Reversibility of Evolution: A Simulation and Mathematical Approach

PARVATHY S
MS13020

A dissertation submitted for the fulfillment of BS-MS dual degree in Science

Indian Institute of Science Education and Research Mohali
April 2018

Certificate of Examination

This is to certify that the dissertation titled **"Reversibility of Evolution: A Simulation and Mathematical Approach"** submitted by **Parvathy S** (Reg. No. MS13020) for the partial fulfillment of BS-MS dual degree program of the Institute, has been examined by the thesis committee duly appointed by the Institute. The committee finds the work done by the candidate satisfactory and recommends that the report be accepted.

Dr. Abhishek Chaudhuri   Dr. Rhitoban Ray Choudhury

Dr. N. G. Prasad (Supervisor)   Prof. Arvind (Co- supervisor)   Prof. Shobha Madan (Local Guide)

Dated: April 20, 2018

Declaration

The work presented in this dissertation has been carried out by me under the guidance of Dr. N. G. Prasad, Prof. Arvind and Prof. Shobha Madan at the Indian Institute of Science Education and Research Mohali.
This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgement of collaborative research and discussions. This thesis is a bonafide record of original work done by me and all sources listed within have been detailed in the bibliography.

Parvathy S
(Candidate)
Dated: April 20, 2018

In my capacity as the supervisor of the candidate's project work, I certify that the above statements by the candidate are true to the best of my knowledge.

Dr. N. G. Prasad	Prof. Arvind	Prof. Shobha Madan
(Supervisor)	(Co- supervisor)	(Local Guide)

Acknowledgements

First and foremost, I would like to thank my thesis supervisors Dr. N.G. Prasad, Prof. Arvind and Prof. Shobha Madan, without whose help and supervision, this thesis would have never been possible. The discussions that I had with them has enhanced my capabilities at a professional level. Also I would like to thank my thesis committee members Dr. Abhishek Chaudhuri and Dr. Rhitoban Ray Choudhury for their valuable suggestions and criticism of my work.

I owe my deepest gratitude to Pranay Rungta for helping me with the computational aspects. The discussions I had with him was extremely useful to improve my computational skills.

I would like to thank all my friends for supporting and helping me during the entire period of the project. I am also thankful to my seniors for their useful advice and guidance. I would like to acknowledge the moral support and encouragement that I have received from my family.

Finally, I am thankful to IISER Mohali for providing me Infrastructure and Computer Centre for all the technical support. I would like to acknowledge DST INSPIRE, Government of India for the financial support.

Parvathy S

Contents

List of Figures

List of Tables

Listings

Abbreviations

No	**N**umber
Vs	**V**ersus
Freq	**F**requency

Abstract

Evolution is the change in the characteristics of a population with successive generations. When selection is imposed to the population, some individuals reach sexual maturity at higher rates or produce more number of offspring proportional to their fitness values.

We focus on one aspect of evolution: its reversibility. The great evolutionist Stephen Jay Gould describes the experiment ”replaying life's tape” to talk on the notion of reversibility. A series of papers by Henrique Teotonio and Michael R.Rose addresses reverse evolution from an experimental point of view. Evolution is reversible if the traits observed in a population goes back to their ancestral state.

In this project, the question ”Is evolution reversible?” has been addressed using simulation. The fitness of a genotype is varied in a sinusoidal manner (fitness values come back to their initial value in some generations) and we see how does the slow rate of change of fitness and fast rate of change of fitness affect the course of evolution and its reversibility. We also build a mathematical model of evolution to predict how the frequencies accross generations would be.

It has been seen that either evolution is reversible or frequency of alleles follows a decreasing trend. The Fourier basis considered to build the model is not the right choice as the operator that maps the fitness function to the frequency function is not linear.

Chapter 1

Introduction

A group of interbreeding individuals along with their offspring form a population. In a population, adults produce gametes, they combine to form zygotes, they develop to comprise the next generation of adults. In this process, traits beneficial for survival are passed on more frequently than those which are detrimental. This changes the characteristics of the population with each generation, which is evolution. In a population genetics perspective, the change in the frequency of alleles with generation can be termed as evolution. This change in the traits is due to the corresponding changes in the genetic variants that affect them. If there is no selection, mutation, migration, chance events and when mating occurs randomly, there would be no change in the population and hence no evolution takes place.

Once we impose selection to the population, certain individuals (having a particular phenotype) become adults at higher rates or produces more number of offspring than other individuals (with a different phenotype). As a consequence individuals have unequal chances of surviving to adulthood and contributing to the next generation. We assume a one-one correspondence between phenotype and genotype. Selection occurs due to change in fitness, which is the ability of an individual to survive and reproduce. One would like to see which alleles become more common in the population and how the course of evolution will look like.[1]

When a rare allele is dominant, the rate of evolution is fastest. Its effect will be expressed in the heterozygotes and selection can act for or against it depending on whether it is beneficial or deleterious. If the rare allele is recessive, it hides from selection as its effect is seen only in homozygote recessive. To conclude, if the beneficial allele is dominant but rare, its frequency increases quickly but if it is common, elimination of deleterious allele will be slow. If the beneficial allele is recessive, its frequency increases slowly but reaches fixation rapidly once it is common. In case of heterozygote superiority, the population remains in a stable equilibrium.

An important aspect of evolution is its reversibility. According to Stephen Jay Gould, he refers to an experiment 'replaying life's tape'.[2] It is described as follows: rewind the life's tape, erasing everything that has happened and go back to any place and time. We play the tape again and see how its repetition looks like. If each time we play the tape the occurrence of events are similar to the original, then evolution is reversible. We follow this notion of reversibility here.

Questions on reversibility have been addressed experimentally in a series of papers by Henrique Teotonio and Michael R. Rose using *Drosophila melanogaster* populations.[3] Different groups are maintained, each one subjected to different selection conditions. They have been selected for early reproduction, mid-life reproduction, late reproduction, survival under complete starvation and so on. It has been shown that reverse evolution back to the initial state occurs in some cases. They have tried to determine possible genetic mechanisms for reversibility of evolution- the ones that facilitate and limits the process. [4] [5] These include checking the effects of genetic variation, mutation, epistasis, genotype-by-environment interaction and pleiotropy.

In this project, we take up the question of reversibility of evolution approaching it using simulation and mathematically. The effect of a reversibly varying environment has been analyzed. We consider a population in which selection has been imposed. A change in environment, like temperature or light, will have an effect on an individuals fitness values. A linearly or sinusoidally changing fitness values have been assigned to the genotype in such a way that their rate of change varies from slow to fast. Instead of following a particular trait, which is in turn determined by genotype, we measure the frequency of alleles after each generation. Frequency plots are analyzed to see the effect of reversibility. Finally, we try to produce a mathematical model for predicting the course of evolution, or more specifically, the allele frequencies after each generation.

Chapter 2

Methods: Computational and Mathematical Approach

In this project, we try to understand the aspects of reversibility of evolution. We approach this problem in two ways, first by using simulation and then mathematically. Darwinian evolution has been modeled using C++ programming based on certain assumptions. The model is described in detail below.

2.1 Model

Consider a population of constant population size such that for each individual we are concerned with its genes. Two models have been considered here- the 2-gene model and the 4-gene model. For the 2-gene model, each individual has 2 genes, say A and B and for each gene, there are two alleles, A0, A1 and B0, B1. The individuals are diploid, hence they carry two copies of gene A and two of gene B. The possible genotypes are A0A0, A0A1, A1A1, B0B0, B0B1, and B1B1. The total number of types of genotypes possible for the individuals are 16 since the first two positions are filled by A0 or A1 and the last two positions are filled by B0 or B1. We consider for now that there are 10 individuals of each type. The total population size will now be 160. We also assume that each individual mates at most once by finding its pair randomly and each pair produces 10 offspring by random recombination of their alleles.

We now impose selection to the population. Now, individuals of each type have unequal chances of surviving to adulthood and contributing to the next generation. The probability that an individual reaches adulthood depends on the total fitness of the individual. There is a fitness value assigned for each genotype and, unless specified, the total fitness of the individual is the average of the genotype fitness values. When adults mate and produce offspring, the number of offspring of each type reaching adulthood varies according to their fitness values. We then normalize the population to maintain constant population size. At generation 0, since there are equal number of individuals of each type, frequency of each allele is 0.5. After running the simulation we measure the frequency of allele after each generation. We also assume a one-one correspondence between the genotype and phenotype. Hence whatever trait we are looking for, its change is reflected in the genotype and hence in the frequency of alleles.

For the 4-gene model, for each individual we are concerned with 4 genes, A,B,C and D. For each gene, there are 4 alleles- A0,A1,A2,A3; B0,B1,B2,B3; C0,C1,C2,C3 and D0,D1,D2,D3. There are 16 genotypes for each of the four genes and the total number of genotypes of the individuals would be $4^8 = 65536$. The total population size would now be 655360. The remaining characteristics of the model is same as that of the 2 gene model. The two models have been summarized in table 2.1.

	Model 1	Model 2
No. of genes	2- A, B	4- A, B, C, D
No. of alleles for each gene	2- A0, A1, B0, B1	4- A0, A1, A2, A3, B0, B1, B2, B3, C0, C1, C2, C3, D0, D1, D2, D3
Total no. of genotypes	$2^4 = 16$	$4^8 = 65536$
Population size	160	655360

Table 2.1: Model summarized

2.2 Computational approach

A general code has been written for the model described above used for the 2-gene model. The required parameters have been iterated and/ or changed accordingly for the 4-gene model. The program has three parts- ”main.cpp”, ”population.cpp” and ”phenotype.cpp”.

In ”phenotype.cpp”, the different types of genotypes have been assigned. Functions have been written for their characteristics which include finding the type given an index and vice versa, assigning fitness values and calculating fitness of individual and so on.

In ”population.cpp”, functions for the characteristics of the population have been written. These include initializing the allele frequency, finding pair for each individual, producing offspring by random recombination of alleles, imposing selection. finding the new allele frequencies, normalizing the population size, assigning the new parent generation, and saving the allele frequencies after each generation. In the ”main.cpp” program, the different parameters have been set such as the number of types of each genotype, initial allele frequency, number of children per pair and the number of generations. It also creates the population, one for parent and one for children to do the required operations. It also calls the functions in the needed order to run the simulation. The code for the three programs has been given in Appendix A.

The graphs for the frequency plots and fitness plots have been plotted as needed. Once the code is written, we first verify it using classical models and then further check for reversibility of evolution. There are three parts where we use the simulation- one for the basic evolutionary models, second for checking the reversibility by varying the fitness and lastly for applying threshold fitness.

2.2.1 Classical evolutionary models

The simulation has been verified by using the code for testing classical evolutionary models for the 2-gene case. These include the following.

1. Genotype fitness fixed: The genotype fitness is kept fixed for each of the six genotypes. We consider the following combination of the fitness values which can be categorized as in table 2.2, where s and t are real numbers between 0 and 1.

   We assign every possible combination for genes A and B from the 5 cases and run the simulation. We look at the change in the frequency of alleles over a period of time.

	A0A0	A0A1	A1A1
(i) Additive	1	1- $\frac{s}{2}$	1-s
(ii) Dominant and beneficial	1	1	1-s
(iii) Dominant and deleterious	1	1-s	1-s
(iv) Heterozygous superiority	1-s	1	1-t
(v) Homozygous superiority	1	1-s	1

TABLE 2.2: Different fitness values assignment to each genotype

2. Allele fitness fixed: Fitness is considered as a function of alleles. Total fitness of an individual is the average of the fitness of four alleles. We consider the fitness cases as seen in table 2.3.

Fitness type	Allele 1	Allele 2
(i)	1	1-s
(ii)	1-s	1
(iii)	1-s	1-s

TABLE 2.3: Different fitness values assignment to each allele

   For each of A and B, we ascribe the three types of combination of allele fitness and see how the frequency of allele varies by looking at the generation vs. frequency plot.

3. Interaction of genes: We consider two cases here, one when the presence of A increases the fitness of corresponding genotypes of B and the other when the presence of A suppresses the fitness of genotypes of B. Fitness is increased by doubling the fitness of genotype when fitness is less than 0.5. Suppression of fitness is done by squaring the fitness when it is between 0 and 1, and interchanging 0 and 1 for each other. We then see the frequency patterns obtained.

4. Fitness is multiplicative: The total fitness of the individual is the product of genotype fitness. When fitness is calculated in this manner we look at the change in frequency of alleles after each generation.

For the 4- gene model, the fitness have been kept constant for genotypes, which have been assigned randomly. The resulting generation vs. frequency plot is observed.

2.2.2 Changing environment and reversibility

After the simulation have been verified, we now move on to find the answer to the question ”Does evolution retraces it’s path ?” under reversibly changing environment. This is done by changing the fitness of genotypes in a reversible manner. We vary the fitness values of A0A0, keeping the fitness of genotypes A0A1 and A1A1 fixed. Since the frequency behaviour of B is independent of that of A, it is enough to see how either one of them behaves in a reversibly changing environment. We vary the fitness of A0A0 and measure the frequency of A0 after each generation for 400 generations.

1. **Linearly changing fitness**

   For the 2- gene model, the fitness of A0A0 is changed linearly starting from 0, increasing till 1 and then returning to 0 in a certain number of generations. The rate of change of fitness is increased from 400 generations to fitness coming back in 5 generations. We look into various cases where fitness comes back in 400, 100, 50 and in 10 generations. For each case, we see how the frequency of alleles changes. Is the population able to adapt to the changing environment or do we see different frequency patterns? Does the frequency of allele A0 return back to their initial value as the fitness does? These are some of the questions that we are trying to address.

2. **Sinusoidally changing fitness**

   It is not always the case that change would be as sharp as linear change. Here we change the fitness smoothly in a sinusoidal manner such that the fitness values come back in 400 generations, 200 generations, 50 generations and then in 5 generations. We see the effect of slowly changing environment versus fast changing environment in the reversibility of evolution.

   For both these approaches, in the program ”phenotype.cpp” we add some extra lines of code to assign the fitness of A0A0 in a linearly or sinusoidally changing manner and then update the fitness of A0A0 accordingly. We normalize the fitness values so that they lie between 0 and 1. The portion of the code for doing this is listed in Appendix A.

3. **Effect of population size**

   So far the simulation has been done for a single population of size 160. To reduce the effect of small population size and reduced sample size, we run the simulation for 50 populations of the same kind and take the average frequency of allele A0 after each generation. We increase the population size from 160 to 1600 and to 16000. We then vary the fitness sinusoidally and compare the frequency patterns in these three cases.

   For running the simulation for 50 times, we provide an iterating loop at the beginning of the code and set the parameters to their initial value each time one run is over for 400 generations. The average of the frequency values are calculated and the graphs in all the cases have been plotted using Gnuplot.

4. **Effect of initial allele frequency**

   We focus on a population size of 16000. To see the effect of initial allele frequency on the frequency pattern obtained, we assign different initial numbers to the types of genotypes and see the corresponding frequency pattern in these cases. Different initial frequencies assigned are 0.275, 0.375,0.425 and so on.

5. **Effect of fixed fitness values of genotypes**

   So far we have been changing the fitness of one genotype, keeping the fitness of others fixed. Initially 1 and 0.5 were assigned to A0A1 and A1A1 respectively. We now assign other fitness values to A0A1 and A1A1 and see the corresponding frequency pattern observed. The assignments given are described in table 2.4.

Case	A0A1	A1A1
a	0.5	1
b	1	1
c	0.75	0.5
d	0.25	1
e	0.5	0.5

Table 2.4: Fitness values assigned to genotypes A0A1 and A1A1

6. **Sinusodialy changing fitness for 4-gene model**

   After having done these simulations for the 2-gene model we now move on to the 4-gene model. We vary the fitness of A0A0 for different assignments of fitness of other genotypes of A as given in table 2.5.

	Case	A0	A1	A2	A3	Case	A0	A1	A2	A3
A0	1	vary	0.5	0.5	0.5	2	vary	0.5	0.75	0.5
A1		0.5	0.5	0.5	0.5		0.5	0.25	0.75	0.5
A2		0.5	0.5	0.5	0.5		0.75	0.75	0.25	0.75
A3		0.5	0.5	0.5	0.5		0.5	0.5	0.75	0.25
	Case	A0	A1	A2	A3	Case	A0	A1	A2	A3
A0	3	vary	0.75	0.75	0.75	4	vary	0.5	0.75	0.25
A1		0.75	0.25	0.5	0.5		0.5	0.1	0.45	0.95
A2		0.75	0.5	0.25	0.5		0.75	0.45	0.1	0.35
A3		0.75	0.5	0.5	0.25		0.25	0.95	0.35	0.1

Table 2.5: Different fitness values assigned to genotypes

   For each case, we vary the fitness of A0A0 sinusoidally from a slow rate of change of fitness to fast rate of change of fitness. After observing the frequency pattern observed for each case, we try to see if the same pattern is observed for different assignments or if there is an effect of the fitness values in the type of pattern observed.

2.2.3 Constraining survival rate by threshold fitness

After seeing the effects of a changing environment on the variation in the frequency of alleles we add one more constraint to the survivability of individuals. At times individuals need a certain minimum value of fitness to reach adulthood. We call this the threshold fitness, the fitness value below which the individuals die and do not survive to become adults. For both the 2-gene and 4-gene model, we first apply threshold fitness values from 0.1 to 0.7 along with varying fitness. For the 2-gene model, we consider two cases:

1. A0A0 -vary, A0A1- 1 and A1A1-0.5

2. A0A0- vary, A0A1- 0.5 and A1A1- 0.5

For the 4- gene model we consider the 4 cases as described in table 2.5. For each case, we apply different fitness threshold values, and for each value, we look at the frequency pattern observed as we increase the rate of change of fitness.

After having done the simulations for the different cases mentioned, we now move on to the mathematical approach which is described in the next section.

2.3 Mathematical approach

From the simulations done, we have obtained the data, say $\underline{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_N)$, where α corresponds to frequency of alleles and coordinates correspond to time or generation. For each fitness function provided, we obtain a corresponding frequency function. Fitness function or frequency function is the function of the fitness values or frequency values corresponding to N generations respectively.

We consider a finite dimensional vector space, say C^N such that our fitness function and frequency function belongs to this vector space. Since we mostly deal with sinusoidal functions, we consider Fourier basis in finite dimension. Let us call F to be our fitness function. We choose the basis as follows.

$$\begin{aligned}
F_1(t) &= (e^{2\pi i/N}, e^{2\pi i2/N}, \ldots, e^{2\pi iN/N}) \\
F_2(t) &= (e^{2\pi 2i/N}, e^{2\pi 2i2/N}, \ldots, e^{2\pi 2iN/N}) \\
&\vdots \\
F_j(t) &= (e^{2\pi ji/N}, e^{2\pi ji2/N}, \ldots, e^{2\pi jiN/N}) \\
&\vdots \\
F_N(t) &= (e^{2\pi Ni/N}, e^{2\pi Ni2/N}, \ldots, e^{2\pi NiN/N})
\end{aligned}$$

In general, we choose $\{F_j = ((\ldots, e^{2\pi ijk/N}, \ldots) : k = 1, 2, \ldots, N) : j = 1, 2, \ldots, N\}$ to be the basis for the vector space. In our simulations, $N = 400$. For each of these F_j's, we obtain the corresponding frequency function through simulation. For fitness function F, let $graph(F)$ be the frequency function. Let T be the operator taking fitness function F to the frequency function $graph(F)$. Let us denote the frequency function by TF.

We are trying to answer the question that if we choose any arbitrary fitness function, being any linear combination of basic fitness functions, can we predict how the corresponding frequency function would look like. For addressing this problem, we first see that if we take the linear combination of fitness functions and then look at its frequency function obtained through simulation, is it the same as the graph obtained if we take the linear combination of the corresponding frequency functions of the basic fitness functions. We check the linearity of operator T by comparing the linear combination of simulation and simulation of linear combination, where we are concerned with the pattern.

Let us consider

$$F(g_k) = a_0 + \sum_{j=1}^{N} a_j e^{2\pi i g_k j/N}$$

wherefor coeffcient a_0 we consider different cases as in table 2.6.

Case	a_0	a_j
1	0	1/j
2	0	$1/j^2$
3	0	$1/2^j$
4	0	random/j
5	0	random/j^2
6	0	random/2^j

TABLE 2.6: Coefficients chosen for the linear combination

Here we generate 400 random numbers and use it in the coefficients as needed. 'Random' in the table refers to these random numbers. Since the fitness functions are complex, for the simulation we consider its real or imaginary part (I have considered imaginary part in the simulation) and normalize them so that fitness values lie in between 0 and 1. The code for finding the linear combination F as above is provided in Appendix A. The graphs have been plotted using "matplotlib".

Now, $F(g_k) = \sum_{j=1}^{N} a_j e^{2\pi i g_k j/N}$.

$$TF(g_k) = b_0 + \sum_{j=1}^{N} e^{2\pi i g_k j/N} \qquad \longrightarrow \text{Simulation of linear combination}$$

Since we are not worried about the constant, we consider

$$TF'(g_k) = TF(g_k) - b_0 = \sum_{j=1}^{N} e^{2\pi i g_k j/N} \qquad \text{where } b_0 = \frac{1}{N}\sum_{k=1}^{N} TF(g_k)$$

Let $SF(g_k) = a_0 + \sum_{j=1}^{N} a_j T(e^{2\pi i g_k j/N})$. Under the assumption that $T(a_0) = a_0$,

$$SF(g_k) = \sum_{j=1}^{N} TF(g_k) \longrightarrow \text{Linear combination of simulation}$$

For each case, we have (g, TF') and (g, SF). We need to compare TF' and SF. We see if T is linear in different cases mentioned. For finding the simulation of linear combination we do the same modifications as above code and plot the graph required using "matplotlib". For finding the linear combination of simulation I have used a Python code as below with small modifications as and when required. The code used is provided in Appendix A.

These are the different methods we use in approaching the question addressed in this project. The results and observations are what we discuss in the coming chapter.

Chapter 3

Results and Discussion

In this chapter, I provide the various results obtained and the observations following the results. Though a lot of data have been generated by simulation, some of the selected results have been provided here from the pool of graphs. The results have been given section wise as in chapter 2.

3.1 Results for the classical evolutionary models

1. Genotype fitness fixed: When fitness is fixed for various cases, the figures 3.1 to 3.5 represent the cases when gene A is kept in the additive case and fitness of genotypes of B is assigned as in caption of the figures with reference to table 2.2.

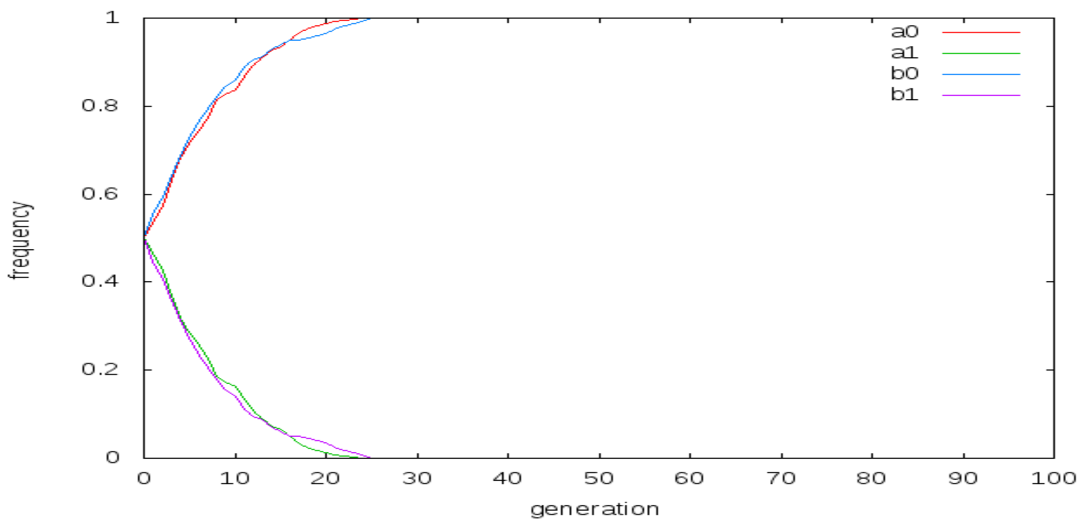


FIGURE 3.1: Gene B - additive

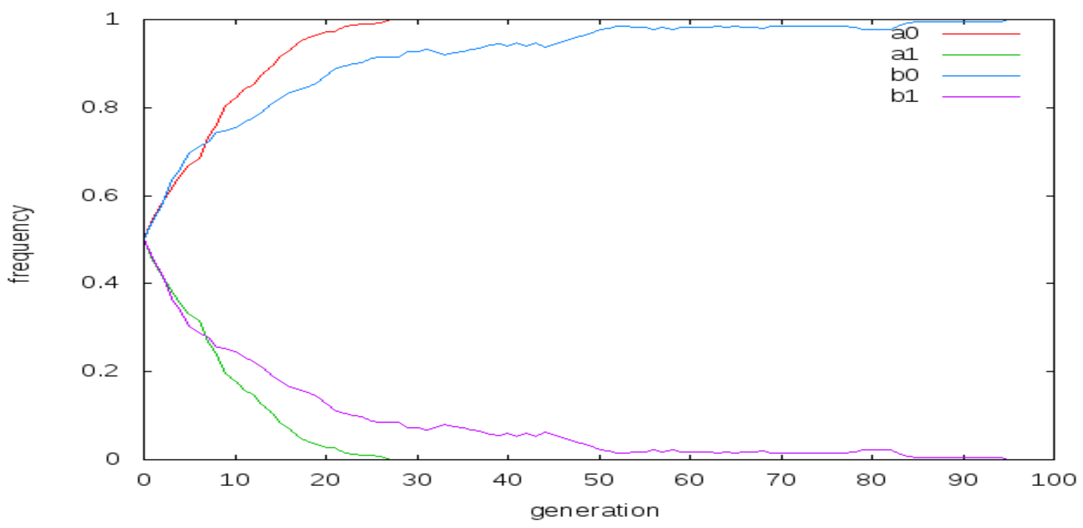


FIGURE 3.2: B0 is dominant and beneficial

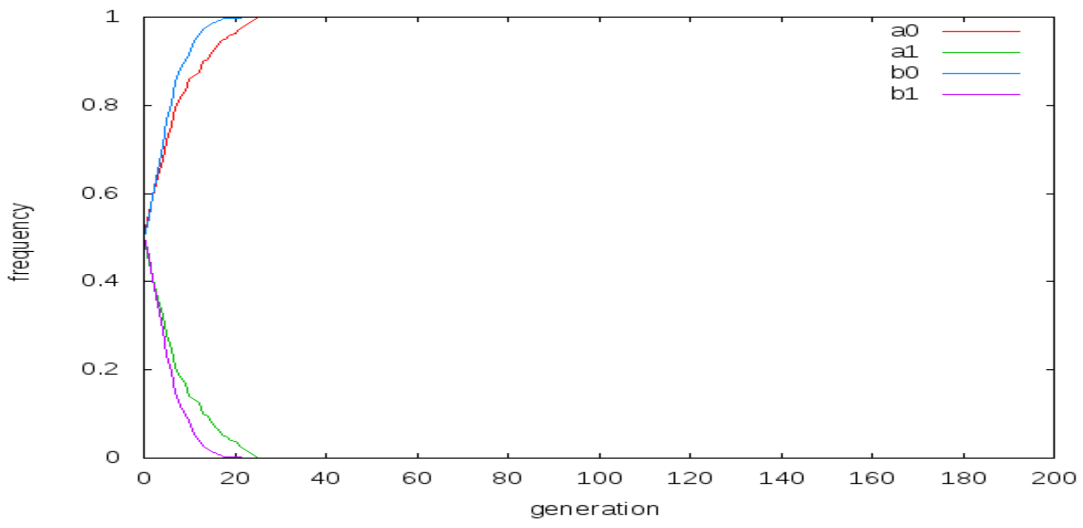


FIGURE 3.3: B1 is dominant and deleterious

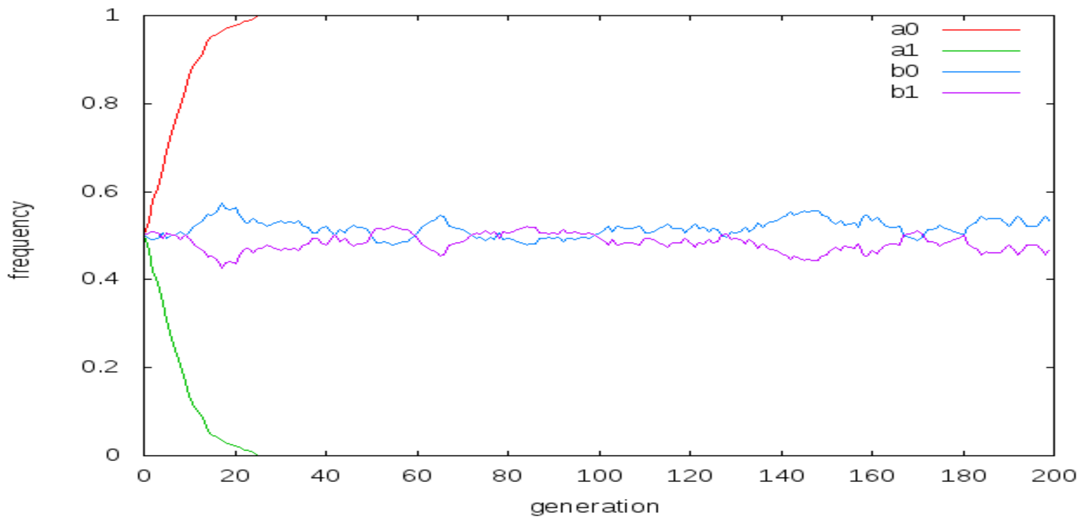


FIGURE 3.4: B- Heterozygous superiority

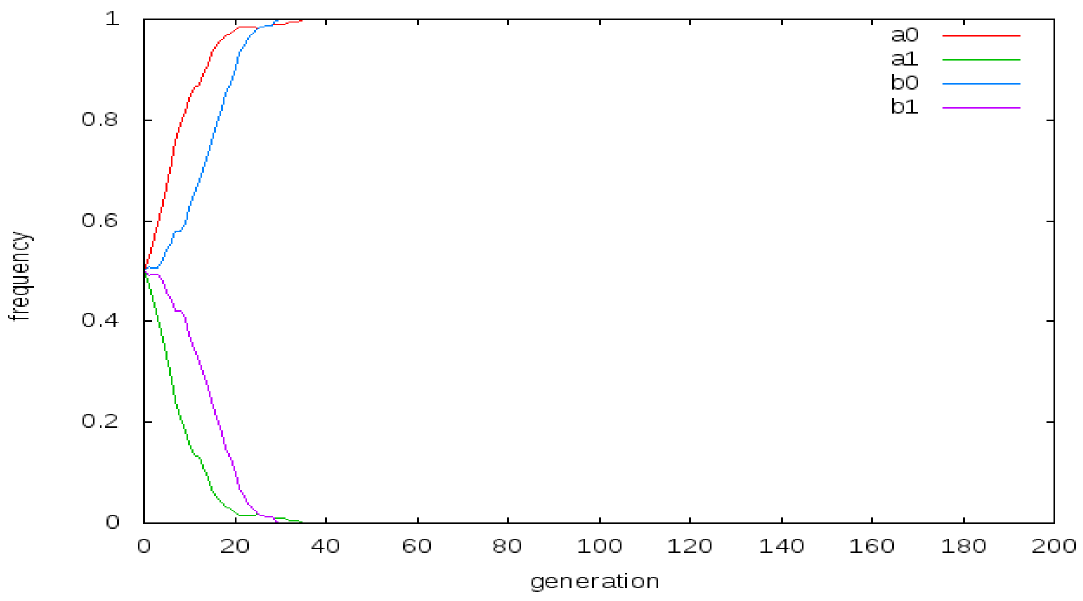


FIGURE 3.5: B - Homozygous superiority

We see that for 3.1, A0 and B0 goes to fixation in about 20 generations since they have the highest fitness. In 3.2, frequency of dominant allele B0 reaches 1 in about 90-100 generations. It takes a longer time to reach fixation since the recessive allele hides from selection as its effect is expressed only in the homozygote recessive. In 3.3, dominant allele B1 is selected against quickly as its effect is expressed in both homozygote and heterozygote genotype. In 3.4, both B0 and B1 remain in the population in a stable equilibrium as it is expected from theory. In 3.5, one of them reaches frequency 1 and other 0 in about 20-40 generation. This is the case of unstable equilibrium.

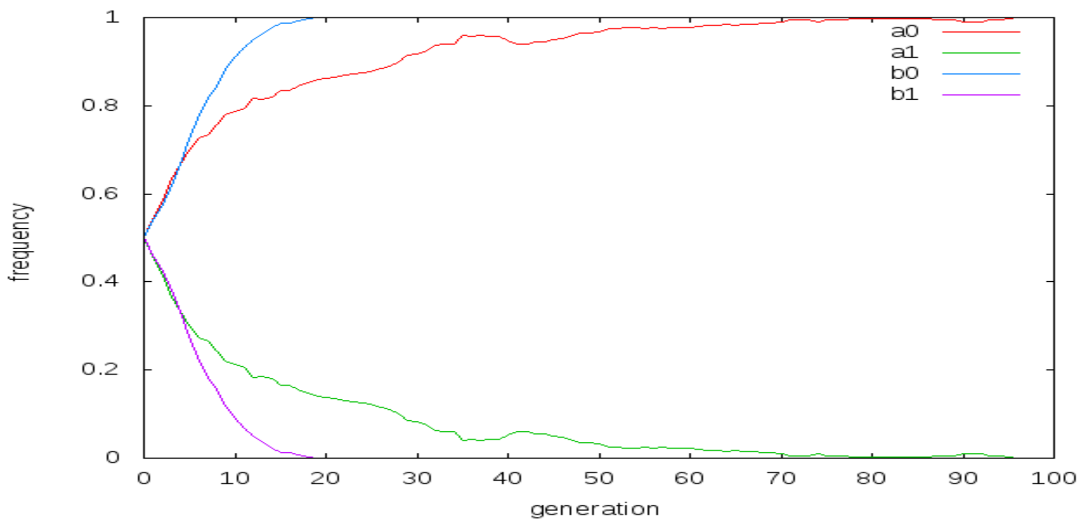


FIGURE 3.6: A0 is dominant and beneficial and B1 is dominant and deleterious

Since we have done simulation for all possible assignments, I give the frequency graph, figure 3.6, for the case when A is in dominant and beneficial case and B is assigned the dominant and deleterious case as in table 2.2.

The way the genes behave is independent of each other. There is no interaction between the two genes. The results for the classical case are those that are expected from the theory. This verifies the simulation written.

2. Allele fitness fixed: The generation versus frequency graph for some of the combination of cases as in 2.3 have been shown in figures 3.7, 3.8 and 3.9.

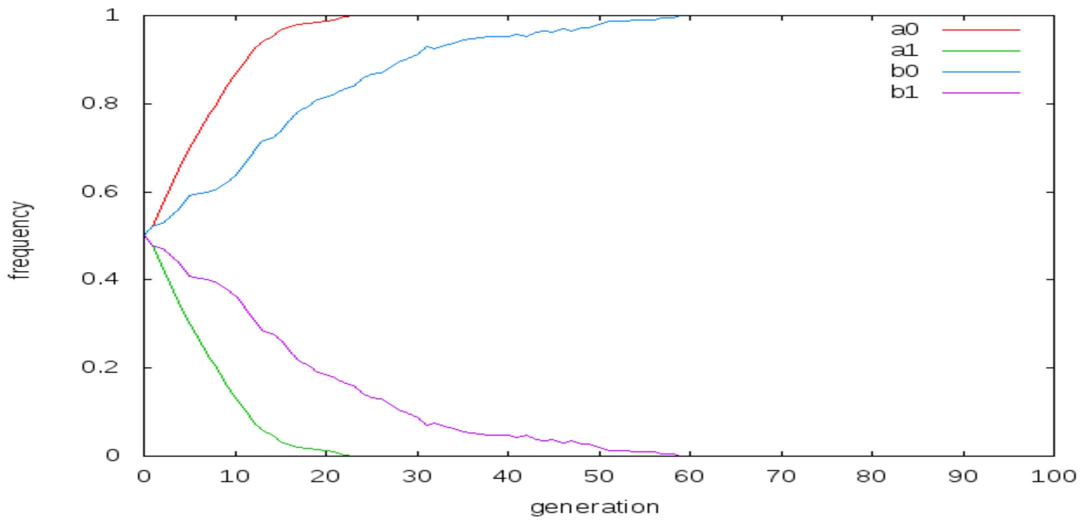


FIGURE 3.7: A- Type 1, B- Type 1

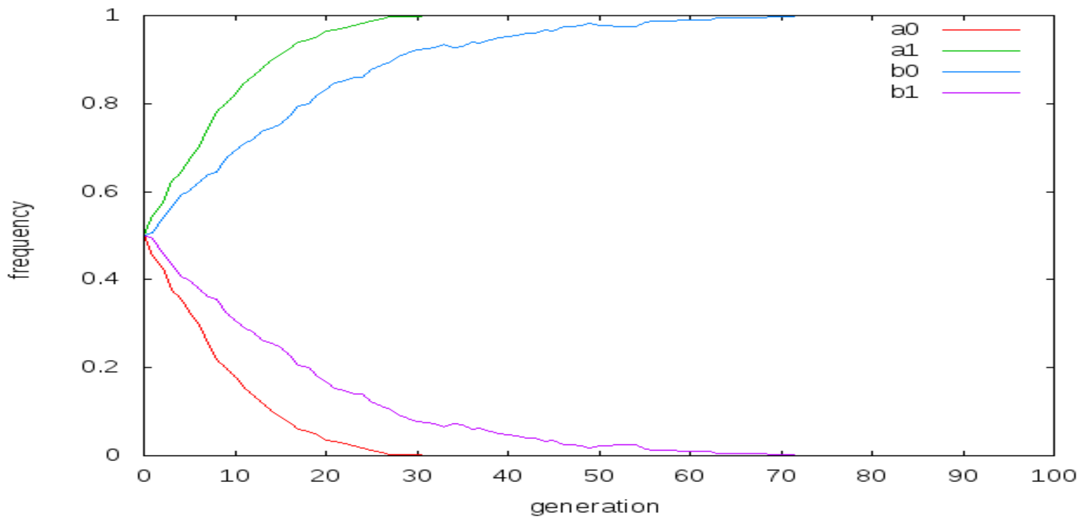


FIGURE 3.8: A- Type 2, B- Type 1

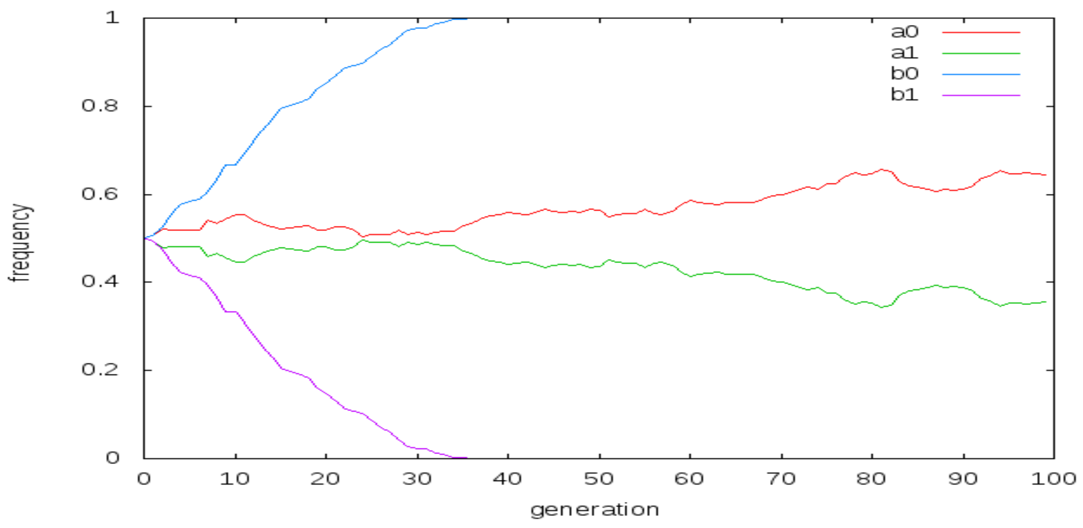


FIGURE 3.9: A- Type 3, B- Type 1

We see that the allele with the highest fitness reaches a frequency 1 in about 30-70 generations. When both alleles have the same fitness, then one of them is selected for or against randomly.

3. Interaction of genes: When the presence of gene A increases the fitness of gene B as described in the methods, the results for the cases as in table 3.1 are given in figures 3.10, 3.11 and 3.12.

Case	A0A0	A0A1	A1A1	B0B0	B0B1	B1B1
1	1	0.75	0.5	1	0.75	1
2	1	0.5	0.5	1	1	1
3	1	0.5	1	1	1	1

TABLE 3.1: Fitness assignment when A increases fitness of B

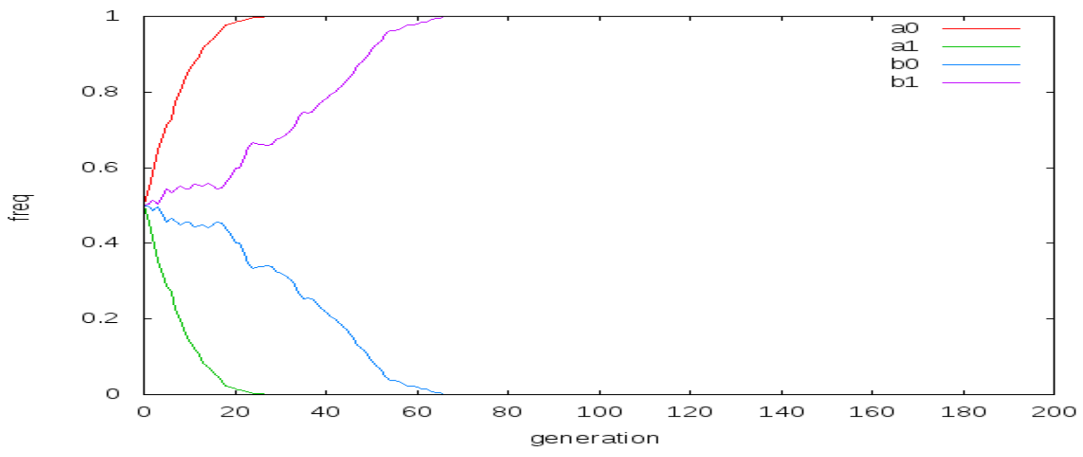


FIGURE 3.10: Case 1 of table 3.1

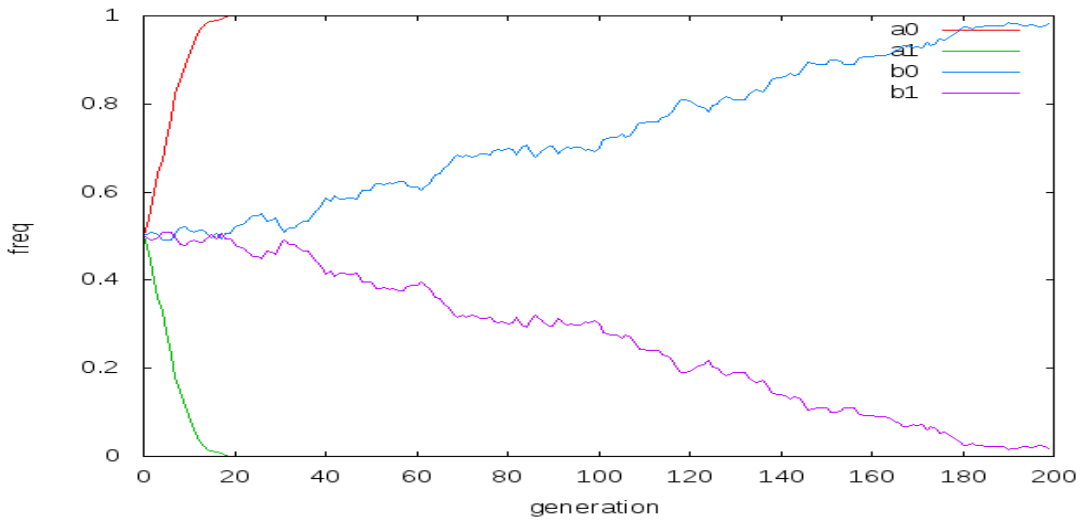


FIGURE 3.11: Case 2 of table 3.1

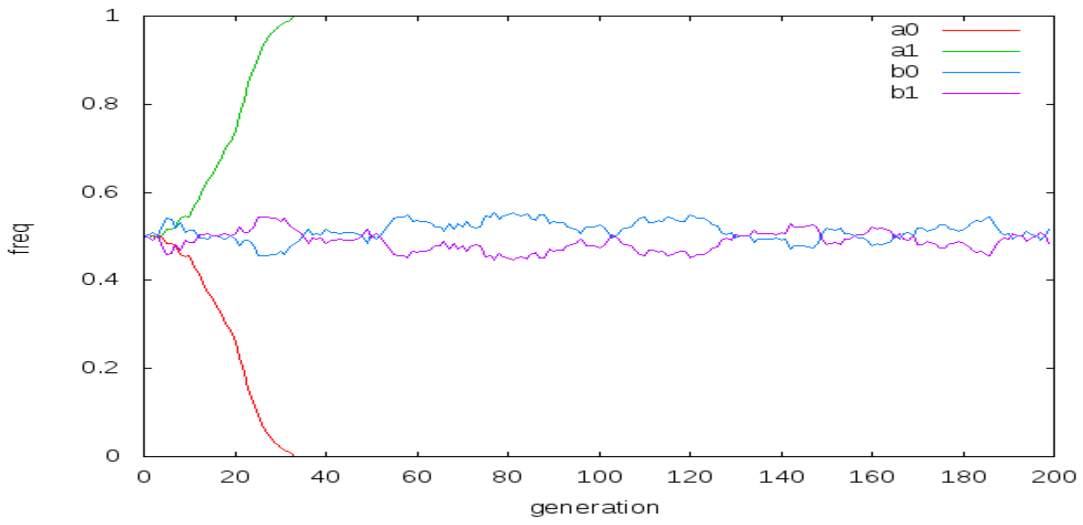


FIGURE 3.12: Case 3 of table 3.1

For gene B, when each genotype fitness is 1, either both alleles remain in the population or one of them gets selected for or against just by chance. In other cases, the change is similar to specific case they belong to as in table 2.2.

Case	A0A0	A0A1	A1A1	B0B0	B0B1	B1B1
1	1	0.75	0.5	0	0.5625	0.25
2	1	1	0.5	0	0	0.25

TABLE 3.2: Fitness assignment when A suppresses fitness of B

When the presence of A suppresses the fitness of B, the results obtained for the cases mentioned in table 3.2 are shown in figures 3.13 and 3.14. The observations are similar to the last case.

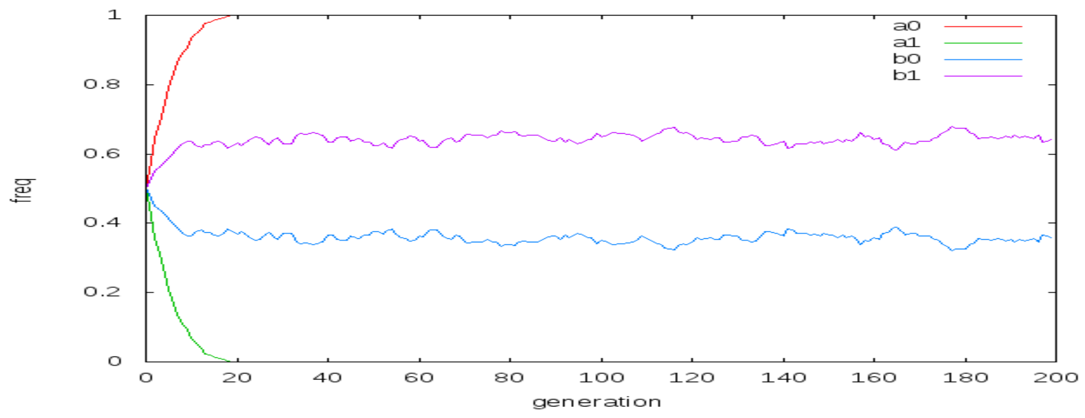


FIGURE 3.13: Case 1 of table 3.2

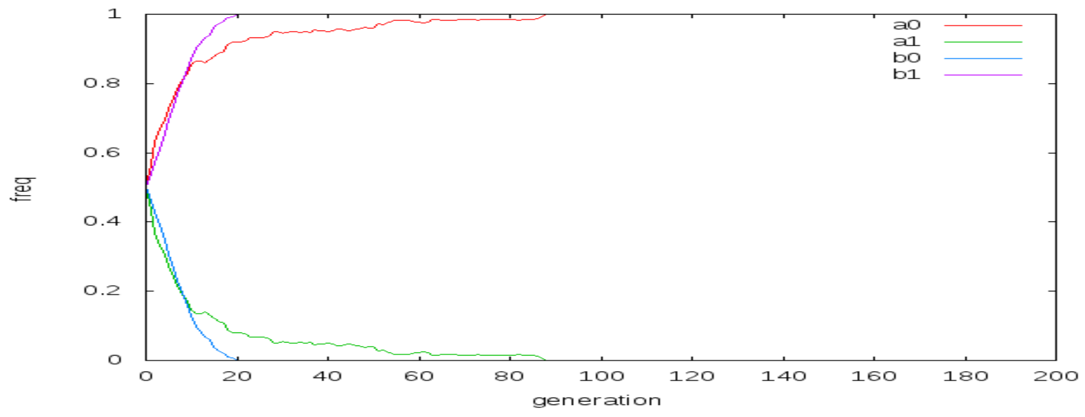


FIGURE 3.14: Case 2 of table 3.2

4. Fitness is multiplicative: For the different combinations as in table 2.2, the results obtained when fitness is multiplicative when A is assigned additive case and B as in caption, are given in figures 3.15 to 3.19. The change in frequency of alleles is irrespective of the presence of two genes. Even if fitness is multiplicative, the frequency is similar to that when fitness is additive and fixed.

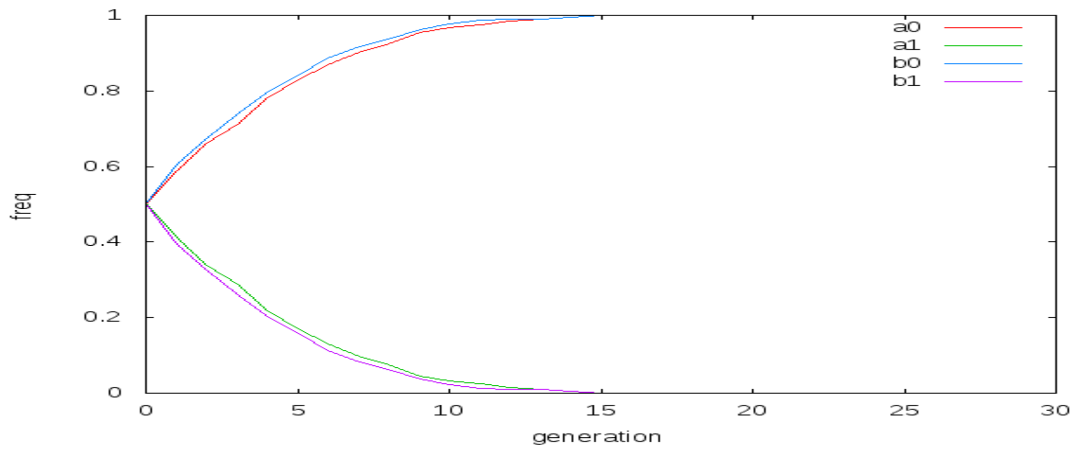


FIGURE 3.15: B is additive

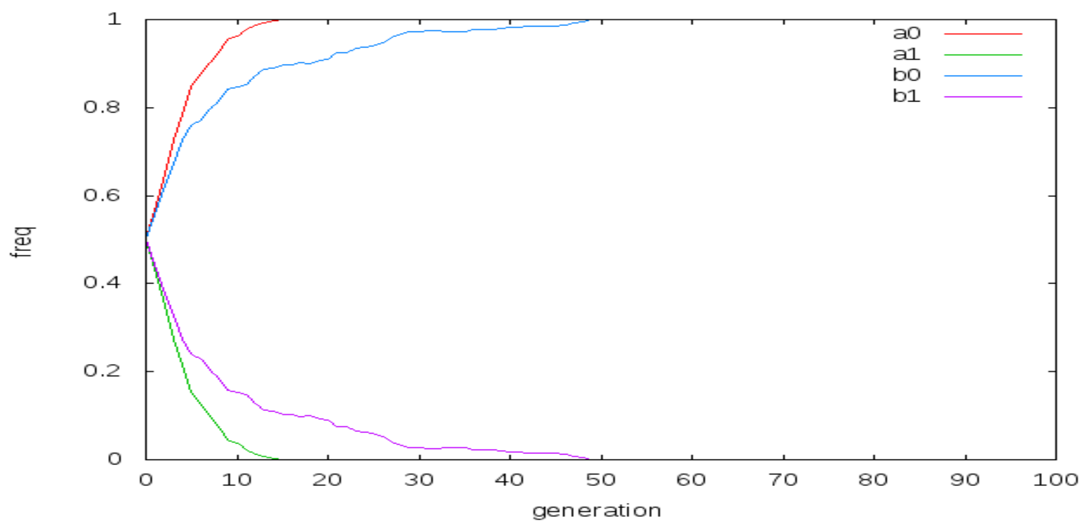


FIGURE 3.16: B0 is dominant and beneficial

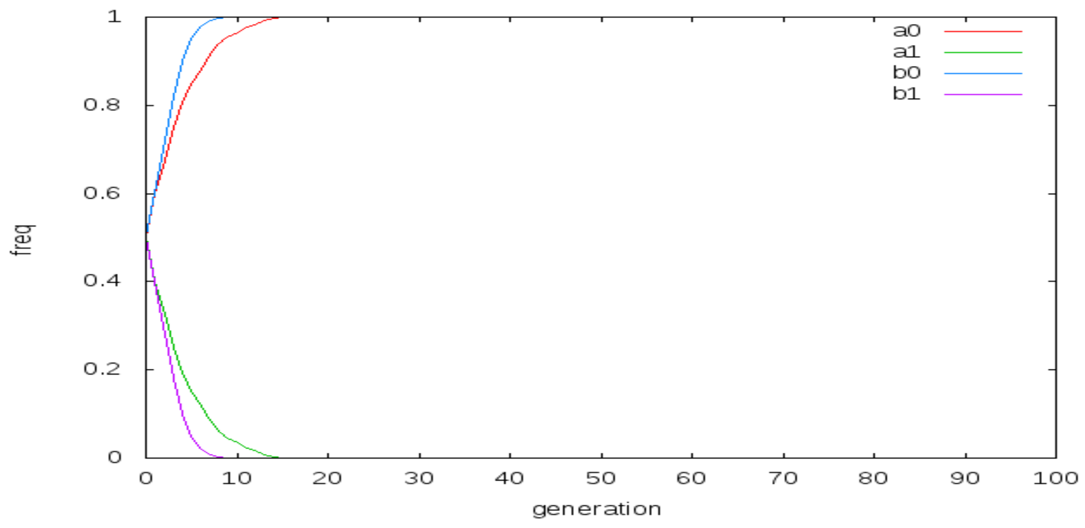


FIGURE 3.17: B1 is dominant and deleterious

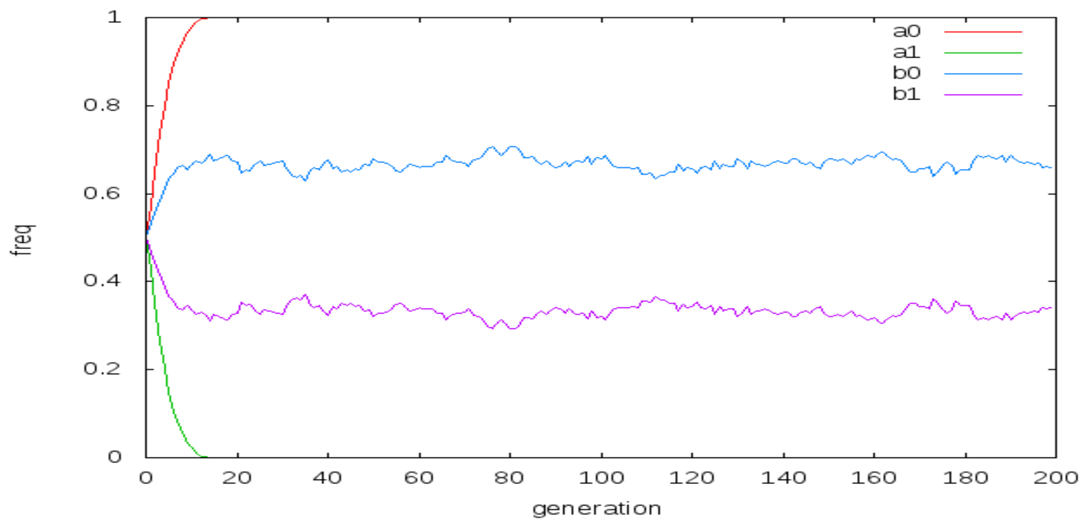


FIGURE 3.18: B - Heterozygous superiority

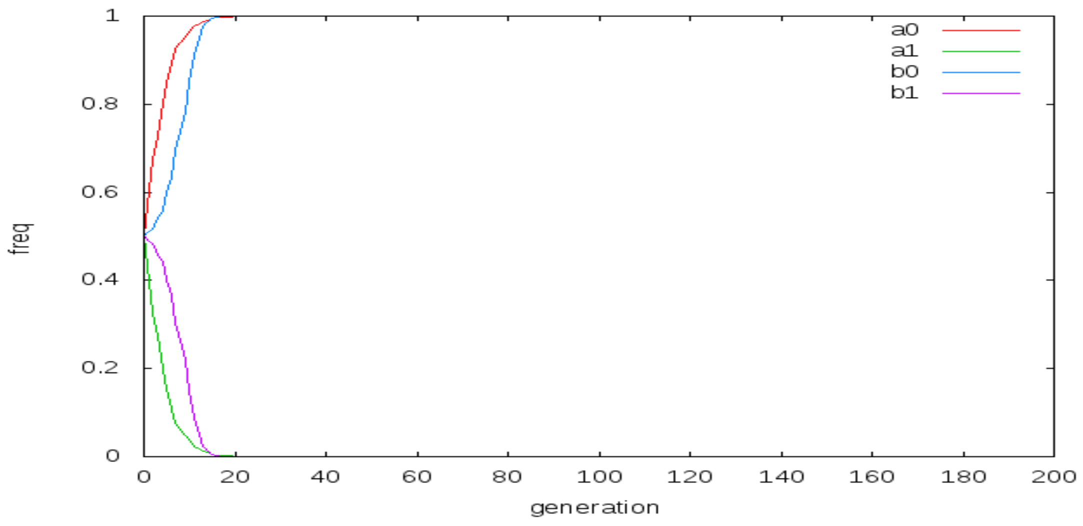


FIGURE 3.19: B- Homozygous superiority

5. 4-gene model- When fitness is fixed: When we assign random values to genotype fitness and then run the simulation, we get the following frequency graphs for genes A, B, C and D as in figure 3.20.

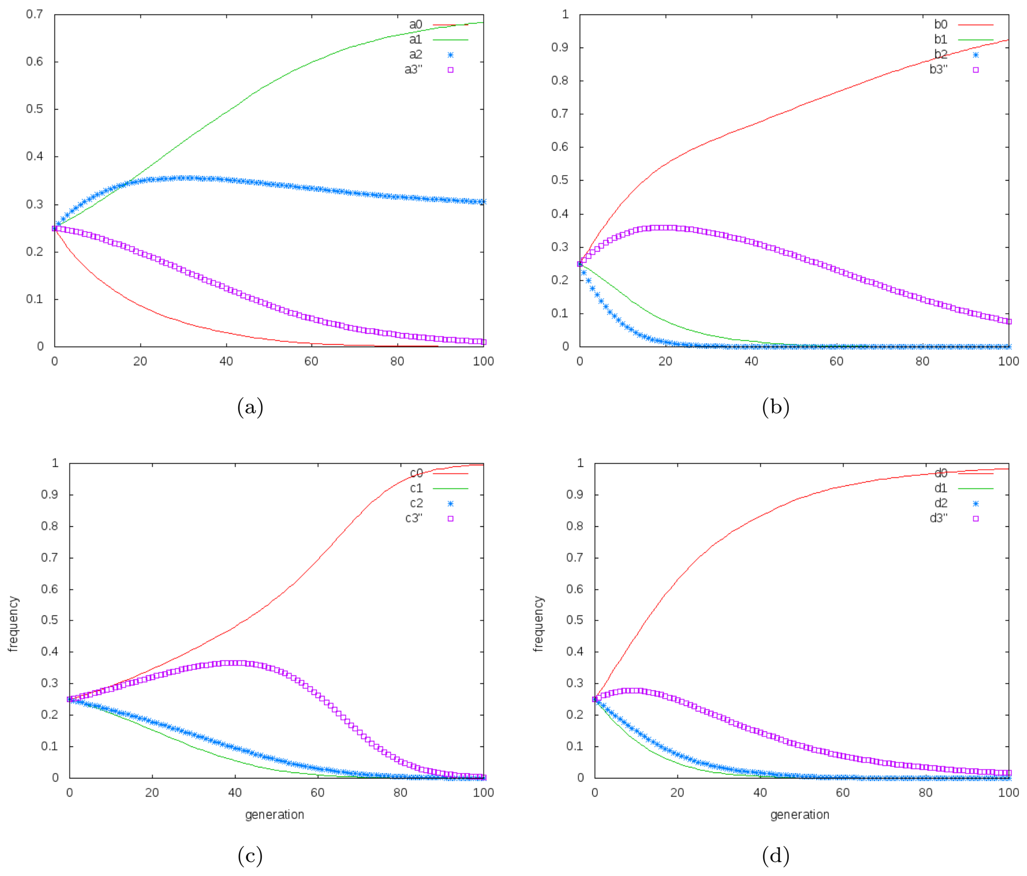
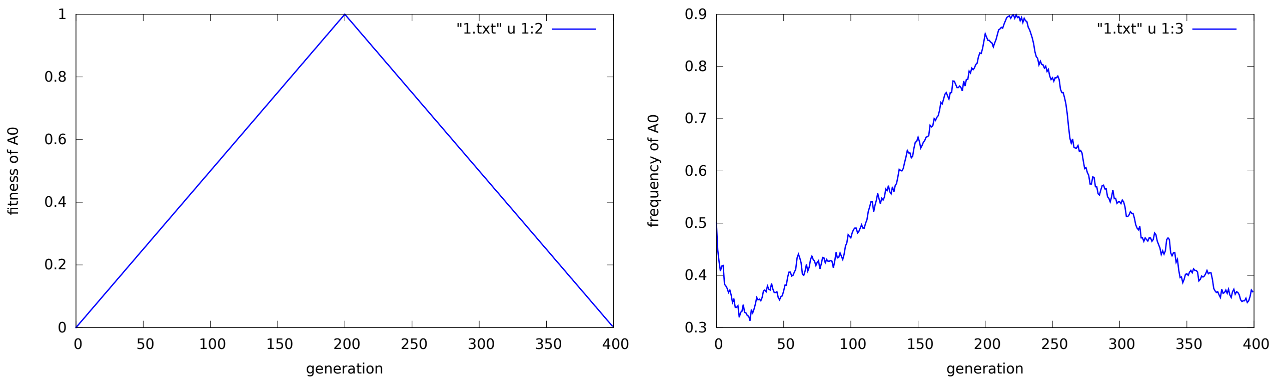


FIGURE 3.20: Frequency graphs for genes: (a) A; (b) B; (c) C; and, (d) D



(a) Fitness graph (b) Frequency graph

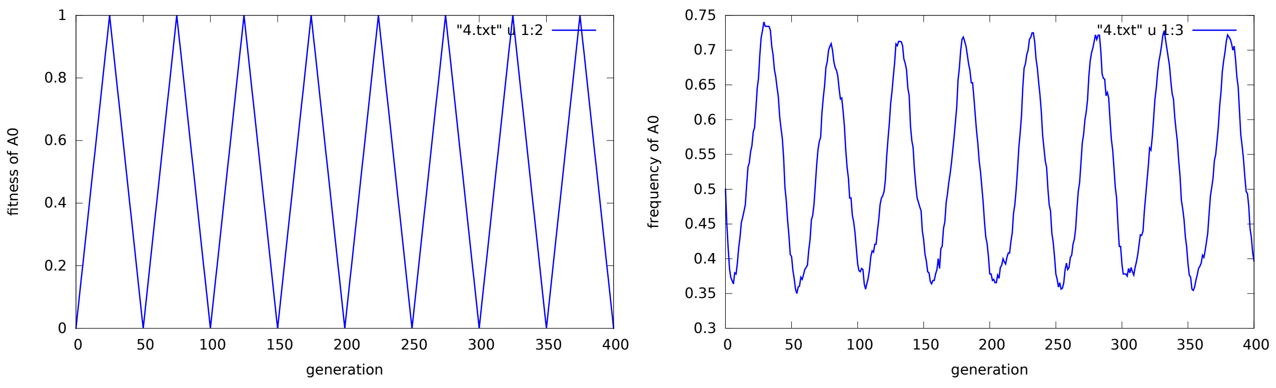
FIGURE 3.21: Rate of change of fitness - 400 generations

3.2 Results for changing environment and reversibility

The results for the 2-gene model are given first.

1. **Linearly changing fitness**

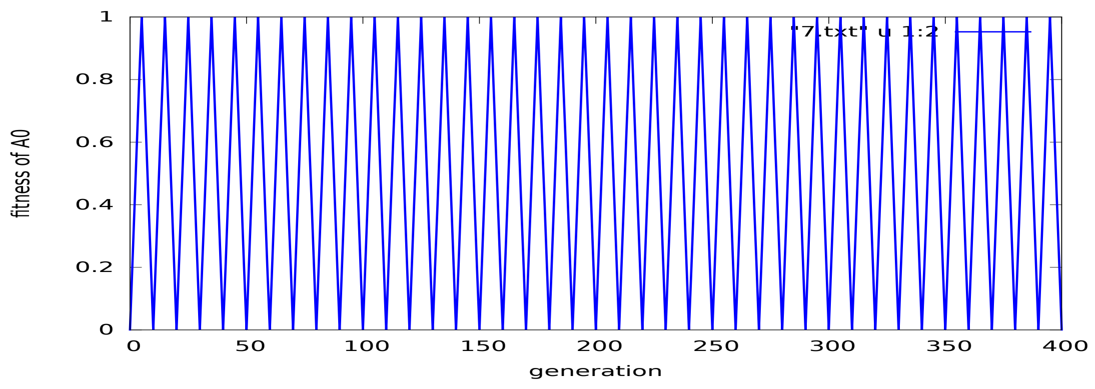
   The fitness graph and its corresponding frequency graphs are given in figures 3.21, 3.22 and 3.23 when fitness comes back in 400, 50 and 10 generations for a population of size 160.



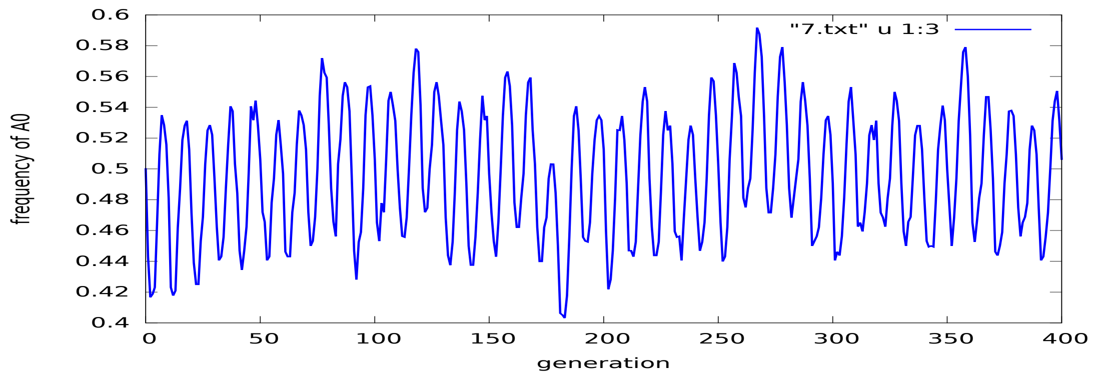
(a) Fitness graph (b) Frequency graph

FIGURE 3.22: Rate of change of fitness - 50 generations

We see that the number of cycles of frequency is proportional to the number of times fitness comes back to 0. As the rate of change of fitness increases, the highest frequency attained decreases from 0.9 to 0.5. Each peak attains a different highest frequency as the rate of change becomes faster. During rapid rate of change of



(a) Fitness graph

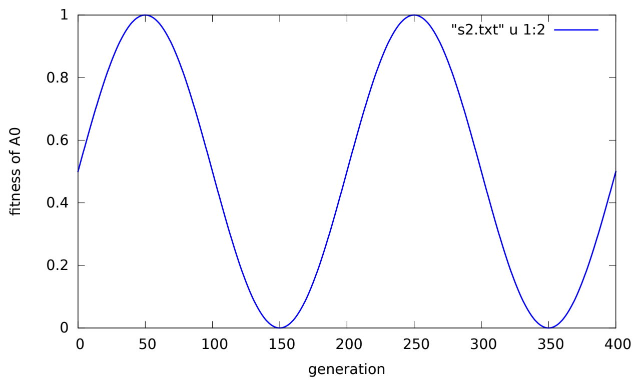


(b) Frequency graph

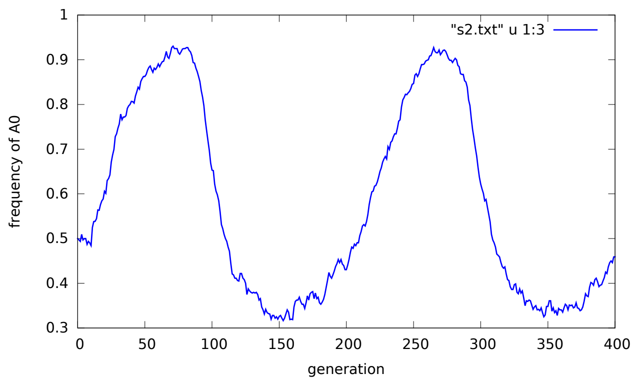
FIGURE 3.23: Rate of change of fitness - 10 generations

fitness, peaks become irregular. During slow rate of change of fitness frequency values come back to their initial values after each cycle.

2. **Sinusoidally changing fitness**



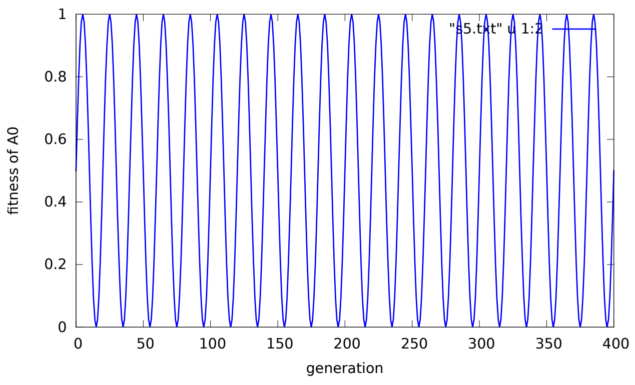
(a) Fitness graph



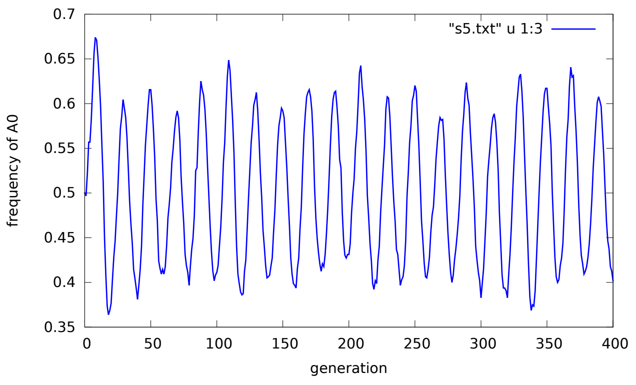
(b) Frequency graph

FIGURE 3.24: Rate of change of fitness - 200 generations

Similar to the above case the fitness and frequency graphs are provided when fitness comes back in 200, 50 and 5 generations in figures 3.24, 3.25 and 3.26.

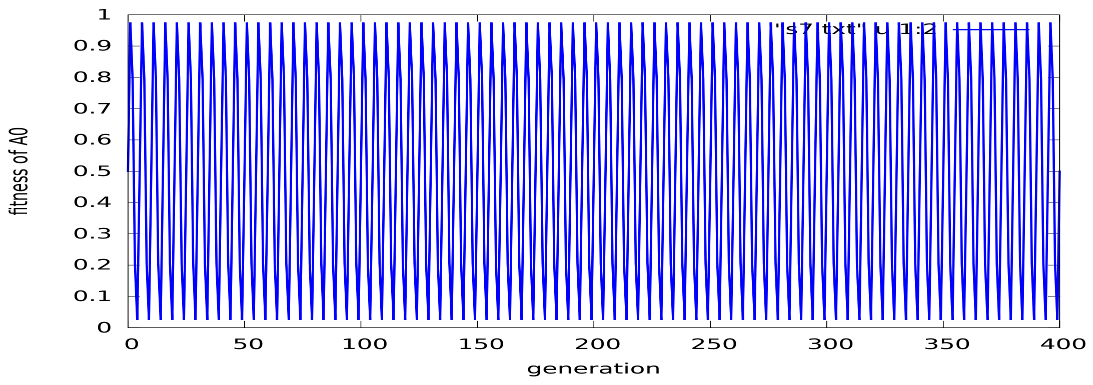


(a) Fitness graph

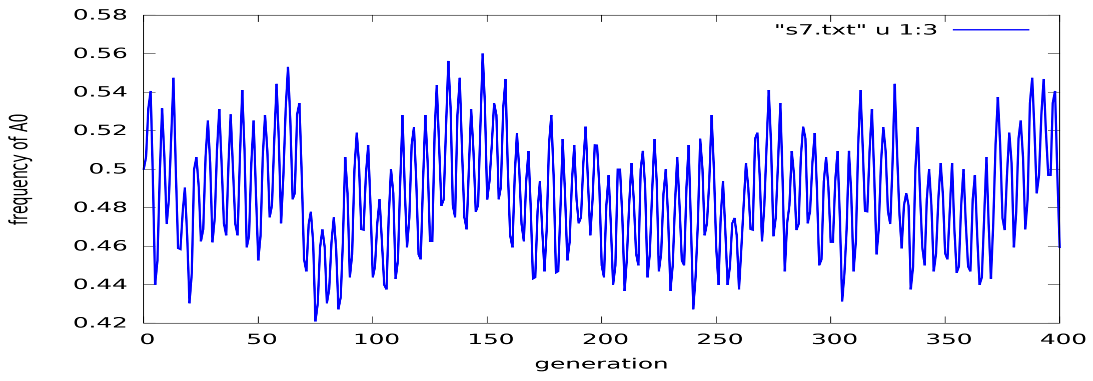


(b) Frequency graph

FIGURE 3.25: Rate of change of fitness - 20 generations



(a) Fitness graph



(b) Frequency graph

FIGURE 3.26: Rate of change of fitness - 5 generations

The number of times frequency comes back to 0.5 is proportional to the number of times fitness comes back to 0.5. As the rate of change of fitness increases, the frequency of highest peak decreases. When change is rapid peaks are attained at different frequencies and change is irregular. At a slow rate of change of fitness, evolution retraces its path.

3. **Effect of population size**

For the case when fitness varies sinusoidally such that it comes back in 5 generations as in figure 3.26(a), figures 3.27, 3.28 and 3.29 show the frequency pattern attained for population sizes of 160, 1600 and 16000 after averaging for 50 generations.

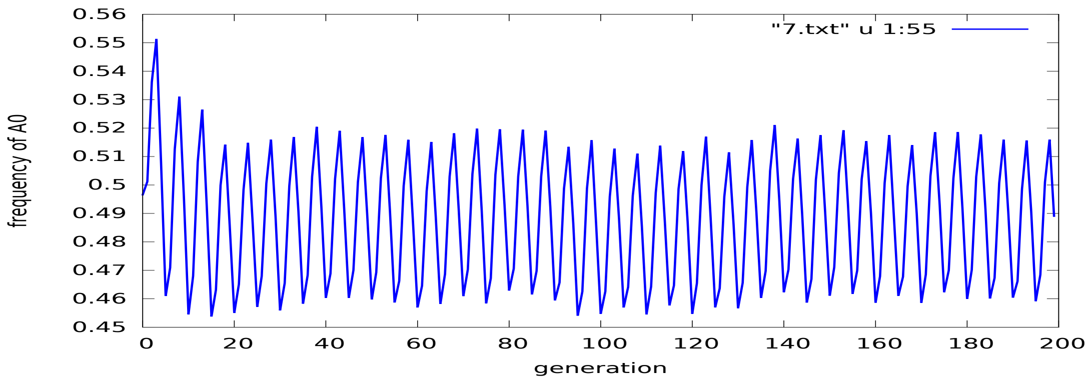


FIGURE 3.27: When population size is 160

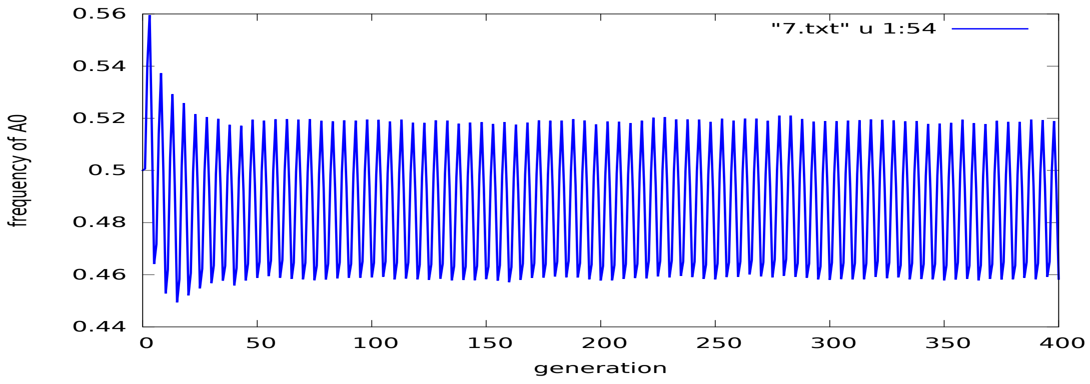


FIGURE 3.28: When population size is 1600

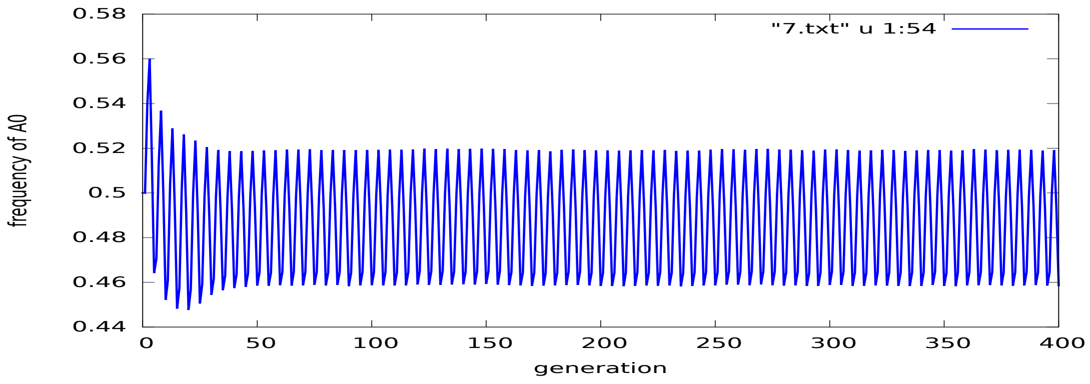


FIGURE 3.29: When population size is 16000

We see that as we increase the sample size, all small effect changes are cancelled off. More uniform peaks are obtained. For smaller rates of change, all peaks attained are of the same frequency. At higher rates for larger population sizes, we see that after some generations, say about 50, frequency oscillates between two fixed values. This initial 50 generation is the time taken by the population to adapt to the environmental condition as 0.5 happened to be the initial frequency by chance.

4. **Effect of initial frequency**

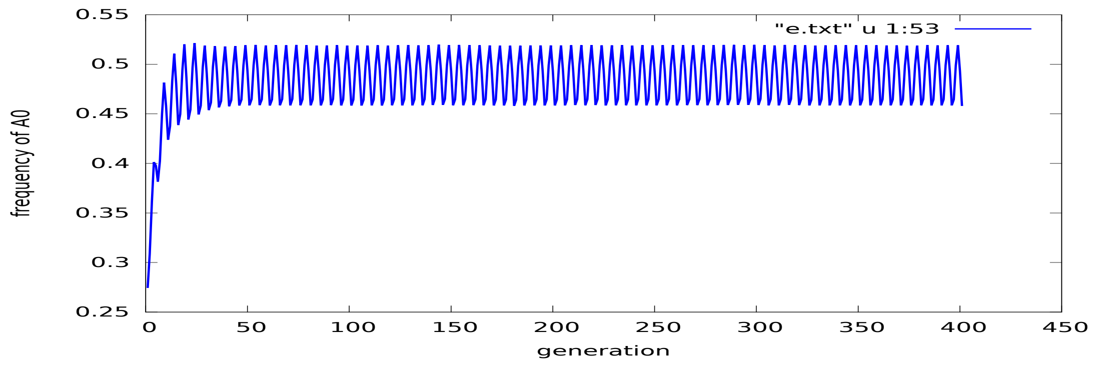


FIGURE 3.30: Initial frequency of A0 = 0.275

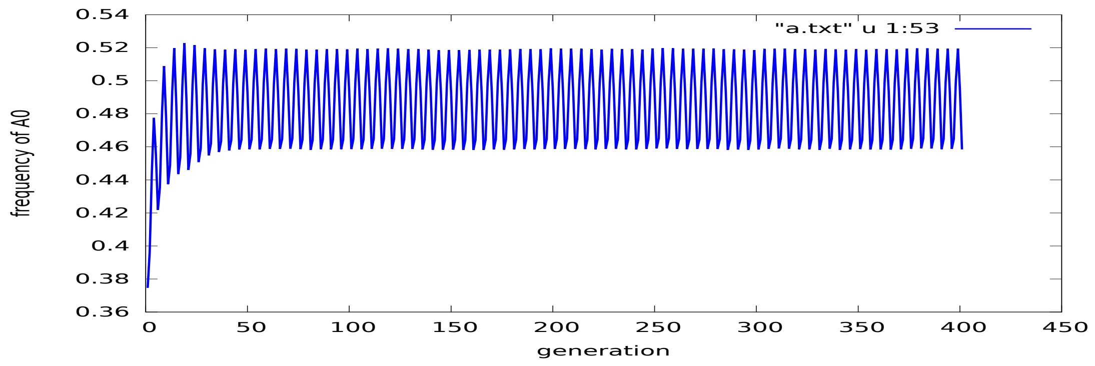


FIGURE 3.31: Initial frequency of A0 = 0.375

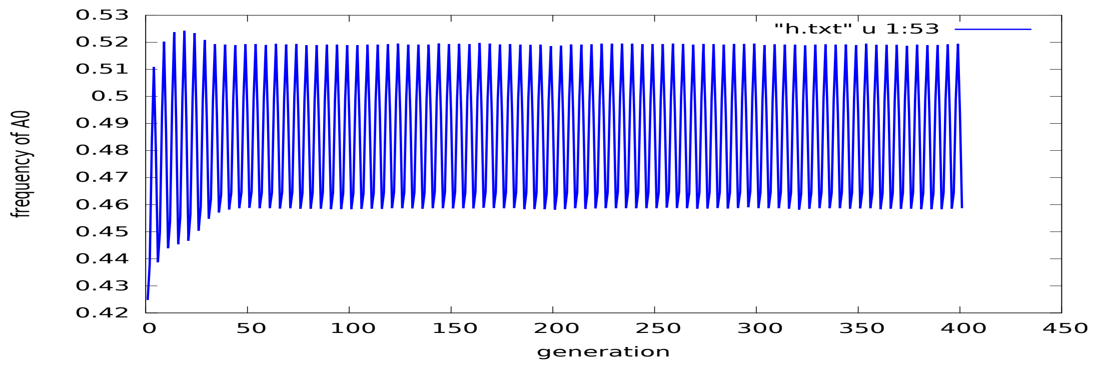


FIGURE 3.32: Initial frequency of A0 = 0.425

The frequency graphs for the different initial frequencies are given in figures 3.30 to 3.32. We see that whatever initial frequency we start with, as long as A0A1 and A1A1 fitness are kept constant, after some generations frequency of A0 oscillate between two fixed values 0.52 and 0.46 all the time.

5. **Effect of fixed fitness values of genotypes A0A1 and A1A1**

For the fitness values as given in captions, the results are given in figures 3.33, 3.34 and 3.35.

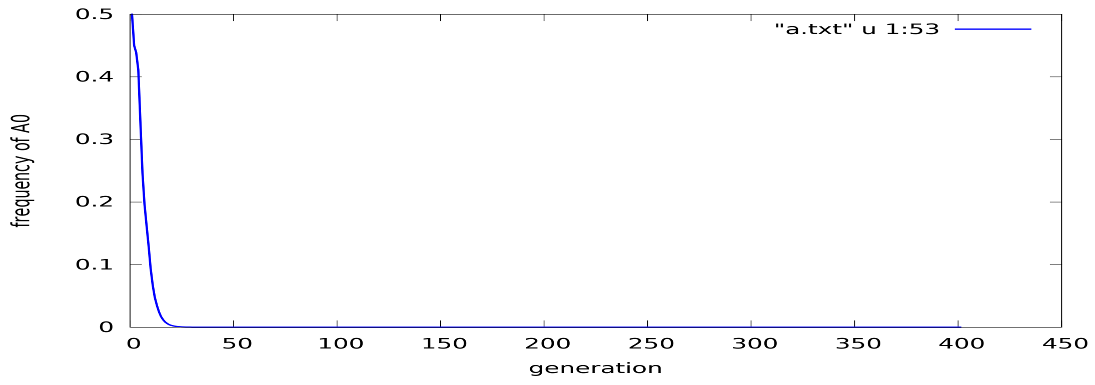


FIGURE 3.33: Case A- A0A1- 0.5, A1A1- 1

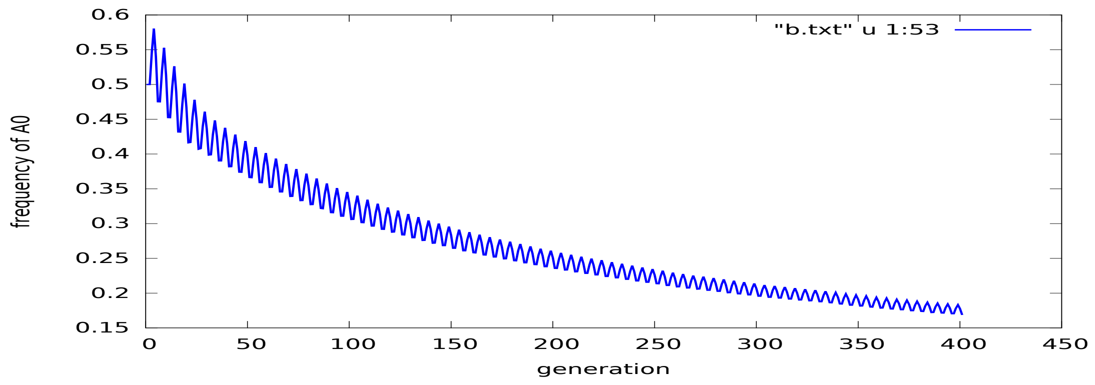


FIGURE 3.34: Case B- A0A1- 1, A1A1- 1

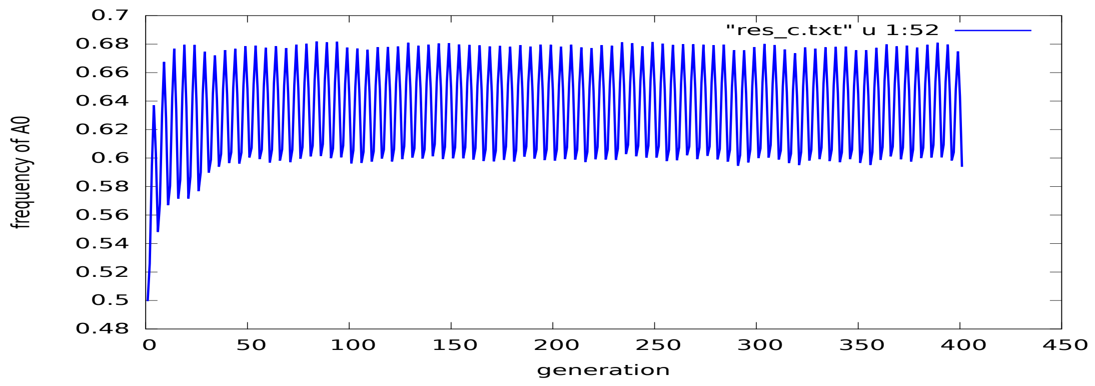


FIGURE 3.35: Case C- A0A1- 0.75, A1A1- 5

In case A, when fitness of A1A1 is 1, the frequency of A0 goes to 0 in a few generations as expected. For case B, when both genotype fitness' is equal we observe a new trend where the frequency of A0 decreases with generation with weakening oscillation after each cycle. Evolution does not retrace its path here. For case C, when heterozygote fitness is more, we obtain a similar trend as previous cases. But now the frequency oscillates between two other values. The pattern of frequency depends on the rate of change of fitness and other fixed fitness values of genotypes.

6. **4-gene model- sinusoidal change of fitness**

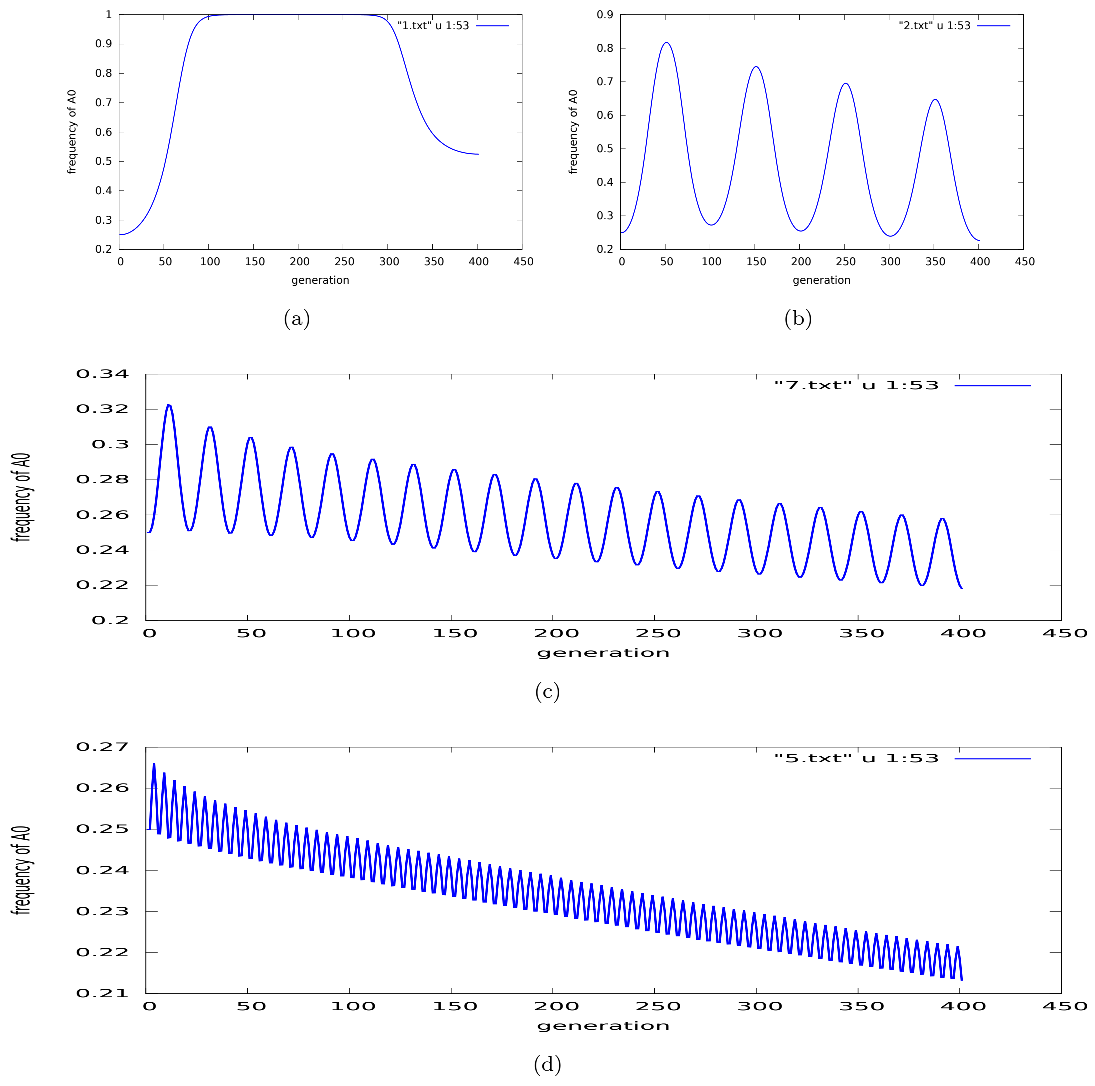
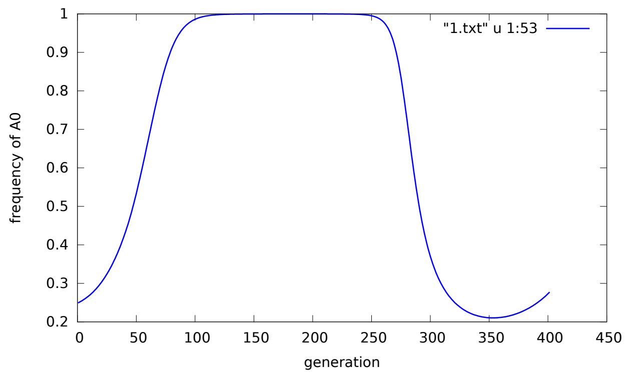


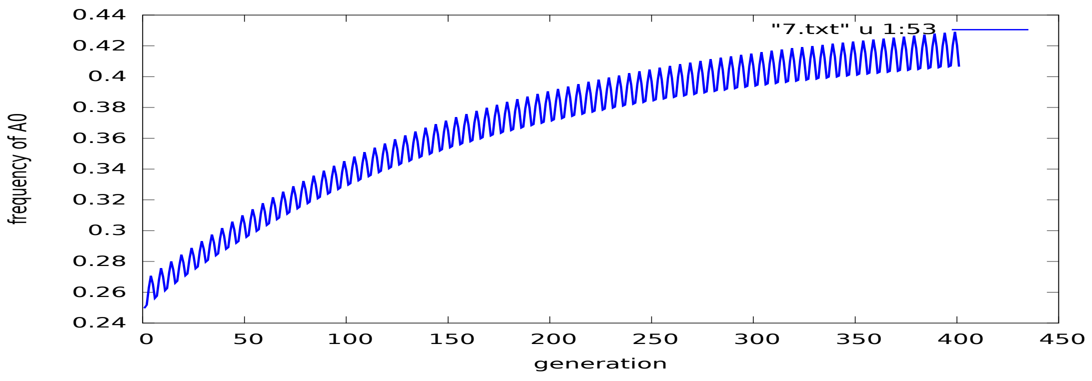
FIGURE 3.36: When rate of change of fitness is 400, 100, 50 and 5 generation.

We now move on to the 4-gene model. For some of the cases as in table 2.5 we give the frequency plots as we increase the rate of change of fitness from 400, 100, 50 and to 5 generations.

Figure 3.36 shows the frequency patterns for Case 1 of the table as the rate of change of fitness increases.The pattern observed is similar to case B of figure 3.35. For case 2 in table 2.5 the graphs are as in figure 3.37.



(a)



(b)

FIGURE 3.37: When rate of change of fitness is 400 and 5 generation.

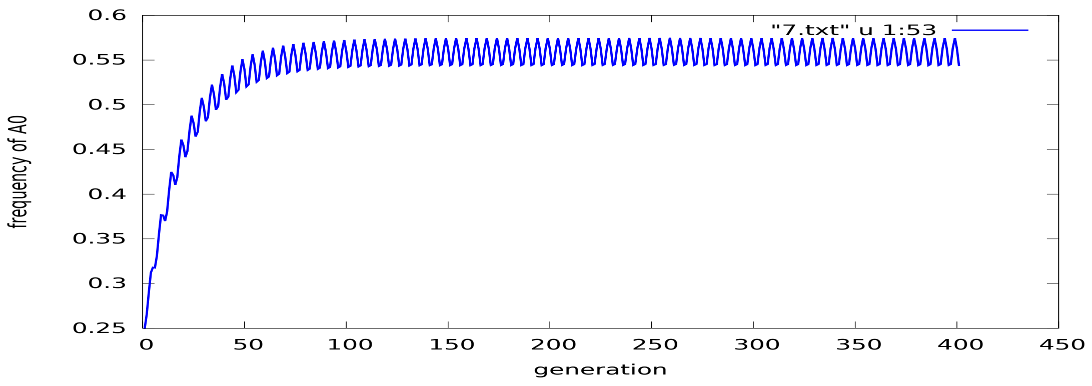


FIGURE 3.38: Case 3 frequency pattern

For different assignments, different trends are observed. For case 3, when fitness comes back in 5 generations frequency pattern observed (figure 3.38) is similar to case C of table 2.5.

3.3 Results for constraining survival rate by threshold fitness

For the 2-gene model, when the fitness of A0A0 =1 and A1A1 = 0.5, we apply threshold fitness of 0.1 to 0.7 with a difference of 0.1. The results for some of the cases are given in figures below.

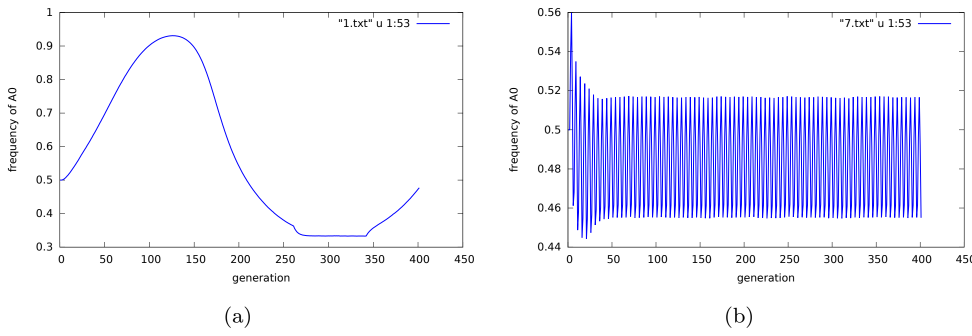


FIGURE 3.39: Th = 0.1(a) rate = 400 generations, (b) rate = 5 generations

For lower threshold of 0.1 we see from figure 3.39 that at low rates of change of fitness, the lower portion of the peak is cut off and for some generations frequency remains constant. The frequency values do not go below a certain value. But there is no effect of the threshold for higher rates of change of fitness. For a higher threshold of 0.5, as in figure 3.40 we see that for a slow and moderate rate of change of fitness, the lower portion is swiped off but like above no change is seen in the highest rate of change.

For case when A0A1= 0.5 and A1A1= 0.5, the frequency pattern obtained for threshold fitness of 0.3 and 0.5 are given in figure 3.41 and 3.42 respectively. We see that when threshold becomes 0.5, the fitness values of genotype A0A0 for which individual survive will be greater than 0.5, which is greater than both A0A1 and A1A1. Hence the frequency of A0 reached 1 quickly with no further variation. The same thing is observed for case 1 of 4- gene model as in table 2.5.

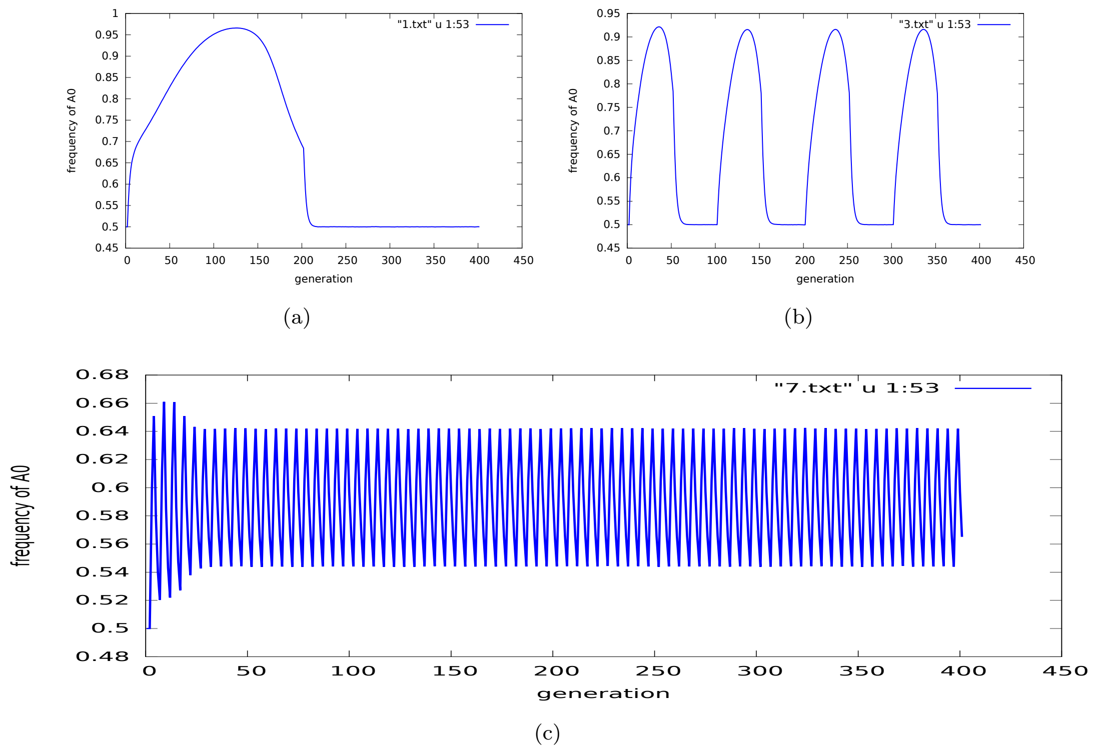


FIGURE 3.40: Th = 0.5(a) rate = 400, (b) rate = 50, (c) 5 generations

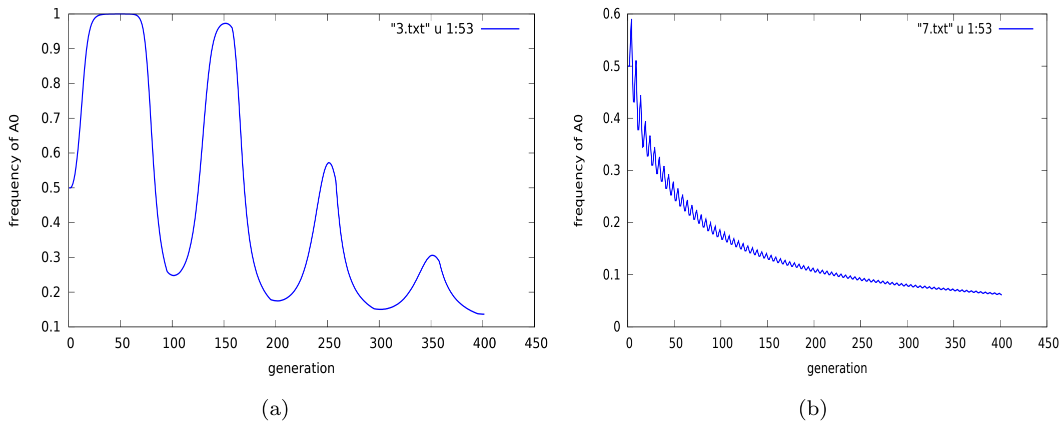
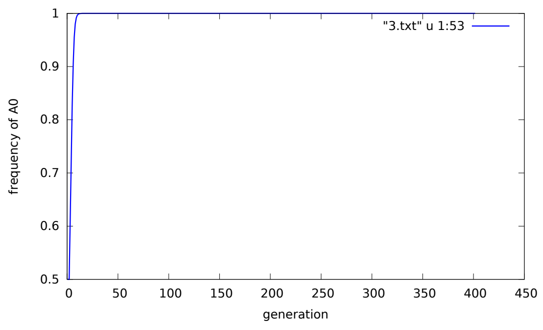


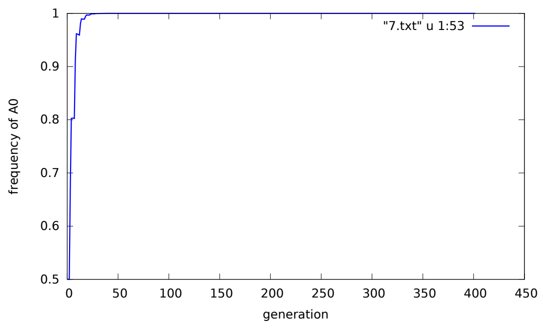
FIGURE 3.41: Th = 0.3(a) rate = 100 generations, (b) rate = 5 generations

For the 4-gene model, for case 2 of table 2.5, when threshold fitness of 0.3 and 0.7 is applied the changes seen are as in figures 3.43 and 3.44.

For a threshold of 0.3, we see that for the smallest rate of change of fitness, the lower portion is swiped off as above. For a moderate rate of change of fitness, a V like pattern is seen in the lower portion of the frequency graph. For higher rates of change, we do not see any significant change. For a threshold of 0.7, a larger portion is swiped off form the lower part for the smallest rate of change of fitness such that frequency does not go

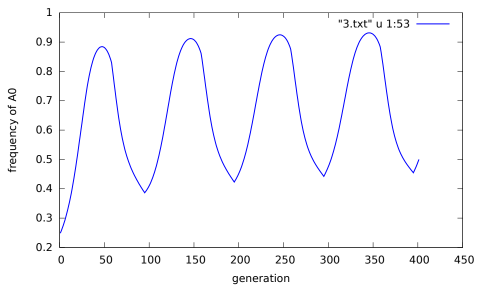


(a)

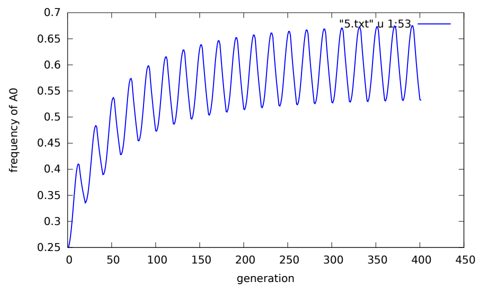


(b)

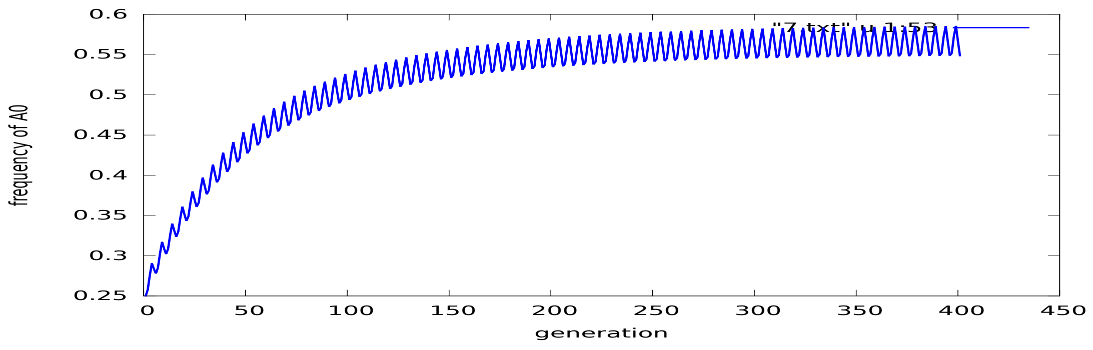
FIGURE 3.42: Th = 0.5(a) rate = 100 generations, (b) rate = 5 generations



(a)



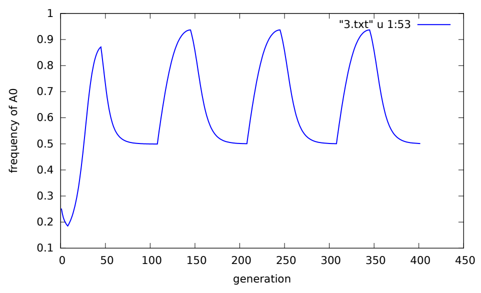
(b)



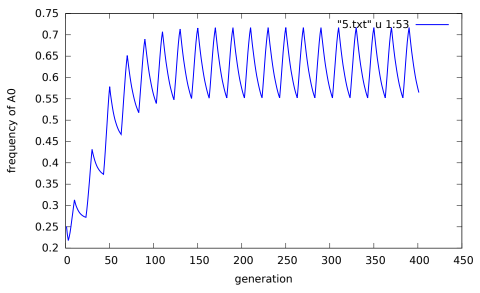
(c)

FIGURE 3.43: Th = 0.3, rate =(a) 100, (b)20, (c) 5 generations

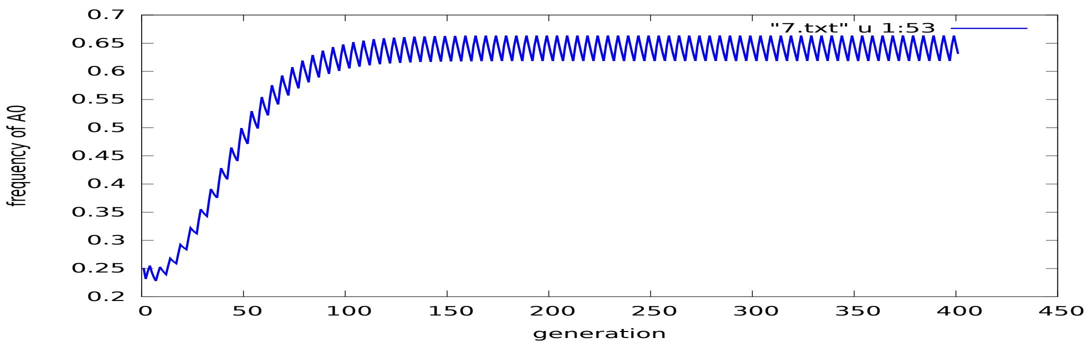
below 0.5. For a moderate rate of change, the same pattern is seen. For others, V like patterns are formed, increasing trend of frequency is no longer present and it begins to oscillate between two fixed values after about 100 generations.For case 3, we observe the same changes as before.



(a)



(b)



(c)

FIGURE 3.44: Th = 0.7, rate =(a) 100, (b)20, (c) 5 generations

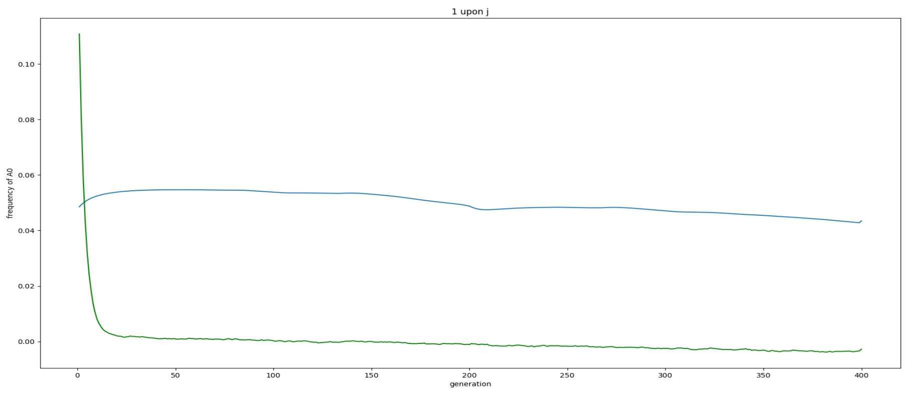


FIGURE 3.45: Case 1 - when coefficients are of the form 1/j

3.4 Results for mathematical approach

For all the basic fitness functions, we have obtained the corresponding frequency functions. We see that the pattern of the corresponding functions looks similar though in some cases evolution is found to be reversible, in others not. Simulation of linear combination and linear combination of simulation have been plotted for all the cases as in table 3.2. For cases 1,2,3 and 6 the results are in the figure below.

In the above graphs, green line indicates the linear combination of simulation and the blue line indicates simulation of linear combination. The two patterns in each graph are not similar. Hence T is not linear. Given any arbitrary linear combination of fitness, we cannot predict how its corresponding fitness function will look like according to this chosen basis.

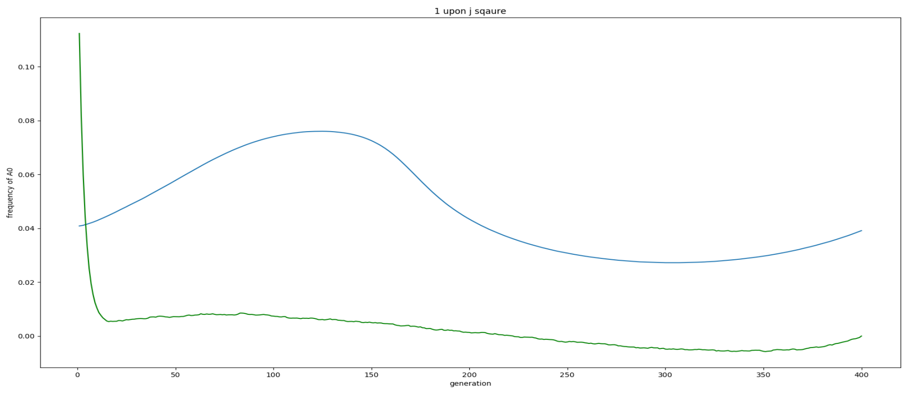


FIGURE 3.46: Case 2 - when coefficients are of the form $1/j^2$

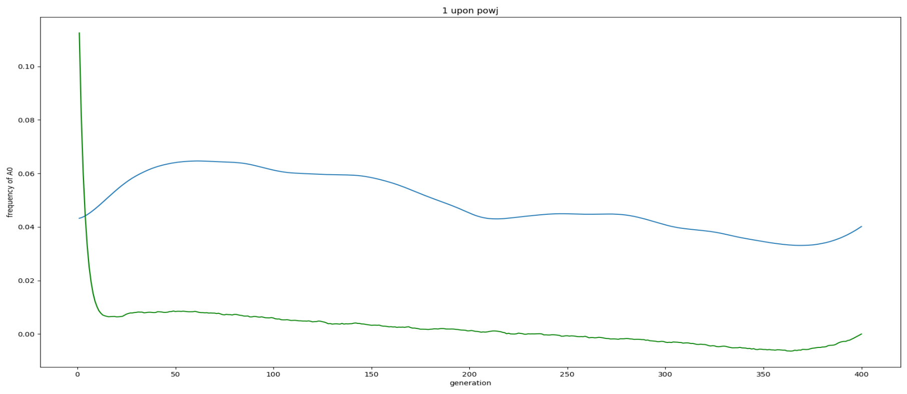


FIGURE 3.47: Case 3 - when coefficients are of the form $1/2^j$

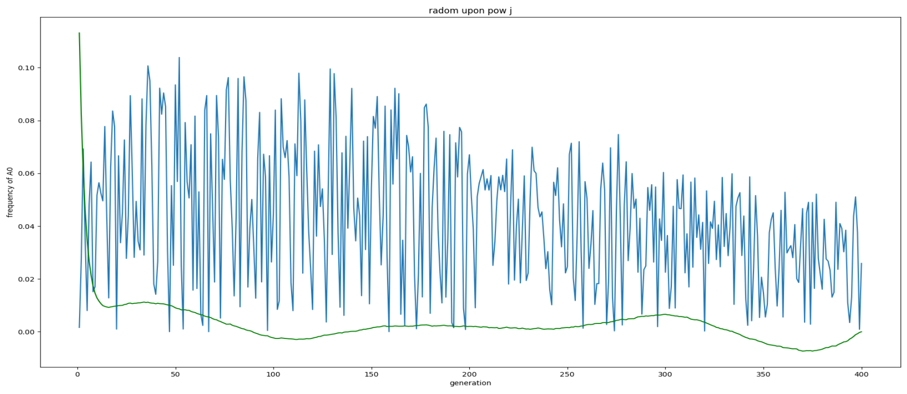


FIGURE 3.48: Case 6 - when coefficients are of the form random$/2^j$

Chapter 4

Conclusions

From the first part of the simulation, where the focus was on classical evolutionary models, the results obtained verify the simulation. By trying to address the question of reversibility, we obtained the following conclusion.
For the 2-gene model, we obtain three results according to fitness values of A0A1 and A1A1.

- A0A1 > A1A1: When the rate of change of fitness in a sinusoidal manner is slow, we see that evolution is reversible. The frequency values come back to their initial value and evolution retraces its path. We focus on larger population size due to the occurrence of regular patterns. When the rate of change of fitness becomes fast, evolution is no longer reversible. But if we omit the initial number of generations which the population takes to adapt to the environment, the remaining frequency pattern suggests that evolution is reversible.

- A0A1 = A1A1: When both the genotypes have the same fitness, decreasing trend of frequency pattern is obtained with weakening oscillations. Evolution is not reversible here.

- A0A1 < A1A1: No variation is observed in this case as the frequency reaches a constant value.

It has also been noted that there is no effect of initial allele frequency to the frequency pattern obtained. The trend depends on the rate of change of fitness of A0A0 and fitness values of other genotypes.

For the 4-gene model, we have obtained all the similar results as of 2-gene model. But we cannot conclude that these are the only frequency patterns that could occur due to the presence of 16 genotypes for each gene.

When we further apply a certain threshold fitness, for low threshold fitness, only slower rate of change of fitness patterns are affected, where lower portion would be swiped off and frequency remains constant for some generations. In some cases, V-like patterns begin to form. There is no effect of the threshold when the rate of change of fitness is fast. For high rate of change of fitness, frequency remains constant for a larger number of generations (when the lower part is swiped off) at higher frequency values for slow rate of change of fitness. When the rate of change of fitness is fast, in some cases, there is yet again no change in the frequency pattern or frequency reaches fixation or follows an increasing trend initially and then oscillate between two fixed values. These results are based on the cases checked for the reversibility of evolution.

When we try to analyze mathematically, we found that T is not linear. With the chosen Fourier basis evolution cannot be modeled. We cannot predict how the frequency pattern will look like or what the course of evolution is given arbitrary fitness functions.

Future directions : We plan to obtain a valid mathematical model for modeling evolution so that given any fitness function, the course of evolution can be predicted. With further improvements, this might also help in the setting up experiments. Secondly, we try to experimentally verify the results obtained using bacterial population by varying temperature and light.

Appendix A

Codes

The general C++ code for the 2-gene model is described below.

```
#include "population.cpp"
#include<fstream>

namespace parameter
{
    const string filename="data.txt";

    constexpr int initial_parent_freq=100;
    constexpr double initial_allel_freq= 1/2.0;

    constexpr int children_per_pair = 10;
    constexpr int generations = 400;
}



void write_header(ostream& os)
{
  os<<"#";
  string g="AB", n="12";
  for(auto i:g)
    for(auto j:n)
        os<<i<<j<<"\t";
  os<<endl;
}

```

```
int main()
{using namespace parameter;

    Phenotype alltypes;

    Population parent(alltypes);
    Population children(alltypes);

    parent.initialize_freq(initial_parent_freq);
    parent.initialize_allel_freq(initial_allel_freq);

  ofstream f(filename);
  write_header(f);
  parent.write_allel_freq(f);

    for(int g=0; g<generations; g++)
    {
      if(g>0)
    {
      children.swap(parent);
      parent.normalise(alltypes.size()*initial_parent_freq);
    }

        cout<<"processing generation : F"<<g+1<<endl;

    parent.populate(children,children_per_pair);
    children.impose_selection();
        children.update_allel_freq();
    children.write_allel_freq(f);
    }
}
```

Listing A.1: Code 1-main.cpp

```
#include "phenotype.cpp"
#include<random>
#include<algorithm>
#include<ctime>
mt19937 generator(time(NULL));

int rand50()
{
    static uniform_int_distribution<int> dist(0,1);
    return dist(generator);
}


class Population
{public:
    Population(const Phenotype& t):
        types(t),
        freq(types.size()),
        allel_freq(2, vector<double>(2))
    {}

    void initialize_freq(const int n)
    { for(int& i:freq) i=n; }

    void initialize_allel_freq(const double t);


    void populate(Population& children, const int siblings=10);
  void impose_selection();
    void update_allel_freq();
  void write_allel_freq(ostream& os);

  void normalise(const double finalsize);
    void swap(Population& p);

private:
    static vector<int> pairing;
    const Phenotype& types;
    vector<int> freq;
    vector<vector<double>> allel_freq;


    void update_indv_allel_freq(const vector<int>& indv, const int ind_freq
    );

    ///producing 1 child from parents p1 and p2
    vector<int> one_child(const vector<int>& p1, const vector<int>& p2);
};

```

```


///-------Implementation of class functions---------
vector<int> Population::pairing(pow(2,4)*51);

void Population::initialize_allel_freq(const double t)
{
    for(auto& i : allel_freq)
    for(auto& j: i)
        j=t;
}

///producing 1 child from parents p1 and p2
vector<int> Population::one_child(const vector<int>& p1, const vector<int>&
     p2)
{
    vector<int> child_p12(p1.size());
    for(int i=0; i<child_p12.size(); i+=2)
    {
        if( rand50() )
            child_p12[i] = p1[i];
        else
            child_p12[i] = p1[i+1];

        if ( rand50() )
            child_p12[i+1] = p2[i];
        else
            child_p12[i+1] = p2[i+1];
    }
    return child_p12;
}

void Population::populate(Population& children, const int siblings)
{
    children.initialize_freq(0);
    pairing.clear();

     //select pairs by shuffling
    for(int i=0; i<freq.size(); i++)
    for(int j=0; j<freq[i]; j++)
        pairing.push_back(i);
    shuffle(pairing.begin(),pairing.end(), generator);
  if(pairing.size()%2==1)
    pairing.resize(pairing.size()-1);

    vector<int> child;
    for(int i=0;i<pairing.size(); i+=2)
    for(int j=0;j<siblings; j++)
    {
        child= one_child(types[ pairing[i] ],
                         types[ pairing[i+1] ]);
        children.freq[ types[child] ]++;
    }
}

void Population::impose_selection()
{
    for(int i=0; i<freq.size(); i++)
        freq[i] = int( round( freq[i]*types.fitness(i) ) );
}


void Population::normalise(const double finalsize)
{
  int total=0;
  for(int i=0; i<freq.size(); i++)
    total+=freq[i];

    /// normalizing
    double scale=finalsize/total;
    for(int i=0; i<freq.size(); i++)
        freq[i]  = int( round(freq[i]*scale) );
}
void Population::swap(Population& p)
{
    freq.swap(p.freq);
    allel_freq.swap(p.allel_freq);
}


void Population::update_indv_allel_freq(const vector<int>& indv, const int
    ind_freq)
```

```
{
    for(int i=0; i<indv.size(); i+=2)
    {
        allel_freq[i/2][ indv[i] ]+=ind_freq;
        allel_freq[i/2][ indv[i+1] ]+=ind_freq;
    }
}
void Population::update_allel_freq()
{
    initialize_allel_freq(0);
    int total=0;
    for(int i=0; i<freq.size(); i++)
  {
        update_indv_allel_freq(types[i],freq[i]);
    total += freq[i];
  }
    for(auto& i: allel_freq)
        for(auto& j: i)
            j/= 2*total;
}

void Population::write_allel_freq(ostream& os)
{
    for(auto& i: allel_freq)
        for(auto& j:i)
            os<<j<<"\t";
    os<<endl;
}
```

Listing A.2: Code 2- population.cpp

```
#include<iostream>
#include<vector>
#include<cmath>
using namespace std;

ostream& operator<<(ostream& os, const vector<int>& ind)
{
    for(int i=0; i<ind.size(); i+=2)
        os<<ind[i]<<" "<<ind[i+1]<<"    ";
    return os;
}
```

```
class Phenotype
{public:
    Phenotype()
    {
        fill_ptypes();
        fill_pfit();
    }

    int size() const
    {return ptypes.size();}

    double fitness(int i) const
    {return pfit[i];}

    /// return ith ptype
    const vector<int>& operator[](int i) const
    {return ptypes[i];}

    /// return index of individual from ptypes
    int operator[](const vector<int>& ind) const;


private:
    void fill_ptypes();
    void fill_pfit();
    double calc_fit(const vector<int>& ind);

    static vector<vector<double>> gfit;/// genotype fitness
    vector<vector<int>> ptypes; /// all types of individuals.
    vector<double> pfit;/// fitness of each type of individual
};




int mainPhenotype()
{
```

```cpp
    Phenotype p;
    const vector<int>& t = p[8];
    cout<< t<<endl;

    cout<<p.fitness(8)<<endl;

    //const vector<int>& ind = p[95];
    cout<<p[ t];
}






///-----------implementation of class functions-----------

/// return index of individual from ptypes
int Phenotype::operator[](const vector<int>& ind) const
{
    int s=0;
    for(int i=0; i<ind.size(); i++)
    {
        int place = ind.size()-1 - i;
        s+= ind[i]*pow(2,place);
    }
    return s;
}




void Phenotype::fill_ptypes()
{
    ptypes.resize(pow(2,4));
    int i=0;
    // all typesize types.
    for ( int a = 0; a < 2; a++)
    for ( int b = 0; b < 2; b++)
    for ( int c = 0; c < 2; c++)
    for ( int d = 0; d < 2; d++)
    {
        ptypes[i] = {a,b,c,d};
        i++;
    }
}

void Phenotype::fill_pfit()
{
    pfit.resize(ptypes.size());
    for(int i=0; i<pfit.size(); i++)
        pfit[i] = calc_fit(ptypes[i]);
}

double Phenotype::calc_fit(const vector<int>& ind)
{
    double s=0;
    for(int i=0; i<ind.size(); i+=2)
        s+= gfit[i/2][ ind[i] + ind[i+1] ];
    s/=2;
    return s;
}

vector<vector<double>> Phenotype::gfit=
{
    {1, 0.5, 1},
    {1, 1, 0.5}

};
```

Listing A.3: Code 3- phenotype.cpp

To change the fitness of genotype A0A0 in a sinusoidal manner as described in Chapter 2, the portion of the code is listed below.

```
double sin_func(int g)
{
    if((g>=0) && (g<=10))
        return (1+sin((g/5.0)*PI))/2.0;
   else return sin_func(g % 10);
}
```

Listing A.4: Code for varying the fitness sinusoidally where fitness comes back in 10 generations

```
    void update_gfit(const int g)
    {
        gfit[0][0][0] = sin_func(g);
        fill_pfit();
    }
```

Listing A.5: Code for updating fitness of A0A0

The function "update_gfit" is called by the "main" program.

The code for finding the linear combination of fitness functions is listed below.

```
float find_fit(int g,vector<float>a)
{
    float pp=0;
    for(int j=1;j<401;j++)
    {
        pp+= (1+sin(2*PI*g*j/400))/(2*a[j]/j);
    }
    return pp;


}

float normalise(vector<float>a)
{
    float xx=0;
    for(int g=1;g<401;g++)
    {
        xx+= (pow(find_fit(g,a),2));

    }

    return sqrt(xx);

}

```

```

double sin_func(int g, vector<float>a)
{
    return find_fit(g,a)/normalise(a);
}
```

Listing A.6: Code for finding linear combination and normalizing

The code for finding the linear combination of simulation is provided below.

```
import matplotlib.pyplot as plt
import numpy as np

filename ='coeff.txt'
c=np.loadtxt(filename)

Matrix = np.zeros(400)
for j in range(1,401):
    filename = '%s.txt'%j
    data = np.loadtxt(filename)
    for k in range(1,401):
        Matrix[k-1] += (data[k][52]/c[k-1])

length=np.sqrt( np.sum(Matrix*Matrix) )
Matrix /=length

data = np.loadtxt('sim_of_lc_random.txt')
for k in range(0,400):
    Matrix[k] = data[k][52] - Matrix[k]

x=np.arange(1,401)
fig = plt.figure(figsize=(20,10))
plt.plot(x,Matrix)
plt.xlabel("generation")
plt.ylabel(" Diff of frequency of A0")
plt.title('Diff of Linear Comination of Simulation_3')
plt.savefig('diff_lc_of_sim_random.png')
```

Listing A.7: Code for finding linear combination and normalizing

Bibliography

[1] Jon C.Herron and Scott Freeman. Evolutionary analysis. *Pearson*.

[2] Stephen Jay Gould. Wonderful life: The burgess shale and the nature of history. *W. W. Norton & Company*, 1989. URL http://link.aip.org/link/?RSI/69/1236/1.

[3] Henrique Teotonio and Michael R. Rose. Variation in the reversibility of evolution. *Nature*, 408:463–466, November 2000. URL https://www.semanticscholar.org/paper/Variation-in-the-reversibility-of-evolution.-Teot%C3%B3nio-Rose/7cb16df97d31bb883987221e421993ffed349734.

[4] Henrique Teotonio and Michael R. Rose. Perspective: reverse evolution. *Evolution*, 55(4):653–660, April 2001. URL https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.0014-3820.2001.tb00800.x.

[5] Henrique Teotonio and Michael R. Rose. Reverse evolution of fitness in *Drosophila melanogaster*. *Journal of Evolutionary Biology*, pages 608–617, 2002. URL https://onlinelibrary.wiley.com/doi/full/10.1046/j.1420-9101.2002.00424.x.