Mathematical Modelling of Drug Resistance and Tumor Growth

Adam Aldridge

Supervised by

Grant Lythe

2019

Abstract

When chemotherapy treatment is administered, sensitive cancer cells are killed but more resistant species are left to grow and multiply. Evolution by forced selection is occurring in diseases all over the world. In the following dissertation, two discrete-time branching process models are formulated and analysed to develop understanding of drug resistance dynamics. Then, a continuous time, Moran style lattice model is proposed for the simulation of tumor growth, in which, the Gillespie algorithm is enlisted to manage realisations of 175 individually defined event types. Finally, the lattice model is used in an implementation of the genetic algorithm to display adaptivity.

Contents

Chapter 1

Biological Introduction

1.1 Cells

[1][2] The building blocks of life; from giant blue whales to tiny tadpoles, every living organism on earth is made of cells.

Casting your mind back to high school, you can probably remember that a cell is basically a small bag filled with various bits that do interesting things. For the purposes of this report, we don't need an extensive knowledge of these inner workings. We do however need to know about DNA. The DeoxyriboNucleic Acid double helix is, simply put, a long string of information. Encoded in the rungs of its ladder are the blue prints for life. Nucleotides, Adenine (A), Thymine (T), Guanine (G), and Cytosine (C) make up each rung in one of the 4 possible base pairs: AT, TA, GC, or CG.

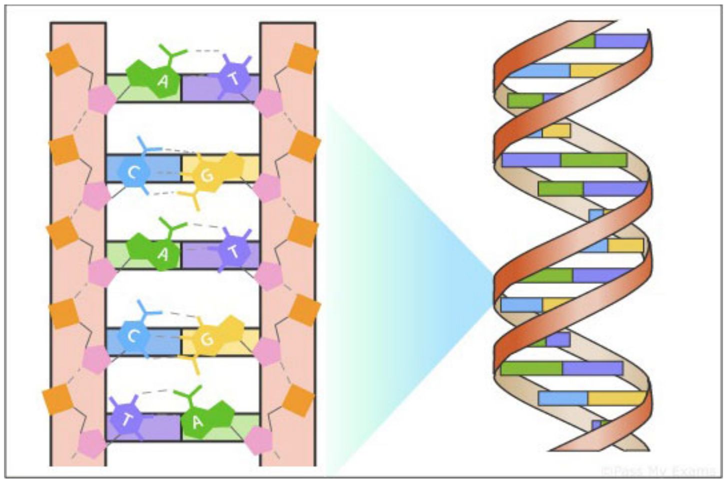


Figure 1.1: The Structure of DNA [3].

Inside the nucleus of every cell making up an organism is the entire body of DNA (the genome). Quite impressive really considering that for example, the human genome contains over 3 billion base pairs and is about 1.8m long, whereas human cells are of order 10^{-5}m in diameter. To succeed at this unthinkable packing task, cells have a special technique. The DNA is first, tightly coiled, ball-like, around proteins called histones making a fibre, chromatin. Then, the chromatin is twisted and coiled repeatedly until it forms worm-like structures that we call chromosomes. Humans pack their genome into 23 pairs of these, 1 of which are the sex chromosomes, labelled XX in women and XY in men.

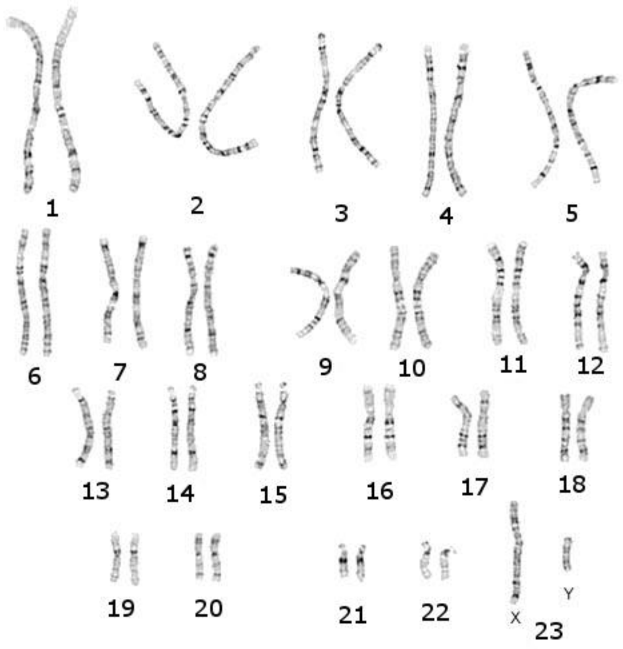


Figure 1.2: Human chromosomes [4].

This sounds awfully difficult. Why on earth would we want to pack a 1.8m string into each of the 37.2 trillion cells in our body? well, as it turns out these strings are quite handy. Using the genome, your cells are able to build the proteins that make up your body. It's like a big instruction manual. Different cells read different chapters; a cell on the surface of your nose reads the chapter titled skin, and it pays particular attention to the section on nose skin. Hang on a moment. Firstly (1), how do cells *read* the genome? And secondly (2), how do the cells know which parts to read and which to ignore? Good questions, here are the answers:

1. To extract the message stored in DNA, enzymes called RNA-Preliminaries pass over its sections. Whilst leaving the DNA as they found it, these enzymes transcribe the message onto a section of messenger-RNA. mRNA is like DNA cut down the rungs; it is a single string of nucleotides. This mRNA then passes through the wall of the nucleus into the main inner cell environment. Here it can meet a ribosome which will look at the whole message, 3 nucleotides at a time, gradually manufacturing proteins from surrounding amino acids.

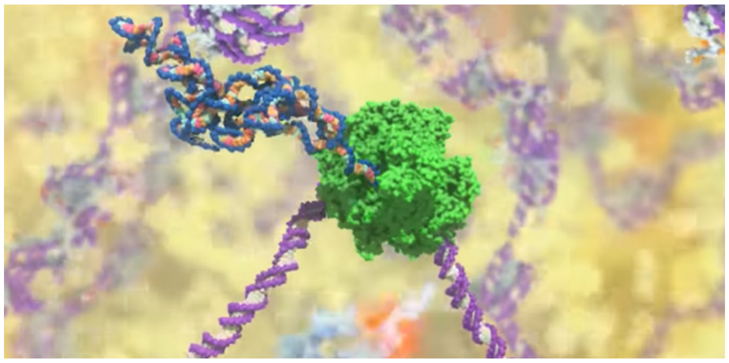


Figure 1.3: RNA-Preliminarie transcribing DNA into mRNA [2].

2. Some sections of DNA are more tightly coiled around histones than others. Sequences that are less tightly wrapped up are more easily accessed by RNA-Preliminaries, causing them to be read with a

higher likelihood. In this mechanism, areas of DNA are rendered active, or inactive. Cells from different parts of the body will not have the same activation patterns; the nose cell will read the nose DNA.

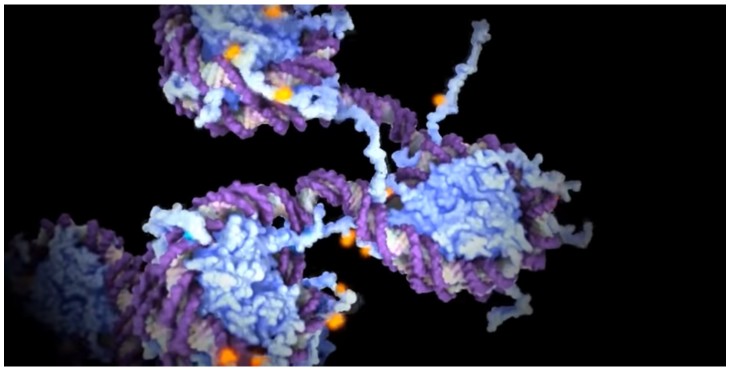


Figure 1.4: DNA wrapped around histone proteins [2].

Aside: Interestingly, these activation patterns in the genome can be, and are being altered all the time by epigenetic marks. Small chemicals which affect histones, causing the DNA balls to become more, or less tightly wrapped. Alterations via epigenetics can occur on relatively short time scales and are influenced significantly by lifestyle and individual behaviour. It has even been shown that just by meditating, you can positively change the way your genome is read. [5]

1.1.1 Cell division

For organisms to grow and maintain themselves, new cells are constantly required. To supply the steady demand, each living cell goes through a process of division, becoming 2 new daughter cells. A cell's life cycle, from birth to division can take hours, days, months, or even years. It depends on the cell type. Regardless of how long it takes, a cell's lifetime can be split into stages. In one such stage, DNA synthesis, the entire genome is copied and arranged into chromosomes, ready to be split. Tasked with copying the entire genome are helicase molecular machines. They rapidly split the double helix down its rungs, building each part back up independently. Amazing as it is, this process can be subject to errors. There is a proofreading mechanism that occurs after a copy, but even this doesn't catch everything. Occasionally, new DNA is left with small errors called mutations. Often, a mutation will be neutral in its effect; not causing a positive or negative change in the behaviour of the new cell. Occasionally, mutations are positive leading to the development of an advantageous trait in the organism. This is one of the fundamental drivers of evolution. When a mutation is not neutral however, it is more likely to be negative, effecting the inner workings of a cell in a problematic way. Cancerous cells are created by in this manner.

1.1.2 Plasmids

Chromosomes may hold the main body of DNA, but they are not the full story. Small extra-chromosomal sections of DNA can also be present in a cell. These are called plasmids. Like the DNA in chromosomes, plasmid DNA can be read, actioned upon, and replicated. Unlike the main body of DNA however, these smaller sections are not included in the complicated mechanism that equally partitions chromosomes. As a result, upon a cell division event, plasmids within the cell are not likely to be shared equally between the progeny. This property, and the dynamics that it cause will be the focus of chapters 3 and 4. Specifically, we will look into double minute (DM) chromosomes. DMs are a particular type of plasmid, found in cancerous cells which hold DNA sequences responsible for the production dihydrofolate reductase (DHFR), an enzyme which has been shown to increase a cancer cell's resistance to the drug methotrexate (MTX).

We now know what plasmids are, but where do they come from? Originally, a plasmid comes into being when it's sequence is separated from the chromosomal DNA. Once the plasmid DNA has departed the a chromosome, it or its replicants are also able to re-enter the source chromosome, or another. This, when plasmids replicate alot, can cause the amplification of a particular DNA sequence in the genome.

1.2 Horizontal Gene transfer

Vertical gene transfer refers to the passage of genetic material from parent to daughter cells, pretty trivial. Less trivial is a mechanism called horizontal gene transfer. In this, one cell is able to pass genetic material, specifically plasmids, to a neighbouring cell. In the book analogy of the genome, this is like handing an article to a friend. But how does one cell 'hand' DNA to another cell? Several methods have been discovered, here is one. Plasmids can contain genetic instructions to build something called a sex pilus. This is a tube that can stick out of the host bacteria, into a neighbour. Through the pilus passes the plasmid, delivering genetic information.

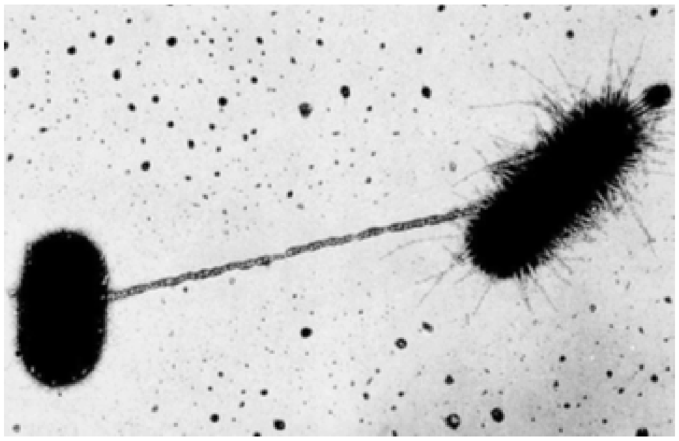


Figure 1.5: A sex pilus, sent from the right cell to the left [6].

This procedure is particularly important because it can facilitate the spread of, for example, drug resistance through a population of cells. Resistance emerging via random mutation alone can be slow, so horizontal gene transfer may accelerate the spread.

Chapter 2

Discrete Time Mathematical Introduction

2.1 Galton Watson Process

[13][7] Branching processes come in different varieties; they can unfold in discrete or continuous time, and they can describe one, or several types of particle. The Galton Watson process, introduced by Francis Galton in 1889 is the most basic:

A single initial ancestor lives for exactly 1 unit of time. At the moment of death, it produces a random number of progeny dependent on a defined probability distribution. Each of these newborns is destined to the exact fate of their parent; one unit of time alive followed by simultaneous death and reproduction. As well as leading the same lifestyle as their single parent, the offspring don't have any interaction with each other. As a result, they can each be considered the single ancestor for a new line of decent. These new sub-processes will be independent identically distributed (iid) copies of each other, meaning that although their population sizes may diverge with time, their respective probability of reaching any given future state will be equal.

For many applications in modelling the simplicity of the Galton Watson process is not realistic. Notably, when cells in a living organism divide, they don't do so instantly and all at once. However, this lack of complexity allows for a comprehensive mathematical analysis. Such analysis can be a stepping stone in understanding more complex, and realistic phenomena.

Since each generation is perfectly in sync and lives for an equal time, the process can be given a discrete time index where $n = 0, 1, 2, \dots$ represents the generation number. Suppose at the end of life, particles produce a random number of offspring $\mathbf{X}$, with

$$\mathcal{P}[\mathbf{X} = k] = p_k, \qquad k = 0, 1, 2, \dots$$

Down the line, whether the population survives or becomes extinct will depend on this distribution. $\mathbf{Z}_n$ is used to define the total number of particles in generation n, so $\mathbf{Z}_0 = 1$. It follows that the number of particles in any given generation $\mathbf{Z}_{n+1}$ is uniquely dependent on that same number for the previous generation $\mathbf{Z}_n$; the state of the system at time $t = n + 1$ is only affected by the state of the system at time $t = n$. This memorylessness is known as the Markov property. Specifically, $\mathbf{Z}_{n+1}$ is obtained as the sum of all offspring in generation n:

$$\mathbf{Z}_{n+1} = \sum_{i=1}^{\mathbf{Z}_n} \mathbf{X}_i. \tag{2.1}$$

Where $\mathbf{X}_i$ represents the number of offspring birthed by the ith member of the nth generation. With time the population will change in size, sometimes it will grow very large, and sometimes it will go extinct. As hinted at above, the likelihood of a given outcome coming to fruition is dependent on the probability distribution for $\mathbf{X}$. By intuition, it is clear that if all the particles in a population have few progeny, that population will fare worse that one where offspring are plentiful. To begin rigourising this idea, let

$$\mathbb{E}(\mathbf{X}) = p_1 + 2p_2 + 3p_3 + \dots = m$$

be the expected value of $\mathbf{X}$. This is the mean number of progeny from a given birth of a single parent. It follows that:

$$\mathbb{E}(\mathbf{Z}_1) = \mathbb{E}(\mathbf{X}) = m.$$

Also, with the linear property of the expectation, $\mathbb{E}(A+B) = \mathbb{E}(A) + \mathbb{E}(B)$, equation (2.1) can be used to show:

$$\begin{aligned}\mathbb{E}(\mathbf{Z}_{n+1}) &= \mathbb{E}(\mathbf{X}_1) + \mathbb{E}(\mathbf{X}_2) + \cdots + \mathbb{E}(\mathbf{X}_{\mathbf{Z}_n}) \\ &= m + m + \cdots + m \\ &= \mathbb{E}(\mathbf{Z}_n)m.\end{aligned}$$

Which in turn, allows the deduction of:

$$\mathbb{E}(\mathbf{Z}_n) = \mathbb{E}(\mathbf{Z}_{n-1})m = \mathbb{E}(\mathbf{Z}_{n-2})m^2 = \cdots = \mathbb{E}(\mathbf{Z}_1)m^{n-1} = m^n.$$

In English, this is saying that the expected number of particles at generation n is realised when each birth prior to that point results in exactly the mean number of offspring. Meaning that, each of the n generations grow the population by a factor of m. A demonstration of this principal can be seen in Figure 2.1, which plots the size of 50 independent populations each with $m = 1.2$ over 20 generations.

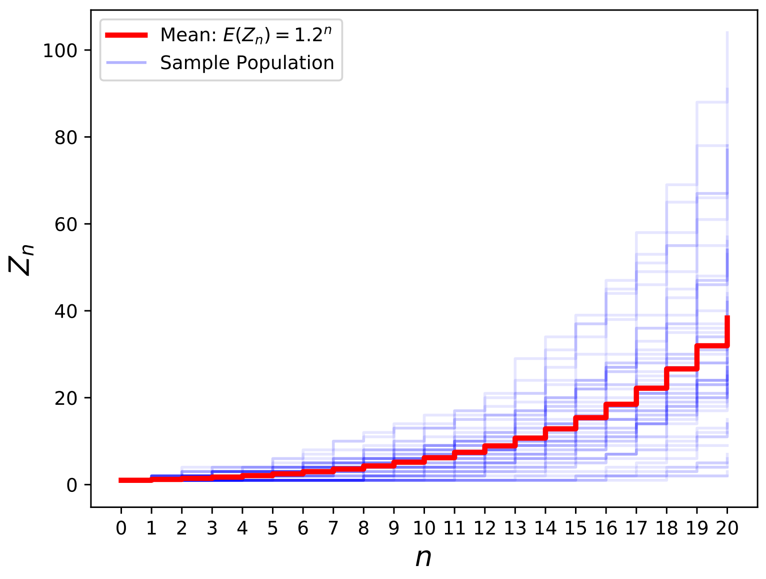


Figure 2.1: $\mathbf{Z}_n$ for 50 populations each reproducing with probabilities: $p_0 = 0$, $p_1 = 0.8$ and $p_2 = 0.2$.

It is observed that at each generation, the population sizes have a distribution that is approximately centred around the expected value.

2.2 The Probability Generating Function

When handling distributions of summed random variables, such as $\mathbf{Z}_n$ for the Galton Watson process, a useful tool is the probability generating function (pgf). Take the random variable for number of progeny $\mathbf{X} \in \mathbb{Z}^+$, distributed by $\mathcal{P}[\mathbf{X} = k] = p_k$. The pgf for $\mathbf{X}$ is defined:

$$\begin{aligned}f(s) = \mathbb{E}(s^{\mathbf{X}}) &= \sum_{i=0}^{\infty} p_i s^i \\ &= p_0 + p_1 s + p_2 s^2 + \cdots\end{aligned}$$

(Sometimes we will write $f_{\mathbf{X}}(s)$ to make the relevant random variable clear). Notice,

$$f(1) = p_0 + p_1 + p_2 + \cdots = 1$$

and

$$f(0) = p_0.$$

The probability generating function $f(s)$ is increasing and passes p_0 at $s = 0$, and 1 at $s = 1$. This holds for any prescribed probability distribution. Here is an example of what it looks like:

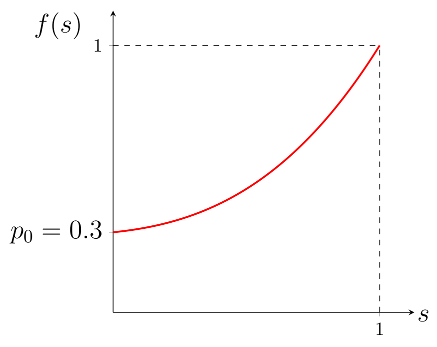


Figure 2.2: A plot of pgf $f(s) = 0.3 + 0.1s + 0.3s^2 + 0.3s^3$.

An extension to the idea of a pgf is possible when multiple random variables are present. Say there are n random variables, $\mathbf{X}_1, \ldots, \mathbf{X}_n$, distributed by,

$$p_{i_1,\ldots,i_n} = \mathcal{P}[\mathbf{X}_1 = i_1, \ldots, \mathbf{X}_n = i_n].$$

The multi-type pgf for this variable set is defined as,

$$F(s_1, \ldots, s_n) = \sum_{i_1,\ldots,i_n} p_{i_1,\ldots,i_n} s_1^{i_1} \cdots s_n^{i_n}.$$

2.3 Extinction and Criticality of the Galton Watson Process

Any Galton Watson process with $p_0 \neq 0$ has some chance of going extinct, but this fate is more likely for some than others. It turns out that a population's probability of ultimate extinction is highly dependent on the mean growth rate m. Three cases for population growth present themselves:

- $m > 1$, supercritical,
- $m = 1$, critical, and
- $m < 1$, subcritical.

As intuition would suggest, in the supercritical case where on average, individuals more than replace themselves, populations are unlikely to go extinct. A demonstration of supercritical growth can be seen above in Figure 2.1. Furthermore, if on average individuals don't fully replace themselves, like when $m < 1$ in the supercritical case, it is clear that extinction is highly likely. It is in fact certain, but we will prove this in a little while. Intuition serves well in understanding the eventual outcomes of both supercritical and subcritical populations, but what about the intermediate case where m is exactly equal to 1. Understanding the evolution of the critical case is a task that side steps the application of common sense. Somewhat surprisingly, populations of the critical type will always go extinct, eventually at least. They may linger for a vast number of generations, but will die out with a condemning probability of one.

Below are two plots showing the individual count $\mathbf{Z}_n$, resulting from simulated growth over several generations. Each plot shows the life and death of 50 independent populations. These plots demonstrate that under both critical (left), and subcritical (right) growth, extinction will occur eventually, despite taking almost 100 generations for one of the populations in the critical simulation.

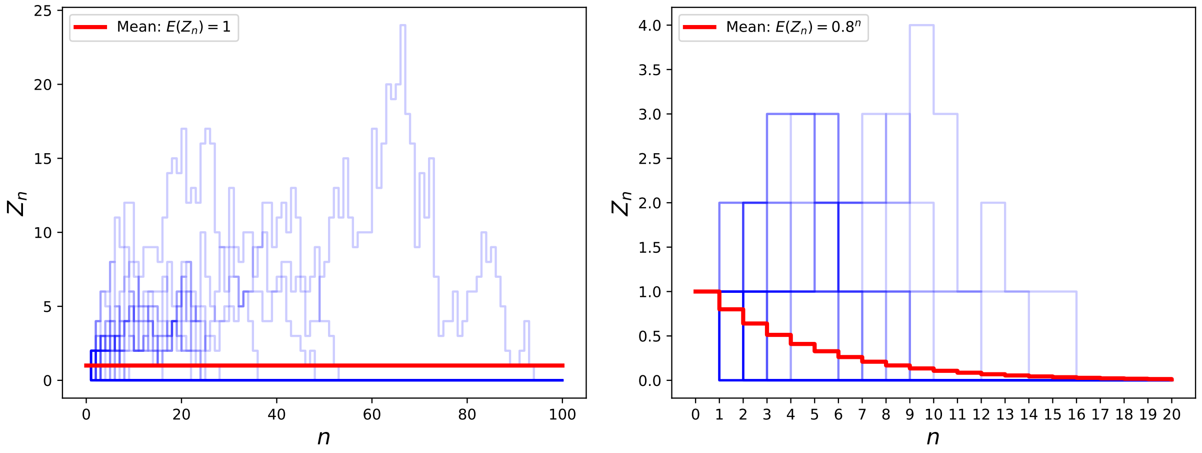


Figure 2.3: $\mathbf{Z}_n$ plotted over a number of generations. Each of the 50 populations in the left plot are reproducing with distribution: $p_0 = 0.4$, $p_1 = 0.2$ and $p_2 = 0.4$. This distribution for populations in the right plot is: $p_0 = 0.3$, $p_1 = 0.6$ and $p_2 = 0.1$.

Beyond knowing simply whether a population will or won't go extinct, it is possible to determine the exact probability that it will die out by a given generation. Furthermore, for the supercritical case we can calculate the chance of ultimate extinction. This analysis will also shed some light on the fact that extinction is certain in the critical case of growth.

To begin with, let's define:

$$u_n = \mathcal{P}[\mathbf{Z}_n = 0|\mathbf{Z}_n = 1],$$

the probability that by generation n a population is extinct.

- Since, after a population dies, it doesn't come back to life, we can say:

$$\mathcal{P}[\mathbf{Z}_i = 0|\mathbf{Z}_j = 0] = 1, \qquad \text{if } j < i.$$

- Also, provided $p_0 \neq 0$, at any given generation n, the probability of extinction by the next generation $n+1$ is non zero and equal to:

$$u_{n+1} = p_0^{\mathbf{Z}_n} > 0.$$

  All $\mathbf{Z}_n$ individuals could die without children.

With knowledge of the above two points, it is deducible that u_n is an increasing function of n.

Consider a population that starts at generation 1 with k individuals. The likelihood of that population's extinction by generation n will be equal to the probability of k iid populations with a single ancestor all going extinct:

$$\mathcal{P}[\mathbf{Z}_n = 0|\mathbf{Z}_0 = k] = u_n^k.$$

Suppose we know a population contains k individuals when $n = 1$, we can write,

$$\mathcal{P}[\mathbf{Z}_n = 0|\mathbf{Z}_1 = k] = u_{n-1}^k.$$

If we have one initial ancestor,

$$\mathcal{P}[\mathbf{Z}_1 = k|\mathbf{Z}_0 = 1] = \mathcal{P}[\mathbf{X} = k] = p_k.$$

Putting things together, we can obtain an expression for u_n in terms of u_{n-1} by summing over all the possible values of k.

$$\begin{aligned}
u_n &= p_1\mathcal{P}[\mathbf{Z}_n = 0|\mathbf{Z}_1 = 1] + p_2\mathcal{P}[\mathbf{Z}_n = 0|\mathbf{Z}_2 = 2] + \cdots \\
&= p_1 u_{n-1} + p_2 u_{n-1}^2 + p_3 u_{n-1}^3 + \cdots \\
&= f(u_{n-1})
\end{aligned}$$

We collect u_n when substituting u_{n-1} into the probability generating function for $\mathbf{X}$. Thus, using the relationship,

$$u_n = f(f(u_{n-2})) = \underbrace{f(\cdots f}_{n-1}(u_1)\cdots)$$

where $u_1 = p_0$, we can find the probability of extinction at any given generation. Furthermore, if we consider the limit where $n \to \infty$, we have $n = n - 1$ and by extension $u_n = u_{n-1} = u_\infty$. The probability of ultimate extinction satisfies the equation,

$$\boxed{u_\infty = f(u_\infty).} \tag{2.2}$$

As $f(s)$ is a polynomial which can have an order higher than 1, there may be many solutions to this equations. In the case where there are multiple values of u_∞ that satisfy equation 2.2, which one is the correct probability of ultimate extinction? We know that since it is a probability, $0 \geq u_\infty \leq 1$. That fact can be used to discount some candidate solutions. Also, because of the uniformly convex nature of $f(s)$ between 0 and 1, there can be at most two roots in this region. 1 is in fact always a solution as $f(1) = 1$, so when extinction is not certain the only option left is the remaining root in the range [0,1). Graphically speaking, u_∞ is located at the smallest positive value of s for which the pgf is intersected by the line $y(s) = s$. A representation of this is seen in Figure 2.4.

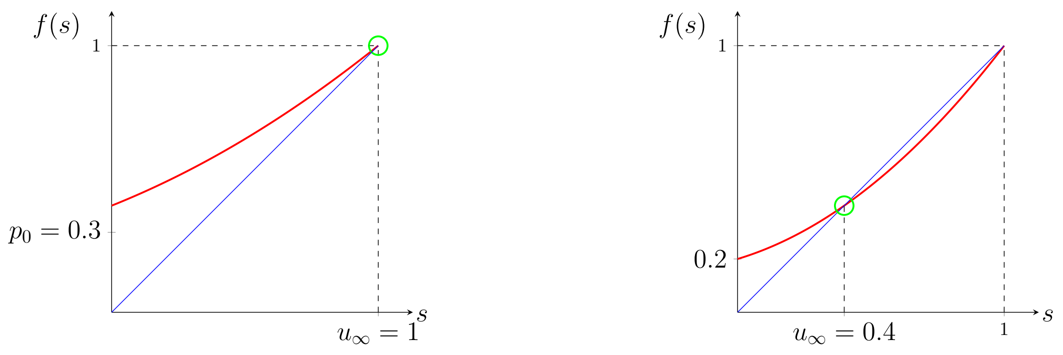


Figure 2.4: The probability generating function plotted simultaneously with the line $y(s) = s$ for two probability distributions: $[p_0, p_1, p_2]$ = [0.4, 0.4, 0.2] (left), and [0.2, 0.3, 0.5] (right). Intersection points locate u_∞

So in the population relevant to the left plot, ultimate extinction is certain; and for that of the right it will occur with probability 0.4.

Pay close attention to the slope of the pgfs at $s = 1$, $f'(1)$. In the case where ultimate extinction is going to happen with probability 1 (left), the slope of $f(s)$ at $s = 1$ can be seen to be <u>more shallow</u> that that of $y(s) = s$. This property, with the fact that the pgf is convex in [0,1], causes there to be no other intersection in the interval. For there to be an root of equation (2.2) in range [0,1), the slope of the pgf at 1 must be <u>steeper</u> than that of $y(s) = s$. In other words $f'(1) > 1 \iff u_\infty < 1$. Recall that $f'(1) = m$:

$$m > 1 \iff u_\infty < 1$$

We have made a connection, the above reasoning serves as a geometric proof for the rules of extinction stated at the beginning of this subsection. Furthermore it completes the understanding of why eventual extinction is guaranteed in the subcritical, <u>and the critical</u> cases of population growth.

2.4 Special Properties of the Probability Generating Function

2.4.1 Mean and Variance

Collecting the mean is simple, we can differentiate $f(s)$ and set $s = 1$:

$$f'(s) = \sum_{i=1}^{\infty} i p_i s^{i-1}$$

$$\implies f'(1) = p_1 + 2p_2 + 3p_3 + \cdots = \underline{\mathbb{E}(\mathbf{X})}.$$

In addition to the expectation of $\mathbf{X}$, the variance can also be obtained. To begin with,

$$f''(s) = \sum_{i=2}^{\infty} i(i-1) p_i s^{i-2}$$

and so,

$$\begin{aligned} f'(1) + f''(1) &= \sum_{i=1}^{\infty} i p_i + \sum_{i=2}^{\infty} i(i-1) p_i \\ &= p_1 + \sum_{i=2}^{\infty} [i + i(i-1)] p_i \\ &= p_1 + \sum_{i=2}^{\infty} i^2 p_i = \sum_{i=1}^{\infty} i^2 p_i \\ &= \mathbb{E}(\mathbf{X}^2). \end{aligned}$$

With that, and knowing the variance of $\mathbf{X}$ to be $\mathbb{V}(\mathbf{X}) = \mathbb{E}(\mathbf{X}^2) - \mathbb{E}(\mathbf{X})^2$, we can define,

$$\mathbb{V}(\mathbf{X}) = f'(1) + f''(1) - f'(1)^2$$

2.4.2 PGF for summed random variables

When we have a value that is the sum of two or more independent random variables, say, $\mathbf{W} = \mathbf{A} + \mathbf{B}$ then the pgf of that value is the product of the the pgfs for its summed components:

$$\begin{aligned} f_{\mathbf{W}}(s) &= \mathbb{E}(s^{\mathbf{W}}) = \mathbb{E}(s^{\mathbf{A}+\mathbf{B}}) \\ &= \mathbb{E}(s^{\mathbf{A}})\mathbb{E}(s^{\mathbf{B}}) \\ &= f_{\mathbf{A}}(s) f_{\mathbf{B}}(s). \end{aligned}$$

2.4.3 The PGF for Random Sums of Independent Random Variables

We know from equation (2.1) that $\mathbf{Z}_{n+1}$ is the sum of all the $\mathbf{Z}_n$ litters of offspring in generation n. Each birth will produce a random number of offspring, but since $\mathbf{Z}_n$ is a random number in of itself, there will also be a random number of births. Following this, $\mathbf{Z}_{n+1}$ is the result of a sum of a <u>random number</u> of <u>random variables</u>. Analysing the pgf for $\mathbf{Z}_{n+1}$ we find:

$$\begin{aligned} f_{\mathbf{Z}_{n+1}}(s) &= \mathbb{E}(s^{\mathbf{Z}_{n+1}}) \\ &= \sum_{i=0}^{\infty} \mathcal{P}[\mathbf{Z}_n = i] \mathbb{E}(s^{\mathbf{Z}_{n+1}} | \mathbf{Z}_n = i) \end{aligned} \tag{2.3}$$

We know that $\mathbb{E}(\mathbf{Z}_{n+1} | \mathbf{Z}_n = i)$ is simply the population count that results when all i individuals produce exactly the expected number of progeny, $(\mathbb{E}(\mathbf{X}))^i$. By the same line of reasoning we can say that:

$$\mathbb{E}(s^{\mathbf{Z}_{n+1}} | \mathbf{Z}_n = i) = (\mathbb{E}(s^{\mathbf{X}}))^i = (f_{\mathbf{X}}(s))^i.$$

Which in turn can be combined with equation (2.3) to say:

$$\begin{aligned} f_{\mathbf{Z}_{n+1}}(s) &= \sum_{i=0}^{\infty} \mathcal{P}[\mathbf{Z}_n = i](f_{\mathbf{X}}(s))^i \\ &= f_{\mathbf{Z}_n}(f_{\mathbf{X}}(s)) \end{aligned}$$

Let's denote $f_{\mathbf{Z}_n}(s) = f_n(s)$ and $f_{\mathbf{Z}_1}(s) = f_{\mathbf{X}}(s) = f(s)$:

$$\boxed{f_{n+1}(s) = f_n(f(s))} \tag{2.4}$$

Furthermore:

$$\begin{aligned} f_n(s) &= f_{n-1}(f(s)) = f_{n-2}(f_{n-1}(f(s))) = \cdots \\ &= \underbrace{f(f(\cdots f}_{n \text{ times}}(s)\cdots)) \end{aligned}$$

Extending this idea, equation (2.4) can be genralised to:

$$f_{a+b}(s) = f_a(f_b(s)) = f_b(f_a(s))$$

Application: When analysing the GW process, a challenge arises: What is the probability of a population containing exactly k individuals in the nth generation; $p_k^{(n)} = \mathcal{P}[\mathbf{Z}_n = k]$? Aside from the trivial case where $p_0 + p_1 = 1$, this can be tricky to determine. Say for example, $k = 3$ and $n = 5$. In 5 generations there are many possible birth death sequences that could result in 3 siblings. Finding $p_3^{(5)}$ would involve mapping out every possible combination of events and summing the probabilities.

Using equation (2.4) it is possible to short cut this procedure. $p_k^{(n)}$ is simply the coefficient of s^k in $f_n(s)$. $p_k^{(n)}$ can be collected by differentiating $f_n(s)$, k times and setting $s = 0$;

$$p_k^{(n)} = \frac{d^k f_n}{ds^k}(0). \tag{2.5}$$

2.4.4 Variance of the nth Generation in the GW Process

Now we have the means to evaluate $f_{\mathbf{Z}_n}(s)$ for $n > 1$ it is possible to calculate $\mathbb{V}(\mathbf{Z}_n)$, the variance of population size at generation n as a function of the mean m, and the variance $\mathbb{V}(\mathbf{Z}_1) = \sigma^2$ in the first generation. To clarify,

$$\mathbb{V}(\mathbf{Z}_1) = m - m^2 + f''(1) = \sigma^2$$

Deriving this result centers around evaluating each of the terms on the RHS of:

$$\mathbb{V}(\mathbf{Z}_n) = f_n'(1) + f_n''(1) - f_n'(1)^2.$$

It gets somewhat involved so we can break it down into several steps. To begin with, we can apply the chain rule and the fact that $f'(1) = m$, to say:

$$f_n'(1) = f_{n-1}(1)f'(1) = \cdots = m^n.$$

Now, let's evaluate $f_n''(s)$ with $s = 1$:

$$\begin{aligned} f_n''(s) &= [\underbrace{f(f(\cdots f}_{n \text{ times}}(s)\cdots))]'' \\ &= [f'(\underbrace{f(\cdots f}_{n-1}(s)\cdots))f'(\underbrace{f(\cdots f}_{n-2}f(s)\cdots))\cdots f'(s)]' \\ &= f''(\cdots \underset{n-1}{f}(s)\cdots)f'(\cdots \underset{n-2}{f}(s)\cdots)^2 \cdots f'(s)^2 + \\ &\quad f'(\cdots \underset{n-1}{f}(s)\cdots)f''(\cdots \underset{n-2}{f}(s)\cdots)f'(\cdots \underset{n-3}{f}(s)\cdots)^2 \cdots f'(s)^2 + \cdots \\ &\quad f'(\cdots \underset{n-1}{f}(s)\cdots)f'(\cdots \underset{n-2}{f}(s)\cdots)\cdots f'(f(s))f''(s). \end{aligned}$$

Since $f(1) = 1$, by extension $f(\underset{a}{\cdots} f(1) \cdots) = 1$ for any $a \in \mathbb{Z}^+$. With this, and the fact that $f'(1) = m$ the above evaluated at $s = 1$ becomes:

$$\begin{aligned} f_n''(1) &= f''(1)m^{2(n-1)} + f''(1)m^{2(n-1)-1} + \cdots + f''(1)m^{2(n-1)-(n-1)} \\ &= f''(1)m^{2(n-1)} \sum_{i=0}^{n-1} m^{-i}. \end{aligned}$$

Then, using the geometric series summation rule,

$$\sum_{i=0}^{n-1} (m^{-1})^i = \frac{1 - m^{-n}}{1 - m^{-1}},$$

we obtain:

$$\begin{aligned} f_n''(1) &= f''(1)m^{2(n-1)} \left(\frac{1 - m^{-n}}{1 - m^{-1}} \right) \\ &= f''(1)m^{2(n-1)} \left(\frac{m - m^{1-n}}{m - 1} \right) \\ &= f''(1)m^{n-1} \left(\frac{m^n - 1}{m - 1} \right). \end{aligned}$$

It is important to note that for the special case of $m = 1$, the geometric series sum results in n, meaning, in this case, $f_n''(1) = f''(1)n$.

Before we go for the final result, it is helpful to first find $G(m)$ such that:

$$m^n - m^{2n} = (m - m^2)G(m).$$

The reason for this will become clear. Rearranging, we get:

$$\begin{aligned} G(m) &= \frac{m^n - m^{2n}}{m - m^2} = \frac{m^{2n} - m^n}{m^2 - m} = \frac{m^n}{m} \left(\frac{m^n - 1}{m - 1} \right) \\ &= m^{n-1} \left(\frac{m^n - 1}{m - 1} \right) \end{aligned}$$

Notice, this factor has shown up above in $f_n''(1) = f''(1)G(m)$. We now have all the ammunition required to obtain the result for $\mathbb{V}(\mathbf{Z}_n)$:

$$\begin{aligned} \mathbb{V}(\mathbf{Z}_n) &= f_n'(1) + f_n''(1) - f_n'(1)^2 \\ &= m^n + f_n''(1) - m^{2n} \\ &= m^n - m^{2n} + f_n''(1) \\ &= (m - m^2)G(m) + f''(1)G(m) \\ &= (m - m^2 + f''(1))G(m) \end{aligned}$$

The left factor here is of course $\mathbb{V}(\mathbf{Z}_1) = \sigma^2$. Summarising, we have:

$$\boxed{\mathbb{V}(\mathbf{Z}_n) = \begin{cases} \sigma^2 m^{n-1} \left(\dfrac{m^n - 1}{m - 1} \right), & m \neq 1 \\ n\sigma^2, & m = 1 \end{cases}}$$

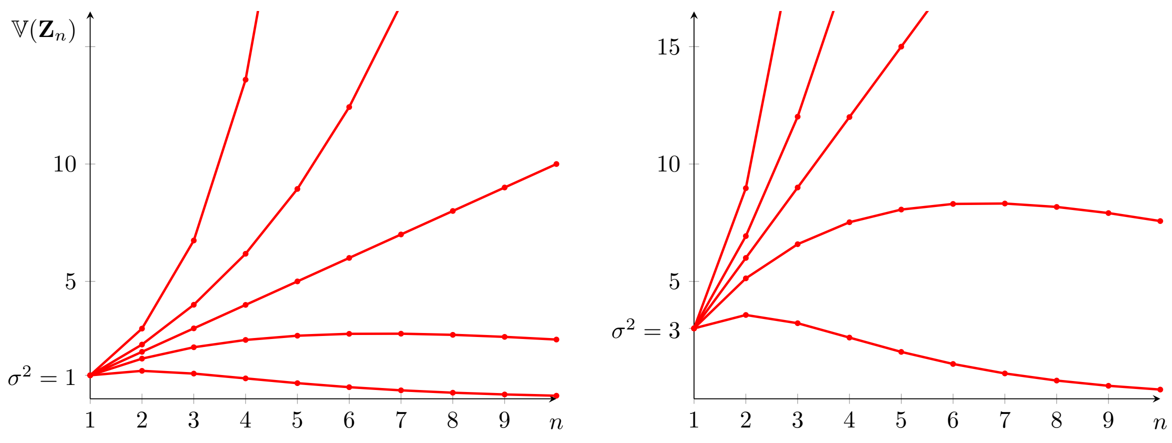


Figure 2.5: $\mathbb{V}(\mathbf{Z}_n)$ plotted against n with $\sigma^2 = 1$ (left), and 3 (right), each for multiple values of m. In order, from top to bottom, the lines in each plot have m values of 1.3, 1.1, 1, 0.9, and 0.7.

Basically,

- if $m > 1$ variance in population sizes will positively accelerate with passing generations,
- with $m = 1$ variance will grow linearly at a rate defined by the initial variance σ^2, and
- when $m < 1$, although $\mathbb{V}(\mathbf{Z}_n)$ may initially increase, ultimately it will tend to 0 as all the populations die out.

Chapter 3

Discrete Time Modelling of Drug Resistance I

When the anti-cancer drug methotrexate (MTX) is introduced to a population of cancerous cells, it is possible that not all of the target cells will be eliminated. Some cancer cells are more resistant than others. This resistance, as stated in the biological introduction, depends on the rate that cells produce the enzyme, dihydrofolate reductase (DHFR), which can vary within a population. A cells production of DHFR is induced by specific sequences of nucleotides. Often, these sequences are located on small extra-chromosomal DNA elements called double minute chromosomes. Separate from the main bulk of DNA held in regular chromosomes, these nucleotide strings act and divide independently. Normal chromosomes have centromeres which are used in a complicated mechanism to ensure that when a cell divides, its chromosomes are copied and distributed equally between the resulting progeny. Double minute chromosomes on the other hand don't contain centromeres so aren't included in this mechanism. As a result, they are generally not divided and shared equally between daughter cells, and so, MTX drug resistance is prone to stochastic fluctuations between generations in a particular cell lineage. Analysis of these fluctuations can be approached via the following mathematical model.

3.1 Galton Watson Process Model

Consider a cell containing a single double minute chromosome. As stated above, the division of double minutes is not fixed to those of their host cell. At a cell division event, a resident double minute will divide with probability (wp) a, and not divide wp $1 - a$. If the double minute does not divide, it will be passed to either daughter cell with an equal probability of $1/2$. If however it does divide into two, both will go to a single cell wp α, or, wp $1 - \alpha$ every new cell will receive one. Each of the two resulting daughter cells will contain one of the following:

- No replicas of the original double minute wp $(1 - a)/2 + a\alpha/2$
- One replica wp $(1 - a)/2 + a(1 - \alpha)$
- Two replicas wp $a\alpha/2$

Therefore, the number of double minutes in the nth generation of a particular cell lineage is a Galton Watson process with PGF:

$$f(s) = [(1 - a)/2 + a\alpha/2] + [(1 - a)/2 + a(1 - \alpha)]s + [a\alpha/2]s^2,$$

or

$$f(s) = d + (1 - b - d)s + bs^2, \tag{3.1}$$

where $b = a\alpha/2$ and $d = (1 - a)/2 + a\alpha/2$. When, in an experiment carried out by [8], a population of cells that has grown in the presence of MTX is subsequently introduced to an environment void of the drug, the number of double minutes and as a result the drug resistance has been shown to decrease. This implies that the Galton Watson process for this model is subcritical ($m < 1$). Since $m = f'(1)$ we have:

$$m = (1 - b - d) + 2b < 1$$
$$\implies b < d$$

The above model set up was supposed by Kimmel and Alexrod in [7], expanding on their theory we can use the recursive property of pgfs (see (2.4) and (2.5)) to find the probability, given the following initial condition,

A single ancestor cell at generation 0 contains 1 double minute chromosome. (3.2)

that at a given generation of a cell lineage, the current cell has a particular number of DHFR producing double minute chromosomes. Hence, a particular level of resistance to MTX.

Example *With initial state (3.2), what is the probability that at generation 2 a particular cell will contain 4 double minutes?*

The desired probability is located at the coefficient of the s^4 term in the pgf for a cell lineage at generation 2, $f_2(s)$, which can be found via (2.4). However, the full calculation of $f_2(s)$ turns out not to be necessary as we only need the 4th order term:

$$\begin{aligned} f_2(s) &= d + (1 - b - d)f(s) + bf(s)^2 \\ &= \cdots + [a\alpha/2]^2 s^4. \end{aligned}$$

The answer is $(a\alpha)^2/4$.

To be clear, the fact that the PGF (3.1) describes only a particular cell lineage means that at a given generation, it is only possible to use it to collect the probabilities atributed to the possible predicaments of an individual cell in that generation. It can not be used to predict the exact state of the population as a whole.

3.2 The Basic Model: A Question of Resistance Existance

Introducing some notation, let $\mathbf{D}_n$ be the number of double minute chromosomes in a specific cell at generation n. Also, let

$$q_k^{(n)} = \mathcal{P}[\mathbf{D}_n = k | \mathbf{D}_0 = 1].$$

Be the probability that a given cell will contain k double minutes at generation n, given (3.2). Note, the solution to the above example is written, $q_4^{(2)} = (a\alpha)^2/4$. Using equation (2.5), we can collect $q_k^{(n)}$ from $f_n(s)$ via,

$$q_k^{(n)} = \frac{d^k f_n}{ds^k}(0).$$

In this model, each double minute chromosome produces the drug resisting enzyme DHFR at an equal rate. Considering a cell with $\mathbf{D}_n$ double minutes, given a particular intensity of the drug, I, let the cell be defined as resistant to the drug if,

$$\mathbf{D}_n \geq I.$$

Cells resistant to a drug will continue normally in its presence whereas non-resistant, sensitive cells will die immediately. This behaviour is not completely realistic, but will do for the model. A more realistic behaviour will be addressed later.

An important question to ask when administering drugs to patients with cancer is: Will the drug remove their cancer? Or, more specifically posed: Has the cancer in the patient become resistant to the drug we intend to use? Some insight towards the answer to this can be obtained from our simple model. In the interest of uncovering this insight, it is helpful to further confine the task:

Supposing initial condition (3.2), calculate the probability, $R_I(n)$ that at generation n there exists at least one cell resident in the population which holds $\underline{\text{at least } \mathbf{D}_n = I}$ double minutes, causing the population to be resistant to drug strength I. (3.3)

By finding this probability, $R_I(n)$ we can develop an understanding of how resistance existence shapes with time.

To tackle this task, it is sufficient to find, first, the likelihood that every cell in the population is sensitive, and then, via the complement of this, the chance that at least one cell is resistant. It may seem intuitive that the first of these results can simply be achieved if we take the probability that a single cell is sensitive and raise it to the power 2^n, the population count. At first glance, this seems like a straightforward path to the solution. But on closer inspection, there is a fundamental issue. Doing the above securely would require that each cell attains its double minute count completely independent of its siblings. This may appear to be the case, but it's not:

Counter Example *With initial state (3.2). What is the probability that a single cell in generation 1 will have* $\mathbf{D}_1 = 0$*, leaving it sensitive to a drug with strength* $I = 1$*?*

Answer:

$$(1-a)/2 + a\alpha/2.$$

This is simply obtained from the first term of (3.1). *Now, what about the probability that both of the two cells are sensative?*

If we make the asumption that they are independent of each other, the solution would be,

$$((1-a)/2 + a\alpha/2)^2.$$

Which is non-zero (aside from the special case where $a = 1$, and $\alpha = 0$). But remember, in our model, double minutes don't vanish, so that one in the initial cell, or its daughters, have to end up somewhere in generation 1. Hence the chance that neither of the two cells contain a double minute cannot, be non-zero. For any a and α. Thus, the assumption of independence has a falsifying contradiction.

Now that our head is clear, we can continue, but before dismissing the incorrect assumption of independence completely, let us continue with it and explore the approximate results that can be found.

3.2.1 Independence Approximation

Let's define a probability $r_I(n)$ that a particular cell at generation n is resistant to a drug with strength I as the summed likelihood of all states in which the cell is resistant,

$$r_I(n) = \sum_{k=I}^{2^n} q_k^{(n)}.$$

It is clear therefore that the probability of a cell being sensitive to drug strength I is simply,

$$1 - r_I(n).$$

Assuming independence, we can say that at generation n the chance of all 2^n cells being sensitive to drug I is,

$$(1 - r_I(n))^{2^n}.$$

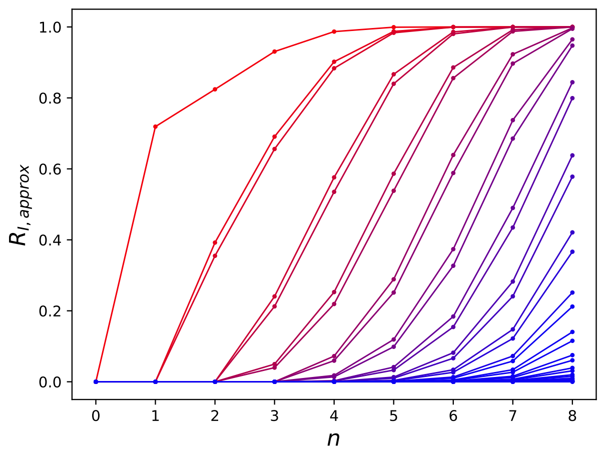
And so, an approximation of the probability, $R_I(n)$ that at least one cell at generation n is resistant can be found from the complement,

$$R_I(n) \approx R_{I,approx}(n) = 1 - (1 - r_I(n))^{2^n}.$$

Which, written fully, becomes,

$$R_{I,approx}(n) = 1 - \left(1 - \sum_{k=I}^{2^n} \frac{d^k f_n}{ds^k}(0)\right)^{2^n}.$$

Below is a plot of the above result for various drug strengths at successive generations. [7] has used experimental data to estimate the parameters $a = 0.94$, and $\alpha = 1$, these are used in all of the plots on the left. The right hand plot uses $a = 0.5$, and $\alpha = 1$.



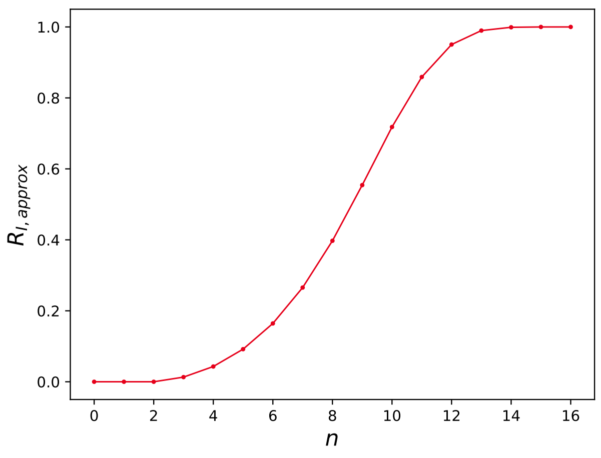


Figure 3.1: $R_{I,approx}(n)$ ploted over multiple generations. For the plotted lines in the left display, I starts at 2 and increases in increments of 1 from red to blue. For the right plot, $I = 6$.

It is observed, unsurprisingly, in Figure 3.1 that the probability of a population containing a particular level of drug resistance increases with time. Furthermore, it is visible, especially in the right plot, that the shape of this increase is similar to that of the sigmoid function, $1/(1-e^{-x})$. See appendix 8.A.2 for the code that was used to generate the vales of R_I.

3.2.2 Exact Analysis

Now we have a rough result, let's rethink task (3.3) without allowing our line of approach to stray into approximation. As a first step, I have formulated the denumerable type probability generating function:

$$\boxed{F_n^{(\mathbf{D}_0)}(x_0,\ldots,x_\lambda) = \sum_{\mathcal{C}_n} p_{c_0\ldots c_\lambda} \prod_{i=0}^{\lambda} x_i^{c_i},}$$

where,

$$p_{c_0\ldots c_\lambda} = \mathcal{P}[c_i \text{ cells have } \mathbf{D}_n = i,\ \text{for } i = 0,\ldots,\lambda]$$

and,

$$\mathcal{C}_n = \{c_0,\ldots,c_\lambda\} \text{ such that } c_i \in \mathbb{Z}\ \forall i,$$

$$\textstyle\sum_i c_i = 2^n, \tag{3.4a}$$

$$\mathbf{D}_0 \leq \textstyle\sum_i i c_i \leq \lambda. \tag{3.4b}$$

$\lambda(n,\mathbf{D}_0) = 2^n\mathbf{D}_0$ is the maximum possible number of double minutes present in generation n; reached if each double minute divides at every cell division. Unlike (3.1), $F_n^{(\mathbf{D}_0)}$ is not limited to describing only the prospective states of a particular cell lineage; in fact, it does this for a whole population. Said completely, it describes the possibilities of a whole population at generation n, which at generation 0 was constituent of a single cell with $\mathbf{D}_0$ double minutes. Working towards the completion of (3.3), we shall consider the initial state of the system to be that stated in (3.2), resulting in,

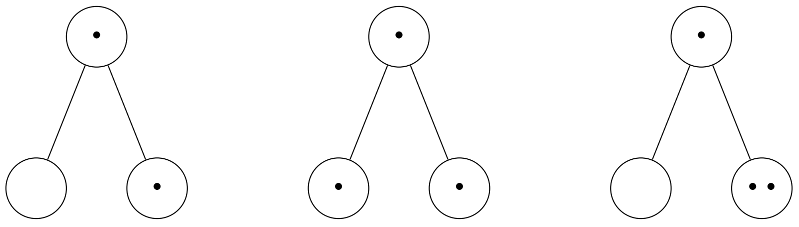
$$F_0^{(1)}(x_0,x_1) = x_1.$$

From now on, when $\mathbf{D}_0 = 1$ the superscript will be dropped so that, $F_n^{(1)} = F_n$. Before proceeding, let's unpack the above definition. To kickstart your understanding, we can the examine function, F_n after a single replication cycle, where $n = 1$. Considering the daughter cells of the single ancestor to be indistinguishable, there are three possible outcomes:

- Wp $p_{110} = (1-a)$ the double minute doesn't divide, and passes to a single daughter.
- Wp $p_{020} = a(1-\alpha)$ it does divide and its progeny are evenly split, resulting two cells with $\mathbf{D}_1 = 1$.
- Wp $p_{101} = a\alpha$ it divides and both new double minutes go to a single daughter.

All of the terms in the probability generating function are related to one of these outcomes, the form of F_1 is stated as follows,

$$F_1(x_0, x_1, x_2) =$$



$$p_{110} \cdot x_0x_1 \quad + \quad p_{020} \cdot x_1^2 \quad + \quad p_{101} \cdot x_0x_2.$$

Clearly, coefficients are the respective probabilities of each outcomes occurring. Looking at the variables you can see that in every term there is one x_i for each cell in the population, the value of i for that cell is descriptive of it's type. Due to the rules of the model, some conceivable outcomes, and thus terms are not possible. For example:

(a) With the ancestor dividing only and exactly into two daughters, there can not be three cells at generation 1. Hence F_1 will not contain $p_{201}x_0^2x_2$.

(b) Since a single double minute can divide into at most 2, there can not be two cells at generation 1 each containing 2 double minutes. Hence, the term $p_{002}x_2^2$ is non existent in F_1 either.

To avoid such issues in the definition of F_n, the set which is summed over, $\mathcal{C}_n$ has been appropriately truncated. Specifically, troublesome terms of the nature of that in (a) are avoided by the condition (3.4a). Which enforces that the total number of cells must be such that every cell in preceding generations has divided into exactly 2, which is 2^n. Furthermore, those excess instances which are of the type in example (b) are dealt with by condition (3.4b), which fixes the range of values that the total number of double minutes is able to traverse.

Ok so, cool, we have this function F_n that holds a complete catalogue of the various population wide outcomes at generation n, with associated probabilities. With F_n it will be possible find the highest level of resistance that is present at generation n, thus completing task (3.3). But before we get too exited, how can we find the terms and respective probabilities of F_n for any particular n? For $n = 1$, this is easy, the outcomes and likelihoods can be simply worked out by hand. An approach of this sort is also technically possible for $n > 1$; with much head scratching I did this for $n = 2$, obtaining:

$$\begin{aligned} F_2(x_0,x_1,x_2,x_3,x_4) = \; & p_{31000} \cdot x_0^3x_1 + p_{30100} \cdot x_0^3x_2 + p_{30010} \cdot x_0^3x_3 + p_{30001} \cdot x_0^3x_4 + \\ & p_{22000} \cdot x_0^2x_1^2 + p_{21100} \cdot x_0^2x_1x_2 + p_{20200} \cdot x_0^2x_2^2 + p_{21010} \cdot x_0^2x_1x_3 + \\ & p_{13000} \cdot x_0x_1^3 + p_{12100} \cdot x_0x_1^2x_2 + p_{04000} \cdot x_1^4, \end{aligned} \tag{3.5}$$

where,

$$\begin{aligned} p_{31000} &= (1-a)[1-a] & p_{04000} &= a^2(1-\alpha)^2[a(1-\alpha)] \\ p_{30100} &= a\alpha[1-a] + \tfrac{1}{2}(1-a)^2[a\alpha] & p_{21010} &= 2a^2\alpha(1-\alpha)[a\alpha] \\ p_{30010} &= 2a(1-a)\alpha[a\alpha] & p_{13000} &= 2a(1-a)\alpha[a(1-\alpha)] \\ p_{30001} &= \tfrac{1}{2}a^2\alpha^2[a\alpha] & p_{12100} &= 2a^2\alpha(1-\alpha)[a(1-\alpha)] \\ p_{22000} &= a(1-\alpha)[1-a] + (1-a)^2[a(1-\alpha)] + \tfrac{1}{2}(1-a)^2[a\alpha] & p_{20200} &= (\tfrac{1}{2}a^2\alpha^2 + a^2(1-\alpha)^2)[a\alpha] \\ p_{21100} &= 2a(1-a)(1-\alpha)[a\alpha] + 2a(1-a\alpha)[a(1-\alpha)] & &\quad + a^2\alpha^2[a(1-\alpha)]. \end{aligned}$$

Although, as n increases, the number of circles and dots you have do draw rapidly becomes ridiculous. Even finding F_3 in this manner is a vast undertaking. A more efficient and tractable approach is to use a systematic procedure which can iterate F_n in to F_{n+1}. Doing this on F_1, n times will result in the desired form of F_n. Due to the nature of F_n, especially the fact that it is constituent of multiple variables, such a proceedure is not trivial.

Iterating F_n in to F_{n+1}

When calculating F_n by hand, the most lucid and thus, the go-to approach is to map out all of the outcomes in a probability tree. I used this method when transforming F_1 in to F_2.

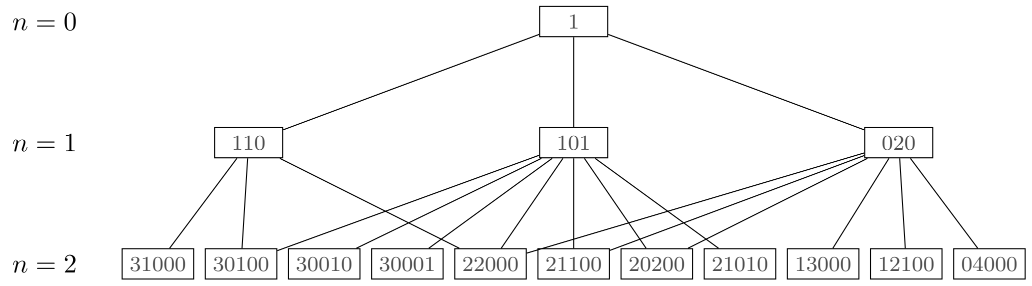


Figure 3.2: A probability tree where each node represents a possible state of the population at generation n. The digits shown in the nodes are the numbers $c_0 \dots c_{2^n}$ which discribe the population.

It was a case of looking at every node in generation 1, considering all of the possibilities stemming from it, and calculating their associated probabilities. Once done, obtaining F_2 was accomplished by encoding the possible outcomes into terms of the relevant form. I will now give a more systematic description of this process, from which, a general rule will be drawn.

To begin, we consider, every outcome-term in F_1; and within those, each cell, represented by an x_i. Zoomed in thus far, the continuation is to consider that cell, on its own, and evaluate all of its possible one-replication developments based on i, the double minute count. Doing so results in a function of the form, $F_1^{(i)}$. With these functions, each x_i in that outcome-term is then substituted by $F_1^{(i)}$. Performing this, for a given outcome-term in F_1 will produce a number of new terms which together, constitute a subset of F_2. Combining these resulting subsets from every outcome-term in F_1 will complete F_2.

In equation form, the above procedure can be summarised by,

$$\begin{aligned} F_2 &= F_1(F_1^{(0)}, F_1^{(1)}, F_1^{(2)}) \\ &= p_{110} \cdot F_1^{(0)} F_1^{(1)} + p_{020} \cdot (F_1^{(1)})^2 + p_{101} \cdot F_1^{(0)} F_1^{(2)} \end{aligned}$$

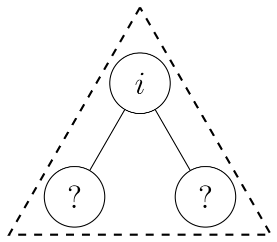
Extending this idea we can say generally that,

$$\boxed{F_{n+1} = F_n(F_1^{(0)}, \dots, F_1^{(2^n)}).} \tag{3.6}$$

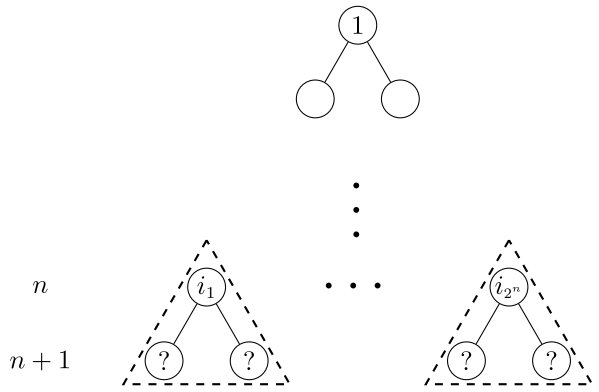
In fact, we could generalise further to say,

$$F_{n+1}^{(\mathbf{D}_0)} = F_n^{(\mathbf{D}_0)}(F_1^{(0)}, \dots, F_1^{(2^n \mathbf{D}_0)}).$$

But this will not be necessary for our aim. Before getting ahead of ourselves, it is worth pausing to cement why the generalisation to equation (3.6) is possible. If we look at the function $F_1^{(i)}$ as a triangle,



where the ancestor cell has i double minutes and the two daughters can have various amounts. Then if we look at a family tree of cells up to generation n. As in the step from F_1 to F_2, each cell can be considered independently; its one-replication outcomes can be investigated to produce a relevant $F_1^{(i)}$. In visual terms, the advancement from one generation to the next is completed by tacking on, 2^n of these triangles to the family tree.



With equation (3.6), it may seem like we're home free. The solution is within our grasp. But those with a keen eye will say: Hang on a moment, yes yes equation (3.6) is all well and good, but how do we find the form of $F_1^{(i)}$ for any i? This of course is essential to the procedure.

Obtaining $F_1^{(i)}$ for any i

Finding, $F_1^{(i)}$ turns out to be a challenge in of itself. To continue, let's spell out the task in clear. If at generation 0 there is one cell with i resident double minutes, what are the possible one replication cycle developments and respective probabilities which lead to the terms and resulting form of $F_1^{(i)}$. As it turns out, this problem can be solved by temporarily dropping our convention that cells are indistinguishable. So here we go. Let each of the parents progeny be labelled individually. Suppose there is one daughter cell called, I; and another called, II. In a given outcome of the system, cell I will have ξ double minutes and cell II will have η. Now, for this we can define a two type probability generating function,

$$\boxed{g_i(u,v) = \sum_{\tilde{\mathcal{C}}_i} \rho_{\xi\eta}^{(i)} u^{\xi} v^{\eta}}$$

where,

$$\rho_{\xi\eta}^{(i)} = \mathcal{P}[\xi \text{ double minutes in I and } \eta \text{ in II} \mid i \text{ in their parent}]$$

and,

$$\tilde{\mathcal{C}}_i = \{\xi, \eta\} \text{ such that } \xi, \eta \in \mathbb{Z}, \text{ and } i \leq (\xi + \eta) \leq 2i.$$

Which outlines the possibilities and probabilities of different one replication outcomes for a single cell with i double minutes. The condition on $\xi + \eta$ enforces the minimal and maximum numbers of double minutes. As an example, here is the form of $g_1(u,v)$ (for simplicity let's reduce the notation so that $\rho_{\xi\eta}^{(1)} = \rho_{\xi\eta}$),

$$g_1(u,v) = \rho_{10}u + \rho_{01}v + \rho_{11}uv + \rho_{20}u^2 + \rho_{02}v^2 \tag{3.7}$$

where,

$$\rho_{10} = \rho_{01} = \tfrac{1}{2}(1-a), \qquad \rho_{11} = a(1-\alpha), \qquad \rho_{20} = \rho_{02} = \tfrac{1}{2}a\alpha.$$

In addition to this, the trivial case of $i = 0$ results in,

$$g_0 = u^0 v^0 = 1$$

as both daughters will be empty. Due to the fact that particles described by it are distinguishable, $g_i(u, v)$ has a useful property:

$$g_i \cdot g_j = g_{i+j}.$$

Which by extension, leads to,

$$g_i = (g_1)^i \tag{3.8}$$

Example *Using (3.7), calculate the probability that after one replication, a single ancestor cell with 2 resident double minutes produces one cell with 1 double minute, and one with 2.*

Considering that the two progeny are distinguished as cell I and cell II, Solving this will be a case of summing two probabilities. $\rho^{(2)}_{21}$ for the outcome where cell I is the one with 2, and $\rho^{(2)}_{12}$ the opposite arrangement. These values can be located as the coefficients of particular terms in g_2. Squaring g_1, they are obtained as follows:

$$\begin{aligned} g_2(u, v) &= (g_1)^2 \\ &= \cdots + \rho^{(2)}_{21} u^2 v + \rho^{(2)}_{12} u v^2 + \cdots . \end{aligned}$$

Some calculation reveals,

$$\begin{aligned} \rho^{(2)}_{21} &= 2\rho_{10}\rho_{11} + 2\rho_{01}\rho_{20} \\ &= a(1-a)[1 - \tfrac{1}{2}\alpha] \end{aligned} \qquad \begin{aligned} \rho^{(2)}_{12} &= 2\rho_{10}\rho_{02} + 2\rho_{01}\rho_{11} \\ &= a(1-a)[1 - \tfrac{1}{2}\alpha] \end{aligned}$$

Meaning that our answer is,

$$2a(1-a)[1 - \tfrac{1}{2}\alpha]$$

To build understanding, the first of these can be read as follows. The probability of there being 2 double minutes in I and 1 in II, given 2 in the parent, is equal to the combined probabilities of:

- One double minute in the ancestor giving I, 1 and II, 0, whilst the the other gives 1 to each.
- The first ancestral double minute gives 0 to I, and 1 to II, with the second giving both 2 to I.

Each of those routes can happen in 2 ways, hence the coefficients.

Notice how, in the above example, two symmetrically similar probabilities of distinguishably defined outcomes were summed to produce the overall probability of a system state which has been defined indistinguishably. This exact method is how, when applied across the whole function, we can traverse from g_i, to $F^{(i)}_1$. It's kind of like folding g_i along the edge of its topologically unique terms. Let me explain.

With g_i for any i firmly in our grasp, we can now use it to obtain $F^{(i)}_1$. For this, we must simply reverse our transition into distinguishably defined cells (sorry I and II, we don't value your individual personalities any more). To do this, terms which are symmetrically similar must be combined; $\rho^{(2)}_{20} u^2$ and $\rho^{(2)}_{02} v^2$ are seen to be the same. Laid out, the procedure looks like this,

$$\boxed{F^{(i)}_1(x_0, \ldots, x_{2i}) = g_i(u, v)\Big|_{(u^j \text{ and } v^j) \,\to\, x_j}} \tag{3.9}$$

When making this transformation, it is important not to forget the hidden x_0s. For example, $\rho_{10}u$ should become $\rho_{10}x_1x_0$, not just $\rho_{10}x_1$. For practice, let's convert g_1.

$$\begin{aligned} g_1(u,v) &= \rho_{10}uv^0 + \rho_{01}u^0v + \rho_{11}uv + \rho_{20}u^2v^0 + \rho_{02}u^0v^2 \\ &= (\rho_{10}+\rho_{01})x_1x_0 + \rho_{11}x_1^2 + (\rho_{20}+\rho_{02})x_2x_0 \\ &= (1-a)\cdot x_1x_0 + a(1-\alpha)\cdot x_1^2 + a\alpha\cdot x_2x_0 \\ &= F_1^{(1)}(x_0,x_1,x_2) \end{aligned}$$

We now have a secure means of obtaining $F_1^{(i)}$ for any i: Use (3.8) to turn g_1 into g_i, then apply (3.9) turning g_i into $F_1^{(i)}$.

Collecting $R_I(n)$ from F_n

Thus far we have generated the capability of acquiring F_n for any n. With this we can now collect the resistance existence probability, $R_I(n)$ for any drug strength I, and any generation n, completing task (3.3) analytically.

To harvest the desired probability from F_n, given I, we must combine the coefficients of all of the terms in F_n which are connected to states in which there exists at least one cell resistant to I. This can be done in the following way:

1. From F_n eliminate all terms representing populations that are resistant,

$$\hat{F}_n = F_n\Big|_{x_j = 0 \text{ for } j \geq I.}$$

2. Combine all remaining coefficients, obtaining the probability of sensitivity to drug I,

$$1 - R_I(n) = \hat{F}_n\Big|_{x_j = 1 \text{ for } j < I.}$$

3. Take to complement, obtaining $R_I(n)$,

$$R_I(n) = 1 - (1 - R_I(n)).$$

Example *Using the above method on F_2, outlined in equation (3.5), calculate $R_2(2)$.*

Firstly,

$$\begin{aligned} \hat{F}_2 &= F_2\Big|_{x_j = 0 \text{ for } j = 2,3,4} \\ &= p_{31000}x0^3x_1 + p_{22000}x_0^2x_1^2 + p_{13000}x_0x_1^3 + p_{04000}x_1^4. \end{aligned}$$

Then,

$$\hat{F}_2\Big|_{x_j = 1 \text{ for } j = 0,1} = p_{31000} + p_{22000} + p_{13000} + p_{04000}.$$

Finaly, taking the complement and substituing the probability values, we obtain the answer,

$$R_2(2) = 1 - \Big\{ \big[(1-a)^2\big] + \big[a(1-\alpha)(1-a) + (1-a)^2(a(1-\alpha)) + \tfrac{1}{2}(1-a)^2(a\alpha)\big] + \big[2a^2(1-a)\alpha(1-\alpha)\big] + \big[a^3(1-\alpha)^3\big] \Big\}.$$

Suppose, $a = 0.94$ and $\alpha = 1$, the value becomes,

$$R_2(2) = 0.99471.$$

With double minutes almost always dividing and being passed to a single daughter, resistance increases rapidly. Consider a different case where they do always divide, but rarely pass to a single progeny; if $a = 1$, and $\alpha = 0.1$, we get,

$$R_2(2) = 0.271$$

Which is the probability that at every division, all double minutes, divide and split equally. This is the only way to avoid level 2 resistance in this case.

Analytic results of $R_I(n)$

Combining all of the steps outlined in this exact analysis, we have a rather convoluted analytic formula for $R_I(n)$:

$$\boxed{R_I(n) = 1 - F_n\Big|_{x_j = \begin{cases} 0, & j \geq I \\ 1, & j < I \end{cases}}}$$

where,

$$\boxed{F_n = F_1\Big|_{x_j \to F_1^{(j)} \text{ for } j = 0,\ldots,2^1} \overset{n-1}{\iddots} \Big|_{x_j \to F_1^{(j)} \text{ for } j = 0,\ldots,2^{n-1}}}$$

and,

$$\boxed{F_1^{(j)} = (g_1)^j\Big|_{u^k, v^k \to x_k}}$$

As it turns out, using this formula by hand gets pretty arduous after $n = 2$, so, to save on mental reserves I have enlisted a computer to do it for me. This in its self was a non-trivial task. Finding F_n involves doing polynomial substitutions with polynomials that have varying numbers, but generally quite a lot, of variable types. In absence of appropriate software available to do this, I wrote some myself. The Python code for it can be found in appendix 8.A.1. With this the following plots were produced. In all of these plots $a = 0.94$ and $\alpha = 1$ as estimated by [7].

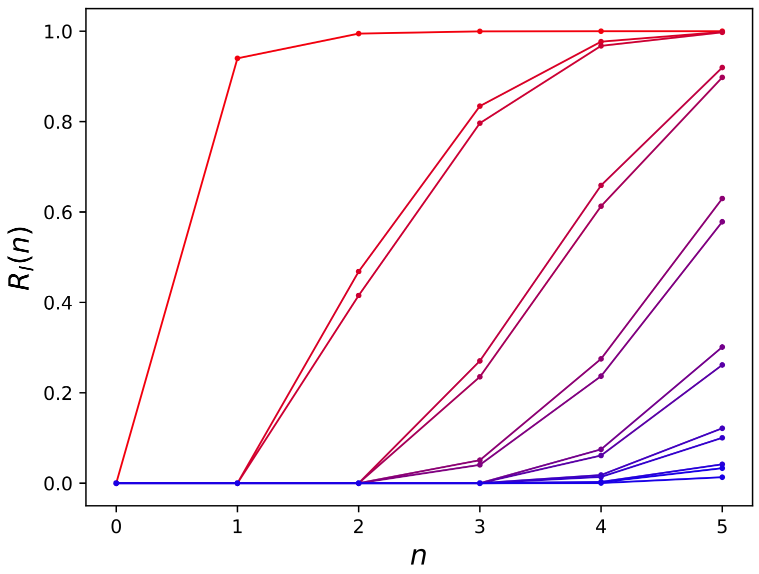


Figure 3.3: $R_I(n)$ plotted for 5 generations using the analytic method. From red to blue I starts at 2 and increases in increments of 1.

As you can see, the above plot of $R_I(n)$ only goes up to generation 5. Setting this maximum was not in fact a choice of aesthetics, it's actually where the computer got stuck. Yes, as it turns out, the count of terms in F_n explodes to in-computable numbers; in order, from generation 1 to 5, F_n consists of, 3, 11, 66, 914, and 43819 terms. The process can be sped up a bit by eliminating terms which have associated probabilities so small that they have been set to 0 by Python's float rounding. There are in fact, quite a few of of these that accumulate, leading to a real performance enhancement. Unfortunately though, it is concluded that with increasing generations, this type of analysis quickly becomes unfeasible to carry out within a human life span. That is of course only true if we don't first formulate the means of immortality.

A valid question to ask at this point would be: How do we know for sure that all of the exact analysis stated, programmed, and applied is correct. To verify the validity of these results I enlisted a computational method of proof. With simple rules, the system of successive cell divisions and resulting populations is relatively straightforward to simulate. Once the functionality to simulate a single population is attained, the resistance existence of that population can be measured. Then, after repeating this process many times, it is then possible, by calculating mean averages, to estimate accurate values of $R_I(n)$ for various I. By making the number of simulations arbitrarily large, the estimated values of $R_I(n)$ can be made arbitrarily precise. Below is a comparison of these estimated values of $R_I(n)$ set against those obtained analytically. In producing the estimates, 1000 simulations were completed.

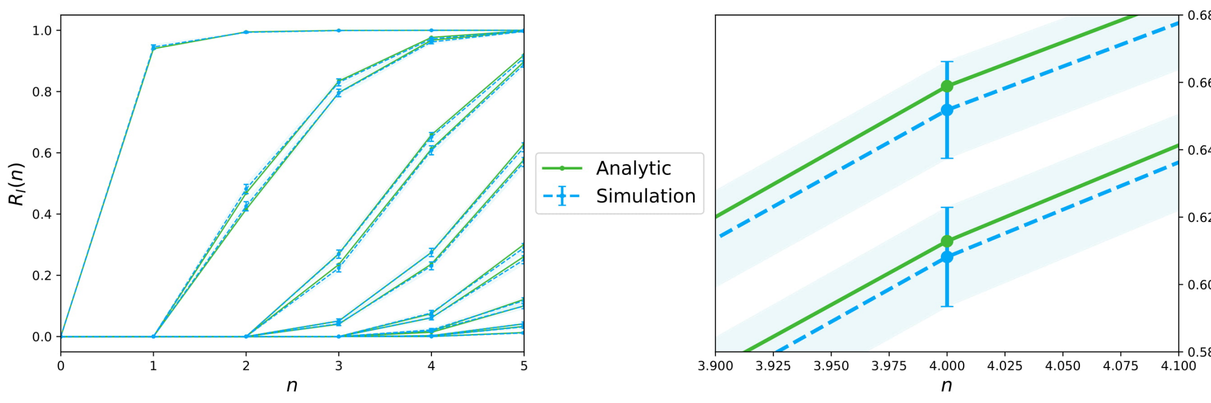


Figure 3.4: A comparison of $R_I(n)$ values resulting from analytic, and numeric methods.

Extending above and below the values obtained by simulation are error bars, these spread plus and minus one standard deviation from the sample mean. It is observed that the analytic results fit nicely with those produced by estimation. Furthermore, it was found when increasing the number of individual simulations, that the estimated values converge closer to the analytic results; when 100,000 simulations are used the difference shrinks to $O(10^{-4})$. Matching nicely, these simulation results inspire confidence in validity of the analysis.

You may notice, that the above analytic values of $R_I(n)$ differ from those obtained by the independence approximation. As shown by Figure 3.5, this is indeed the case.

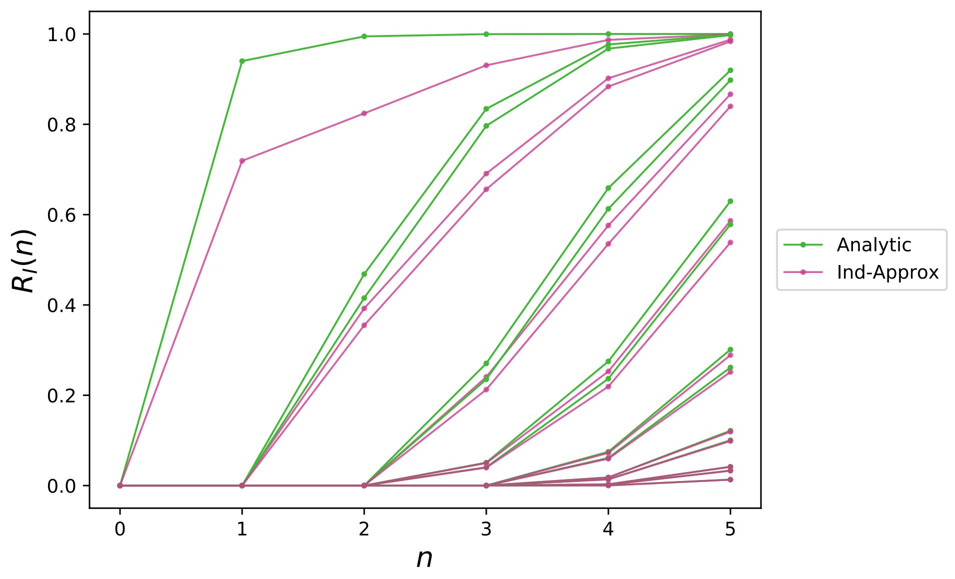


Figure 3.5: A comparison of $R_I(n)$ values resulting from analytic, and approximate methods.

Although, for increasing generations, and values of I, there seems to be a decrease in the disparity of the results. The independence approximation improves when the population is larger.

Concluding, it seems, the best approach in calculating $R_I(n)$ depends on n. For small $n \leq 5$, the analytic method offers precise reults. Then, for 6 and beyond the independence assumption offers a good approximation.

3.3 Simulating the Basic GW Process Model

To gain further understanding of the behaviours of this model, I have run more targeted simulations. I was curious about the nature of resistance diffusion after the application of a drug.

In the following simulation, at generation 0 there are five cells, four of which contain 0 double minutes, and one with 1. This population was propagated with the dynamics stated above for 22 generations. During generation 6 a drug of strength 3 was applied to the population. As in the previous experiment a, and α are set to the estimated values of 0.94 and 1 respectively.

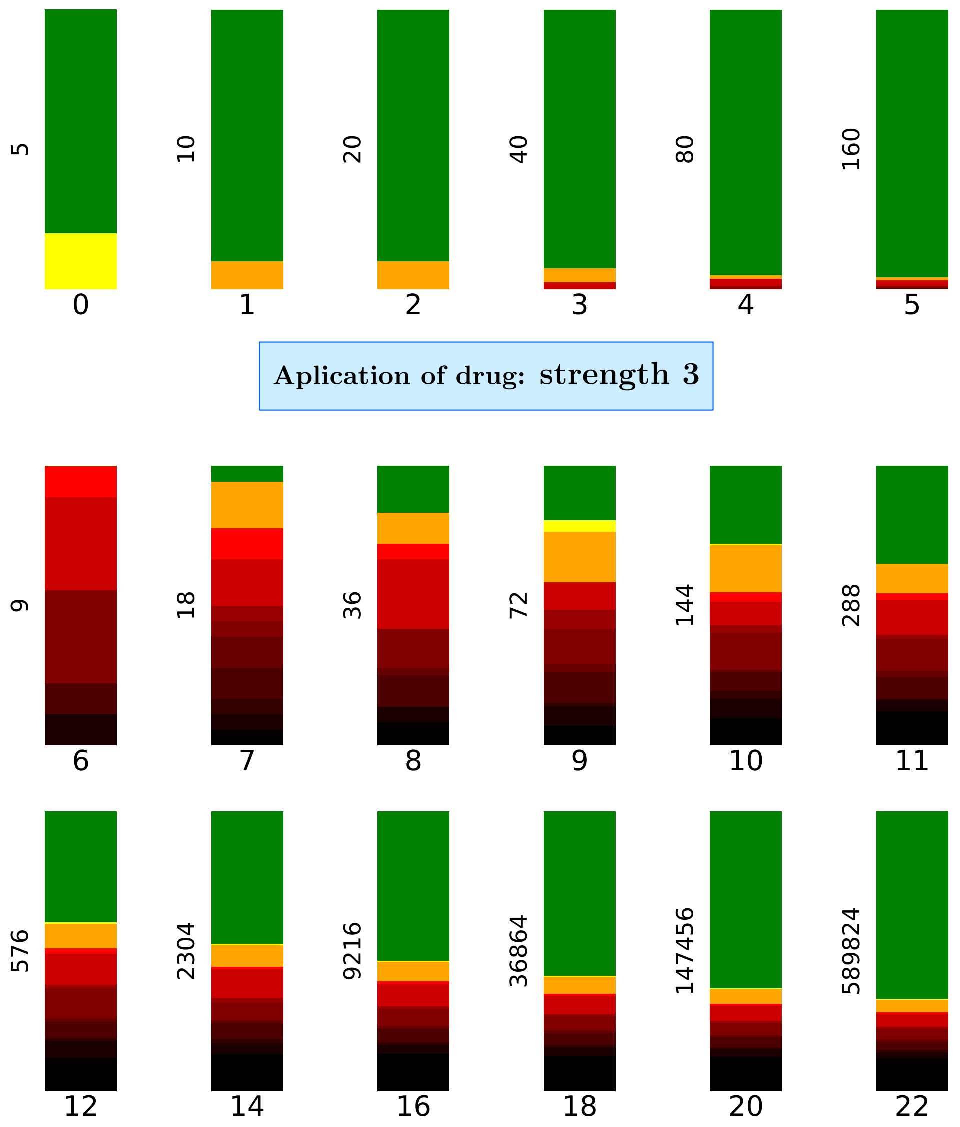


Figure 3.6: A representation of drug resistance ratios in a cell population over 22 generations with a strength 3 drug applied during generation 6.

In Figure 3.6, every bar is representative of the population at a generation n, with n labelled on the bottom edge. On the left edge of the bars is a number displaying the total quantity of cells in the population at the respective generation. All of the rectangles are of the same physical dimension, this is because they simply represent the ratios of each cell resistance type. For example, the first plot can be seen to be four fifths green (representing cells with $\mathbf{D}_0 = 0$), and one fifth yellow (for those with $\mathbf{D}_0 = 1$). Cells with higher double minute counts of 2, 3, 4 so on and so forth are respectively represented by orange, red, dark red, followed by successively darker reds until, at the count of 11 and beyond, black. See appendix 8.B for the core code that was used.

As you can see, the only cells alive at the end of generation 6 are those with 3 or more double minute chromosomes, $\mathbf{D}_6 \geq 3$, which are the ones that are, in theory, producing sufficient DHFR enzyme to withstand the dose of MTX. During the following generations, in absence of MTX, there is a gradual reduction of the portion of cells which have resistance.

A close look at some of the later generations shows that odd numbers of $\mathbf{D}_n$ are unlikely to be present. This is an artefact of the parameter choice. With $\alpha = 1$, on double minute division, both progeny will invariably go to the same cell; and with $a = 0.94$, double minutes divide at almost every instance of mitosis, leading to even double minute counts.

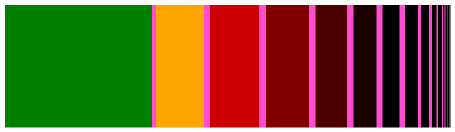


Figure 3.7: Generation 15 of the simulation with odd double minute numbers highlighted in pink.

3.4 Simplifying assumptions

Although the above mathematical model is nice and tractable to analyse, further exploration of the literature [9] has exposed it to contain some critical simplifications of reality. Primarily the model assumes:

1. An equal growth rate for all cells. *Whether a cell has 2 or 20 double minutes, it will divide along with all the rest.* As it turns out, more resistant cells which produce inflated quantities of DHFR also divide less frequently than sensitive cells. Eliminating this dynamic from the model allows for another problem to emerge:

2. No upper limit on the number of double minutes in a cell. *With the right sequence of divisions, cells can accumulate arbitrarily large numbers of double minutes.* Of course, this doesn't happen in reality. High enough levels of DHFR will prevent a cell from dividing. There may not be a hard limit on the number of double minutes in a cell, but it can't grow indefinitely. After [9] exposed cells to MTX for several generations a mean average of 28 double minutes per cell was recorded.

3. Static drug dynamics. *A threshold value is used to decide whether, on application of drug strength I, a cell will die, or carry on unaffected.* This dynamic is completely un-realistic. What actually happens in the presence of MTX is that cells have an <u>increased death rate</u> and a <u>decreased division rate</u>, and the amount that the drug affects a cell in this way will depend on that cell's particular level of resistance.

In addition to these reductions corresponding to fundamental dynamics of the system, there are some further biological events and complexities that the model doesn't include:

4. Varying numbers of DHFR producing neucleotide sequences per double minute chromosome. A single double minute can produce a relative lot or little of the enzyme. Generally constituent in one will be 1 to 5 of these sequences.

5. Re-integration to the main body of DNA. In populations that have been grown in MTX for extended period of time, it has been observed that, sometimes, resistance isn't lost on the removal of the drug. This occurs because, on ocation, the extra-chromosomal elements of DNA can re-enter the mass of chromosomal DNA, causing a fixation of the enzyme production.

As well as within the cell, we also have further dynamics relating to the interaction of cells.

6. Scarsity of resources. No population of cells has acsess to an infinite quantity of food, so it is inevitable that it will grow up to a stable limit by way of a dynamic we call logistic growth. More on that later.

7. Horizontal resistance transfer Cells with resistance causing double minutes can transfer these gene sequences to neighboring cells, as described in the biological background, via the use of a sex pilus.

Now these simplifications are understood we can push our understanding further. To this aim I have developed a more complex model.

Chapter 4

Discrete Time Modelling of Drug Resistance II: Dynamic Probabilities

4.1 Setting up the model

Progressing further towards reality, the following model steps beyond assumptions 1, 2, and 3 in section 3.4. Cells have variable birth and death probabilities. We haven't yet moved into continuous time however; like the basic model, this one is also a Galton Watson process with every cell acting simultaneously at the end of their fixed generation. At each population changing event, every cell will either die, wp p_0; birth a single daughter, wp p_1; or divide into two wp p_2. This may seem familiar. Although, these probabilities can now vary with differing drug applications and double minute numbers. Let's consider a drug strength I, and a cells double minute count, regardless of generation as $\mathbf{D}$. Since the dynamics of division will change when a drug becomes present, we will divide the probabilities accordingly into $p_i^{(\text{no drug})}$, and $p_i^{(\text{drug})}$. For space purposes, the superscripts are abbreviated. Dependencies of the probabilities are as follows:

- When no drug is present a cell with $\mathbf{D}$ double minuets acts in line with,

$$\begin{aligned} p_2^{(\text{nd})} &= p_2^{(\text{nd})}(\mathbf{D}) \\ p_1^{(\text{nd})} &= p_1^{(\text{nd})}(\mathbf{D}) \\ p_0^{(\text{nd})} &= p_0^{(\text{nd})} = \text{ constant.} \end{aligned}$$

- If drug strength I is applied, the same cell will be dictated by,

$$p_i^{(\text{d})} = p_i^{(\text{d})}(\mathbf{D}, I) \text{ for all } i.$$

The form of these probabilities will not be derived rigorously from some fundamental laws, but instead they are going to be fixed with functional forms in such a way that they exhibit the desired dynamics. In the process of deciding these functional forms, we lay out exactly how they are to behave, then choose functions that match these behaviours. Ill run you through the thought process.

4.1.1 No Drug Case

, to advance assumption 1 in 3.4, we wish that the number of double minutes will affect a cell's division rate. Specifically, with increased $\mathbf{D}$, $p_2^{(\text{nd})}$ should decrease to an asymptote at 0. An inverse exponential will fit the bill.

$$p_2^{(\text{nd})} \propto e^{-\tau \mathbf{D}}$$

τ is a parameter that will control the degree to which double minutes reduce a cells division rate; large τ, big reduction. Now, two things are to be said. First, $p_0^{(\text{nd})}$ is set to a constant,

$$p_0^{(\text{nd})} = \beta$$

allowing resistance to disappear from the system. Second, and this is true for both the no drug and the drug cases, p_1 is not tuned independently but is simply set to the complement of the other two probabilities combined;

$$p_1 = 1 - (p_0 + p_2).$$

With those points made, it becomes clear that the combination of $p_0^{(\text{nd})}$, and $p_2^{(\text{nd})}$ can not exceed 1. This property can be enforced by fixing the value of $p_2^{(\text{nd})}$ at $\mathbf{D} = 0$ (its relevant maximum) to $1 - \beta$. Resulting in:

$$p_2^{(\text{nd})} = (1 - \beta)e^{-\tau \mathbf{D}}$$

Which, for $\beta = \tau = 0.2$ looks like:

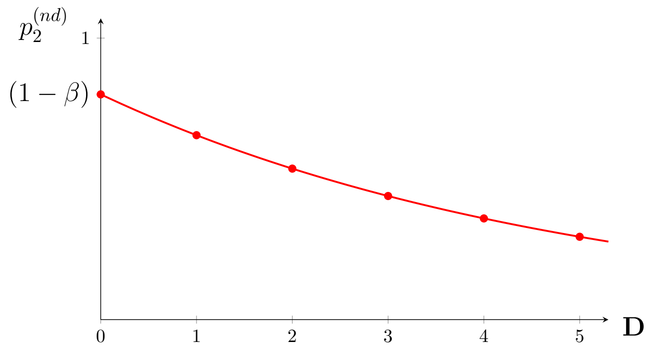


Figure 4.1: $p_2^{(nd)}(\mathbf{D})$ plotted against $\mathbf{D}$ with illustrative values of $\beta = \tau = 0.2$.

4.1.2 Case Where MTX is Present

, more dynamics are to be included. Laying them on the table, we have:

- $p_0^{(\text{d})}$, the death probability is to depend on the sensitivity of the cell to the present drug. Especially, when this sensitivity is low $p_0^{(\text{d})}$ must tend to the standard death likelihood of β; and for high sensitivity, it must tend to 1. Importantly, we want the variation between these two extremes to be smooth.

- $p_2^{(\text{d})}$ needs to, as before the drug was applied, contain an element of decrease with progressive values of $\mathbf{D}$. This however is not the full story; the model should also include the fact that a cell's division is inhibited by a drug which it is sensitive to. So, along with the stated element of decrease, $p_2^{(\text{d})}$ will be tuned by an additional reducing factor dependent on sensitivity.

Before we get stuck in, it will be of great use to first define a measure of sensitivity relating to a particular cell with $\mathbf{D}$ double minutes and a drug of strength I. Let's define sensitivity $\phi(\mathbf{D}, I) \in \mathbb{Z}$ as,

$$\phi = I - \mathbf{D}$$

For example, a cell with 4 double minute, in the presence of a drug with strength 2 will have sensitivity $\phi = -2$.

Examining the conditions that are to be met by $p_0^{(\text{d})}$, it can be visualised that we are after something like this:

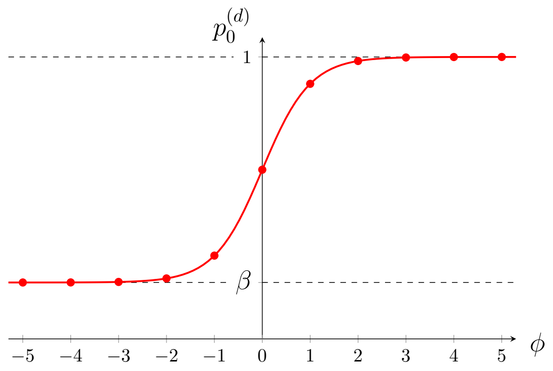


Figure 4.2: $p_0^{(d)}(\phi)$ plotted against ϕ with ilustrative values of $\beta = 0.2$, and $\nu = 2$.

Well, why not use this exactly. A variation on the sigmoid function,

$$p_0^{(d)} = \frac{1-\beta}{1+e^{-\nu\phi}} + \beta$$

In this, ν is a parameter that tunes the steepness of the function's transition between its asymptotes, β and 1. Tuning ν can blur or sharpen the boundary of a drugs affect. In the limit of $\nu \to \infty$, $p_0^{(d)}(\phi)$ becomes a step function with a special point at $\phi = 0$,

$$\lim_{\nu\to\infty} p_0^{(d)}(\phi) = \rho(\phi) = \begin{cases} \beta, & \phi < 0 \\ \frac{1}{2}, & \phi = 0 \\ 1, & \phi > 0. \end{cases}$$

Aside This limit shows a nice link with the basic model. Because we only care about integer inputs for $p_0^{(d)}(\phi)$, with a slight variable shift $\varphi = \phi - \delta$, where $0 < \delta < 1$ the function $\rho(\varphi)$, for integer φ is,

$$\rho(\varphi) = \begin{cases} \beta, & \phi \leq 0 \\ 1, & \phi > 0. \end{cases}$$

With β set to 0, we can generate the precise conditions of the basic model: In a drug free environment,

$$p_2^{(nd)} = 1,$$

and when MTX *is* present,

$$p_0^{(d)} = \rho(\varphi)\big|_{\beta=0}$$
$$p_2^{(d)} = 1 - p_0^{(d)}.$$

Now, we draw our attention to the formulation of $p_2^{(d)}$. When a cell is very resistant to a drug, we want it to act as it would if there were no drug. That is to say, when $p_0^{(d)} \to \beta$, $p_2^{(d)}$ should tend to $p_2^{(nd)}$. Furthermore, if the cell is actually very sensitive to a present drug, it should be the case that $p_2^{(d)} \to 0$. The following form ot $p_2^{(d)}$ decreases linearly with an increase in $p_0^{(d)}$, whilst fitting these two boundary conditions exactly,

$$p_2^{(d)} = p_2^{(nd)} \left(\frac{1 - p_0^{(d)}}{1-\beta} \right);$$

when $p_0^{(d)} = \beta$,

$$p_2^{(d)} = p_2^{(nd)} \left(\frac{1-\beta}{1-\beta} \right) = p_2^{(nd)},$$

and when $p_0^{(d)} = 1$,

$$p_2^{(d)} = p_2^{(nd)} \left(\frac{1-1}{1-\beta} \right) = 0.$$

Said in full,

$$\begin{aligned} p_2^{(d)} &= (1-\beta)e^{-\tau \mathbf{D}} \left(\frac{1-p_0^{(d)}}{1-\beta} \right) \\ &= e^{-\tau \mathbf{D}} \left(1 - \beta - \frac{1-\beta}{1+e^{-\nu\phi}} \right) \\ &= (1-\beta)e^{-\tau \mathbf{D}} \left(\frac{e^{-\nu\phi}}{1+e^{-\nu\phi}} \right) \\ &= (1-\beta) \left(\frac{e^{-\tau \mathbf{D}}}{1+e^{\nu\phi}} \right). \end{aligned}$$

Which, plotted against ϕ, for different values of $\mathbf{D}$, looks like:

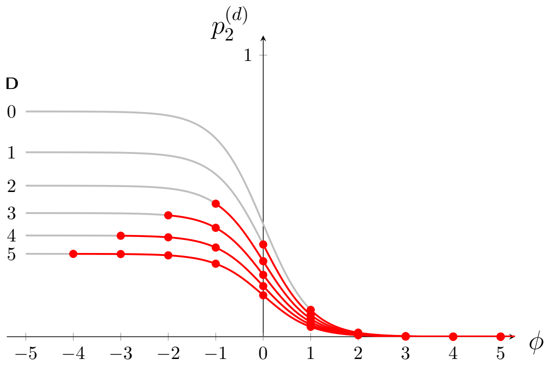


Figure 4.3: $p_2^{(d)}(\mathbf{D}, \phi)$ plotted against ϕ for various $\mathbf{D}$, with illustrative values of $\beta = \tau = 0.2$, and $\nu = 2$.

As you can see, some portion of each line is red, whilst the rest is grey. Because I, the drug strength must be greater than 1, for a given $\mathbf{D}$, ϕ will have a realistic minimum value of $1 - \mathbf{D}$. Which is reached, as you might expect, when $I = 1$. This red section of each line is positioned to display the range of values for which ϕ is physical.

4.1.3 Summarising the Model

We have the probabilities for both the drug and drug free cases as follows:

No Drug

$$\boxed{\begin{aligned} p_0^{(nd)} &= \beta \\ p_2^{(nd)} &= (1-\beta)e^{-\tau \mathbf{D}} \\ p_1^{(nd)} &= 1 - (p_0^{(nd)} + p_2^{(nd)}) \end{aligned}}$$

Drug

$$\boxed{\begin{aligned} p_0^{(d)} &= \frac{1-\beta}{1+e^{-\nu\phi}} + \beta \\ p_2^{(d)} &= (1-\beta) \left(\frac{e^{-\tau \mathbf{D}}}{1+e^{\nu\phi}} \right) \\ p_1^{(d)} &= 1 - (p_0^{(d)} + p_2^{(d)}) \end{aligned}}$$

Where:

β is the base, no drug death probability,

τ is an amplitude for which double minutes reduce a cell's division probability, and

- ν controls the blurriness of the drug effectiveness boundary, for which on one side, cells are completely sensitive, and on the other are completely resistant.

With these probabilities, we can calculate the mean number of offspring of a cell with $\mathbf{D}$ double minutes in the presence of a drug strength I, $m^{(d)}(\mathbf{D}, I)$ or in the absence of a drug, $m^{(nd)}(\mathbf{D})$. Using $m = \sum ip_i$, we get,

$$m^{(nd)} = (1-\beta)(1+e^{-\tau\mathbf{D}}) \tag{4.1}$$

and,

$$m^{(d)} = (1-\beta)\left(\frac{1+e^{-\tau\mathbf{D}}}{1+e^{\nu\phi}}\right) \tag{4.2}$$

4.2 An Exploration in Survival Likelihoods for Cells of Particular Resistance Levels

To progress understanding of survival likelihoods we will consider particular cell $\mathbf{D}$ values in various drug I environments. More precisely, we will work with a homogeneous collection of cells that have equal and fixed $\mathbf{D}$ values. When one of these cells for example with $\mathbf{D} = 3$ goes through a division, it's two daughters must also have $\mathbf{D} = 3$. This property is present when $a = 1$, and $\alpha = 0$. Under this constriction of the model, the following questions can be posed:

4.2.1 Maximum Resistance in Drug Free Environment

Question 1 *When $\beta > 0$ and $\tau > 0$, what is the maximum level of resistance that can exist in a drug free environment without certainty of extinction?*

Going back to section 2.3, we know that for a population to survive indefinitely, its cells must be dividing in a supercritical fashion. Therefore, we are interested specifically in finding the largest $\mathbf{D}$ which satisfies: $m^{(nd)}(\mathbf{D}) > 1$. Using (4.1) we have:

$$\begin{aligned}(1-\beta)(1+e^{-\tau\mathbf{D}}) &> 1\\ 1-\beta+(1-\beta)e^{-\tau\mathbf{D}} &> 1\\ e^{-\tau\mathbf{D}} &> \frac{\beta}{1-\beta}\\ \mathbf{D} &< -\frac{1}{\tau}\log\left(\frac{\beta}{1-\beta}\right).\end{aligned} \tag{4.3}$$

And so, the highest possible resistance level, $\mathbf{D}_{\max}$ that can survive indefinatly is:

$$\mathbf{D}_{\max} = \left\lfloor -\frac{1}{\tau}\log\left(\frac{\beta}{1-\beta}\right)\right\rfloor \tag{4.4}$$

Notice how in the limit as $\beta \to 0$, $\frac{\beta}{1-\beta} \to 0$, so $\log(\frac{\beta}{1-\beta}) \to -\infty$, and as a result,

$$\mathbf{D}_{\max} \to \infty.$$

This result makes great sense because with $\beta = 0$, death without a drug is impossible, so there can be no maximum resistance level. Furthermore, equation (4.4) also secures the link between points 1 and 2 in section 3.4; if we consider the limit $\tau \to 0$, we also get $\mathbf{D}_{\max} \to \infty$; reducing the dividing efficiency of more resistant cells enforces an upper limit on resistance.

Suppose $\beta = 0.2$ and $\tau = 0.3$, $\mathbf{D}_{\max} = \lfloor 4.621\ldots\rfloor = 4$; cells with $\mathbf{D} = 5$ will die more often than they reproduce. As demonstration, I have simulated two populations over a number of generations, one has $\mathbf{D} = 4$, the other $\mathbf{D} = 5$. We expect that the first will prosper and the second will eventually die.

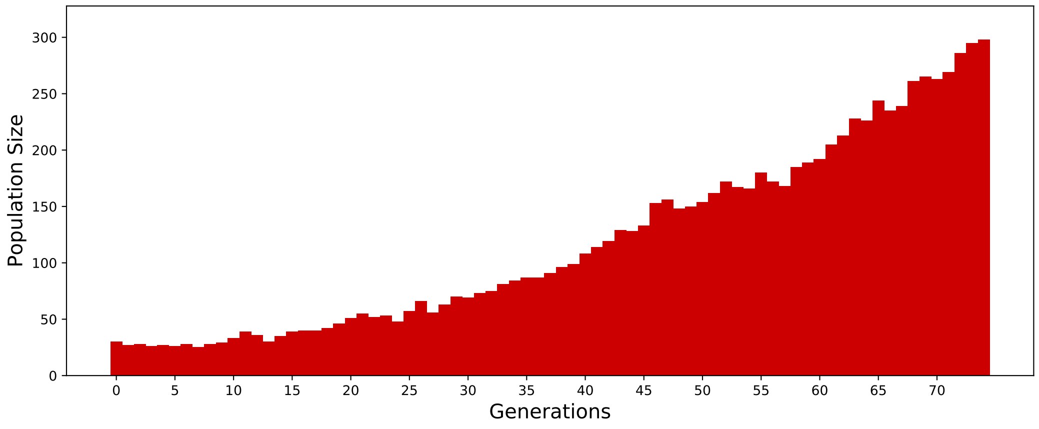


Figure 4.4: A population of cells with $\mathbf{D} = 4$, initially of size 30, propagated in a drug free environment with $\beta = 0.2$ and $\tau = 0.3$ for 74 generations.

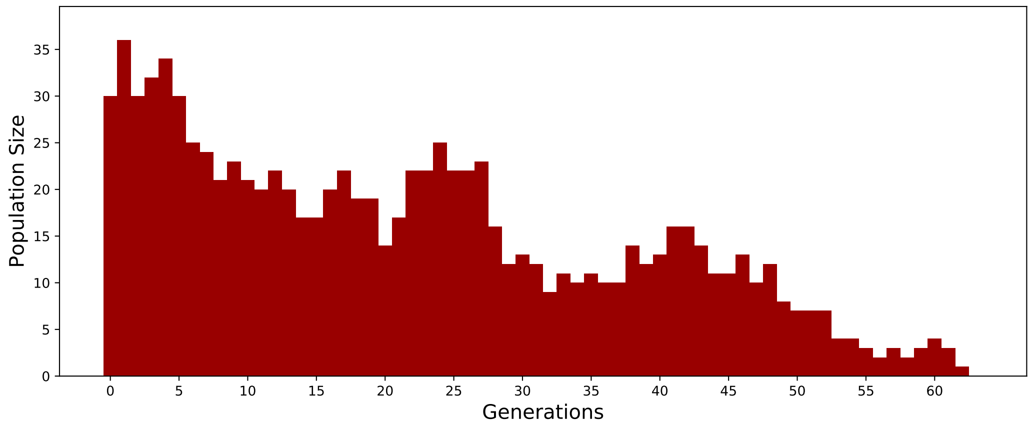


Figure 4.5: A population of cells with $\mathbf{D} = 5$, initially of size 30, propagated in a drug free environment with $\beta = 0.2$ and $\tau = 0.3$ for 62 generations.

A fair point to make is that these are singular examples of a random process; they could be outliers. Instead of seeing them as a computational proof, they must instead be viewed as a loose verification of the analysis.

4.2.2 Avoiding Certain Extinction with Drug Present

Question 2 *What is the possible range of resistance levels that a population of cells can posses to avoid* <u>*certain*</u> *extinction in the presence of* <u>*drug strength I*</u>*?*

To answer this, we can start on the same line of approach that was followed in question 1. Specifically, we wish to find the values of $\mathbf{D}$, that satisfies $m^{(d)}(\mathbf{D}, I) > 1$. Using (4.2),

$$(1-\beta)\left(\frac{1+e^{-\tau\mathbf{D}}}{1+e^{\nu(I-\mathbf{D})}}\right) > 1$$

$$(1-\beta)(1+e^{-\tau\mathbf{D}}) > 1+e^{\nu(I-\mathbf{D})}$$

$$1-\beta+(1-\beta)e^{-\tau\mathbf{D}} > 1+e^{\nu(I-\mathbf{D})}$$

$$\boxed{(1-\beta)e^{-\tau\mathbf{D}} - e^{\nu(I-\mathbf{D})} - \beta > 0} \tag{4.5}$$

Inequality (4.5) is an implicit definition of the possible values of $\mathbf{D}$; any cell resistance $\mathbf{D}$ that satisfies it can survive in a medium of drug strength I. We'll come back to the full form of (4.5) in a while, but for now, let's examine the special case where $\beta = 0$. It becomes,

$$\begin{aligned} e^{-\tau\mathbf{D}} - e^{\nu(I-\mathbf{D})} &> 0 \\ e^{-\tau\mathbf{D}-\nu(I-\mathbf{D})} &> 1 \\ -\tau\mathbf{D} - \nu(I-\mathbf{D}) &> 0 \\ (\nu-\tau)\mathbf{D} - \nu I &> 0 \\ \mathbf{D} &> \frac{I}{1-\frac{\tau}{\nu}} \end{aligned}$$

When $\beta = 0$, the minimum resistance $\mathbf{D}_{\text{min}}$ that is required to survive in the presence of a drug with strength I is,

$$\mathbf{D}_{\text{min}} = \left\lceil \frac{I}{1-\frac{\tau}{\nu}} \right\rceil \tag{4.6}$$

Notice, when $\beta = 0$ there is no upper limit on supercritical values of $\mathbf{D}$, just a lower one. This is because the upper limit is solely enforced by β.

Equation (4.6) reveals some interesting properties of the relationship between τ and ν. Firstly, it is important to note that in above workings, the last step is only valid if $\nu > \tau$. If not, the final inequality instead becomes,

$$\mathbf{D} < \frac{I}{1-\frac{\tau}{\nu}}$$

with the sign flipped, and equation (4.6) must be floored instead of ceilinged. In this case, the results are un-physical; the boundary is now an upper limit, and the maximum possible $\mathbf{D}$ is negative. In other words: if $\nu > \tau$, any resistance level will be killed by any drug strength.

Furthermore when $\nu = \tau$, there is a singularity; $\mathbf{D}_{\text{min}}$ becomes infinite as $\frac{\tau}{\nu} \uparrow 1$. In this case it is also true that any drug will kill any population, although for those with very high resistance levels, the process can take a really long time. As an illustration, here is a population of 30 cells with $\mathbf{D} = 15$, acting by $\nu = \tau = 0.3$, and taking <u>874</u> generations to be killed by a drug of strength 1.

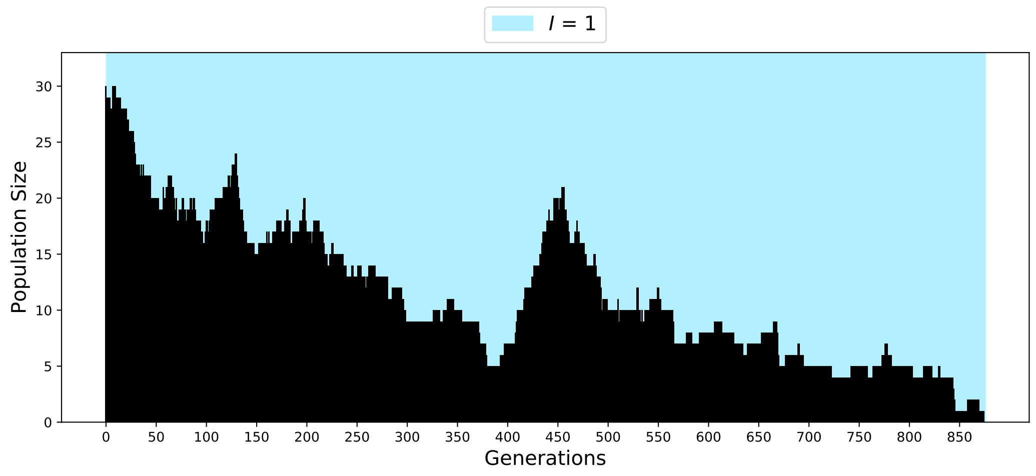


Figure 4.6: A population of cells with a high $\mathbf{D} = 15$, initially of size 30, deteriorating in the presence of low drug strength 1, with $\beta = 0$ and $\nu = \tau = 0.3$ for 874 generations.

Now, suppose that cells act in alignment with $\beta = 0$, $\nu = 1$, and $\tau = 0.4$. If there is a drug of strength 4 present, $\mathbf{D}_{\text{min}} = \lceil 6.666\ldots \rceil = 7$. Under these conditions, populations with resistance level 7 should expand indefinably, whilst those with 6 shall die out. Initial simulation results agree with this:

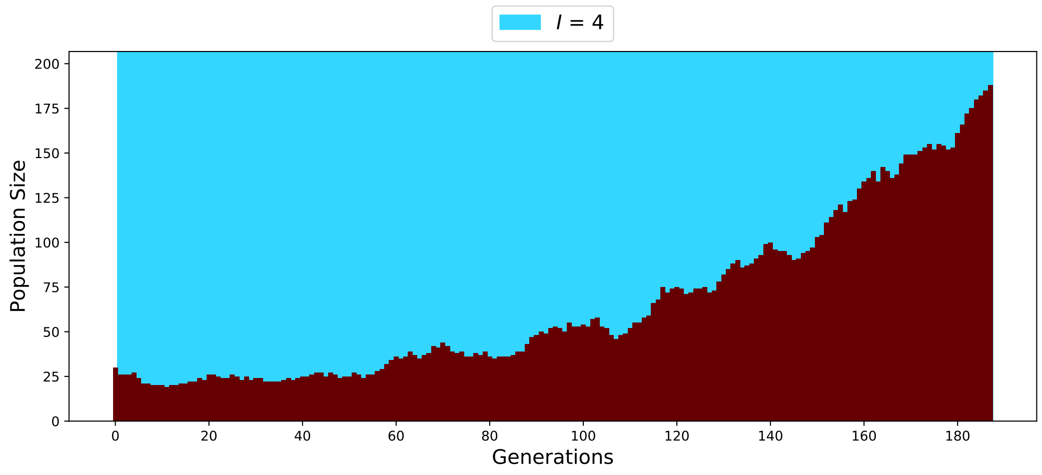


Figure 4.7: A population of cells with $\mathbf{D} = 7$, initially of size 30, grown in a medium containing a drug of strength 4, with $\beta = 0$, $\nu = 1$ and $\tau = 0.4$ for 187 generations.

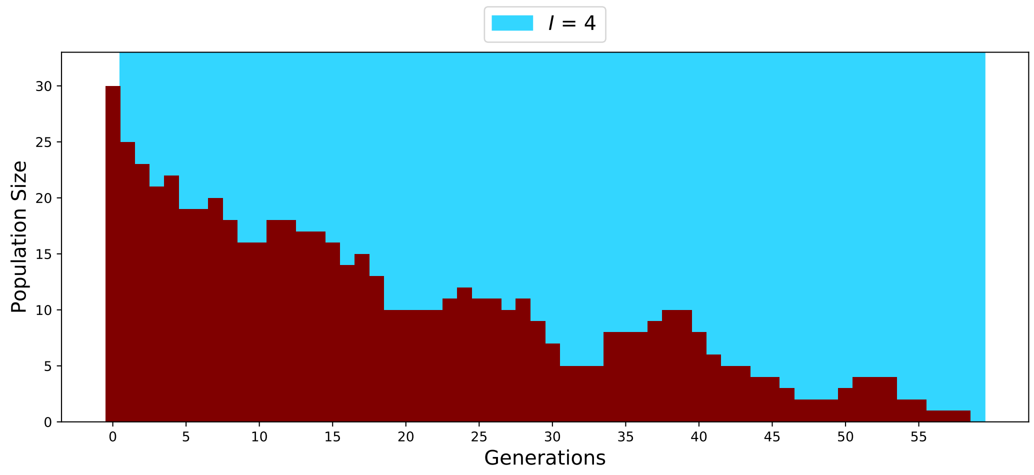


Figure 4.8: A population of cells with $\mathbf{D} = 6$, initially of size 30, dieing out in a drug strength 4 environment, with $\beta = 0$, $\nu = 1$ and $\tau = 0.4$ over 58 generations.

We now have an understanding of the answer to question 2 in the special instance where $\beta = 0$, but what about the more general case of $0 \leq \beta \leq 1$? For further exploration we must redirect our attention to inequality (4.5) with the aim of prying out the implicitly defined range of $\mathbf{D}$ values. In pursuit of this aim we can find the values of $\mathbf{D}$ which satisfy,

$$(1-\beta)e^{-\tau \mathbf{D}} - e^{\nu(I-\mathbf{D})} - \beta = 0$$

for various I. This will produce a line on the $\mathbf{D}$ - I plane. Then, our desired range can be found on one side of the line. As is usually the case, a picture paints a thousand words:

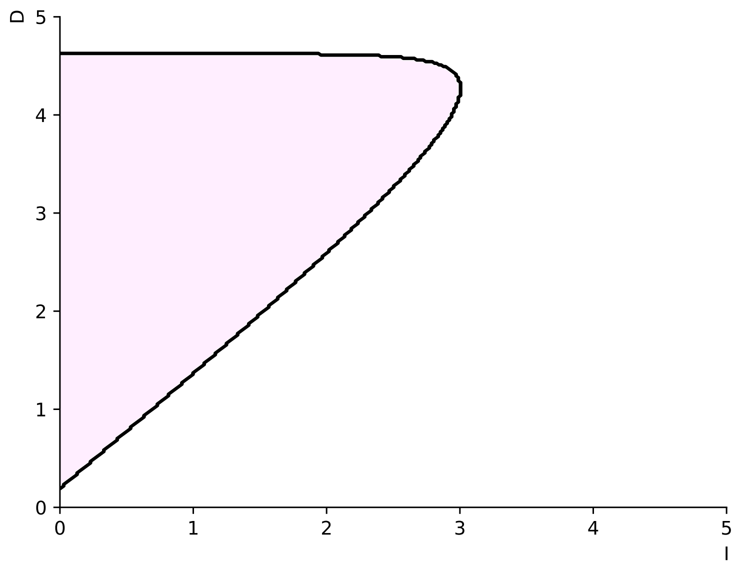


Figure 4.9: A plot of inequality (4.5) produced numerically with sympy.

Figure 4.9, plotted with the sympy Python package displays a pink shaded region in which I, and $\mathbf{D}$ values satisfy inequality (4.5). In an environment of a particular drug strength I, cells will be able to propagate without condemnation of ultimate extinction, if and only if their resistance level $\mathbf{D}$ sits <u>within</u> the shaded region at I. Figure 4.10 offers a nice visualisation of this.

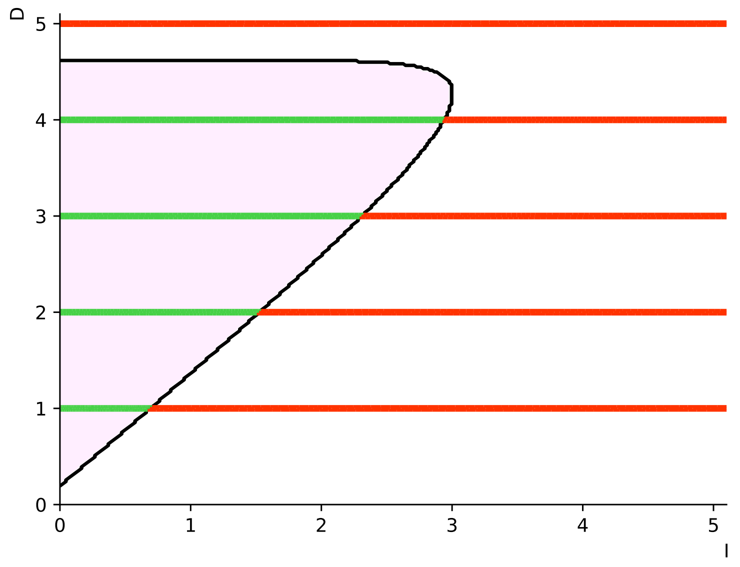


Figure 4.10

In it, green line segments show the range of drug strengths in the presence of which, particular levels of resistance can prosper; $\mathbf{D}$, I combinations displayed in red represent areas of certain cell elimination. Using this plot, we can see, for the case where $\beta = 0.2$, $\nu = 3$, and $\tau = 0.3$, that only cells of $\mathbf{D} = 2$, 3, and 4 are able to survive in a drug environment of $I = 1$. **Note:** We have considered I to be a positive integer, but, the theory would extend nicely to allow for non-integer drug values.

How does the plot of this inequality change when the parameters are varied? Let's see. The following plots show what happens to the black boundary of Figure 4.9 when β, τ, and ν are individually varied. In all of the plots, cyan lines represent higher parameter values and magenta lines correspond to those that are lower. Alterations in β and τ are seen below.

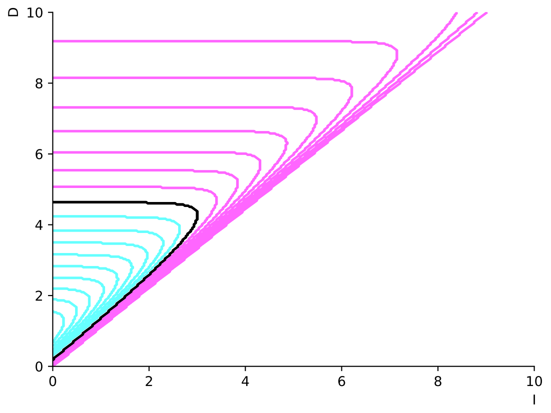


Figure 4.11: β varied from 0 to 1 in steps of 0.02.

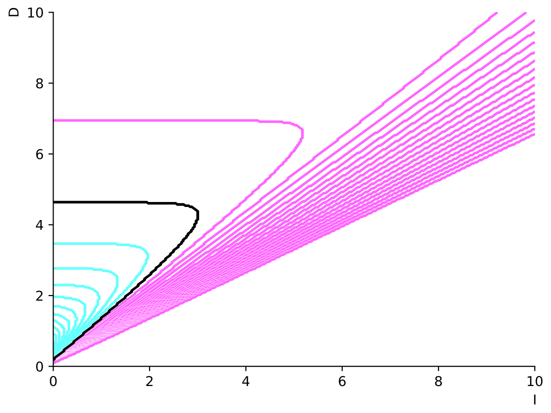


Figure 4.12: τ varied from 0.1 to 2 in steps of 0.1.

In both of these cases, it is seen that the upper limit of resistance is changed in a non-linear fashion. Such a change is not however seen when ν is varied:

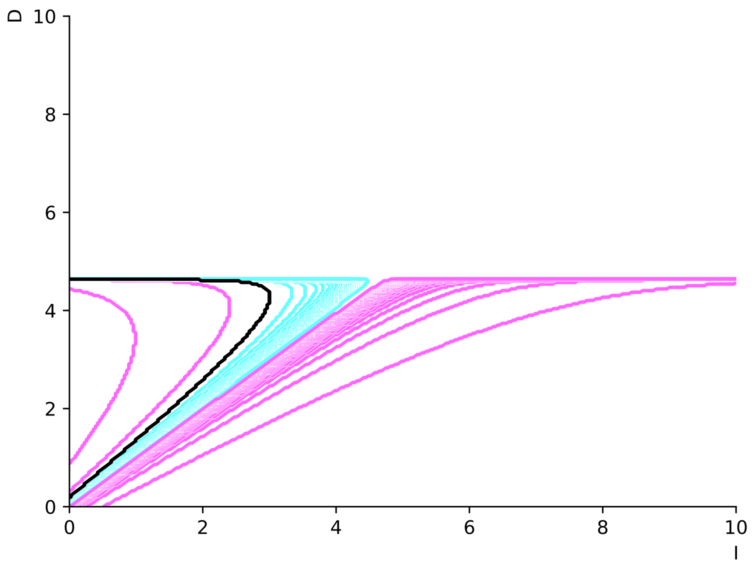


Figure 4.13: ν varied from 1 to 50 in steps of 1.

With high ν, the sharpness of the curve is increased. Also, when ν lowers to the critical value of $\nu = \tau$, the line flips over to the right rendering the standard physical interpretation incorrect.

4.2.3 Probability of Ultimate Extinction When it's Not Certain

Question 3 *If a population is not condemned to definite extinction, what is the likelihood that it will never die out?*

To answer this we can look again to section 2.3, specifically equation (2.2):

$$u_\infty = f(u_\infty).$$

It may seem like this question should be split into two parts, one for the drug case and one where no drug is present. In fact, the answer for each of the situations can be obtained in parallel; both sets of probabilities are of the form,

$$p_0 \qquad\qquad p_2 \qquad\qquad p_1 = 1 - (p_0 + p_2)$$

so we can find each of our desired solutions using:

$$\begin{aligned} u_\infty &= p_0 + (1 - (p_0 + p_2))u_\infty + p_2 u_\infty^2 \\ 0 &= p_0 - (p_0 + p_2)u_\infty + p_2 u_\infty^2. \end{aligned}$$

With an application of the quadratic formula, we collect,

$$u_\infty = \frac{p_0}{p_2} \text{ or } 1$$

Meaning, in both the drug and no drug case, when ultimate extinction is <u>not</u> guaranteed, its likelihood is $\frac{p_0}{p_2}$. This probability becomes relevant when inequalities (4.3) or (4.5) from questions 1 or 2 are satisfied. Which, as has now been made clear, falls in line with the satisfaction of,

$$\begin{aligned} \frac{p_0}{p_2} &< 1 \\ p_0 &< p_2. \end{aligned}$$

Summarising after some algebraic manipulation, we have:

$$\boxed{u_\infty^{(nd)} = \begin{cases} \dfrac{\beta}{1-\beta} e^{\tau \mathbf{D}}, & p_0^{(nd)} < p_2^{(nd)} \\ 1, & p_0^{(nd)} \geq p_2^{(nd)} \end{cases}} \qquad \boxed{u_\infty^{(d)} = \begin{cases} \dfrac{e^{\tau \mathbf{D}}}{1-\beta} \left(e^{\nu\phi} + \beta\right), & p_0^{(d)} < p_2^{(d)} \\ 1, & p_0^{(d)} \geq p_2^{(d)} \end{cases}}$$

4.3 Simulating the Dynamic Model

To gain a better feel for how the dynamic model behaves I have coded it into a Python simulation (See appendix 8.C for the fundamentals). In it, you can advance generations with prescribed strength of drug turned on or off. Here is an example of its results:

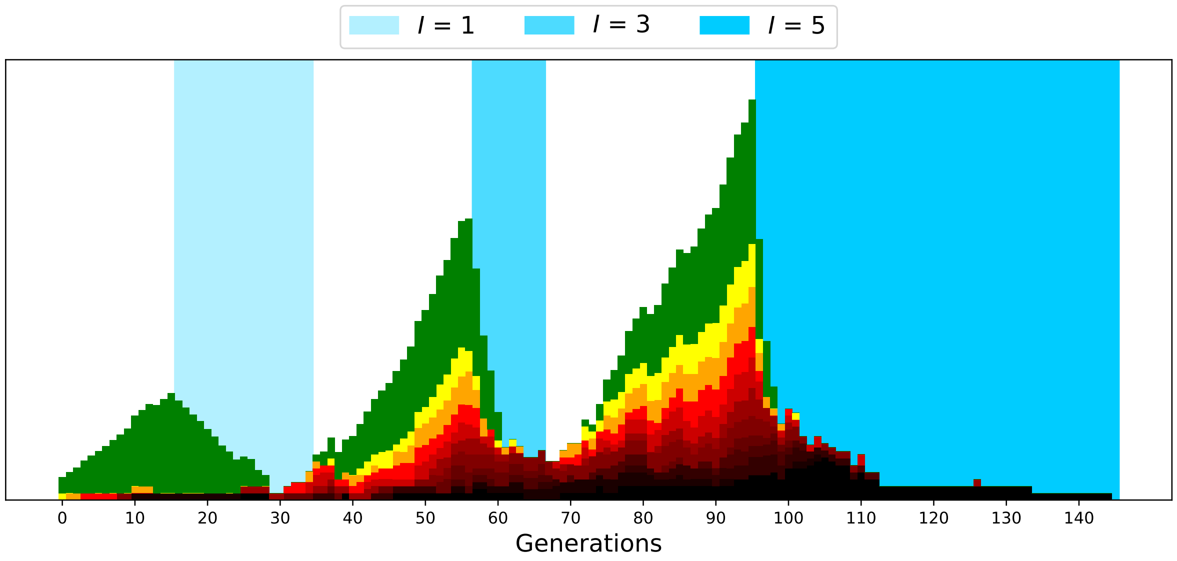


Figure 4.14: Population initialy containing four cells with $\mathbf{D} = 0$ and one with $\mathbf{D} = 1$, propagated under the rules of the dynamic model for 144 generations. Drug strengths 1, 3, and 5 are applied at various times shown in blue. Parameters were set as follows: $[a,\ \alpha,\ \beta,\ \tau,\ \nu] = [0.9,\ 0.5,\ 0.1,\ 0.3,\ 1]$.

Colours here are identical to those in the basic model simulation; green for $\mathbf{D} = 0$ and so on. You may wonder why there are no y axis markings. This is because of the specific plotting style used; bars shown are not linearly scaled based on their respective population sizes, each coloured section of every bar is sized based on the log of the population count for that resistance type. So the hight of the bars is actually a sum of loged values. Instead of for an acurate measure of population counts, this plotting style is chosen to give an idea of how the prevelence of different resistance levels changes with time and drug applications.

Figure 4.14 is a nice ilustration of how drug use can foster a increasing proportion of resistant cells. After drug strength $I = 1$ is removed, the ratio of resistant to non resistant cells quickly increases. And the story only gets worse after the next application of $I = 3$.

Chapter 5

Continuous Time Mathematical Introduction

Until this point the report has only represented the world in discrete steps of time. We will now evolve this limitation, stepping boldly into the realm of continuous time. In our discrete modelling, events were confined to occur at specified locations in time; every cell simultaneously died and gave birth, instantly, and all at once.

In a continuous time process, the moment at which an event occurs is not fixed. In a language familiar to modelling of cell divisions, the lifetime of a given cell is decided as a random variable. Specifically, one that is exponentially distributed.

5.1 Exponentially Distributed Random Variables

[13] Before getting into the modelling, we must first understand what an exponentially distributed random variable (EDRV) actually is. Suppose that the random variable is the time until a cell dies. To say that this is exponentially distributed means the following: If a cell is born at time 0, the probability that it has not died by time t is,

$$p(t) = e^{-\mu t},$$

where here, the rate μ is a death rate. Generally, this probability that *an event*, happening with rate μ, hasn't happened by time t looks like,

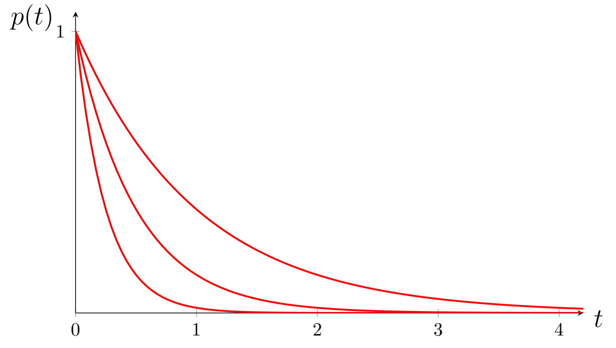


Figure 5.1: $p(t) = e^{-\mu t}$ plotted for $t > 0$, with $\mu = 1$, 2, and 4 from top to bottom respectively.

With a higher rate μ, the probability at a given time that an event hasn't happened yet is reduced; or, with a higher rate, events happen sooner.

5.1.1 Mean

Applying $\mathbb{E}(t) = \sum_t tp(t)$, we can calculate the expected time at which an event will occur. Since t is continuous however, the sum must instead be evaluated as an integral, becoming area under the curve:

$$\begin{aligned}\mathbb{E}(t) &= \int_0^\infty p(t)dt \\ &= \int_0^\infty e^{-\mu t}dt \\ &= \left[-\frac{1}{\mu}e^{-\mu t}\right]_0^\infty \\ &= \frac{1}{\mu}\end{aligned}$$

We expect an event with rate μ to take at least $t = \frac{1}{\mu}$ to happen. Some calculation,

$$p(\tfrac{1}{\mu}) = e^{-\mu\frac{1}{\mu}},$$

reveals that this expected outcome will occur with probability $e^{-1} = 0.367\ldots$, regardless of μ.

5.1.2 Multiple Events

So far, that which has been said focuses on a single event. In the case where events can happen more than once, instead of considering the time till *the event*, our working random variable becomes the time from the previous event to the next event. Graphically, it's like setting t back to 0 at each happening.

There are many examples of processes in which events can happen repeatedly. The time till the next tick on a Geiger counter is an exponentially distributed random variable; as is the interval between rain drops hitting a pond.

5.1.3 Variable Rates

Back at the start of summer I was at Glastonbury, a music festival. Here, the toilets were grouped in blocks, varying in size, but generally of about 20. In the areas of the festival with higher traffic, these would become saturated; the rate at which people arrived became greater than the rate at which people were using the toilets. Waiting in line, assessing the speed of each queue, I pondered that the time people take in the toilet, and thus, the time until the next space becomes free, is an exponentially distributed random variable.

Suppose the time one person takes on one toilet is an EDRV with rate τ. That is to say, the probability that at time t they will not be finished is,

$$e^{-\tau t}.$$

Ok well, what about if there are 20 occupied toilets? If we make the assumption that everyone acts at the same rate, τ (Although I think there is scope for including a man-woman rate distinction in the model.) the probability that no toilet is made available by time t is then:

$$\left(e^{-\tau t}\right)^{20}.$$

Which is equivalent to

$$e^{-(20\tau)t}.$$

When there are say, n objects that can experience an event with rate μ, the rate attached to the EDRV for events of the whole group is then, $n\mu$, denoted μ_n from now on.

Example There is a population of 100 cells, each cell has a death rate of $\mu = 0.01$. *What is the probability that after 2 minutes, <u>at least one</u> cell has died?*

With 100 cells, the population-wide death rate is $\mu_{100} = 1$. If we first find the likelihood that no cell dies in 2 minutes,

$$e^{-\mu_{100} \times 2}.$$

Our solution can be obtained from the complement, the probability that NOT(no cell dies),

$$1 - e^{-1 \times 2} = 0.864 \ldots .$$

Now, suppose that in the initial 2 minute interval, 3 cells have died. *What is the probability that after a further 2 minutes, no more cells die?*

Since the population has been reduced to 97, the whole-population death rate is now, $\mu_{97} = 0.97$. With this, the solution is,

$$e^{-0.97 \times 2} = 0.143 \ldots .$$

Events can cause a change in the rate at which following events occur.

5.2 The Gillespie Algorithm

Imagine a process in which 3 varieties of event, a, b, and c can occur. Where $\mu^{(a)}$, $\mu^{(b)}$, and $\mu^{(c)}$ are the respective rates at which each event type happens. Applying the Gillespie Algorithm, this class of process can be simulated, thereby generating numerical realisations.

I will now explain this Algorithm in the context of our example process. Its workings divide into three main steps:

1. Decide how long to wait before the next event.
2. Choose which event type will happen next.
3. After the decided time period, execute the chosen event. Then, if the stopping criteria is not met, go back to step 1.

It will seem intuitive that the decisions made in step 1 and 2 should depend on rates $\mu^{(a)}$, $\mu^{(b)}$, and $\mu^{(c)}$. But how?

Let's examine the second step first. When selecting an a, b, or c we wish that, for example, an event with a higher rate will be chosen more often than one with a lower rate. Putting it more precisely, we want the decision to be weighted in exact proportion to the rates $\mu^{(a)}$, $\mu^{(b)}$, and $\mu^{(c)}$. Normalising, we say that events a, b, and c will be chosen with respective probabilities,

$$\frac{\mu^{(a)}}{M}, \ \frac{\mu^{(b)}}{M}, \ \text{and} \ \frac{\mu^{(c)}}{M}, \ \text{where } M = \mu^{(a)} + \mu^{(b)} + \mu^{(c)}.$$

Shifting our focus to step 1. If the combination of rates M is large, we expect the chosen wait time, on average to be short. It should furthermore be exponentially distributed. Achieving these properties, the Gillespie Algorithm uses the definition of an EDRV,

$$p = e^{-\mu t} \quad \Longrightarrow \quad t = -\frac{ln(p)}{\mu}.$$

On the production of a uniformly distributed random number between 0 and 1, U, an appropriately distributed wait time is chosen,

$$t = -\frac{ln(U)}{\mu}.$$

Here some example results of a Gillespie simulation for cell division.

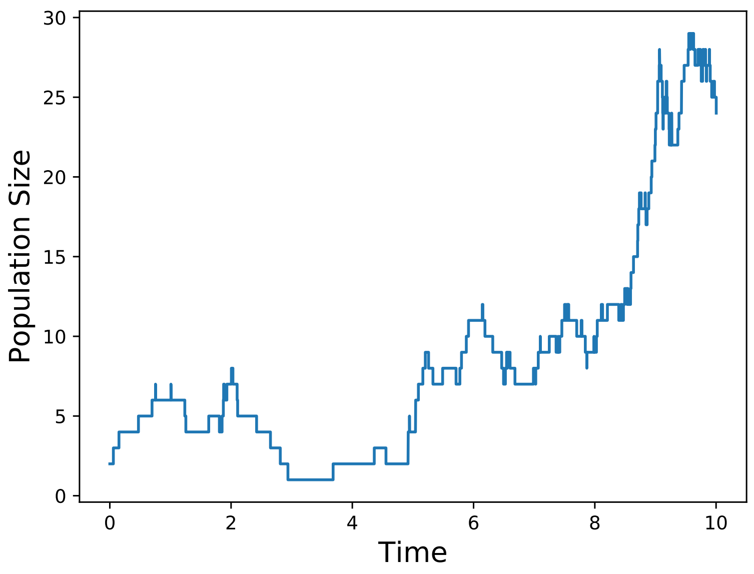


Figure 5.2: A population, initially of 2 cells, propagated for a period of 10 time units using the Gillespie Algorithm. The birth and death rate constants are 1.08 and 1 respectively.

5.2.1 Logistic Growth

When modelling cell divisions, an important point can be easily overlooked. Food is finite. Cells use food to reproduce, so when it starts to run out, a reduction in growth must follow. Also, the more cells there are eating through supplies, the faster they will run out. A given supply of food will support only a limited number of cells. When extending a model to account for this, we introduce something called a carrying capacity, K. When a population reaches a count of K, it can grow no further. With this, the birth rate of a population will not only increase with more cells, but also, when the limit K is approached, it will decrease. Specifically, the birth rate is of the form,

$$\mu_n^{(Logistic)} = \mu n \left(1 - \frac{n}{K}\right)$$

Using the Gillespie Algorithm, we can visualise this behaviour:

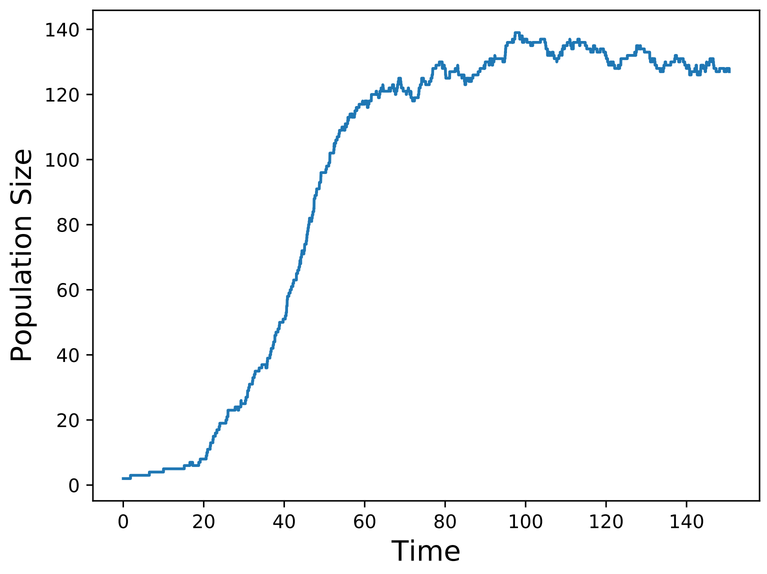


Figure 5.3: A population of 2 cells, propagated under logistic growth with $K = 150$. The Gillespie Algorithm was used for a period of 150 time units. The birth and death rate constants are 0.1 and 0.01 respectively.

Chapter 6

Continuous Time Modelling of Tumor Growth

The following model is a Moran process; there is a fixed number of cells occupying a 2 dimensional rectangular lattice. Initially, all but one of these are healthy; placed in the centre is a single cancerous cell.

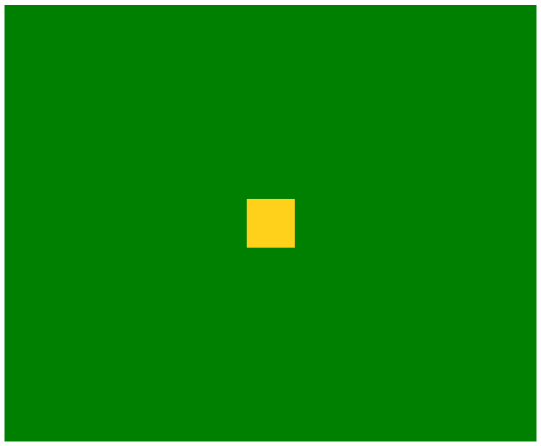


Figure 6.1: A single sensitive cancer cell surrounded by healthy cells, and centred on an 11 by 9 lattice.

6.1 Cell Division Dynamics

There are multiple types of cancer cell in the model, but to get traction on your understanding, we will focus for now on the most basic. These are called, sensitive, labelled as such for their complete vulnerability to drugs.

When the continuous time process begins, the initial cell may divide. The time it takes to do so is an EDRV with a prescribed one-cell birth rate. Say at time $t = 5$ it splits.

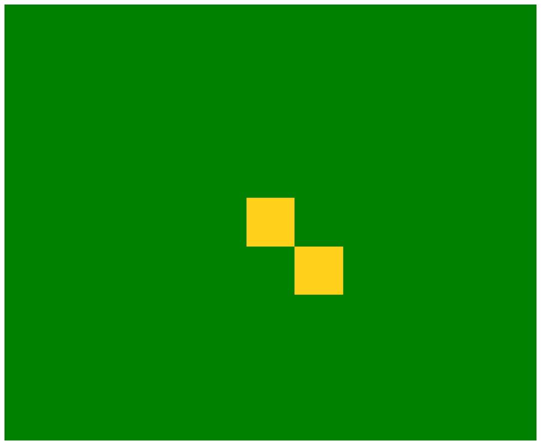


Figure 6.2: Two sensitive cells encapsulated by healthy cells in an 11 by 9 lattice, .

There are now two cells on the lattice. With this, you might expect that the rate used to decide when the next cell will be born is double that which it was originally. Furthermore, it seems intuitive that with n sensitive cells on the lattice, the population-wide birth rate should be n times the one-cell birth rate. This is, importantly, not the case. The reason is to do with a limited blood flow to the centre of a tumor, but more on that later. As of now, I would like to suggest a question:

When a cell divides, where do its offspring go?

Of the two daughters created at a division event, one will simply take the place of its parent. Her sister on the other hand must find new accommodation. Possible locations are both found in, and limited to the 3 by 3 doughnut of squares which surround the position of the late parent. Of these 8 possible places, the home hunting sister is only able to select one that is currently occupied by a healthy cell. From those that are, she chooses at random, and replaces the healthy inhabitant.

Those keen among you will spy a corollary of this division dynamic: Cancerous cells which are completely surrounded (That is to say, each of the 8 squares in their enclosing doughnut is occupied by another cancer cell), cannot divide. At all.

6.1.1 Blood Flow

To grow and divide, cells require access to blood and the vital nutrients that it provides. In healthy tissue, this lifeline is delivered by an intricate network of tubular vessels. Capillaries, veins, and arteries develop alongside cells, permeating their structure, ensuring a reliable supply.

When cancer cells divide and grow, they form ball like clusters. Inside these tumors, the same diligent construction and distribution of blood vessels is not carried out. As a result, towards the centre of a tumour, cancerous cells receive less and less access to blood flow. Two key results are presents in the case where a cell lacks blood flow:

- With less nutrients, its growth rate is reduced. Leading to death if the deficiency is sustained for sufficient time.
- When a drug is applied to the blood stream, its effect on the cell will be reduced.

To account for both of these dynamics, the model assigns a number to each living cell which is indicative of its current blood flow. The decision of this number is simple. Given a cancerous cell on the lattice, if we look at the 3 by 3 doughnut of square around it, its blood flow number is equal to the count of healthy cells in that doughnut. Maximally for example, the first cancer cell has a blood flow number of 8, because it is completely surrounded by healthy cells. Minimally on the other hand, a cancerous cell that is fully embed inside a tumour will have a blood flow number of 0. See Figure 6.4 for further examples.

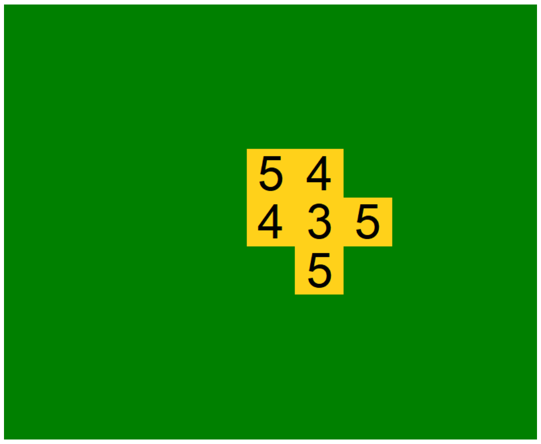


Figure 6.3: A 6 cell tumor in which each cell is labelled with its blood flow number.

Using this number, the individual birth and death rate for a given cell is adjusted. More specifically, each cell has a fixed base birth rate, and 4 potential base death rates (One for each drug combination: None, drug 1, drug 2, drugs 1 & 2.). Calculating the working, acted upon birth and death rates for a single cell, its base rates are simply multiplied by its blood flow number. Then there is a summation over particular cell types. For example, cells which are both sensitive and have a blood flow number of 5 will be grouped into a sub population. Say this sub population has 30 members, the birth rate for that particular group will be $30 \times 5 \times \mu$, where μ is the base birth rate for sensitive cells. As a whole, the tumor will be made up of multiple sub populations, each with varying birth and death rates. The Gillespie algorithm used takes all of these into account when deciding what to do next, and when to do it.

6.2 Cell Types

So far, only sensitive cells have been introduced. In addition to these, the model includes various others: Necrotic cells are those which have died due to a lack of blood supply. Resistant cells, of types 1, 2, and 12 suffer less in the presence of drugs. Finally, metastatic cells have an altered growth dynamic, allowing them to separate from a cluster, and form a new tumor.

6.2.1 Necrotic

When an active cancerous cell loses its last source of blood flow, achieving a blood flow number of 0, it becomes a candidate for a necrotic switch event. A transition from an active cancer cell into a dead **necrotic cell**. These events, like births and deaths have an attached rate.

When a necrotic switch event is decided to happen, the specific cell which is chosen depends on waiting time. Cells which have been necrotic candidates for the longest time are the ones which are chosen. If there are multiple cells which have been waiting both equally long, and the longest, one of them is selected at random. The reason for this choice of dynamic is that it causes the formation of a necrotic core.

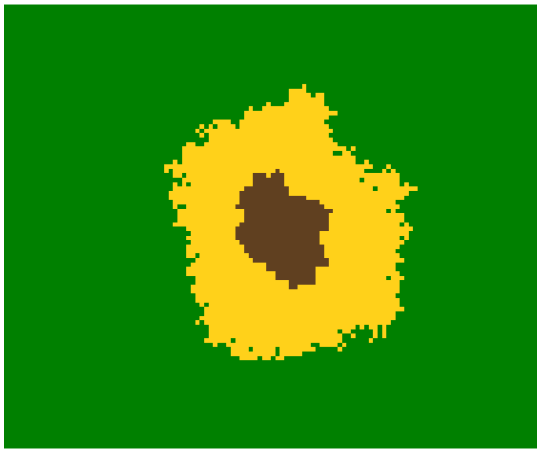


Figure 6.4: A tumor consisting of sensitive cells with a necrotic core. Situated in a 120 by 100 lattice.

As observed in reality. If instead, necrotic candidates were chosen completely are random, the outcome would be a tumor randomly dotted all over with necrotic cells.

6.2.2 Resistant

Introduced via mutation of sensitive cells, there are two initial types of resistant cell. resistance type 1, which can weather an application of drug 1; and type 2 which does the same for drug 2. From each of these cell types, mutation into a doubly resistant cell is also possible.

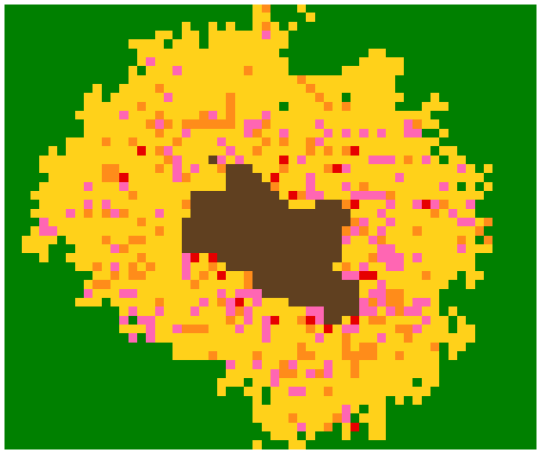


Figure 6.5: A tumor in which various resistance mutations have occurred.

In the chosen model parameters, the various drug application specific base death rates are as follows:

Drug Application	Base Death Rates			
	SN	R1	R2	R12
None	0	0	0	0
1	$\frac{10}{7}$	$\frac{1}{50}$	1	$\frac{1}{34}$
2	$\frac{10}{7}$	1	$\frac{1}{50}$	$\frac{1}{34}$
1 & 2	$\frac{20}{7}$	$\frac{51}{50}$	$\frac{51}{50}$	$\frac{2}{34}$

Singly resistant cells fair slightly better than sensitive cells in the drug they are not resistant to, and doubly resistant cells fair worse than singly resistant cells in the drug they are adapted to. When both drugs are present, the base death rates are defined as the sum of those from the individual drug usages.

Shown by the following images is the growth of a model tumor. In it there is a small rate of mutation from sensitive, to resistance 1 cells. Immediately after the first image was captured, drug 1 was applied, and left active for the remainder of the simulation.

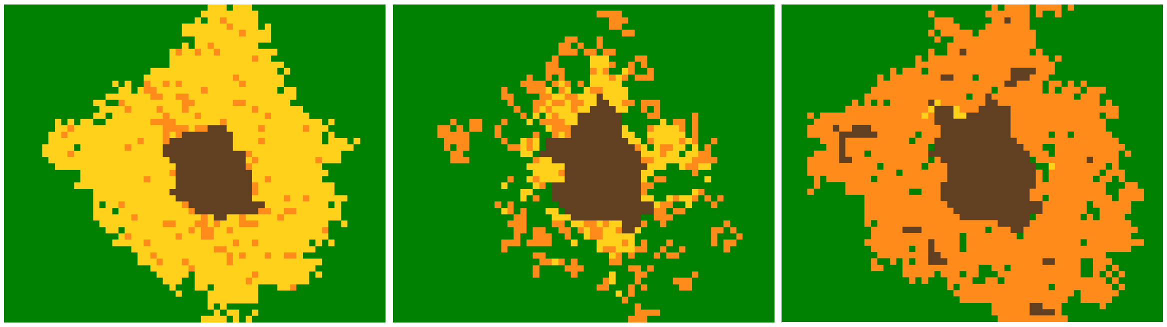


Figure 6.6: A partially resistant tumor undergoing the application of drug 1.

Seeing the above display, it may seem that resistant cells are very powerful; they're like sensitive cells but don't die as fast. In actuality, this view is incomplete. Yes, resistant cells are harder to kill, but, they also grow slower. The base birth rate for resistance 1 and 2 cells is $\frac{1}{3}$ that of sensitive cells, and for doubly resistant cells, this reduction multiple is $\frac{1}{5}$.

6.2.3 Metastatic

When it first develops, cancer is often confined to a single tumor. If the cancer undergoes metastasis however, caused by a further genetic mutation, it may spread around the body, developing into multiple tumors. To accomplish this, it will send metastatic cells into the blood stream. Most, not protected by their parent tumor, will perish. Some however, may find a new home, and begin a fresh colony.

The presented model represents metastasis in the following way. Sensitive cells are given a chance to mutate into a metastatic cell. Although metastatic cells are very similar to sensitive cells; they share both birth, and death rates, there is a key difference. At a division event, these cells have special dynamics: Firstly, both progeny of a metastatic cell are sensitive cells, the mutation is lost after division. Also, the daughter that moves to a new position tries to spread much further. Specifically, it will seek to occupy one of 12 pre-defined long-range locations. These are positioned in a broken circular fashion around the parent cell's position, at a radius of 15 squares.

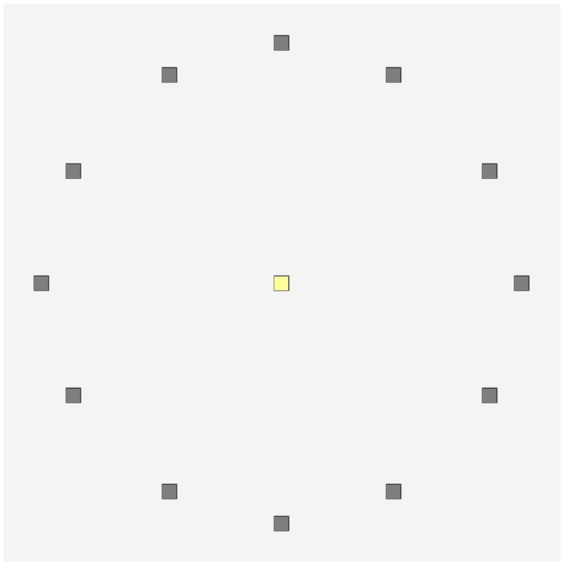


Figure 6.7: A representation of the primary birth locations that a dividing metastatic cell considers.

If none of these positions are available, containing a healthy cell, the daughter will simply move to one of the 8 cells in the doughnut centred on the parents old position.

Fair criticism of this chosen dynamic will include the fact that it is overly regular and very short range. Real metastatic cell can travel far and wide all over the body, so 15 cells is pretty close. Despite not giving a full representation of reality, the dynamic achieves two important goals. Most significantly, it gives the desired advantage to a tumor. By branching out into new clusters, the cancer can access a higher overall blood flow to its cells, accelerating growth. Secondly, the method is computationally cheap. One of the biggest challenges in programming this model was to make it run in real time. Along with this feature choice, another critical speed advantage was achieved with a heavy focus on special numpy array indexing, as opposed to loops.

Below is a simulation example in which the main tumor has produced some metastatic cells. From these, three new tumors have formed.

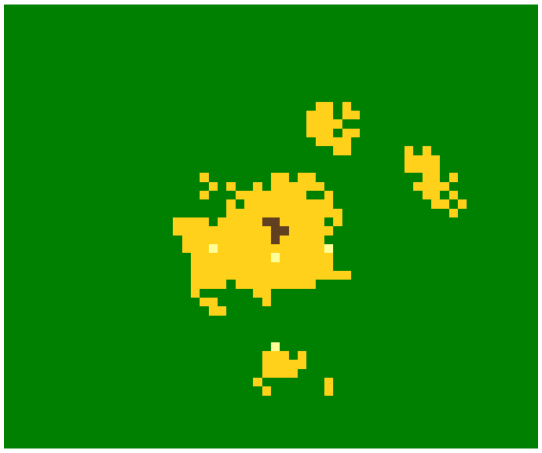


Figure 6.8

As mentioned, this spreading out offers the cancer with an increased overall blood supply. Although, this is good for growth, it can also leave cells more susceptible to drugs.

6.3 Rates and the Gillespie Algorithm

With a total of 175 possible events, this model is pretty complicated. I will now summarise these events and explain the process by which the Gillespie algorithm was implemented.

5 of the 7 cell types are able to divide and die; necrotic and healthy cells are non-active. With 8 possible blood flow numbers, each active cell type has 8 birth rates, resulting in $5 \times 8 = 40$ birth rates in total. Death rates can only be non-zero when at least one drug is present. With 3 possible drug application combinations, 5 active cell types, and 8 blood flow numbers, there are $3 \times 5 \times 8 = 120$ possible death rates. In addition to births and deaths, there can be mutations and necrotic switch events. Necro-events add 5 rates to the total, one for each cell type, since only cells with a blood flow of 0 can become necrotic. Combining all of the so far stated rates with the 10 mutation rates,

	Forward					Back				
From	SN	SN	R1	R2	SN	R1	R2	R12	R12	ME
To	R1	R2	R12	R12	ME	SN	SN	R1	R2	SN

Figure 6.9: Representation of all 10 mutation types. 5 forward mutations and 5 back.

the model finds itself with $40 + 120 + 5 + 10 = 175$ rates relating to the same number of individually defined events. This magnitude posed a great coding challenge. Especially as the rates need to be updated according to current cell counts after every change to the lattice.

To wrap the Gillespie algorithm around this multiplicity, I broke it down into parts. Instead of simply making a single choice of which event will happen next, the program makes a chain of decisions. It can be visualised as a branching tree. Each fork is a decision to make, and the leaves at the end are the events. Firstly, the program chooses what type of event will happen, based on a probability set weighted with the summed rates of each sector. Suppose that it decides to complete a birth event. From there, it will make a choice on what species of cell to divide, again with probabilities weighted by summed rates from each sub group. If at that point, it chooses to birth a sensitive cell, its final decision will be of what blood flow number should that sensitive cell be.

When a Necrotic switch event is chosen to occur, the following decision tree is smaller since only one blood flow level, 0 can be selected.

Code

The full Python script that is used to simulate this model can be found in appendix 8.D.

Chapter 7

Strategic Evolution with the Genetic Algorithm

Ok, we have this complex model for tumor growth controlled by a whole bunch of parameters. Most of these parameters are fixed; birth death and necrotic switch event base rates are all pre-defined. What about mutation rates? I think it is fair to say that a tumor's set of mutation rates can be called its strategy. The others, are the rules of the game. These are how it plays. Maybe the cancer favours a rapid growth but high vulnerability tactic with significant metastatic mutations. On the other hand, it might prefer a slow but protected growth strategy, implementing high resistance levels. The best tumor growth strategy will depend on the type of, and the timing by which drugs are applied.

Hold on a minute. How do you define the goodness of, and therefore the *best* strategy? Theoretically, you could do this how ever you want. You could say that the strategy which causes the tumor to die first is the best strategy. Evolutionarily however, this makes little sense. Generally, If a strategy causes an individual to die faster, there will be less of that individual's type about. Eventually, the genome holding the strategy will die out. Nature has pre-defined the metric; the goodness of a strategy is measured by the likelihood that it gives the individual of surviving for the longest, and spreading its genome the most. Of coarse, we are talking about a model, and in a model the creator has freedom to do anything, but since we are modelling a natural phenomenon, let's stick with nature's way. The best strategy is the one by which a tumor will grow the most in a given time period. The metric will be laid out more rigorously in a little while, but for the time being this will do.

As of now, we can pose an interesting question:

> For an arbitrary, but well defined drug application regime, what is a tumors best mutation strarergy?

Here we have a 10 dimensional optimisation problem; we wish to maximise the goodness of the 10 mutation rate set, given a particular drug routine. This kind of multidimensional optimisation task can prove intractable for standard mathematical analysis. Instead, we shall approach it with a more suited tool from computer science.

7.1 The Genetic Algorithm

[10] In an incredibly fitting fashion, the genetic algorithm can be used to explore a large solution space with principals inspired heavily by natural selection. The procedure works in the following way:

1. An initial 'population' of solutions is generated at random.
2. The goodness, or fitness as it will now be called, of each solution is assessed, and numerically graded.
3. High fitness individuals are selected to create the next generation of solutions.

4. The ‘genetic information’ of these individuals is combined and modified, by way of crossover, and mutation. A new population is born.
5. If stopping criteria is not met, the new generation is is returned to step 2 .

In our case, the genetic information is the set of mutation rates. When two parent solutions are combined in crossover, they are split in identical locations, then, a swapping of respective solution parts is carried out.

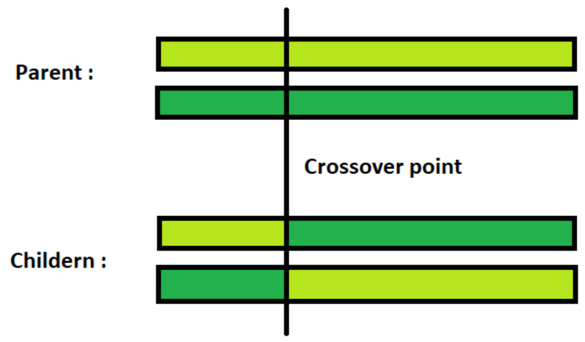


Figure 7.1: Illustration of genetic crossover [11].

This crossover point sits directly on the boundary between forward and backward rates. After crossover, mutations in the genetic information can occur. At this point it is very important to notice the difference between a mutation of a rate in the genetic algorithm, and a mutation of a cell in the model. To enhance the mutation dynamic, the string of 10 numbers is converted into a binary string, in which each of the numbers is represented by a 5 digit, Gray coded, binary number. Gray coding is used, as it was by [12] because it causes bit-flip mutations, generally, to not change the number by much.

7.1.1 Specific Definition of the Fitness Function Used

A suitable fitness function is critical to success of a genetic algorithm implementation. In this usage, the fitness function uses several components. The run time, R of a simulation, a pre-specified maximum run time, $R_{\max}$, and the proportion of cells which are healthy at the completion of a run. With these, the fitness function is split into 3 parts, laid out as follows:

- 🔵 If the cancer dies out in a test, its fitness will the the run time which it survived for,

$$Fitness = R.$$

- 🟢 If it survives until the maximum time but does not take over, that is to say, $h > 0$ healthy cells are left at the end, its fitness will be,

$$Fitness = 2R_{\max} - \frac{h}{N} R_{\max}.$$

  Where N is the total number of cells in the lattice. Notice, the minimum value of this is exactly the maximum value of the previous case. It also follows that the maximum value of this meets the minimum value of the next case.

- 🔴 When the tumor does in fact take over before the maximum run time, its fitness will be negatively proportional to the time it took,

$$Fitness = 3R_{\max} - R.$$

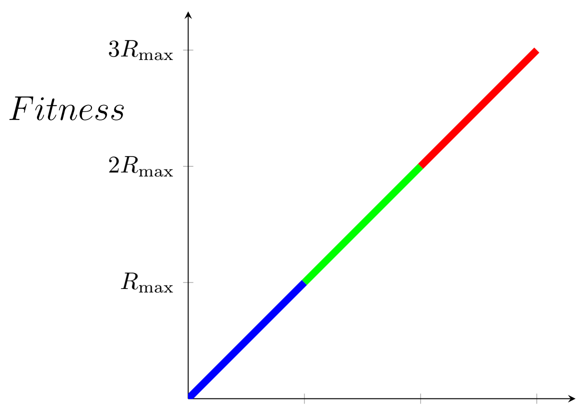


Figure 7.2: A representation of how the three categories of fitness definition fit together.

7.1.2 Challenges Faced when Implementing the Genetic Algorithm

When evolving tumor growth strategies, there were a couple of factors that posed significant challenge. Both of these were related to fitness evaluation.

Firstly, the model is, at its core, a random process. A rate set that performs well in one test may get its tumor wiped out in the next. The issue is especially prevalent on the fringes of survival. In chapter 2, we saw that when a population growing with $m = 1$, is expected not to grow, or shrink, its simulated realisations can vary wildly. See the left plot in figure 2.3 for an example.

Randomness in the genetic algorithm can be positive sometimes; it can free the process from being stuck in non global maxima. There is a distinction to be made however. This benefit is only obtained when the randomness is introduced to the population formation process. It is not of any advantage, in fact it is of detriment when the algorithm cannot make accurate distinctions in the quality of solutions. For this reason, I have opted to combat the random behaviour stated. To assess the fitness of a single rate-set, 20 repeats of the evaluation are completed, then the average is taken. Less repeats were tried initially, but uncertainty was still an issue. Actually, even at 20 evaluations, volatility was present, but progress could be made.

As well as randomness, another major difficulty was speed. Or lack of it, to be precise. Simply put, the simulation, and therefore fitness evaluations, are expensive. They take a long time to complete. In the genetic algorithm, it is desirable to have many candidate solutions in a population, north of 50 is good. But when each test takes 1 minute, and has to be done 20 times for a single individual, and there are 50 individuals in a generation, and, upwards of 40 generations need to be advanced, the maths doesn't work. $1 \times 20 \times 50 \times 40 = 40000$ minutes, or about 27.7 days.

Working past the issue, a few measures were taken to reduce evaluation run times:

- Only 10 candidate solutions were presented in a given population.
- The evaluation of Individuals in a generation was parallelized.
- Maximum run times were restricted.
- The lattice was kept small at 60 by 50 squares.

These decisions made it possible to run the required tests at sufficiently reduced time scales to achieve meaningful genetic algorithm results.

7.2 Genetic Algorithm Results

7.2.1 Chosen Drug Application Regimes

In the experiments, three treatment plans were explored:

1. No drugs at all.
2. Periodic application of drug 1. Alternations occur at intervals of 0.2 run-time units.
3. Both drugs 1 & 2 fixed on.

To give the tumors a fighting chance, plans 2, and 3 were only initiated after a threshold number of 25 cancerous cells was reached. We will now analyse, in reverse order, the results from genetic algorithm trainings with these drug application plans.

7.2.2 Both Drugs Fixed On

Taking only about 14 generations to converge, the strategy to combat drug plan 3 was the fastest to train.

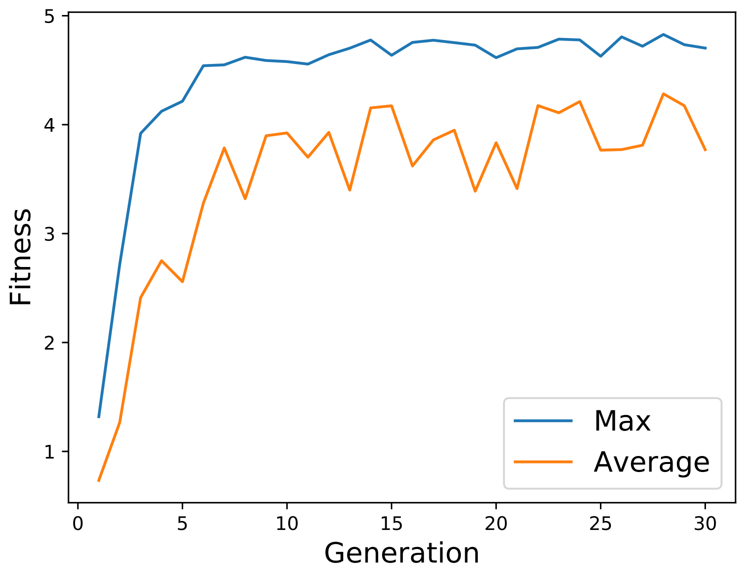


Figure 7.3: Maximum and average fitness values of each generation over training period.

Here is the final muation rate set:

	Forward					Back				
From	SN	SN	R1	R2	SN	R1	R2	R12	R12	ME
To	R1	R2	R12	R12	ME	SN	SN	R1	R2	SN
Rates	17	22	13	26	26	3	14	0	0	3

Figure 7.4: Winning set of mutation rates. Obtained after 30 generations of training with drug plan 3.

The evolved technique is, as one might expect, heavily focused on double resistance mutations. Less predictably on the other hand, is a partial preference for metastatic mutations. Why might this be? Metastatic cells increase a cancer's susceptibility to drugs. Well, as is seen in Figure 7.5, this slight metastatic preference allows the cancer to quickly spread out, before the 25 cell threshold is reached. From there, multiple doubly resistant tumors can form, increasing the potential for growth.

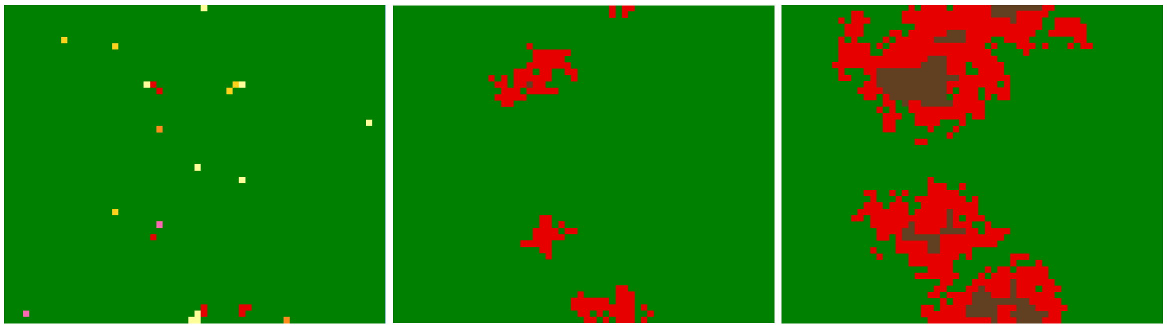


Figure 7.5: Example simulation of plan 3 with mutation rates set to those that won.

Another notable feature of the obtained solution is that there is no possible back mutation from doubly resistant cells. If there was, I assume that it would be a disadvantage when the drugs are on.

7.2.3 Drug 1 Pulsing

A key distinction between the previous fitness graph and the one shown here in Figure 7.6 is that this one is much more spiky. Does that mean more randomness was present in these plan 2 experiments? Not necessarily, a plausible explanation is the much smaller range of fitness values here. Small fluctuations seem bigger.

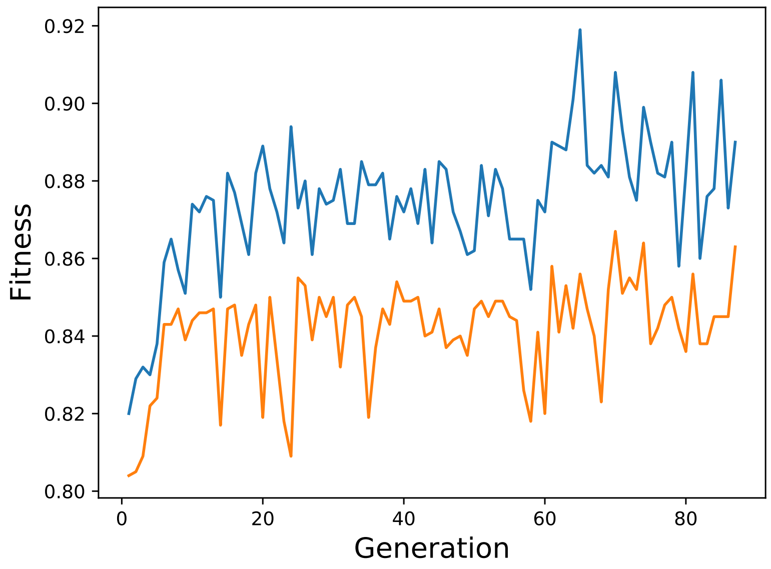


Figure 7.6: Maximum and average fitness values of each generation over training period.

Despite a higher volatility, the genetic algorithm delivers a rate set which does make strategic sense.

	Forward					Back				
From	SN	SN	R1	R2	SN	R1	R2	R12	R12	ME
To	R1	R2	R12	R12	ME	SN	SN	R1	R2	SN
Rates	16	0	1	13	31	2	16	20	6	10

Figure 7.7: Winning set of mutation rates. Obtained after 87 generations of training with drug plan 2.

Faced with intermittent drug 1 exposures, the cancer learns to direct some mutations towards resistant 1 cells, and others in the direction of metastatic cells. Growing rapidly in drug free times while quickly locking in progress with resistance.

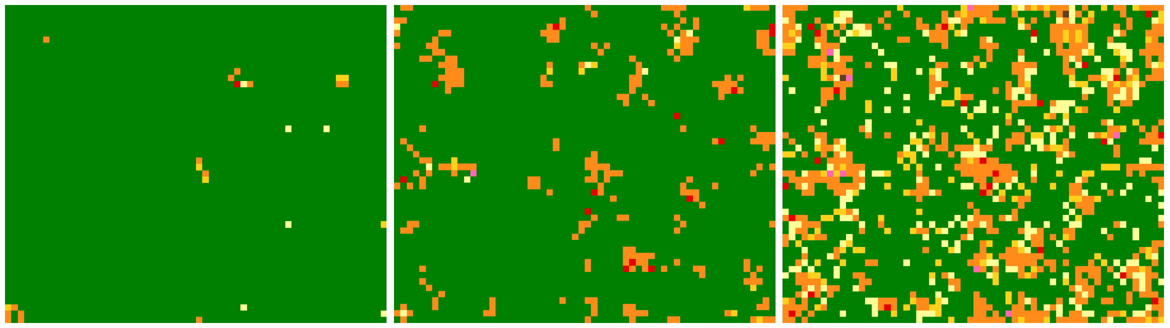


Figure 7.8: Example simulation of plan 2 with mutation rates set to those that won.

7.2.4 No Drug

When training a plan 1 strategy, something unexpected happened: Evolution reversed; between generations 22 and 48, fitness values were hovering around the 0.35 mark. Then, all of a sudden, the level fell to around the region of 0.3375, a drop of 35%.

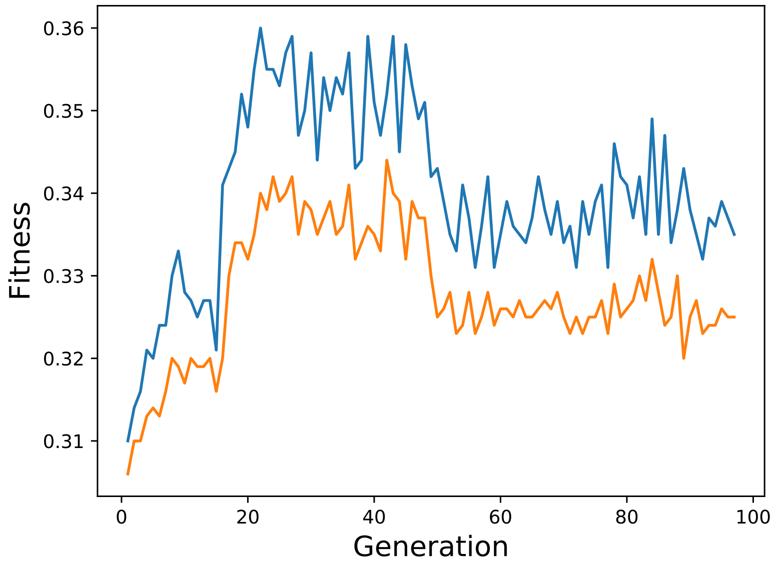
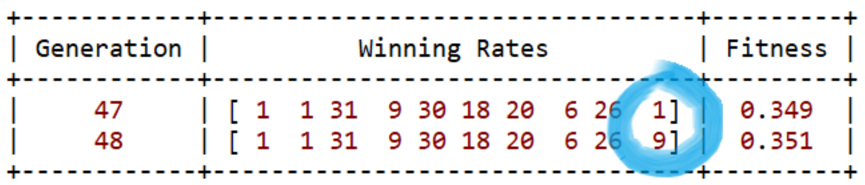


Figure 7.9: Maximum and average fitness values of each generation over training period.

Looking into the genetic algorithm log files is can be seen that around the time of the crash, a nonsensical rate change occurred. From generations 47 to 48 the ME→SN back mutation rate went from 1, to 9.



```
+------------+-----------------------------------+----------+
| Generation |           Winning Rates           | Fitness  |
+------------+-----------------------------------+----------+
|     47     | [ 1  1 31  9 30 18 20  6 26  1]   |  0.349   |
|     48     | [ 1  1 31  9 30 18 20  6 26  9]   |  0.351   |
+------------+-----------------------------------+----------+
```

Resulting in less metastatic cells, which is clearly a disadvantage. Why did this happen? Interference from Random fluctuations. How do we know? Well suppose we re-test the rate-set of generation 48. If fluctuations aren't to blame, the re-evaluated fitness should be close to its previously tested value of 0.351. Averaging over 100 evaluations instead of 20, to gain a more accurate result, the re-assessed fitness value for generation 48's rate-set came in at 0.327 . . . .

Ignoring the issue, the following rate-set was taken from generation 43, before the crash.

	Forward					Back				
From	SN	SN	R1	R2	SN	R1	R2	R12	R12	ME
To	R1	R2	R12	R12	ME	SN	SN	R1	R2	SN
Rates	0	0	31	9	30	18	20	9	5	1

Figure 7.10: Winning set of mutation rates. Obtained after 43 generations of training with drug plan 1.

It shows a heavy focus on speed increasing metastatic mutations, and no emphasis at all on growth reducing resistance mutations. Interestingly, with a lack of mutation into resistant cells, all other resistance based mutation rates become meaningless. Thus, any change to these rates can be labelled a neutral mutation.

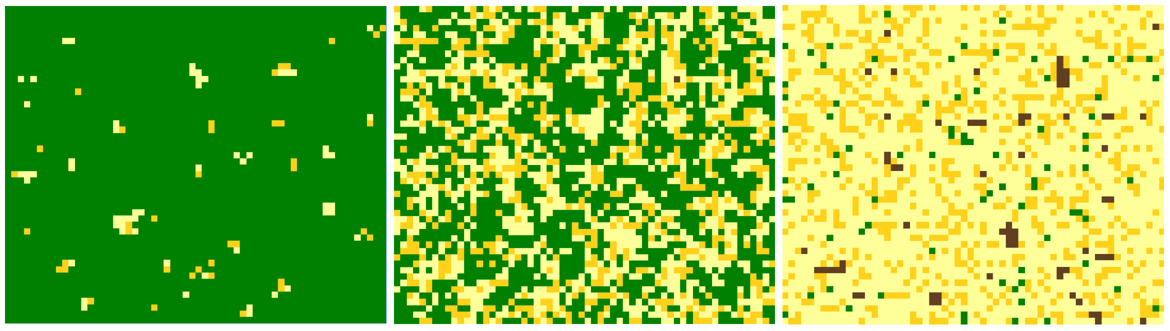


Figure 7.11: Example simulation of plan 1 with mutation rates set to those that won.

END

Chapter 8

Appendix

8.A Resistance Existence

Preamble, Variables, and Initial Polynomials

```
import matplotlib.pyplot as plt
from numpy.random import choice
import numpy as np
import json

colours = ['#f2000d', '#e6001a', '#d90026',
           '#cc0033', '#bf0040', '#b2004c',
           '#a60059', '#990066', '#8c0073',
           '#800080', '#73008c', '#660099',
           '#5900a6', '#4d00b2', '#4000bf',
           '#3300cc', '#2600d9', '#1900e6']

gens = 5
max_I = 15
a = 0.94
al = 1
nSim = 5000

F = {'{"x0": 1, "x1": 1}': 1-a, '{"x0": 1, "x2": 1}': a*al,    # F_1
     '{"x1": 2}': a*(1-al)}

g = {'{"u": 1, "v": 0}': (1-a)/2, '{"u": 0, "v": 1}': (1-a)/2, # g_1
     '{"u": 1, "v": 1}': a*(1-al),'{"u": 2, "v": 0}': a*al/2,
     '{"u": 0, "v": 2}': a*al/2}

f = np.poly1d([a*al/2, (1-a)/2+a*(1-al), a*al/2+(1-a)/2])      # ind-approx f_1
```

8.A.1 Exact Analysis

Functions to Manipulate Polynomials

```
def p1_times_p2(p1, p2):
   '''Multiplies two polynomials.'''
   new_p = {}          # new polynomial
   for St1 in p1:      # string term 1 (variables)
      for St2 in p2:   # string term 2
         t1, t2 = json.loads(St1), json.loads(St2)
         coeff = p1[St1]*p2[St2]  # multiplies coefficients
```

```
        for var in t2:              # multiplies variables
            if var in t1:
                t1[var] += t2[var]
            else:
                t1[var] = t2[var]
        St = json.dumps(t1, sort_keys=True) # converts dict to string

        if St in new_p:             # injects a new term, combining like terms
            new_p[St] += coeff
        else:
            new_p[St] = coeff

    return(new_p)


def polypower(poly, k):
    '''Raises a polynomial to a power k.'''
    i = 1
    poly_k = poly
    while i < k:
        poly_k = p1_times_p2(poly_k, poly)
        i += 1

    return(poly_k)


def gi_to_Fi(gi):
    '''Converts a gi into a Fi.'''
    Fi = {}
    for St1 in gi:
        t1 = json.loads(St1)
        t2 = {}             # converted term (variables)
        for varKey in t1: # fills converted term
            var = 'x'+str(t1[varKey])
            if var in t2:
                t2[var] += 1
            else:
                t2[var] = 1

        St2 = json.dumps(t2, sort_keys=True)
        if St2 in Fi:      # fills Gn combining like terms
            Fi[St2] += gi[St1]
        else:
            Fi[St2] = gi[St1]

    return(Fi)


def get_Fi(i):
    '''Produces Gn for a given n.'''
    if i == 0:
        return({'{"x0": 2}': 1}) # Fi = x0^2 when i = 0
    gi = polypower(g, i)

    return(gi_to_Fi(gi))


def iterate_F(F1):
    '''Iterates F_(n) into F_(n+1).'''
    F2 = {}    # F_n+1
    for St1 in F1:
        t1 = json.loads(St1) # single term_in F_n
        tp = {} # polynomial that will replace the variables in_term t1
        for varKey in t1:
            i = int(varKey[1:])
            Fi = get_Fi(i)
            Fik = polypower(Fi, t1[varKey])
            if tp == {}: # multiplies Fi polynomials
                tp = Fik  # frist part of the term begins a new position in_tp
            else:
                tp = p1_times_p2(tp, Fik) # sucsessive parts multiply what exists

        for St2 in tp: # fills new F with terms and_apropriate coefficients
            if St2 in F2:
                F2[St2] += tp[St2]*F1[St1] # like terms are combined
            else:
                F2[St2] = tp[St2]*F1[St1]

    return(F2)


def save_F(Fn, n):
    '''Saves Fn to a text file so it doesn't
     need to be computed more than once.'''
    file = open('F'+str(n)+'a'+str(a)+
                'al'+str(al)+'.txt','w')
    file.write(json.dumps(Fn))
    file.close()


def produce_Fn(n):
    '''Produces Fn for a given n'''
    m = 1
    Fn = F
    save_F(Fn, 1)
    while m < n:
        Fn = iterate_F(Fn)
        null_terms = [] # eliminates trailing terms with coeffs of 0
        for term in Fn:
            if Fn[term] == 0:
                null_terms.append(term)
        for term in null_terms:
            del Fn[term]

        m += 1
        save_F(Fn, m)

    return(Fn)
```

Resistance Calculations

```
def get_R_I(Fn, I):
    '''Finds the resistance existence probability,
     R_I for a given generation n, and drug strength I'''
    R_I = 0
    for St1 in Fn:
        t1 = json.loads(St1)
        for varKey in t1:
          n = int(varKey[1:])
          if n >= I:
             R_I += Fn[St1] # include the coefficient probability
             break

    return(R_I)


def fill_R():
    '''Fills a list, R with R_I for I=2 to max_I
    for generation 0 to gens. '''
    R = [[0] for i in range(max_I-1)] # 0 resistance probability at gen 0

    for n in range(1, gens+1):
        try:
          file = open('F'+str(n)+'a'+str(a)+
                'al'+str(al)+'.txt','r')
        except:
          produce_Fn(gens)
          file = open('F'+str(n)+'a'+str(a)+
                'al'+str(al)+'.txt','r')
        Fn = json.loads(file.readline())
        file.close()

        for i in range(max_I-1):
          I = i + 2
          R_I = get_R_I(Fn, I)
          R[i].append(R_I)
    file = open('gens'+str(gens)+'_maxI'+str(max_I)+
             '_a'+str(a)+'_al'+str(al)+'.txt','w')
    file.write(json.dumps(R))
    file.close()
```

8.A.2 Independent Approximation

Functions to get Probabilities

```
def fill_Ra_gen_n(Ra, n, pgf):
   '''Calculates resistance existence probabilities, R_I for gen n.'''
   qvals = list(pgf)
   qvals.reverse()

   for i in range(max_I-1):
      I = i + 2
      res_exist_prob = 1-(sum(qvals[0:I]))**(2**n)
      Ra[I-2].append(res_exist_prob)

   return(Ra)


def fill_Ra():
   '''Calculates R_Is for generations.'''
   n   = 1
   pgf = f
   Ra = [[0] for i in range(max_I-1)]
   Ra = fill_Ra_gen_n(Ra, 1, pgf)
   while n < gens:
      n += 1
      pgf = np.polyval(f, pgf) # iterate polynomial
```

```
        Ra = fill_Ra_gen_n(Ra, n, pgf)

    return(Ra)
```

8.A.3 Simulation Analysis

```
def one_generation(cell_dict):
    new_cell_dict = {}

    # Division Process
    for DM_number in cell_dict:
        for cell in range(cell_dict[DM_number]):
            progeny_DM = np.zeros(2)
            for DM in range(int(DM_number)):
                new_DMs = choice([1,2], 1 ,p=[1-a, a])[0]
                favored_cell = choice([0, 1])
                if new_DMs == 1:
                    progeny_DM[favored_cell] += 1
                else:
                    even_split = choice([True, False], 1, p=[1-al, al])[0]
                    if even_split:
                        progeny_DM += np.ones(2)
                    else:
                        progeny_DM[favored_cell] += 2

            for p in progeny_DM:
                try:
                    new_cell_dict[p] += 1
                except:
                    new_cell_dict[p] = 1

    return(new_cell_dict)


def single_sim():
    Rs = [[0] for i in range(max_I-1)]
    cell_dict = {1:1}
    for gen in range(gens):
        cell_dict = one_generation(cell_dict)
        max_R = max(cell_dict.keys())
        for i in range(max_I-1):
            I = i + 2
            if max_R >= I:
                Rs[i].append(1)
            else:
                Rs[i].append(0)
    return(Rs)


def average_over_many_sims():
    '''Mean max resistance average of n generations'''
    data = []
    for i in range(nSim):
        result = single_sim()
        data.append(result)
    data_array = np.array(data)

    mean = np.mean(data_array, axis = 0)
    file = open('GensMean'+str(gens)+'_maxI'+str(max_I)+
           '_a'+str(a)+'_al'+str(al)+'_simRepetes'+str(nSim)+'.txt','w')
    file.write(json.dumps([list(row) for row in mean]))
    file.close()
```

8.B Basic Model Simulation Code

```

a     = 0.94
alpha = 1
cell_dict = {0:4,1:1}  # initial population

def apply_drug(cell_dict):
   '''Remove sensitive cells form cell_dict when drug is active.'''
   kill = [] # cells to kill
   threshold = drug_strenth.get()
   for DM_number in cell_dict.keys():
      if DM_number < threshold:
         kill.append(DM_number)
   for cell in kill:
      del(cell_dict[cell])


def one_generation_basic(cell_dict):
   '''Advances population one generation.'''
   new_cell_dict = {}
   # loops over every cell type
   for DM_number in cell_dict:
      # division Process
      for cell in range(cell_dict[DM_number]):
         progeny_DM = np.zeros(2)
         for DM in range(int(DM_number)):
            new_DMs = choice([1,2], 1 ,p=[1-a, a])[0] # number of new daughters
            favored_cell = choice([0, 1]) # cell DMs go to if_not evenly split
            if new_DMs == 1:
               progeny_DM[favored_cell] += 1
            else:
               even_split = choice([True, False], 1, p=[1-alpha, alpha])[0]
               if even_split:
                  progeny_DM += np.ones(2)
               else:
                  progeny_DM[favored_cell] += 2
         # fills new_cell_dict with new cells
         for p in progeny_DM:
            try:
               new_cell_dict[p] += 1
            except:
               new_cell_dict[p] = 1
   # drug Based Selection
   if drug:
      apply_drug(new_cell_dict)

   return(new_cell_dict)
```

8.C Dynamic Model Simulation Code

```
a     = 0.9
alpha = 0.5
b     = 0.1
t     = 0.3
v     = 1
cell_dict = {0:4,1:1} # initial popuation

def add_cell(cell_dict, DM):
   '''Adds a specified cell to cell_dict'''
   try:
      cell_dict[DM] += 1
   except:
      cell_dict[DM] = 1

   return(cell_dict)


def one_generation_dynamic(cell_dict):
   '''Advances population one generation under the rules of
    the dynamic Model.'''
   new_cell_dict = {}
   # loops over every cell type
   for DM in cell_dict:
      # sets the birth-death probabilities
      if not(drug):
         p0 = b
         p2 = (1-b)*np.exp(-t*DM)
         p1 = 1 - p0 - p2
      else:
         DS = drug_strenth.get()        # drug strenth
         drug_intevals[gens[-1]+1] = DS # to plot blue area
         p0 = (1-b)/(1 + np.exp(-v*(DS-DM))) + b # DS-DM = sensetivity
         p2 = (1-b)*np.exp(-t*DM) * ((1-p0)/(1-b))
         p1 = 1 - (p0 + p2)
      # division Process
      for cell in range(cell_dict[DM]):
         # cell division
         ofspring = choice([0,1,2], 1 ,p=[p0,p1,p2])[0]
         if ofspring == 0:
            continue
         if ofspring == 1:
            new_cell_dict = add_cell(new_cell_dict, DM)
            continue
         # double minute division
         progeny_DM = np.zeros(ofspring)
         for n in range(int(DM)):
            new_DMs = choice([1,2], 1 ,p=[1-a, a])[0] #number of new double minutes
            if new_DMs == 1 or ofspring == 1:
               progeny_DM[0] += new_DMs
            else:
               even_split = choice([True, False], 1, p=[1-alpha, alpha])[0]
               if even_split:
                  progeny_DM += np.ones(2)
               else:
                  progeny_DM[0] += 2
         # fills new_cell_dict with new cells
```

```
        for p in progeny_DM:
            new_cell_dict = add_cell(new_cell_dict, p)
    # prints max cell double minute count which is present
    print(int(max(new_cell_dict.keys()) if new_cell_dict != {} else 0))
    return(new_cell_dict)
```

8.D Tumor Growth Model

Preamble

```
import tkinter as tk
import numpy as np
from numpy.random import choice, uniform
import tkinter.messagebox
from multiprocessing import Pool
from copy import deepcopy
```

8.D.1 Variables and Parameters

User Control Panel

```

mutation_rates = np.array([0,0,0,0,0,0,0,0,0,0,0]) # default mutation rates

ncol, nrow = 60, 50 # lattice dimentions
```

Static Variables

```

width = 800
d = width/ncol
tor = np.array([[nrow, ncol]]).T  # used to facilitate toroidal boundary
c_dim = np.array([nrow,ncol])
dim   = c_dim*(width/max(c_dim))
idens  = np.array(['sn','r1','r2','rb','ms'])
colours = {'he':'#008000','sn':'#ffd11a','r1':'#ff8c1a',
           'r2':'#ff66b3','rb':'#e60000','ms':'#ffff99',
           'nc':'#604020'}
# circles
circle = np.array([np.array([0,0,1,1,1,-1,-1,-1]),
                   np.array([1,-1,1,0,-1,1,0,-1])])
pointCirc31 = np.array([np.array([15,-15,0,0,7,-7,7,-7,13,13,-13,-13]),
                        np.array([0,0,15,-15,13,13,-13,-13,7,-7,7,-7])])
# birth rates
bConst = 3
birth_miltiplier  = bConst * np.array([np.array([i for
                             i in np.arange(9)])/j for j in [1,3,3,5,1]])
# death srates
dConst = 5
death_multiplier1 = dConst * np.array([np.array([i for
                             i in np.arange(9)])/j for j in [0.7,50,1,34,0.7]])
death_multiplier2 = dConst * np.array([np.array([i for
                             i in np.arange(9)])/j for j in [0.7,1,50,34,0.7]])
# necrotic switch constant
necroSpeed      = 0.5
# mutation rates
mOriginal = mutation_rates
mut_idens = np.array([['sn', 'sn', 'r1', 'r2', 'sn',
                       'r1', 'r2', 'rb', 'rb', 'ms'],  # mutate from
                      ['r1', 'r2', 'rb', 'rb', 'ms',
                        'sn', 'sn', 'r1', 'r2', 'sn']]) # mutate into
```

Dynamic Variables

```
cell_array  = np.array([['xhe']*ncol]*nrow)
cell_tally  = np.array([[0]*9]*5)
necro_timer = np.array([[0]*ncol]*nrow)
threshold = True
plan = None
running = False
labled = False
runTime = 0

# first cancerous cell
cell_array[int(nrow/2)][int(ncol/2)] = '8sn'
cell_tally[0,8] += 1
```

8.D.2 Simulation Code

Drawing

```
def init_squares():
   '''Draws the initial grid of green squares.'''
   for i in range(ncol):
      for j in range(nrow):
         vis.create_rectangle(i*d, j*d+d, i*d+d, j*d,
                        fill=colours[cell_array[j][i][1:]], outline="")


def draw_cell(iden, row, col):
   '''Draws a particular cell.'''
   vis.create_rectangle(col*d, row*d+d, col*d+d, row*d,
                       fill=colours[iden], outline="")
```

Growth Facilitating Functions

```
def get_circ_tup(cir, cen, tor):
   '''Finds the indices of the cells that are in the 8, circling the
    current cell.'''
   cir += cen      # centres circle on current cell
   cir %= tor      # adjusts to accommodate toroidal boundary conditions
   return(cir)


def pick_a_cell(typ):
   '''Chooses a particular cell of a given type.'''
   locs = np.dstack(np.where(cell_array == typ))[0] # find all cells of typ
   r, c = locs[choice(np.arange(locs.shape[0]))]    # chooses a cell to divide
   return(r, c)


def place_cell_short_range(r, c):
   '''Chooses a new place for a cell in the close vacinity of its parent.'''
   cen = np.array([[r, c]]).T        # coords of the chosen cell
   # looks for a healthy cell in_the vicinity
   cir = np.copy(circle)
   cir = get_circ_tup(cir, cen, tor) # adjust circle to be centered on cell
   sample = cell_array[tuple(cir)]   # cells in_a circle around parent
```

```
    sample = np.where(sample == 'xhe')
    ind = choice(sample[0]) # random choice among options
    return(cir.T[ind])      # new cell position


def update_surround(idenOriginal, tor, nr, nc):
    '''Updates the values of the cells which are imidiatley surrounding a
    cell which has just undergone an event.'''
    global cell_tally, cell_array
    cen = np.array([[nr, nc]]).T
    cir = np.copy(circle)
    cir = get_circ_tup(cir, cen, tor)
    sample = cell_array[tuple(cir)] # cells surrounding new daughter cell
    locs = np.where(np.char.find(sample, 'x') == -1)
    daughter_level = np.count_nonzero(sample == 'xhe')
    typs = sample[locs]
    # reduces blood flow level of surrounding cells by one
    new_typs = np.array(list(map(lambda a: str(int(a[0])-1)+a[1:], typs)))
    levels   = np.array(list(map(lambda a: int(a[0]), typs)))
    idenees  = np.array(list(map(lambda a: a[1:], typs)))
    where    = np.array(list(map(lambda a: np.where(idens == a)[0], idenees )))
    # updates memory with the placed daughter cell
    cell_array[nr, nc] = str(daughter_level) + idenOriginal
    cell_tally[np.where(idens == idenOriginal), daughter_level]   += 1
    # updates cell_tally and_cell_array with changes to surrounding cells
    cell_array[tuple(cir.T[locs].T)] = new_typs
    for i in np.arange(levels.shape[0]):
        cell_tally[where[i], levels[i]-1] += 1
        cell_tally[where[i], levels[i]]   -= 1


def advance_necro_timer():
    '''Advances the timer which decides which cell will become necrotic next.'''
    global necro_timer
    # finds every cell that has no bloof flow
    die_map = np.char.find(cell_array, '0')
    locs    = np.where(die_map == 0)
    necro_timer[locs] += 1
```

Main Event Functions

```
def grow_one(typ):
    '''Finds a parent for and places a new cell of a given type.'''
    global cell_tally, cell_array
    r, c = pick_a_cell(typ)
    # finds a place for_the new cell
    nr, nc = place_cell_short_range(r, c)
    # updates surrounding cell levels
    update_surround(typ[1:], tor, nr, nc)
    # draw new cell
    if drawing:
        draw_cell(typ[1:], nr, nc)


def convert_to_necrotic():
    '''Chooses a cell that has had the longest wait to die and converts it
     to necrotic.'''
    global cell_tally, cell_array, necro_timer
    longest_wait = np.max(necro_timer) # subset of candidates with longest wait
    locs = np.dstack(np.where(necro_timer == longest_wait))[0] # their positions
    r, c = locs[choice(np.arange(locs.shape[0]))] # choses one at random
    iden = cell_array[r, c][1:]
    # update memory and_draw cell
    cell_array[r, c]  = 'xnc'
    necro_timer[r, c] = 0
    cell_tally[np.where(idens == iden), 0] -= 1
    if drawing:
        draw_cell('nc', r, c)


def mutate(mutIndex, level):
    '''Executes a given type of mutation. Chooses a cell at random.'''
    global cell_tally, cell_array
    # mutation type
    startIden, endIden = mut_idens[0,mutIndex], mut_idens[1,mutIndex]
    startTyp = str(level) + startIden # mutate_from
    endTyp   = str(level) + endIden   # mutate into
    r, c = pick_a_cell(startTyp)      # particular cell choice
    # update memory and_draw cell
    cell_array[r, c]  = endTyp
    cell_tally[np.where(idens == startIden), level] -= 1
    cell_tally[np.where(idens == endIden), level]   += 1
    if drawing:
        draw_cell(endIden, r, c)


def grow_one_mst(typ):
    '''Same as grow_one but is specifically designed for metastatic cells '''
    global cell_tally, cell_array
    r, c = pick_a_cell(typ)
    cen = np.array([[r, c]]).T      # coords of the chosen cell
    try:
        # tries to put new metastatic cell far away
        cir = np.copy(pointCirc31) # deep copy
        cir = get_circ_tup(cir, cen, tor) # adjust circle to be centered on cell
        sample = cell_array[tuple(cir)]   # cells in_a circle around parent
        sample = np.where(sample == 'xhe')
        ind = choice(sample[0])
        nr, nc = cir.T[ind] # new cell position
    except:
        nr, nc = place_cell_short_range(r, c)
    # updates surrounding cell levels
    update_surround('sn', tor, nr, nc)
    # draw new cell
    if drawing:
        draw_cell('sn', nr, nc)


def kill_one(typ):
    '''Selects a cell of the given type to kill. then replaces it with a
    healthy cell.'''
    global cell_tally, cell_array, necro_timer
    r, c = pick_a_cell(typ)
    # updates sensitivity of surrounding cells
    cen = np.array([[r, c]]).T
    cir = np.copy(circle)
    cir = get_circ_tup(cir, cen, tor)
    sample = cell_array[tuple(cir)] # cells surrounding new daughter cell
    locs = np.where(np.char.find(sample, 'x') == -1)
    typs = sample[locs]
    # adds 1 to the blood flow numbers of surrounding cells
    new_typs = np.array(list(map(lambda a: str(int(a[0])+1)+a[1:], typs)))
    levels   = np.array(list(map(lambda a: int(a[0]), typs)))
    idenees  = np.array(list(map(lambda a: a[1:], typs)))
    where    = np.array(list(map(lambda a: np.where(idens == a)[0], idenees )))
    # updates memory of the killed cell
    cell_array[r, c] = 'xhe'
    cell_tally[np.where(idens == typ[1:]), int(typ[0])] -= 1
    # updates memory on surrounding cells
    circle_indecies = tuple(cir.T[locs].T)
    cell_array[circle_indecies] = new_typs
    necro_timer[circle_indecies] = 0
    necro_timer[r, c] = 0
    for i in np.arange(levels.shape[0]):
        cell_tally[where[i], levels[i]+1] += 1
        cell_tally[where[i], levels[i]]   -= 1
    # draws healthy cell
    if drawing:
        draw_cell('he', r, c)
```

Functions Used in the Run Main Loop

```
def get_rates():
    '''Produces all 175 rates at each timestep to facilitate the Gillespie
    Algorithm.'''
    global bRates, bRatesPartSum, bRatesTotalSum, mRates, mRatesPartSum,\
           mRatesTotalSum, dRates, dRatesPartSum, dRatesTotalSum, rateSum,\
           allRates
    # birth rates
    bRates            = cell_tally * birth_miltiplier
    bRatesPartSum     = np.sum(bRates, axis=1)
    bRatesTotalSum    = np.sum(bRatesPartSum)
    # necrotic switch rate
    necroRates        = necroSpeed * cell_tally[:,0]
    necroRatesTotal   = np.sum(necroRates)
    # mutation rates
    sen, res1, res2, resb, mst = cell_tally[0], cell_tally[1], \
     cell_tally[2], cell_tally[3], cell_tally[4]
    mRates            = np.array([sen, sen, res1, res2, sen, res1, res2,
                                  resb, resb, mst]) * mutation_rates[:, None]
    mRatesPartSum     = np.sum(mRates, axis=1)
    mRatesTotalSum    = np.sum(mRatesPartSum)
    # death rates
    if not(drug1 or drug2):
        dRatesTotalSum = 0
    else:
        dRates     = np.zeros(cell_tally.shape)
        if drug1:
            dRates += cell_tally.astype('float64')*death_multiplier1
        if drug2:
            dRates += cell_tally.astype('float64')*death_multiplier2
        dRatesPartSum    = np.sum(dRates, axis=1)
        dRatesTotalSum   = np.sum(dRatesPartSum)
    # all rates
    allRates = np.array([bRatesTotalSum, necroRatesTotal,
                         mRatesTotalSum, dRatesTotalSum])
    rateSum  = np.sum(allRates)
```

```
def execute(eventIndex):
   '''Given an event type, execute will decide the further specifics of the
   event. It will then carry out its choice.'''
   # birth event
   if eventIndex == 0:
      # choose which cell type
      idenIndex = choice(np.arange(bRatesPartSum.shape[0]), 1,
                     p=bRatesPartSum/bRatesTotalSum)[0]
      # choose which level
      level = choice(np.arange(9), 1,
                     p=bRates[idenIndex]/bRatesPartSum[idenIndex])[0]
      # chosen type
      typ = str(level) + idens[idenIndex]
      if idenIndex == 4: # metastatic
         grow_one_mst(typ)
      else:
         grow_one(typ)
   # necro event
   elif eventIndex == 1:
      convert_to_necrotic()
   # mutation event
   elif eventIndex == 2:
      # choose which mutation type
      mutIndex = choice(np.arange(mRatesPartSum.shape[0]), 1,
                     p=mRatesPartSum/mRatesTotalSum)[0]
      # choose which level
      level = choice(np.arange(9), 1,
                     p=mRates[mutIndex]/mRatesPartSum[mutIndex])[0]
      mutate(mutIndex, level)
   # drug kill event
   elif eventIndex == 3:
      # choose which cell type
      idenIndex = choice(np.arange(dRatesPartSum.shape[0]), 1,
                     p=dRatesPartSum/dRatesTotalSum)[0]
      # choose which level
      level = choice(np.arange(9), 1,
                     p=dRates[idenIndex]/dRatesPartSum[idenIndex])[0]
      # chosen type
      typ = str(level) + idens[idenIndex]
      kill_one(typ)


def auto_drug_regime(t):
   '''Used to apply defined drug application regimes.'''
   global threshold
   if plan == None:
      return
   # after 25 cells become present, begin
   if threshold and np.sum(cell_tally) > 25:
      threshold = False
   # no drug
   if threshold or plan == 1:
      pass
   else:
      # drug 1 alternation
      if plan == 2:
         if int(10*t/2) % 2 == 0:
            if drug1:
                toggle_drug_1()
        else:
            if not(drug1):
                toggle_drug_1()
    # drug 1 and 2 fixed on
    elif plan == 3:
        if not(drug1):
            toggle_drug_1()
        if not(drug2):
            toggle_drug_2()
```

Running the Model

```
def run():
    '''Main driving function.'''
    global drawing, runTime
    drawing = True
    # main loop
    while running:
        auto_drug_regime(runTime)
        get_rates()
        if rateSum != 0:
            U1 = uniform(0,1)
            dt = -np.log(U1)/rateSum # time until next event
            runTime += dt
            advance_necro_timer()
            root.after(int(10000*dt))
        else: # tumour has been eliminated or_taken over
            toggle_run(1)
            result = 'Tumour has been eliminated.' if 'xhe' in cell_array \
                                                else 'Game Over'
            tkinter.messagebox.Message(icon='info', type='ok',
                                       message=result,
                                       title='Results').show()
            continue
        # choose which event type
        eventIndex = choice(np.arange(allRates.shape[0]), 1,
                            p=allRates/rateSum)[0]
        execute(eventIndex)  # execute choice
        root.update()
```

8.D.3 User Control

Button Press Functions

```
def toggle_run(event):
    '''Sarts and stops the continuous time process.'''
    global running, labled
    # start
    if not(running):
        running = True
        goStop.configure(bg='red',fg='white')
        # does the cell blood flow labeling
        if labled:
            locs = np.dstack(np.where(np.char.find(cell_array, 'x') == -1))[0]
            for i, j in locs:
                idenNum = cell_array[i,j]
            draw_cell(idenNum[1:], i, j)
         labled = False
   # stop
   else:
      running = False
      goStop.configure(bg='#e6e6e6',fg='black')


drug1 = False
def toggle_drug_1():
   global drug1
   if drug1:
      drug1 = False
      try:
         drug1_onoff.configure(bg='#e6e6e6',fg='black')
      except:
         pass
   else:
      drug1 = True
      try:
         drug1_onoff.configure(bg='#6666ff',fg='white')
      except:
         pass


drug2 = False
def toggle_drug_2():
   global drug2
   if drug2:
      drug2 = False
      try:
         drug2_onoff.configure(bg='#e6e6e6',fg='black')
      except:
         pass
   else:
      drug2 = True
      try:
         drug2_onoff.configure(bg='#33ccff',fg='white')
      except:
         pass


def label_cells():
   '''Blood flow labeling.'''
   global labled
   labled = True
   if running:
      toggle_run(1)
   locs = np.dstack(np.where(np.char.find(cell_array, 'x') == -1))[0]
   for i, j in locs:
      idenNum = cell_array[i,j]
      draw_cell(idenNum[1:], i, j)
      vis.create_text(((j+0.5)*d, (i+0.5)*d), font=("Purisa", int(0.5*d)),
                  text=idenNum[0])


def plan1():
   global plan, mutation_rates
   if plan != 1:
      reset_all()
      example1.configure(bg='#6600ff',fg='white')
      plan = 1
      mutation_rates = np.array([ 0,  0, 31,  9, 30, 18, 20,  9,  5, 1])
      toggle_run(1)
      run()
   else:
      toggle_run(1)
      run()


def plan2():
   global plan, mutation_rates
   if plan != 2:
      reset_all()
      example2.configure(bg='#6600ff',fg='white')
      plan = 2
      mutation_rates = np.array([16,  0,  1, 13, 31,  2, 16, 20,  6, 10])
      toggle_run(1)
      run()
   else:
      toggle_run(1)
      run()


def plan3():
   global plan, mutation_rates
   if plan != 3:
      reset_all()
      example3.configure(bg='#6600ff',fg='white')
      plan = 3
      mutation_rates = np.array([17, 22, 13, 26, 26,  3, 14,  0,  0,  3])
      toggle_run(1)
      run()
   else:
      toggle_run(1)
      run()


def reset_vals():
   global cell_array, cell_tally, necro_timer, threshold, runTime
   # dynamic variables
   cell_array  = np.array([['xhe']*ncol]*nrow)
   cell_tally  = np.array([[0]*9]*5)
   necro_timer = np.array([[0]*ncol]*nrow)
   # first cancer cell
   cell_array[round(nrow/2)][round(ncol/2)] = '8sn'
   cell_tally[0,8] += 1
   # drug reset
   if drug1:
      toggle_drug_1()
   if drug2:
      toggle_drug_2()
   threshold = True
   runTime = 0


def reset_all():
   global running, plan, mutation_rates
   reset_vals()
   # re-draw display
   init_squares()
   if running:
      running = False
      goStop.configure(bg='#e6e6e6',fg='black')
   plan = None
   mutation_rates = mOriginal
   example1.configure(bg='#e6e6e6',fg='black')
   example2.configure(bg='#e6e6e6',fg='black')
   example3.configure(bg='#e6e6e6',fg='black')


def close():
   global running
   if running:
      toggle_run(1)
   root.destroy()
```

Graphical User Interface

```

GUI = 1
if bool(GUI):
   root = tk.Tk()
   root.title('Tumour Growth Model')

   root.configure(background='white')
   vis  = tk.Canvas(root, highlightthickness=0, background="Black",
                  width=dim[1], height=dim[0])
   ctrl = tk.Frame(root, width=dim[1], height=150, bd=0, background='white')
   vis.pack()
   ctrl.pack()

   b1 = tk.Canvas(ctrl, width=470, bd=0, height=130,
                highlightthickness=0, bg='#999999')
   b2 = tk.Canvas(ctrl, width=dim[1]-10-490, bd=0, height=130,
                highlightthickness=0, bg='#999999')
   b1.place(x=10,y=10)
   b2.place(x=490,y=10)

   goStop = tk.Button(ctrl, text='Run/Stop',bg='#e6e6e6', command=run)
   goStop.bind('<Button 1>', toggle_run)
   goStop.place(x=30, y=50, width=120, height=50)

   drug1_onoff = tk.Button(ctrl, text='Toggle Drug 1', bg='#e6e6e6',
                     command=toggle_drug_1)
   drug2_onoff = tk.Button(ctrl, text='Toggle Drug 2', bg='#e6e6e6',
                      command=toggle_drug_2)
   drug1_onoff.place(x=180, y=23, width=130, height=40.5)
   drug2_onoff.place(x=180, y=86.5, width=130, height=40.5)

   reset       = tk.Button(ctrl, text='Reset', bg='#e6e6e6',command=reset_all)
   reset.place(x=340, y=50, width=120, height=50)

   label       = tk.Button(ctrl,text='Label',bg='#e6e6e6',command=label_cells)
   label.place(x=dim[1]-300, y=90, width=70, height=30)

   example3    = tk.Button(ctrl, text='3', bg='#e6e6e6', command=plan3)
   example2    = tk.Button(ctrl, text='2', bg='#e6e6e6', command=plan2)
```

```
    example1     = tk.Button(ctrl, text='1', bg='#e6e6e6', command=plan1)
    example1.place(x=dim[1]-180, y=90, width=30, height=30)
    example2.place(x=dim[1]-120, y=90, width=30, height=30)
    example3.place(x=dim[1]-60, y=90, width=30, height=30)

    root.protocol("WM_DELETE_WINDOW", close)
    init_squares()
    root.mainloop()
```

8.D.4 Genetic Algorithm Code

```
def test_fitness(rates):
    '''Asesses the fitness of a given set of mutation rates for a perscribed
    drug application plan'''
    global drawing, drug1, drug2, mutation_rates
    mutation_rates = rates
    drawing = False
    runTime = 0
    maxTime = 0.3
    noElim, noTake = (True, True)
    # main loop simular to run function
    while runTime < maxTime and noElim and noTake:
        auto_drug_regime(runTime)
        get_rates()
        # time till next event
        U1 = uniform(0,1)
        alive = 'xhe' in cell_array
        if rateSum != 0 and alive:
            dt = -np.log(U1)/rateSum # time till next event
            advance_necro_timer()
        else:
            noElim, noTake = (False, True) if alive else (True, False)
            continue
        # choose which event type
        eventIndex = choice(np.arange(allRates.shape[0]), 1,
                            p=allRates/rateSum)[0]
        execute(eventIndex) # execute event
        runTime += dt
    # calculate fitness
    k = 1
    totalCells = cell_array.size
    healthy    = np.count_nonzero(cell_array == 'xhe')
    if not(noElim):
        # tumour killed
        # fitness proportional to survival time
        fitness = k*runTime
    elif not(noTake):
        # tumour takes over
        # fitness negatively proportional to take-over time
        fitness = k*(2*maxTime + maxTime - runTime)
    else:
        # tumour survives but does not_take over
        # fitness negatively proportionsl to the number of healthy cells
        fitness = k*(1*maxTime + maxTime - healthy*maxTime/totalCells)
    reset_vals()
    return(fitness)


def initial_set():
    '''Provides an initial solution population.'''
    gen0 = np.array([np.random.randint(0,6, size=10) for i in range(10)],
                    dtype=np.int32)
    return(gen0)


def asess_one(rates):
    '''Asseses the fitness of an individual solution by testing it 20 times
    and taking an average.'''
    fitness = 0
    for i in range(20):
        fitness += test_fitness(rates)
    return(fitness/20)


def asess_fitness(genN):
    '''Parallelized fittnes testing.'''
    p = Pool()
    fitnesses = p.map(asess_one, list(genN))
    return(fitnesses)


def select_parents(genN, fit):
    '''Chooses 2 fittest individuals to be parents for next gen.'''
    fitSort = np.argsort(fit)
    fitSort = np.flip(fitSort, axis=0)
    parents = np.array([genN[i] for i in fitSort[:2]], dtype=np.int32)
    winner = parents[0]
    return(parents, winner)


def binary_to_gray(b):
    g = b[0]
    for k in range(1,len(b)):
        g += str(int(b[k-1] != b[k]))
    return(g)


def gray_to_binary(g):
    b = g[0]
    for k in range(1,len(g)):
        b += str(int(b[k-1] != g[k]))
    return(b)


def real_to_bit_string(mr):
    '''Converts mutation rates list into gray coded binary string.'''
    bitString = ''
    for num in mr:
        b = str(bin(int(num)))[2:]
        g = binary_to_gray(b)
        bitString += ('0'*(5-len(g)) + g)
    return(bitString)


def bit_string_to_real(bs):
    '''Converts gray coded binary string into mutation rates.'''
    real = []
    for i in range(10):
      g = bs[5*i:5*(i+1)]
      b = gray_to_binary(g)
      real.append(int(b,2))
   return(np.array(real))


def mutate_one(mr1, pos):
   '''Mutates with a single decimal change of + or - 1.'''
   mr = deepcopy(mr1)
   old = mr[pos]
   if old == 0:
      change = 1
   elif old == 31:
      change = -1
   else:
      change = -1 if uniform(0,1) > 0.5 else 1
   mr[pos] += change
   return(mr)


def sex(p1, p2):
   '''Combines two parent genomes to form 2 offspring via crossover
   and mutation.'''
   #crossover
   c1 = np.concatenate((p1[:5] , p2[5:]))
   c2 = np.concatenate((p2[:5] , p1[5:]))
   # mutation
   m1, m2 = choice(list(range(10)), 2) # mutation locations
   c1 = mutate_one(c1, m1)
   c2 = mutate_one(c2, m2)
   return(c1, c2)


def create_set(parents, winner):
   '''Produces a new population of individuals from the best 2 from the last
   generation.'''
   new_rates = []
   # 2 new rates from_crossover
   children = sex(parents[0], parents[1])
   new_rates += children
   # 7 new rates gained asexualy from_last winner 3 of which have 2 muts
   winnerVarients = [winner]
   wbs = real_to_bit_string(winner)

   for i in range(6):
      m1 = choice(list(range(50))) # mutation location
      # bit string mutation
      oneMBs = wbs[:m1] +('0'if wbs[m1]=='1'else'1')+ wbs[m1+1:]
      if i >= 4:
         m2 = choice(list(range(50))) # mutation location
         # bit string mutation
         twoMBs = oneMBs[:m2] +('0'if oneMBs[m2]=='1'else'1')+ oneMBs[m2+1:]
         winnerVarients.append(bit_string_to_real(twoMBs))
      else:
         winnerVarients.append(bit_string_to_real(oneMBs))
   new_rates += winnerVarients
   # 1 random rates
   new_rates += [np.random.randint(0,16, size=10)]
   return(np.array(new_rates, dtype=np.int32))
```

```
def genetic_algorithm(p):
   '''Main implementation of the genetic algorithm.'''
   global plan
   plan = p
   fit = []
   parents = []
   winner = None
   try:
      genN = np.loadtxt('lastGenPlan31FINAL_'+str(plan)+'.txt').astype('int32')
   except:
      genN = initial_set()
   k = 0
   # main loop
   while k<10000:
      if __name__ == '__main__':
         print(genN)
         fit = asess_fitness(genN)
         avgFit, maxFit = round(sum(fit)/float(len(fit)),3), round(max(fit),3)
         parents, winner = select_parents(genN, fit)
         with open('fitrecord31FINAL_'+str(plan)+'.txt','a+') as file:
            file.write(str(avgFit)+'    '+str(maxFit)+'     '+str(winner)+'\n')
         print(k,'fitness| average: {},  best: {} | winner| {}'.format(
               avgFit,maxFit,winner))
         genN = create_set(parents, winner)
         np.savetxt('lastGenPlan31FINAL_'+str(plan)+'.txt', genN)
      k += 1
      print('=============================================================='
            '=====================')
```

Bibliography

[1] Deepak Chopra and Rudolph E. Tanzi. Super genes: Unlock the astonishing power of your dna for optimum health and well-being. 2015.

[2] Walter+ElizaHall. Website: https://www.wehi.edu.au/wehi-tv/x-inactivation-and-epigenetics. accessed: 8/2019.

[3] PassmyexAms. Website: http://www.passmyexams.co.uk/gcse/biology/structure-of-dna-molecule.html. accessed: 8/2019.

[4] EuroGentest. Website: https://eurogentest.eshg.org/index.php?id=611. accessed: 8/2019.

[5] Ayman Mukerji Househam, Christine Tara Peterson, Paul J Mills, Deepak Chopra, et al. The effects of stress and meditation on the immune system, human microbiota, and epigenetics. *Adv Mind Body Med*, 31(4):10–25, 2017.

[6] Science Samhita. Website: https://sciencesamhita.com/wp-content/uploads/2017/10/bacterial-conjugation.png. accessed: 8/2019.

[7] Marek Kimmel and David E Axelrod. Branching processes in biology, 2002.

[8] PETER C Brown, STEPHEN M Beverley, and ROBERT T Schimke. Relationship of amplified dihydrofolate reductase genes to double minute chromosomes in unstably resistant mouse fibroblast cell lines. *Molecular and cellular biology*, 1(12):1077–1083, 1981.

[9] Randal J Kaufman, Peter C Brown, and Robert T Schimke. Amplified dihydrofolate reductase genes in unstably methotrexate-resistant cells are associated with double minute chromosomes. *Proceedings of the National Academy of Sciences*, 76(11):5669–5673, 1979.

[10] Michael Affenzeller, Stefan Wagner, Stephan Winkler, and Andreas Beham. *Genetic algorithms and genetic programming: modern concepts and practical applications*. Chapman and Hall/CRC, 2009.

[11] GeeksforGeeks. Website: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/. accessed: 8/2019.

[12] Randall D Beer and John C Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive behavior*, 1(1):91–122, 1992.

[13] Grant Lythe. Mathematical biology lecture notes. 2018.