 1 && Cipro_ElCol <= N/2 && Cipro_ElRow
> N/2
            Cipro_Function(Cipro_ElRow,Cipro_ElCol) = max((ColMax+1 -
Cipro_ElCol),(Cipro_ElRow- RowMax));
        % Lower Right Square
        elseif El_Col_Sum >= N + 1 && Cipro_ElCol > N/2  && Cipro_ElRow
> N/2
            Cipro_Function(Cipro_ElRow,Cipro_ElCol) = max((Cipro_ElCol-
ColMax),(Cipro_ElRow- RowMax));
        end
    end
end
Cipro_Function = 2.115*Cipro_Function/(N/2);
% surf(Cipro_Function);

%%
% The number of total iterations for the model
% Count is a count of the iterations and FreeEnergyCount is every 10000
% iterations
numIters = 700*numel(spin);
Count = 0;
FreeEnergyCount = 0;
%% Energy Matrix
% These matricies are made to record the total energy of the system
EnergyGrid = zeros(N);
PlotEnergy = zeros(1,numIters);
IterCount = [];
% Plot_Entropy_Energy records the total energy of the system for each
iter
% count to be used by the Entropy Energy Historgram later
```

```
Plot_Free_Energy = zeros(1,(numel(spin)/1));
Free_Energy_Recording_Count = numIters/(numel(spin)/1);
Record_Free_Energy = zeros(1, Free_Energy_Recording_Count);
Record_Entropy = zeros(1, Free_Energy_Recording_Count);

%%
% Opening a video file to record the model
v = VideoWriter('BacterialGrowth.avi');
open(v)

%% Begin running the simulation
for iter = 1 : numIters
    % Pick a random spin
    LengthOfSpinArray = randi(numel(spin));
    [row, col] = ind2sub(size(spin), LengthOfSpinArray);

    % Find its nearest neighbours
    % A = to the right [0 1]
    A = circshift(spin, [0 1]);
    % B = to the left [0 -1]
    B = circshift(spin, [0 -1]);
    % C = Shift down
    C = circshift(spin, [1 0]);
    % D = shift up
    D = circshift(spin, [-1 0]);
% The shifts move the surrounding neighbours to wherever the single
spin is surrounded, and then "neighbours" adds up those surrounding
spins at the same matrix element location of the surrounded spin,
(without adding the surrounded spin value to the same matrix element
location).
%     neighbours = A + B + C + D;

    % try to make a finite grid and not a torus shape continuous grid
    if row == N && col ~= 1 && col ~= N
        neighbours = A + B + C;
        neighbours_sqrd = A.*A + B.*B + C.*C;
        OneMinusNeighbour = (1 - A) + (1 - B) + (1 - C);
    elseif row == 1 && col ~= 1 && col ~= N
        neighbours = A + B + D;
        neighbours_sqrd = A.*A + B.*B + D.*D;
        OneMinusNeighbour = (1 - A) + (1 - B) + (1 - D);
    elseif col == 1 && row ~= 1 && row ~= N
        neighbours = B + C + D;
        neighbours_sqrd = B.*B + C.*C + D.*D;
        OneMinusNeighbour = (1 - B) + (1 - C) + (1 - D);
    elseif col == N && row ~= 1 && row ~= N
        neighbours = A + C + D;
        neighbours_sqrd = A.*A + C.*C + D.*D;
        OneMinusNeighbour = (1 - A) + (1 - C) + (1 - D);
    elseif row == 1 && col == 1
        neighbours = B + D;
        neighbours_sqrd = B.*B + D.*D;
        OneMinusNeighbour = (1 - B) + (1 - D);
    elseif row == 1 && col == N
        neighbours = A + D;
        neighbours_sqrd = A.*A + D.*D;
```

```
        OneMinusNeighbour = (1 - A) + (1 - D);
    elseif row == N && col == 1
        neighbours = B + C;
        neighbours_sqrd = B.*B + C.*C;
        OneMinusNeighbour = (1 - B) + (1 - C);
    elseif row == N && col == N
        neighbours = A + C;
        neighbours_sqrd = A.*A + C.*C;
        OneMinusNeighbour = (1 - A) + (1 - C);
    else
        neighbours = A + B + C + D;
        neighbours_sqrd = A.*A + B.*B + C.*C + D.*D;
        OneMinusNeighbour = (1 - A) + (1 - B) + (1 - C) +(1 - D);
    end

%% J values
   J = 1.86;
   Jd = 5.95;
   Jf = Food_Function(row,col);% 30 - 56
   Jc = Cipro_Function(row,col); % 0 - 20.7

   %% Remove Food to kill off bacteria
   if iter > 350*numel(spin)
       Jf = 0;
   end

%% BACTERIAL DEATH PROBABILITY VALUES
%     initial spin neighbour interactions
    death_neighbour_A = (1 - A.*A);
    death_neighbour_B = (1 - B.*B);
    death_neighbour_C = (1 - C.*C);
    death_neighbour_D = (1 - D.*D);
    death_neighbour_S = (1 - spin.*spin);

%   Initial Death Neighbours
    Initial_DN = [];
    Initial_DN = [death_neighbour_A(row,col), ...
        death_neighbour_B(row,col), death_neighbour_C(row,col), ...
        death_neighbour_D(row,col), death_neighbour_S(row,col)];

    if row == N && col ~= 1 && col ~= N
%       bottom row
%       growth_neighbours = (1 + A.*A) + (1 + B.*B) + (1 + C.*C);
        Counter_Food_Affinity =
((A(row,col))^2)*((B(row,col))^2)*((C(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((A(row,col))^2)*((B(row,col))^2)*((C(row,col))^2);
%         Jd = Jd*(Initial_DN(1) + Initial_DN(2) + Initial_DN(3));
    elseif row == 1 && col ~= 1 && col ~= N
%       top row
%       growth_neighbours = (1 - A.*A) + (1 - B.*B) + (1 - D.*D);
        Counter_Food_Affinity =
((A(row,col))^2)*((B(row,col))^2)*((D(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((A(row,col))^2)*((B(row,col))^2)*((D(row,col))^2);
%         Jd = Jd*(Initial_DN(1) + Initial_DN(2) + Initial_DN(4));
```

```
    elseif col == 1 && row ~= 1 && row ~= N
%         very left column
%         growth_neighbours = (1 - B.*B) + (1 - C.*C) + (1 - D.*D);
        Counter_Food_Affinity =
((B(row,col))^2)*((C(row,col))^2)*((D(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((B(row,col))^2)*((C(row,col))^2)*((D(row,col))^2);
%         Jd = Jd*(Initial_DN(2) + Initial_DN(3) + Initial_DN(4));
    elseif col == N && row ~= 1 && row ~= N
%         very right column
%         growth_neighbours = (1 - A.*A) + (1 - C.*C) + (1 - D.*D);
        Counter_Food_Affinity =
((A(row,col))^2)*((C(row,col))^2)*((D(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((A(row,col))^2)*((C(row,col))^2)*((D(row,col))^2);
%         Jd = Jd*(Initial_DN(1) + Initial_DN(3) + Initial_DN(4));
    elseif row == 1 && col == 1
%         top left corner
%         growth_neighbours = (1 - B.*B) + (1 - D.*D);
        Counter_Food_Affinity =
((B(row,col))^2)*((D(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((B(row,col))^2)*((D(row,col))^2);
%         Jd = Jd*(Initial_DN(2) + Initial_DN(4));
    elseif row == 1 && col == N
%         top right corner
%         growth_neighbours = (1 - A.*A) + (1 - D.*D);
        Counter_Food_Affinity =
((A(row,col))^2)*((D(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((A(row,col))^2)*((D(row,col))^2);
%         Jd = Jd*(Initial_DN(1) + Initial_DN(4));
    elseif row == N && col == 1
%         bottom left corner
%         growth_neighbours = (1 - B.*B) + (1 - C.*C);
        Counter_Food_Affinity =
((B(row,col))^2)*((C(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((B(row,col))^2)*((C(row,col))^2);
%         Jd = Jd*(Initial_DN(2) + Initial_DN(3));
    elseif row == N && col == N
%         bottom right corner
%         growth_neighbours = (1 - A.*A) + (1 - C.*C);
        Counter_Food_Affinity =
((A(row,col))^2)*((C(row,col))^2)*((spin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((A(row,col))^2)*((C(row,col))^2);
%         Jd = Jd*(Initial_DN(1) + Initial_DN(3));
    else
%         anywhere on the grid not on the edges
%growth_neighbours = (1 - A.*A) + (1 - B.*B) + (1 - C.*C) + (1 - D.*D);
        Counter_Food_Affinity =
((A(row,col))^2)*((B(row,col))^2)*((C(row,col))^2)*((D(row,col))^2)*((s
pin(row,col))^2);
        Counter_Food_Affinity_Spin1 =
((A(row,col))^2)*((B(row,col))^2)*((C(row,col))^2)*((D(row,col))^2);
```

```
%         Jd = Jd*(Initial_DN(1) + Initial_DN(2) + Initial_DN(3) +
Initial_DN(4));
    end

%%
% SPIN FLIP
% Calculate the energies of flipping a spin
   if spin(row,col) == 0
       % Growth: spin = 0 --> 1
       dE = -J*neighbours(row, col) + Jd - Jf*neighbours_sqrd(row,col)
+ Jc ...
           +
(J/2)*OneMinusNeighbour(row,col)*Counter_Food_Affinity_Spin1*exp(-
Food_Function(1,1) + Jf + 0.095);
       % Growth: spin = 0 --> -1
       %        -J*(si=-1)*neighbours + Jd*(si)^2 - Jf*(si)^2
       dE_0_n1 = J*neighbours(row,col) + Jd -
Jf*neighbours_sqrd(row,col) - Jc ...
           -
(J/2)*OneMinusNeighbour(row,col)*Counter_Food_Affinity*exp(-
Food_Function(1,1) + Jf + 0.095);

       dE_flip = 2*J*spin(row,col)*neighbours(row,col) -
2*Jc*spin(row,col);
   elseif spin(row,col) ~= 0
       % Death: spin = 1 --> 0
       dE = J*spin(row,col)*neighbours(row,col) - Jd +
Jf*neighbours_sqrd(row,col) - Jc*spin(row,col) ...
           - (1 - spin(row,col))*OneMinusNeighbour(row,col)*(J/2)*exp(-
Food_Function(1,1) + Jf + 0.095)*Counter_Food_Affinity;

       % dE_flip = [-J(-si) + Jc(-si)] - [(-J)(si) + Jc(si)]
       dE_flip = 2*J*spin(row,col)*neighbours(row,col) -
2*Jc*spin(row,col) ...
           - 2*spin(row,col)*OneMinusNeighbour(row,col)*(J/2)*exp(-
Food_Function(1,1) + Jf + 0.095)*Counter_Food_Affinity;
       dE_n1_1 = -2*J*neighbours(row,col) + 2*Jc;
   end
% Calculate the probability of flipping a spin based on the change in
energy
   prob = exp(-dE/kT);
   prob_0_n1 = exp(-dE_0_n1/kT);
   Mutant_Potential = exp(-dE_flip/kT);
% Enact the spin flips based on probability
   SystemChange = 0;
   prob_rand = rand();
   if spin(row, col)==0 && prob_rand < prob && dE < dE_0_n1
       spin(row,col) = 1;
       SystemChange = 1;
   elseif spin(row,col)~=0 && prob_rand < prob
       show_prob_rand = prob_rand;
       spin(row,col) = 0;
       SystemChange = 1;
   elseif spin(row,col)~= 0 && prob_rand < Mutant_Potential
       Mutant_Potential;
       spin(row,col) = -spin(row,col);
```

```
        SystemChange = 1;
    elseif spin(row,col) == 0 && dE_0_n1 < dE && prob_rand < prob_0_n1
        spin(row,col) = -1;
        SystemChange = 1;
    end

% Record an image of the system every 1000 iterations and put into
video file
    % "mod(iter,numel(spin)) == 0" selects every interval with length
numel(spin)
    if mod(iter,numel(spin)/10) == 0
        image((spin)*216,'CDataMapping','scaled');
%         xlabel(sprintf('T = %0.2f, M = %0.2f, E = %0.2f', T, Mmean,
TotalE));
%         set(gca,'YTickLabel',[],'XTickLabel',[]);
        axis square; colormap default; drawnow;

        frame = getframe(gcf);
        writeVideo(v,frame)
    end

     %% Initializing the energy for the total system
     %% Measuring the Energy for the total system
     Entropy_iter = mod(iter,numel(spin)/1);
     if Entropy_iter == 0
         Entropy_iter = 10000;
     end

     if iter >= 1 %&& mod(iter,numel(spin)) == 0 %(SystemChange == 1 ||
spin(row,col)~= 0)
         for newrow = 1:N
             for newcol = 1:N
                 if newrow == N && newcol ~= 1 && newcol ~= N
                     neighbours = A(newrow,newcol) + B(newrow,newcol) +
C(newrow,newcol);
                     neighbours_sqrd = ((A(newrow,newcol))^2) +
((B(newrow,newcol))^2) + ((C(newrow,newcol))^2);
                     OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
B(newrow,newcol)) + (1 - C(newrow,newcol));
                     Counter_Food_Affinity =
((A(newrow,newcol))^2)*((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                     Jf = Food_Function(newrow,newcol);
                     Jc = Cipro_Function(newrow,newcol);
                 elseif newrow == 1 && newcol ~= 1 && newcol ~= N
                     neighbours = A(newrow,newcol) + B(newrow,newcol) +
D(newrow,newcol);
                     neighbours_sqrd = ((A(newrow,newcol))^2) +
((B(newrow,newcol))^2) + ((D(newrow,newcol))^2);
                     OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
B(newrow,newcol)) + (1 - D(newrow,newcol));
                     Counter_Food_Affinity =
((A(newrow,newcol))^2)*((B(newrow,newcol))^2)*((D(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                     Jf = Food_Function(newrow,newcol);
                     Jc = Cipro_Function(newrow,newcol);
```

```
                elseif newcol == 1 && newrow ~= 1 && newrow ~= N
                    neighbours = B(newrow,newcol) + C(newrow,newcol) +
D(newrow,newcol);
                    neighbours_sqrd = ((B(newrow,newcol))^2) +
((C(newrow,newcol))^2) + ((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - B(newrow,newcol)) + (1 -
C(newrow,newcol)) + (1 - D(newrow,newcol));
                    Counter_Food_Affinity =
((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((D(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newcol == N && newrow ~= 1 && newrow ~= N
                    neighbours = A(newrow,newcol) + C(newrow,newcol) +
D(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((C(newrow,newcol))^2) + ((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
C(newrow,newcol)) + (1 - D(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((C(newrow,newcol))^2)*((D(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == 1 && newcol == 1
                    neighbours = B(newrow,newcol) + D(newrow,newcol);
                    neighbours_sqrd = ((B(newrow,newcol))^2) +
((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - B(newrow,newcol)) + (1 -
D(newrow,newcol));
                    Counter_Food_Affinity =
((B(newrow,newcol))^2)*((D(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == 1 && newcol == N
                    neighbours = A(newrow,newcol) + D(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
D(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((D(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == N && newcol == 1
                    neighbours = B(newrow,newcol) + C(newrow,newcol);
                    neighbours_sqrd = ((B(newrow,newcol))^2) +
((C(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - B(newrow,newcol)) + (1 -
C(newrow,newcol));
                    Counter_Food_Affinity =
((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
```

```
                elseif newrow == N && newcol == N
                    neighbours = A(newrow,newcol) + C(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((C(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
C(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((C(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                else
                    neighbours = A(newrow,newcol) + B(newrow,newcol) +
C(newrow,newcol) + D(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((B(newrow,newcol))^2) + ((C(newrow,newcol))^2) +
((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
B(newrow,newcol)) + (1 - C(newrow,newcol)) +(1 - D(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((
D(newrow,newcol))^2)*((spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                end

                El_Energy = -(J/2)*spin(newrow,newcol)*neighbours +
Jd*(spin(newrow,newcol))^2 ...
                    - (Jf/2)*neighbours_sqrd*(spin(newrow,newcol))^2 +
Jc*spin(newrow,newcol) ...
                    - (J/2)*(1 -
spin(newrow,newcol))*OneMinusNeighbour*exp(-Food_Function(1,1)/kT +
Jf/kT + (2*0.095)/(10*kT))*Counter_Food_Affinity;
                EnergyGrid(newrow,newcol) = El_Energy;

            end
        end
        for newrow = N:-1:1
            for newcol = N:-1:1
                if newrow == N && newcol ~= 1 && newcol ~= N
                    neighbours = A(newrow,newcol) + B(newrow,newcol) +
C(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((B(newrow,newcol))^2) + ((C(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
B(newrow,newcol)) + (1 - C(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == 1 && newcol ~= 1 && newcol ~= N
                    neighbours = A(newrow,newcol) + B(newrow,newcol) +
D(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((B(newrow,newcol))^2) + ((D(newrow,newcol))^2);
```

```
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
B(newrow,newcol)) + (1 - D(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((B(newrow,newcol))^2)*((D(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newcol == 1 && newrow ~= 1 && newrow ~= N
                    neighbours = B(newrow,newcol) + C(newrow,newcol) +
D(newrow,newcol);
                    neighbours_sqrd = ((B(newrow,newcol))^2) +
((C(newrow,newcol))^2) + ((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - B(newrow,newcol)) + (1 -
C(newrow,newcol)) + (1 - D(newrow,newcol));
                    Counter_Food_Affinity =
((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((D(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newcol == N && newrow ~= 1 && newrow ~= N
                    neighbours = A(newrow,newcol) + C(newrow,newcol) +
D(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((C(newrow,newcol))^2) + ((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
C(newrow,newcol)) + (1 - D(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((C(newrow,newcol))^2)*((D(newrow,newcol))^2)*((
spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == 1 && newcol == 1
                    neighbours = B(newrow,newcol) + D(newrow,newcol);
                    neighbours_sqrd = ((B(newrow,newcol))^2) +
((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - B(newrow,newcol)) + (1 -
D(newrow,newcol));
                    Counter_Food_Affinity =
((B(newrow,newcol))^2)*((D(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == 1 && newcol == N
                    neighbours = A(newrow,newcol) + D(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
D(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((D(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == N && newcol == 1
                    neighbours = B(newrow,newcol) + C(newrow,newcol);
                    neighbours_sqrd = ((B(newrow,newcol))^2) +
((C(newrow,newcol))^2);
```

```
                    OneMinusNeighbour = (1 - B(newrow,newcol)) + (1 -
C(newrow,newcol));
                    Counter_Food_Affinity =
((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                elseif newrow == N && newcol == N
                    neighbours = A(newrow,newcol) + C(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((C(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
C(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((C(newrow,newcol))^2)*((spin(newrow,newcol))^2)
;
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                else
                    neighbours = A(newrow,newcol) + B(newrow,newcol) +
C(newrow,newcol) + D(newrow,newcol);
                    neighbours_sqrd = ((A(newrow,newcol))^2) +
((B(newrow,newcol))^2) + ((C(newrow,newcol))^2) +
((D(newrow,newcol))^2);
                    OneMinusNeighbour = (1 - A(newrow,newcol)) + (1 -
B(newrow,newcol)) + (1 - C(newrow,newcol)) +(1 - D(newrow,newcol));
                    Counter_Food_Affinity =
((A(newrow,newcol))^2)*((B(newrow,newcol))^2)*((C(newrow,newcol))^2)*((
D(newrow,newcol))^2)*((spin(newrow,newcol))^2);
                    Jf = Food_Function(newrow,newcol);
                    Jc = Cipro_Function(newrow,newcol);
                end

                El_Energy = -(J/2)*spin(newrow,newcol)*neighbours +
Jd*(spin(newrow,newcol))^2 ...
                    - (Jf/2)*neighbours_sqrd*(spin(newrow,newcol))^2 +
Jc*spin(newrow,newcol) ...
                    - (J/2)*(1 -
spin(newrow,newcol))*OneMinusNeighbour*exp(-Food_Function(1,1)/kT +
Jf/kT + (2*0.095)/(10*kT))*Counter_Food_Affinity;
                EnergyGrid(newrow,newcol) = El_Energy;
            end
        end
% Calculate total energy of the system for use in plots and histogram
    Tot_Energy = sum(sum(EnergyGrid));
    end
    Plot_Free_Energy(Entropy_iter) = Tot_Energy;
    PlotEnergy(iter) = Tot_Energy;
    Count = Count + 1;

    %% Imaging the energy for the total system

%     if mod(iter,numel(spin)/100) == 0 % selects every 1000 iterations
if N =100
%         image((EnergyGrid)*216,'CDataMapping','scaled');
```

```
% %            xlabel(sprintf('T = %0.2f, M = %0.2f, E = %0.2f', T,
Mmean, TotalE));
% %            set(gca,'YTickLabel',[],'XTickLabel',[]);
%        axis square; colormap default; drawnow;
%
%        frame = getframe(gcf);
%        writeVideo(v,frame)
%     end

%% Calculate Entropy Per Iteration

% if mod(iter,numel(spin)/1) == 0 %&& iter == 300*numel(spin) % selects
every 1000 iterations
%
%     FreeEnergyHistRange = round(range(Plot_Free_Energy),0);
%     if FreeEnergyHistRange == 0
%         FreeEnergyHistRange = 1;
%     end
%     FreeEnergyHistdata = Plot_Free_Energy;
%     RangeOfFreeEnergyData = zeros(FreeEnergyHistRange,1);
%     LowestDataValue = min(FreeEnergyHistdata);
%     for HistElements = 1:length(Plot_Free_Energy)
%         HistogramPosition = round(FreeEnergyHistdata(HistElements) -
LowestDataValue,0);
%         if HistogramPosition == 0
%             HistogramPosition = 1;
%         end
%         RangeOfFreeEnergyData(HistogramPosition) =
RangeOfFreeEnergyData(HistogramPosition) + 1;
%     end
%
% %     data = PlotEnergy;
% %     RangeOfData = zeros(round(range(PlotEnergy),0),1);
% %     if RangeOfData == 0
% %         RangeOfData = 1;
% %     end
% %     LowestDataValue = min(data);
% %     for HistElements = 1:length(PlotEnergy)
% %         HistogramPosition = round(data(HistElements) -
LowestDataValue,0);
% %         if HistogramPosition == 0
% %             HistogramPosition = 1;
% %         end
% %         RangeOfData(HistogramPosition) =
RangeOfData(HistogramPosition) + 1;
% %     end
%
%     Element = [];
%     for HistElements2 = 1:length(RangeOfFreeEnergyData)
%         Element = [Element -round(-(LowestDataValue) -
HistElements2)];
%     end
%     Element = Element';
%
% %     % % Histogram Energy Plot
% %     bar(Element, RangeOfEntropyData, 'BarWidth', 20)
```

```
% % %     xlim([-180000 -165000])
% %     ylim([0 200])%max(RangeOfData)+2000])
% %     grid minor;
% %     title('Frequency vs Total Energy')
% %     xlabel('Total Energy')
% %     ylabel('Frequency')
%
%     Tot_Free_Energy = 0;
%     Tot_Entropy = 0;
%     Free_Energy = zeros(1,length(RangeOfFreeEnergyData));
%     Entropy = zeros(1,length(RangeOfFreeEnergyData));
%     for HistElements = 1:length(RangeOfFreeEnergyData)
%         EnergyValue = Element(HistElements);
%         HistValue = RangeOfFreeEnergyData(HistElements);
%         Free_Energy(HistElements) =
(HistValue/numel(spin))*(EnergyValue ...
%             + kT*log(HistValue/numel(spin)));
% %         if Free_Energy(HistElements) ~= 0
% %             EnergyValue
% %             HistValue/numel(spin)
% %             Current_FreeEn = Free_Energy(HistElements)
% %         end
%         Entropy(HistElements) = -
(HistValue/numel(spin))*log(HistValue/numel(spin));
%     end
%     Free_Energy(isnan(Free_Energy))=0;
%     Entropy(isnan(Entropy))=0;
%     Point = iter/10000
%     Tot_Free_Energy = sum(Free_Energy)
%     Tot_Entropy = sum(Entropy)
%     Tot_Energy
%     FreeEnergyCount = FreeEnergyCount + 1;
%     Record_Free_Energy(FreeEnergyCount) = Tot_Free_Energy;
%     Record_Entropy(FreeEnergyCount) = Tot_Entropy;
%   Plot_Free_Energy = zeros(1,numel(spin)/1);
%
% %     Tot_Entropy_Label = ['Total Entropy = ',
num2str(Total_Entropy)];
% %     % % Entropy Energy Plot
% %     plot(Element, Entropy, '.')
% %     grid minor;
% %     title('Entropy vs Total Energy')
% %     xlabel('Total Energy')
% %     ylabel('Entropy')
% %     legend(Tot_Entropy_Label)
% end


%% Extra

    % Testing ends when mutation occurs
%     if spin(row,col) == -1
%         break
%     end

    if iter == 0.05*numIters
```

```
        disp('5% complete')
    elseif iter == 0.1*numIters
        disp('10% complete')
    elseif iter == 0.15*numIters
        disp('15% complete')
    elseif iter == 0.2*numIters
        disp('20% complete')
    elseif iter == 0.25*numIters
        disp('25% complete')
    elseif iter == 0.33*numIters
        disp('33% complete')
    elseif iter == 0.4*numIters
        disp('40% complete')
    elseif iter == 0.45*numIters
        disp('45% complete')
    elseif iter == 0.5*numIters
        disp('50% complete')
    elseif iter == 0.667*numIters
        disp('66% complete')
    elseif iter == 0.75*numIters
        disp('75% complete')
    elseif iter == 0.8*numIters
        disp('80% complete')
    elseif iter == 0.85*numIters
        disp('85% complete')
    elseif iter == 0.9*numIters
        disp('90% complete')
    elseif iter == 0.95*numIters
        disp('95% complete')
    end
end

% Close the video file
close(v);

% Create an array of iterations for plots
IterCount = linspace(1, numIters, Count);
Free_Energy_IterCount = linspace(1, numIters, FreeEnergyCount);

%% Creating Energy and Free Energy Histogram

% % "PlotEnergy" is the energy matrix created which records the total
energy
%     % of the system for each iteration
% % "data" is the absolute value of all the energy values calculated
for PlotEnergy
% data = PlotEnergy;
% % data = abs(PlotEnergy);
% % The range of energy for the histogram "range(PlotEnergy)"
% % "round" rounds any decimal value to the nearest integer
% % RangeOfData creats an array of zeros representing the range of
energies
% RangeOfData = zeros(round(range(PlotEnergy),0),1);
% % The lowest value for the range of data in the "data" array
% LowestDataValue = min(data);
%
```

```
% % "length(PlotEnergy)" is the number of elements in PlotEnergy
% for HistElements = 1:length(PlotEnergy)
%     HistogramPosition = round(data(HistElements) -
LowestDataValue,0);
%     if HistogramPosition == 0
% %         SHOW_DATA = data(HistElements)
%         HistogramPosition = 1;
%     end
%     % Put an energy value into the "bin"
%     RangeOfData(HistogramPosition) = RangeOfData(HistogramPosition) +
1;
% end
%
% Element = [];
% for HistElements2 = 1:length(RangeOfData)
%     Element = [Element -round(-(LowestDataValue) - HistElements2)];
% end
% Element = Element';

%% Total Free Energy

% SumAll = 0;
% Tot_Free_Energy = 0;
% Free_Energy = zeros(1,length(RangeOfData));
% for HistElements = 1:length(RangeOfData)
%     EnergyValue = Element(HistElements);
%     HistValue = RangeOfData(HistElements);
%     Free_Energy(HistElements) =
(HistValue/sum(RangeOfData))*(EnergyValue ...
%         + kT*log(HistValue/sum(RangeOfData)));
%     SumAll = SumAll + RangeOfData(HistElements);
% end
% Free_Energy(isnan(Free_Energy))=0;
% % SumAll
% % Free_Energy
% Tot_Free_Energy = sum(Free_Energy)
% Exp_Free_Energy = exp((1/kT)*Tot_Free_Energy);

%% Free Energy per iteration
%
% Tot_Free_Energy = 0;
% Free_Energy = zeros(1,length(PlotEnergy));
% for HistElements = 1:length(PlotEnergy)
%     EnergyValue = PlotEnergy(HistElements);
%     HistValue = RangeOfData(HistElements);
%     Free_Energy(HistElements) =
(HistValue/sum(RangeOfData))*(EnergyValue ...
%         + kT*log(HistValue/sum(RangeOfData)));
%     SumAll = SumAll + RangeOfData(HistElements);
% end
% Free_Energy(isnan(Free_Energy))=0;
%
% Tot_Free_Energy = sum(Free_Energy);
% Exp_Free_Energy = exp((1/kT)*Tot_Free_Energy);

%%
```

```
% % Density of States
% Density_of_States = zeros(1,length(RangeOfData));
% for HistElements = 1:length(RangeOfData)
%     EnergyValue = Element(HistElements);
%     HistValue = RangeOfData(HistElements);
%     Density_of_States(HistElements) = ...
%         (HistValue/sum(RangeOfData))*exp(EnergyValue*(1/kT) ...
%         - Tot_Free_Energy);
%     if HistElements = 0
%         Energy = Element(HistElements)
%         DoS = Density_of_States(HistElements)
%     end
% end
%
% plot(Element, Density_of_States, '.')
% xlim([-170000 2000])
% % ylim([0 0.00001])
% grid minor;
% title('Total Energy vs DoS')
% xlabel('Total Energy')
% ylabel('Density of States')


%% Plots

% % % Histogram Energy Plot
% bar(Element, RangeOfData, 'BarWidth', 20)
% % xlim([-1000 2000])
% ylim([0 1000])%max(RangeOfData)+2000])
% grid minor;
% title('Frequency vs Total Energy')
% xlabel('Total Energy')
% ylabel('Frequency')

% % % Iteration vs Energy Plot
% figure(f1);
% plot(IterCount, PlotEnergy, '.')
% grid minor;
% title('Total Energy vs Iterations')
% xlabel('Iterations')
% ylabel('Total Energy')

% % % Iteration vs Entropy Plot
% plot(Entropy_IterCount, Record_Entropy, '.')
% grid minor;
% title('Total Entropy vs Iterations')
% xlabel('Iterations')
% ylabel('Total Entropy')

% % % Free Energy Plot
% % Tot_Free_Energy_Label = ['Total Free Energy = ',
num2str(Tot_Free_Energy)];
% figure(f2);
% plot(Free_Energy_IterCount, Record_Free_Energy, '.')
% grid minor;
% title('Total Free Energy vs Iteration')
```

```
% xlabel('Iteration')
% ylabel('Free Energy')
% % legend(Tot_Free_Energy_Label)

% figure(f1);
% % % Iteration vs Energy Plot
% plot(IterCount, PlotEnergy, '.')
% hold on
% plot(Free_Energy_IterCount, Record_Free_Energy, '.')
% grid minor;
% title('Total Energy and Free Energy vs Iterations')
% xlabel('Iterations')
% ylabel('Total Energy or Free Energy')
% hold off
%
% figure(f2);
% % % Iteration vs Entropy Plot
% plot(Free_Energy_IterCount, Record_Entropy, '.')
% grid minor;
% title('Total Entropy vs Iterations')
% xlabel('Iterations')
% ylabel('Total Entropy')
```

## B

## The SSA Model for Mutant Fixation at Goldilocks Point

```
%*********** SSA For Mutant Fixation at Goldilocks Point
%***** This version calculates move based on unit distance dx = 1

% Create array for plotting
itertot = 1; % total iteration
ntot = 300; % total particle number
ttim = 1500; % total time
nl = 601;   % Total Position if particle position of particles
Even = mod(nl,2);
if Even == 0 % Even number
    disp('Cannot use this value. nl has to be odd.');
    return;
end
xd = 5; % diffusion coefficient
dx = 1; % unit position
xcons = 1/(2*xd);
dt = dx*dx*xcons; % Fick's diffusion length with RMS coefficient,
                  % D = (1/(2*dimension))d/dt(rms(dx^2))
                  % In this model, dimension = 1
                  % dx = sqrt(2*D*dt) = 1
% a measure of how far the concentration has propagated
% in the x-direction by diffusion in time t
Fit_m1 = 20; % The fitness of the bacteria is 2 for the first mutation
Fit_m2 = 100; % The fitness of the bacteria is 3 for the second
mutation
Fit_m3 = 200; % The fitness of the bacteria is 3 for the second
mutation
%% INSEERT FOOD CONCENTRATION
Food_Function = zeros(nl,1);
LeftConc = (nl-1)/2; % The max concentration of food is 40
RightConc = (nl-1)/2;
for Food_Pos = 1:nl
    if Food_Pos <= ((nl-1)/2) %From 1 to (Middle - 1)
%         Food_Function(Food_Pos) = (LeftConc+2 - Food_Pos);
%         % Change Linear Function to Exponential
%         Food_Function(Food_Pos) = exp(log(((nl-
1)/2)+1)*Food_Function(Food_Pos)/(((nl-1)/2)+1));
        Food_Function(Food_Pos) = 0;
    elseif Food_Pos >= ((nl-1)/2)+1 % From Middle to Max (41 being the
middle)
        Food_Function(Food_Pos) = (Food_Pos - RightConc);
        % Change Linear Function to Exponential
        Food_Function(Food_Pos) = exp(log(((nl-
1)/2)+1)*Food_Function(Food_Pos)/(((nl-1)/2)+1));
    end
end
plot(Food_Function)
xlim([1 nl])
```

```
xbias = 0.01 + 5*Food_Function/(((nl-1)/2) + 1);
% Removing baseline value
% xbias = Food_Function/(((nl-1)/2) + 1);
Max_xbias = 4.5; % Food Gradient
for il = 1: nl
    if xbias(il) > Max_xbias
        xbias(il) = Max_xbias;
    end
end
plot(xbias)
xlim([1 nl])

%% Constant Food Function
% for Food_Pos = 1:nl
%     Food_Function(Food_Pos) = 0.03;
% end
% xbias = Food_Function;

%% INSEERT CIPROFLOXACIN CONCENTRATION
Cipro_Function = zeros(nl,1);
LCipConc = (nl-1)/2; % The max concentration of food is 40
RCiptConc = (nl-1)/2;
for Cipro_Pos = 1:nl
    if Cipro_Pos <= ((nl-1)/2) %From 1 to (Middle - 1)
%         Cipro_Function(Cipro_Pos) = (LCipConc+2 - Cipro_Pos);
%         % Change Linear Function to Exponential
%         Cipro_Function(Cipro_Pos) = exp(log(((nl-
1)/2)+1)*Cipro_Function(Cipro_Pos)/(((nl-1)/2)+1));
        Cipro_Function(Cipro_Pos) = 0;
    elseif Cipro_Pos >= ((nl-1)/2)+1 % From Middle to Max (41 being the
middle)
        Cipro_Function(Cipro_Pos) = (Cipro_Pos - RCiptConc);
        % Change Linear Function to Exponential
        Cipro_Function(Cipro_Pos) = exp(log(((nl-
1)/2)+1)*Cipro_Function(Cipro_Pos)/(((nl-1)/2)+1));
    end
end

plot(Cipro_Function)
xlim([1 nl])
% Adjusting for a new volume of cipro
Cip_xbias = 7*(Cipro_Function/(((nl-1)/2) + 1));

plot(Cip_xbias)
xlim([1 nl])

%% Constant Cipro Function
% for Cipro_Pos = 1:nl
%     Cipro_Function(Cipro_Pos) = 0.02;
% end
% Cip_xbias = Cipro_Function;
% % Cip_xbias = zeros(nl,1);

%% initialize position space
w = zeros(nl,ttim); % Initialize wild type baceria array
mut1 = zeros(nl,ttim); % Initialize mutant 1 array
```

```
mut2 = zeros(nl,ttim); % Initialize mutant 2 array
mut3 = zeros(nl,ttim); % Initialize mutant 3 array
CC = 400; % Carrying Capacity
Death_Prob = 0.001; % The Death Rate
Mut_Rate = 0.001; % Rate of Mutation
Migration_Prob = 0.1; % Probability of Migration


%%  Set iteration loop
% Open video file to record
v = VideoWriter('BacteriaMutantFix.avi');
open(v)

for iter = 1:itertot; % set iterationloop
% first initialization
% x is a zeros matrix with dimensions "n1" by "ttim"
x = zeros(nl,ttim);
xx = zeros(nl,ttim);
m1 = zeros(nl,ttim);
mm1 = zeros(nl,ttim);
m2 = zeros(nl,ttim);
mm2 = zeros(nl,ttim);
m3 = zeros(nl,ttim);
mm3 = zeros(nl,ttim);

% initialize in the middle x = (nl-1)/2+1
x(((nl-1)/2+1),1) = ntot; % Wild type bacteria placed in the center
xx(((nl-1)/2+1),1) = ntot;
m1(((nl-1)/2+1),1) = 0; % The mutant bacteria are not
mm1(((nl-1)/2+1),1) = 0;%   placed in the initial model
m2(((nl-1)/2+1),1) = 0;
mm2(((nl-1)/2+1),1) = 0;
m3(((nl-1)/2+1),1) = 0;
mm3(((nl-1)/2+1),1) = 0;

xttim = 1;
itim = 1;
next_num = itim + 1;
    % while dx is constant, dt is what helps move the particles around
        % what shifts the particles is different lengths of time and
not a
        % constant time with various distances travelled
        % dt varies while dx = 1
    while (itim < ttim) % set time while loop

        mutation_time = 0; % Used to check the first mutation
        % ttimold is the time before adding dt to itim, (xttim = itim +
dt)
        ttimold = itim;
        nxcount = 0; % count number of particles
        % particle postion loop through all particle at postion il
        for il = 1:nl % select position loop

%% Wild Type Bacteria
             % BEGIN particle diffusion/raction
```

```
             % nxloop selects the particle at the old time and old
position
                % xx(nl,ttim) to determine which way it moves then
records
                % it into new position x(nl,ttim)
              % nxloop records the total number of particles at the
                % original position and the old time
              nxloop = xx(il,ttimold);
              for ixl = 1:nxloop   % begin particle position loop
                 nxcount = nxcount + 1; % count number of particles
                 direction1 = rand(); % Random # between -1 and 1
                 dir = 1;
                 if (direction1 < 0.5)
                     dir = -1;
                 end
                 distance = rand();
                 dxx = (dx*dir)*distance;
                 xxx = il + dxx;
                 if (xxx < 1)
                     xxx = 1; % The bacteria have stopped moving and
settled
                              % On the next loop the particles have a
chance
                              % of moving backwards or staying
                 end
                 if (xxx > nl)
                     xxx = nl;
                 end
                 xxx = round(xxx);
% GROWTH, DEATH, MIGRATION, AND CARRYING CAPACITY
rep = rand();
mig = rand();
die = rand();
mut = rand();
Food_Conc = xbias;
Food_Factor = Food_Conc(il);
Cipro_Conc = Cip_xbias;
Cipro = Cipro_Conc(il); % Cipro concentration is adjusted here
K = Max_xbias/4; % The Monod Constant is the value of food when growth
rate
                 % is 1/2 the maximum growth rate
                 % Measured in number of bacterial yields per
microlitre
                 % Max_xbias/3 assumes it is at 1/3 the food
concentration
% Max growth is set to 1 for this equation
Monod = 1.25*(Food_Factor)/(K + Food_Factor);
Max_Monod = 1.25*(Max_xbias/(K + Max_xbias)); % Maximum Growth Rate
% Fast-Growth Targeting Anitibiotic MIC
MICW = 1*(0.03 - 0.027*(Monod/Max_Monod)); % for Wild Type Bacteria
% for FGTA the MIC decreases with growth rate
  % when the growth rate is faster the concentration needed to inhibit
growth
  % decreases and less antibiotic is needed
% The pharmacodynamic function describes bacterial inhibition by the
antibiotic
```

```
   % because of MIC, the greater the growth rate, the less amount of
cipro is
   % needed to inhibit the bacteria
   % with diminishing returns, more antibiotic is needed to eliminate
   % fewer bacteria, so as food concentration increases growth rate the
   % cipro has a greater effect of eliminating bacteria
% Monod and the Pharmacodynamic Function describe the probability of
% replication for the selected bacterium
Pharma_Function = 1 - ((Cipro)/MICW)^2;
% Geographic restriction can prevent the selected bacterium to grow
Pop_Factor = (1 - (x(il,itim) + m1(il,itim) + m2(il,itim) +
m3(il,itim))/CC);
if (x(il,itim) + m1(il,itim) + m2(il,itim) + m3(il,itim)) > CC
    Pop_Factor = 0;
end
% The probability of Replication is a combination of three factors like
in
% equation 41
Replication_Prob = Monod*Pharma_Function*Pop_Factor;

                if (rep < Replication_Prob)
                    % THIS IS WHERE MUTATION TAKES PLACE
                    if (mut > 0.0002*Mut_Rate) && (mut < Mut_Rate)
                        m1(il,itim) = m1(il,itim) + 1;
                    else
                        x(il,itim) = x(il,itim) + 1;
                    end
                elseif (die < Death_Prob) && (x(il,itim) > 0) % Death
Rate
                    % Death rate needs to be proportional to food
                    % concentration, fitness, cipro concentration, with
a
                    % minor constant death rate.
                    x(il,itim) = x(il,itim) - 1;
% The fixed rate should be a turnover rate where at steady state when
there
% is no cipro and there is food, the steady state is reached
                else
                    x(il,itim) = x(il,itim);
                    % Each position can grow up to CC
                    if (x(xxx,itim) < CC) && (mig < Migration_Prob)
                        % Remove particle from this position
                        x(il,itim) = x(il,itim) - 1;
                        % Place particle in this position
                        x(xxx,itim) = x(xxx,itim) + 1;
                    end
                end
            end % end particle position loop

%% Mutant 1
            mxloop = mm1(il,ttimold);
            for ixl = 1:mxloop   % begin particle position loop
               nxcount = nxcount + 1; % count number of particles
               % Migration
               direction1 = rand(); % Random # between -1 and 1
               dir = 1;
```

```
                if (direction1 < 0.5)
                    dir = -1;
                end
                distance = rand();
                dxx = (dx*dir)*distance;
                xxx = il + dxx;
                if (xxx < 1)
                    xxx = 1; % The bacteria have stopped moving and
settled
                end
                if (xxx > nl)
                    xxx = nl;
                end
                xxx = round(xxx);

% GROWTH, DEATH, MIGRATION, AND CARRYING CAPACITY
rep = rand();
mig = rand();
die = rand();
mut = rand();
Food_Conc = xbias;
Food_Factor = Food_Conc(il);
Cipro_Conc = Cip_xbias;
Cipro = Cipro_Conc(il); % Cipro concentration is adjusted here
K = Max_xbias/4;
Monod = 1.25*(Food_Factor)/(K + Food_Factor); % Max growth is set to 1
for this equation
Max_Monod = 1.25*(Max_xbias/(K + Max_xbias)); % Maximum Growth Rate
MIC1 = Fit_m1*(0.03 - 0.027*(Monod/Max_Monod)); % Fast-Growth Targeting
Anitibiotic MIC
Pharma_Function = 1 - ((Cipro)/MIC1)^2;
Pop_Factor = (1 - (x(il,itim) + m1(il,itim) + m2(il,itim) +
m3(il,itim))/CC);
if (x(il,itim) + m1(il,itim) + m2(il,itim) + m3(il,itim)) > CC
    Pop_Factor = 0;
end
Replication_Prob = Monod*Pharma_Function*Pop_Factor;

                if (rep < Replication_Prob)
                    if mut < 0.0002*Mut_Rate
                        x(il,itim) = x(il,itim) + 1;
                    % THIS IS WHERE MUTATION TAKES PLACE
                    elseif (mut > 0.01*Mut_Rate) && (mut < Mut_Rate)
                        m2(il,itim) = m2(il,itim) + 1;
                    else
                        m1(il,itim) = m1(il,itim) + 1;
                    end
                elseif (die < Death_Prob) && (m1(il,itim) > 0) % Death
Rate
                    m1(il,itim) = m1(il,itim) - 1;
                else
                    m1(il,itim) = m1(il,itim);
                    % Each position can grow up to CC
                    if (m1(xxx,itim) < CC) && (mig < Migration_Prob)
                        m1(il,itim) = m1(il,itim) - 1;
                        m1(xxx,itim) = m1(xxx,itim) + 1;
```

```
                end
            end
        end % end particle position loop

%% Mutant 2
        mxloop = mm2(il,ttimold);
        for ixl = 1:mxloop   % begin particle position loop
            nxcount = nxcount + 1; % count number of particles
            % Migration
            direction1 = rand(); % Random # between -1 and 1
            dir = 1;
            if (direction1 < 0.5)
                dir = -1;
            end
            distance = rand();
            dxx = (dx*dir)*distance;
            xxx = il + dxx;
            if (xxx < 1)
                xxx = 1; % The bacteria have stopped moving and
settled
            end
            if (xxx > nl)
                xxx = nl;
            end
            xxx = round(xxx);
% GROWTH, DEATH, MIGRATION, AND CARRYING CAPACITY
rep = rand();
mig = rand();
die = rand();
mut = rand();
Food_Conc = xbias;
Food_Factor = Food_Conc(il);
Cipro_Conc = Cip_xbias;
Cipro = Cipro_Conc(il); % Cipro concentration is adjusted here
K = Max_xbias/4;
Monod = 1.25*(Food_Factor)/(K + Food_Factor); % Max growth is set to 1
for this equation
Max_Monod = 1.25*(Max_xbias/(K + Max_xbias)); % Maximum Growth Rate
MIC2 = Fit_m2*(0.03 - 0.027*(Monod/Max_Monod)); % Fast-Growth Targeting
Anitibiotic MIC
Pharma_Function = 1 - ((Cipro)/MIC2)^2;
Pop_Factor = (1 - (x(il,itim) + m1(il,itim) + m2(il,itim) +
m3(il,itim))/CC);
if (x(il,itim) + m1(il,itim) + m2(il,itim) + m3(il,itim)) > CC
    Pop_Factor = 0;
end
Replication_Prob = Monod*Pharma_Function*Pop_Factor;

            if (rep < Replication_Prob)
                if mut < 0.0002*Mut_Rate
                    m1(il,itim) = m1(il,itim) + 1;
  % THIS IS WHERE MUTATION TAKES PLACE
                elseif (mut < Mut_Rate)
                    m3(il,itim) = m3(il,itim) + 1;
                else
                    m2(il,itim) = m2(il,itim) + 1;
```

```
                    end
                elseif (die < Death_Prob) && (m2(il,itim) > 0) % Death
Rate
                    m2(il,itim) = m2(il,itim) - 1;
                else
                    m2(il,itim) = m2(il,itim);
                    % Each position can grow up to CC
                    if (m2(xxx,itim) < CC) && (mig < Migration_Prob)
                        m2(il,itim) = m2(il,itim) - 1;
                        m2(xxx,itim) = m2(xxx,itim) + 1;
                    end
                end
            end % end particle position loop
%% Mutant 3
            mxloop = mm3(il,ttimold);
            for ixl = 1:mxloop   % begin particle position loop
                nxcount = nxcount + 1; % count number of particles
                % Migration
                direction1 = rand(); % Random # between -1 and 1
                dir = 1;
                if (direction1 < 0.5)
                    dir = -1;
                end
                distance = rand();
                dxx = (dx*dir)*distance;
                xxx = il + dxx;
                if (xxx < 1)
                    xxx = 1; % The bacteria have stopped moving and
settled
                end
                if (xxx > nl)
                    xxx = nl;
                end
                xxx = round(xxx);
% GROWTH, DEATH, MIGRATION, AND CARRYING CAPACITY
rep = rand();
mig = rand();
die = rand();
mut = rand();
Food_Conc = xbias;
Food_Factor = Food_Conc(il);
Cipro_Conc = Cip_xbias;
Cipro = Cipro_Conc(il); % Cipro concentration is adjusted here
K = Max_xbias/4;
% Max growth is set to 1 for this equation
Monod = 1.25*(Food_Factor)/(K + Food_Factor);
Max_Monod = 1.25*(Max_xbias/(K + Max_xbias)); % Maximum Growth Rate
% Fast-Growth Targeting Anitibiotic MIC
MIC3 = Fit_m3*(0.03 - 0.027*(Monod/Max_Monod));
Pharma_Function = 1 - ((Cipro)/MIC3)^2;
Pop_Factor = (1 - (x(il,itim) + m1(il,itim) + m2(il,itim) +
m3(il,itim))/CC);
if (x(il,itim) + m1(il,itim) + m2(il,itim) + m3(il,itim)) > CC
    Pop_Factor = 0;
end
Replication_Prob = Monod*Pharma_Function*Pop_Factor;
```

```
                if (rep < Replication_Prob)
                    if mut < 0.0002*Mut_Rate %Go back one genotype
                        m2(il,itim) = m2(il,itim) + 1;
                    else
                        m3(il,itim) = m3(il,itim) + 1;
                    end
                % Death Rate
                elseif (die < Death_Prob) && (m3(il,itim) > 0)
                    m3(il,itim) = m3(il,itim) - 1;
                else
                    m3(il,itim) = m3(il,itim);
                    % Each position can grow up to CC
                    if (mig < Migration_Prob)
                        m3(il,itim) = m3(il,itim) - 1;
                        m3(xxx,itim) = m3(xxx,itim) + 1;
                    end
                end
             end % end particle position loop
        end % end position loop

        % now all the particles have moved by +/- dx=1
        % time has moved by + dt=(dx*dx)/(2*xd)
            % if dx = 1, xd = 5, then dt = 0.1
        xttim = xttim + dt; % new time
        if xttim < next_num
            itim = next_num - 1;
        elseif xttim >= next_num
            itim = next_num;
            next_num = next_num + 1;
        end
        ndum = nxcount; % count number of particles

        for ill = 1:nl; % select position loop

      % CheckCC combines the bacterium for each position and
      % re-normalizes them so that they're within the carrying capacity
           CheckCC = x(ill,itim) + m1(ill,itim) + m2(ill,itim) +
m3(ill,itim);
           if CheckCC > CC
               x(ill,itim) = (x(ill,itim)/CheckCC)*CC;
               m1(ill,itim) = (m1(ill,itim)/CheckCC)*CC;
               m2(ill,itim) = (m2(ill,itim)/CheckCC)*CC;
               m3(ill,itim) = (m3(ill,itim)/CheckCC)*CC;
               x(ill,itim) = round(x(ill,itim));
               m1(ill,itim) = round(m1(ill,itim));
               m2(ill,itim) = round(m2(ill,itim));
               m3(ill,itim) = round(m3(ill,itim));
           end

           % before this, w and mut1 were originally set to zero
           % records number of particle at postion w and others
           w(ill,itim) = x(ill,itim); %w(ill,itim) + x(ill,itim);
           mut1(ill,itim) = m1(ill,itim);
           mut2(ill,itim) = m2(ill,itim);
           mut3(ill,itim) = m3(ill,itim);
          % nxloop records the total number of particles at the
```

```
          % original position and the old time, nxloop = xx(il,ttimold)
            % now xx shifts to the new value of x for all positions ill
               %  to be used for the next while loop iteration
           xx(ill,itim) = x(ill,itim);
           mm1(ill,itim) = m1(ill,itim);
           mm2(ill,itim) = m2(ill,itim);
           mm3(ill,itim) = m3(ill,itim);
       % For this "for" loop only, re-record the values of the old time
           % itim into the new time ixxx
           % if dt = 0.1, then this cycle continues 10 more times
           % before it reaches itim  + 1, continuously updating the
           % number of particles into the new time
           % the values at a new position get moved from time itim to
           % the new time ixxx when it's at the boundary of itim + 0.9
           ixxx = itim + 1;
           if xttim >= itim + 0.9
                x(ill,ixxx) = x(ill,itim);
                m1(ill,ixxx) = m1(ill,itim);
                m2(ill,ixxx) = m2(ill,itim);
                m3(ill,ixxx) = m3(ill,itim);
           end
       end % end system position loop

% Record a video of the model
    if mod(itim,5) == 0

       xpos = zeros(nl,1);
       xpos(1) = 1;
       for il = 2:nl
           xpos(il) = xpos(il-1) + 1;
       end

       xdata1=xpos(1:nl);
       ydata1=w(1:nl,(itim));
       ydata1(ydata1==0)=nan;
       xdata2=xpos(1:nl);
       mutdata2=mut1(1:nl,(itim));
       mutdata2(mutdata2==0)=nan;
       xdata3=xpos(1:nl);
       mutdata3=mut2(1:nl,(itim));
       mutdata3(mutdata3==0)=nan;
       xdata4=xpos(1:nl);
       mutdata4=mut3(1:nl,(itim));
       mutdata4(mutdata4==0)=nan;

plot(xdata1,ydata1,xdata2,mutdata2,xdata3,mutdata3,xdata4,mutdata4,'Lin
eWidth',2);
       xlim([1 nl])
       ylim([0 (CC+100)])
       legend({'Wild Type Fitness = 1','Mutant 1 Fitness = 20','Mutant
2 Fitness = 100','Mutant 3 Fitness = 200'},'Location','northwest');
       xlabel ('Position (unitless)', 'fontsize', 16);
       ylabel ('Number of Cells', 'fontsize',16);
       title(['Time is ',num2str(itim)])

       frame = getframe(gcf);
```

```
        writeVideo(v,frame)
        drawnow;
    end

    end %  end of time while loop
   % After the time while loop, the next iteration goes back and adds
another ntot number of particles to the system starting at time 1,
while the previous particles in iteration 1 continue with their
original position
   % REMEMBER: The total number of particles added to the system on
   iteration 2 is added at ttim = 1.  This is when the particles in the
   center only start to move, while particles in iteration 1 are no
   longer moving.
   % Both iteration 1 and any further iterations have roughly the same
   displacement of particles from the initial position and onward,
   (ttim + dt).  The average number of particles gets normalized back
   to 500 below.
end % end iteration loop

% Close the video file recorded
close(v);

% ttim is the total time
% Normalize y for total amount of iterations
for itim = 1:ttim
    for il = 1:nl
       % With xcons = 1/(2*xd), the time loops through 10 times, adding
         500 particles to each loop
        w(il,itim) = w(il,itim)/(itertot);
        mut1(ill,itim) = mut1(ill,itim)/(itertot);
        mut2(ill,itim) = mut2(ill,itim)/(itertot);
        mut3(ill,itim) = mut3(ill,itim)/(itertot);
    end
end
% y(1:nl,1)
% Total_Particles = sum(y(1:nl,1))

xpos = zeros(nl,1);
xpos(1) = 1;
for il = 2:nl
    xpos(il) = xpos(il-1) + 1;
end

% Image of the model after it has finished
xdata1=xpos(1:nl);
ydata1=w(1:nl,(ttim - 1));
ydata1(ydata1==0)=nan;
xdata2=xpos(1:nl);
mutdata2=mut1(1:nl,(ttim - 1));
mutdata2(mutdata2==0)=nan;
xdata3=xpos(1:nl);
mutdata3=mut2(1:nl,(ttim - 1));
mutdata3(mutdata3==0)=nan;
xdata4=xpos(1:nl);
mutdata4=mut3(1:nl,(ttim - 1));
mutdata4(mutdata4==0)=nan;
```

```
plot(xdata1,ydata1,xdata2,mutdata2,xdata3,mutdata3,xdata4,mutdata4,'Lin
eWidth',2);
xlim([1 nl])
ylim([0 (CC+100)])
legend({'Wild Type Fitness = 1','Mutant 1 Fitness = 20','Mutant 2
Fitness = 100','Mutant 3 Fitness = 200'},'Location','northwest');
xlabel ('Position (unitless)', 'fontsize', 16);
ylabel ('Number of Cells', 'fontsize',16);
```