Models of Sexual Selection with Explicit Genetics

Adarsh Krishnan

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

Indian Institute of Science Education and Research Mohali

April 2019

Certificate of Examination

This is to certify that the dissertation titled "Models of Sexual Selection with Explicit Genetics" submitted by Mr. Adarsh Krishnan (Reg. No. **MS14177**) 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. N G Prasad
(Supervisor)

Prof Arvind
(Co-Supervisor)

Dr. Rhitoban Ray Choudhury

Dr. Abhishek Chaudhuri

Dated: April 26, 2019

Declaration

The work in this dissertation has been carried out by me under the guidance of Dr. N G Prasad and Prof. Arvind 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.

Adarsh Krishnan
(Candidate)

Dated: April 26, 2019

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
(Supervisor)

Prof Arvind
(Co-Supervisor)

Acknowledgement

No amount of words can equal the love and support that my friends and family have provided for me throughout the last year. However, I am so much indebted to some of these people that its hard not to thank them in writing.

I thank Dr N. G. Prasad and Dr Arvind for the opportunity and mentorship that they provided. I also thank Dr Rhitoban Ray Choudhury and Dr Abhishek Chaudhuri for being valuable critics of my work.

I thank Ms Parul Jangal for helping me troubleshoot the program and giving me suggestions. I also thank Mr Manas Samant for providing me with the necessary help to get my work started.

Adarsh Krishnan

List of Figures

List of Tables

Contents

Abstract

Theoretical modelling provides insights that can be useful in furnishing statistical tools that would assist empiricists in checking the feasibility of their studies. This thesis focuses on developing explicit genetics models of sexual selection.

In the first chapter, an introduction to the field is given along with the need for explicit genetic models. In the second chapter, the results of the mathematical model for runaway sexual selection developed by Dr Sergey Gavrilets is verified. In the third chapter, a model based on random mating is developed. In the final chapter, a modified version of the previous model which includes a sexual selection of parents is developed, and its various scenarios are studied. In the end, an annex is provided with all the Python codes and its explanations.

The framework developed can be used for many more scenarios dealing with explicit genetics and thus serves as a foundation for further explorations.

Chapter 1

Introduction

1.1 Natural & Sexual Selection

Biological reproduction is not a perfect process, and due to this, each progeny is uniquely flawed. All populations of organisms have entails a certain amount of variation within. It is caused partly by the random mutations arising in an individual's genome and the other variations caused by the individual's interaction with their environment throughout its life which would then be inherited by its offsprings. Such changes often change the individual's destiny for better or worse depending on how it manifests. The slight variation that an individual process gives it a differential advantage or disadvantage in survival and reproductive success. Thus individuals with good variations survive, and the population grows. This mechanism is called Natural selection, and it is what drives evolution - the variation in traits that are heritable over generations. According to Charles Darwin, Natural Selection is in stark contrast with the artificial or intentional selection that breeders produce.

Natural selection manifests on all heritable phenotypic trait and other environmental aspects such as sexual selection and competition with members of different as well as the same species. Sexual selection arises due to the struggle for mating partners and usually acts even before fecundity selection. Natural selection through any other means apart from sexual selection, including kin selection, contest, etc., come under Ecological selection. Some prefer to view selection processes as two distinct wings

where one is Sexual selection, and the other is Ecological selection (analogous to Natural selection). Selection makes many individuals resort to drastic means for sex. Sexual selection has the potential to produce features that are detrimental to the individual's survival. For example, extravagant and colourful tail feathers or fins are likely to attract predators as well as interested members of the opposite sex.

Sexual selection is said to occur when individuals of particular biological sex prefer mates of the other sex to mate with (intersexual selection) while at the same time competing with individuals of the same sex for access to individuals of the opposite sex (intrasexual selection). Over time, such selections ensure that some individuals are more successful reproducing among other individuals of the same population due to he or she being more attractive or favours more attractive partners to mate with and procreate.

The concept of sexual selection was put forward by Charles Darwin and Alfred Russel Wallace and since the inception has been a topic of debate. The problem of peacock's trunk is such a well-discussed topic. Despite lowering the Darwinian fitness of the peacock, why does the peacock posses such a long trunk? All these questions were explained by a new concept called the runaway sexual selection. It attributes a sense of aesthetics in higher organisms, and an exaggerated trait serves as a stronger signal. If this signal is strong enough, the female preference for this signal is enough to undermine natural selection. Therefore, the female preference for a long trunk in the case of peacocks leads to most males descendants producing long trunk, and most females having a preference for that trait. Thus male posses these traits that demonstrate their fitness even in the cost of lowering of their fitness for an advantage in sexual selection.

Another important concept is that of Sexual dimorphism, which refers to the condition where both the sexes of a species distinctly differ in their phenotypic manifestation. The differences can be seen in characteristics including weight, colour and size or behavioural and cognitive differences which may have a differential advantage or disadvantage when it comes to sexual selection.

1.2 Why do we need theoretical models?

Theoretical modelling and analysis is an integral part of modern biological research. The insights from dynamic biological models (of selection) can be useful in furnishing statistical tools that would assist empiricists in checking the feasibility of their studies. Theory and models can be used to generate immediately testable quantitative predictions. It can also help to elaborate empirically-derived biological patterns and so on and to test verbal hypotheses in Evolutionary biology and population genetics. However, any modelling approach has its limitations, and any meaningful insight will mostly arise from a comparison among different strategies and different mathematical models that reveals deeper generalities. Therefore, more models and more applications of these models are needed.

Chapter 2

Population Genetics Model

2.1 Gavrilets' runaway sexual selection

In his model (Gavrilets 01), S. Gavrilets considered the male stimulus as a quantitative trait (y) and it has a distribution in the population given by $g(y)$ with an average value of $\bar{y}$; whereas the females are characterised by response curves $\Psi(y)$ and female resistance is given by x which is defined as the y value at which the probability of mating is half. The distribution of female resistance in a population is f(x) with a mean of $\bar{x}$.

The probability of mating between a male (y) and a female x is described by

$$\Psi(y - x) \tag{2.1}$$

The average proportion of males that can stimulate an 'x' female

$$P(x) = \int \Psi(y - x)g(y)dy \tag{2.2}$$

The average proportion of females that would mate a 'y' male

$$Q(x) = \int \Psi(y - x)f(x)dx \tag{2.3}$$

The female response curve can be expressed by the following function.

$$\Psi(z) = \frac{1}{2}\tanh\left[\epsilon \times z + 1\right] \tag{2.4}$$

Constants	***Values***
ϵ	0.2
V_x	1
V_y	1
a	0.1
b	0.05
P_{opt}	0.2

Table 2.1: Parameter values given in (Gavrilets 01)

The dynamic equations for the changes in traits between two subsequent generations are

$$\Delta x = V_x[a\Psi'(\Psi - P_{opt})] \tag{2.5}$$

$$\Delta y = V_y[b\Psi'] \tag{2.6}$$

where V_x and V_y are the additive genetic variances. Ψ and Ψ' are evaluated at $z = \bar{y} - \bar{x}$. After substituting the values of the parameters given in the table 2.1 our equations simplifies to the following.

$$\Psi(z) = \Psi(y - x) = 0.5 * \tanh\left[0.2 * z + 1\right] \tag{2.7}$$

where $z = \bar{y} - \bar{x}$. Also,

$$\Delta x = 0.1 * \Psi'(\Psi - 0.2) \tag{2.8}$$

$$\Delta y = 0.05 * \Psi' \tag{2.9}$$

During my correspondence with Dr. Gavrilets, he provided me the initial values he used while running the simulations; they are given in the table below.

$\boldsymbol{X}$	0.0	5.0	10.0	15.0	0.0	5.0	10.0
$\boldsymbol{Y}$	0.0	0.0	5.0	10.0	5.0	10.0	15.0

Table 2.2: Initial values used in (Gavrilets 01)

Using these initial values and the parameters that are given, I can evaluate all the functions and calculate the per generation change in the quantitative trait values for both males and females. Then, I continue to repeat the loop for about 100000 times. Then, I repeated the whole steps using the next pair of initial values. The algorithmic description is given below.

2.1.1 Algorithm

Algorithm 1 To simulate runaway sexual selection using Gavrilets' model.

1: Initialise x and y with the values given.
2: Define a response curve $\Psi(y - x)$.
3: Initialise empty lists tx and ty
4: **for** Repeat n times. **do**
5: $z = \bar{y} - \bar{x}$
6: $p = 0.5 * np.tanh((0.2 * z) + 1)$
7: $k = np.cosh(1 + (0.2 * z))$
8: $p' = 0.1/(k * k)$
9: $\Delta x = 0.25 * p' * (p - 0.2)$
10: $\Delta y = 0.05 * p'$
11: Append $\bar{x} = x + \Delta x$ to tx.
12: Append $\bar{y} = y + \Delta y$ to ty.
13: Plot (tx,ty)

Now, using this algorithm, we develop a python code which is described in section 5.2, we simulate the model described in Gavrilets' paper. The result is shown below.

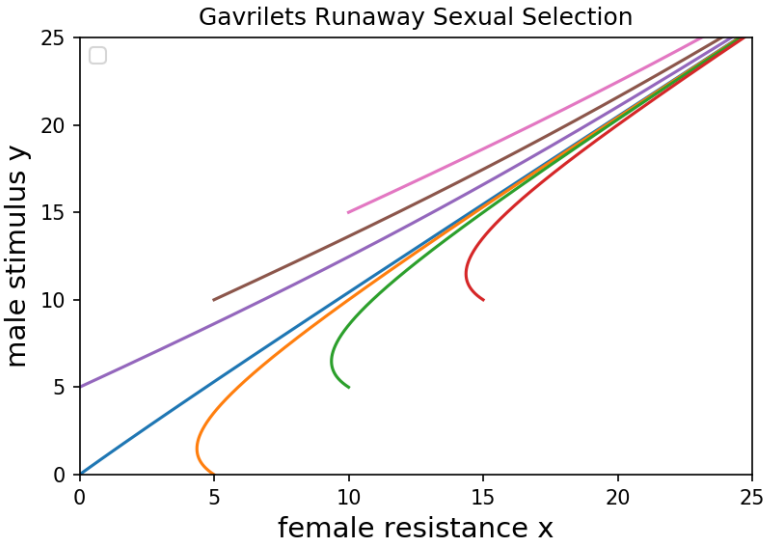


Figure 2.1. Gavrilets model for runaway sexual selection

From the above graph, we understand that in the scenario of runaway sexual selection, the male trait value increases directly proportional to the female resistance and continues to grow linearly.

2.1.2 The need for explicit genetic models

Population genetic models such as the one described in (Gavrilets 01) gives us a statistical understanding of the population. However, it fails to supply any further details. This is where the need for an explicit genetic model comes in. It can provide insights into the evolution and dynamics of each allele and genotype over the generations. One can infer which genotype or allele is superior among the rest, easily by looking at the results of an explicit genetic model.

Chapter 3

Random Mating Model

3.1 Explicit Genetic Model for Random Selection

The need for an explicit genetic model has been justified in the previous chapter 2.1.2. However, developing an explicit model has its challenges. How do we define each allele such that we can implement it in a computational code? How do we code for reproduction? How do we make an algorithm for sexual selection?

These questions can become intimidating if you try to tackle all of them together. It is, therefore, essential to break these problems down and solve one at a time. To begin with, we need a solid foundation which creates a population and evolves randomly. The model described in this chapter is such a model. It does not provide much biological insight on its own. However, it serves as a bedrock which we can configure for sexual selection in the next chapter.

A1B1	P
A2B1	Q
A1B2	R
A2B2	S

Table 3.1: Genotype substitutions

For ease of computation, we give a label to each haplotype given in table 3.1. Such that we can list all the possible individuals as given in table 3.2.

PQ	PR	PS	QR	QS
RS	PP	RR	SS	QQ

Table 3.2: All possible genotypes for individuals

This model creates an initial parent generation from the list of all possible individuals. Then, it selects a certain number of males and females to become parents. Note that all such selection is made randomly with no preference for any genotype. Each couple will produce 'x' number of children. All of these children are denoted by one long list. Out of all these children, we select a certain number of children to advance to the next generation. This step acts as a carrying capacity. The model repeats these steps over 'n' generations. Each time after reproduction, all genotype frequencies and allele frequencies are calculated. Finally, the model gives us a plot with the probability of finding each type of genotype or allele in the y-axis and the generation number in the x-axis.

3.1.1 Assumptions

All models work with some level of simplification. In the random mating model, we make the following assumptions.

1. From each generation, only a certain number of males and females will mate and are chosen randomly.
2. All parents will have a certain predetermined number of children.
3. Selection is only random and arises:
   (a) when choosing eligible mates, and
   (b) selecting the genotype of their children.
4. There is no breeding between generations, and the old generation is removed after the next generation is formed.

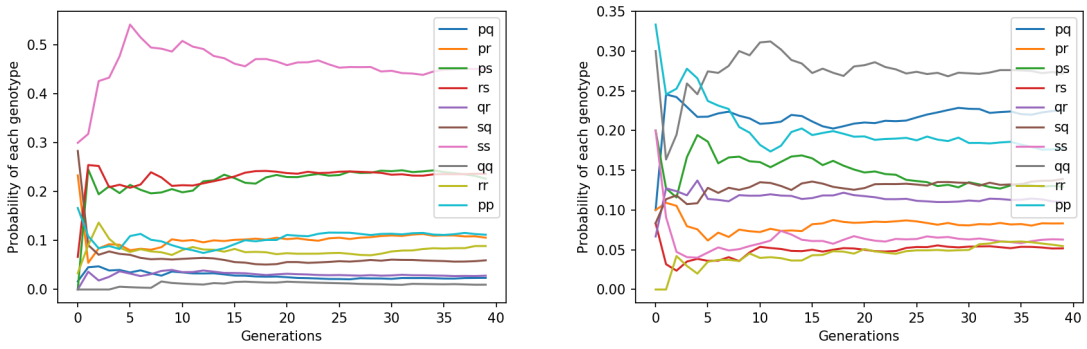
3.1.2 Algorithm

Algorithm 2 To create a population that mates randomly.

1: Create an initial generation from gene pool using random selection.
2: From the current generation, **randomly** select a male and female parent.
3: From the list of all possible recombination of parents, **randomly** select genotypes for children.
4: Remove parents from the current generation and add the children to a new generation after counting the frequencies of each genotype.
5: Switch the current generation with the new generation and repeat the process.

3.1.3 Results

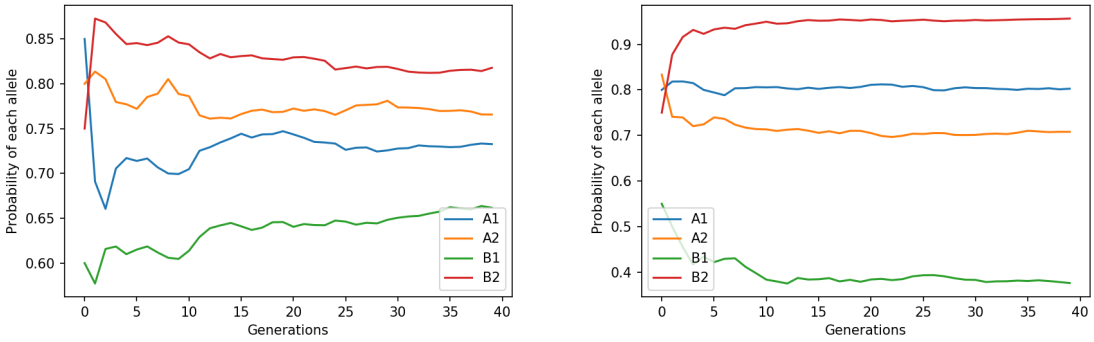
Since all the selections featured in this model are based on the pseudo-random choice from the Numpy package in python, the results that we get vary each time we run it.



(a) Instance 1 **(b)** Instance 2

Figure 3.1. Genotype frequency from Random Selection

It may be noted that the population reaches fixation rapidly and each genotype settles to a certain probability. Some may go to negligible values. It is because although the selection is random, over the generations, the one that received higher probability, in the beginning, tends to have a bias due to its higher frequency in the subsequent generations.



(a) Instance 1 **(b)** Instance 2

Figure 3.2. Allelic frequency from Random Selection

Now that we have developed our foundations, we can explore new possibilities that can provide some valuable biological insights.

Chapter 4

Explicit Sexual Selection Model

4.1 Coding sexual selection

To create a model that works via sexual selection, we start with our previous random mating model and modify the parts of it that selects the parents. It is conceptually accurate to implement sexual selection while choosing mating partners because of the exclusive nature of the selection.

4.1.1 Assumptions

Although the assumptions that we make in this model varies only slightly from what we did in 3.1.1, the change in the output is significant.

1. From each generation, only a certain number of males and females will mate and are chosen randomly.
2. All parents will have a certain predetermined number of children.
3. Parents are selected based on their attractiveness via sexual selection.
4. Selection is random while selecting the genotype of children.
5. There is no breeding between generations, and the old generation is removed after the next generation is formed.

In this model, we use a certain predetermined probability for each possible genotype while implementing sexual selection. Note that the sum of all these probabilities should equal one. We define a function to select all male and/or female parents. This function selects parents from the existing new generation using a certain (predetermined) biases. The model lends itself to heavy customisation. One can choose individual probability for each genotype for both parents separately. In this chapter, we explore various scenarios that feature mostly the same algorithm but have different biological meaning.

4.1.2 Algorithm

Algorithm 3 To create a population that mates via sexual selection.

Require: Probability Definition

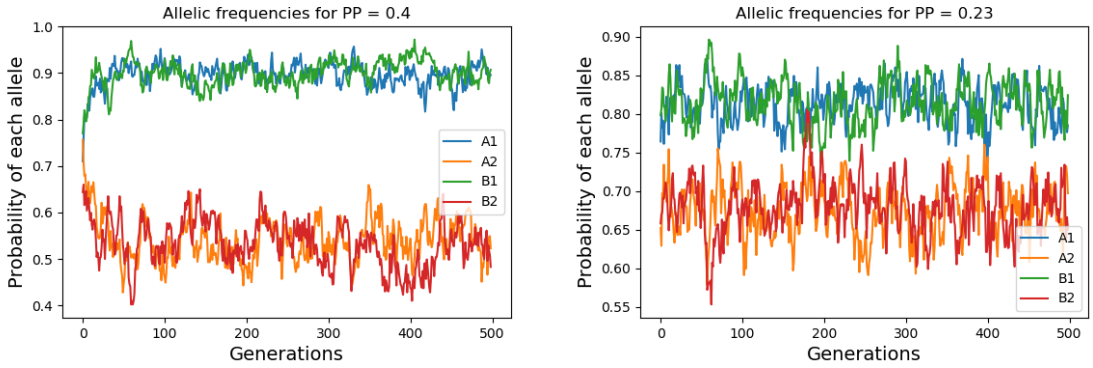
1: Create an initial generation from gene pool using random selection.
2: From the current generation, select males and female parents using **sexual selection**.
3: **function** SEXUAL SELECTION(parentgeneration)
4: Find frequencies of all genotype.
5: List of genotypes repeated with frequencies.
6: List of probability.
7: Using numpy.random.choice() and the two lists, select parents.
8: From the list of all possible recombination of parents, **randomly** select genotypes for children.
9: Remove parents from the current generation and add the children to a new generation after counting the frequencies of each genotype.
10: Switch the current generation with the new generation and repeat the process.

4.2 One sex is sexually selected

In this scenario, we select one of the parents using our sexual selection algorithm while the other parent is chosen randomly. We arbitrarily decide that **PP** is the ideal male and **SS** is the ideal female although our model inherently has no male or female parameter that can make this distinction.

4.2.1 Only males are sexually selected

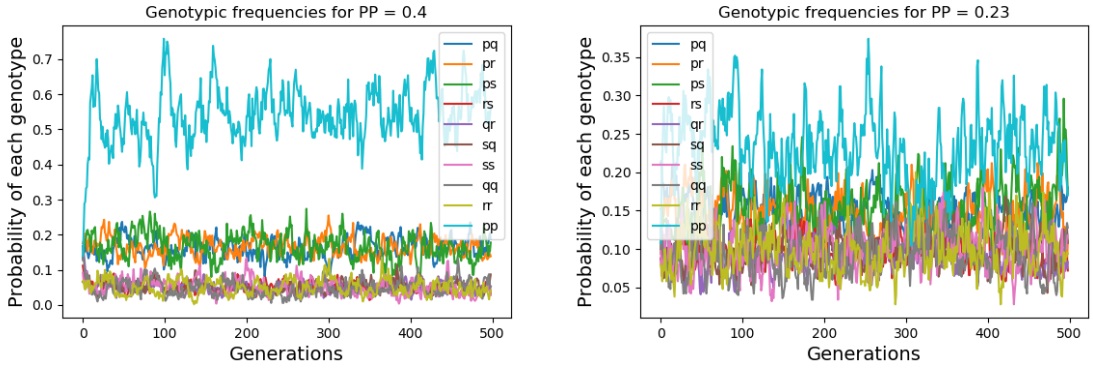
Here, the **PP** males are the most desirable and all males find their mates randomly. We can see two cases below, one where the selection for the ideal male is strong and another where the selection is comparatively weaker.



(a) P(PP) = 0.4 & others 0.04 **(b)** P(PP) = 0.23 & others 0.07

Figure 4.1. Allelic frequency when males are sexually selected.

We observe that the alleles corresponding to **PP**, **A1B1** reaches fixation fast and has a higher probability, in both the cases.



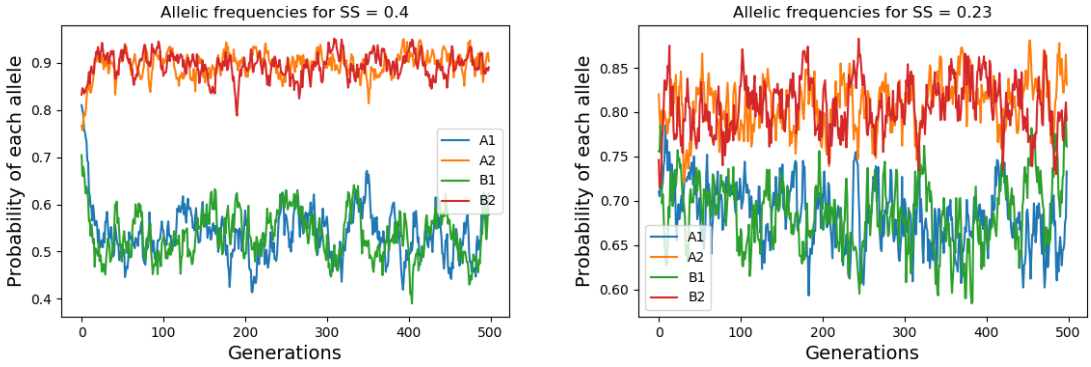
(a) P(PP) = 0.4 & others 0.04 **(b)** P(PP) = 0.23 & all others 0.07

Figure 4.2. Genotypic frequency when males are sexually selected.

We observe that the population swiftly reaches fixation and **PP** becomes the most probable genotype. After that comes all genotype that has **P** in it. All the others have a negligible probability.

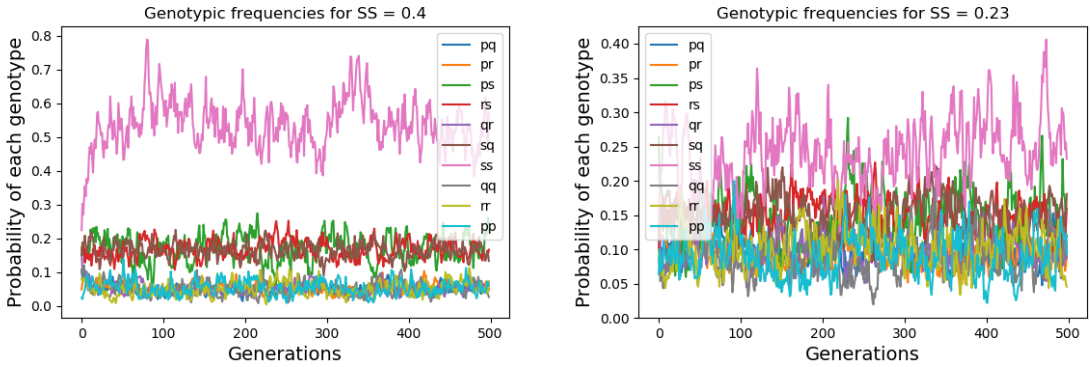
4.2.2 Only females are sexually selected

Here, the **SS** females are the most desirable and all females find their mates randomly. Similar to males we have two cases, one where the selection for the ideal female is strong and another where it is comparatively weaker.



(a) P(SS) = 0.4 & others 0.04 **(b)** P(SS) = 0.23 & others 0.07

Figure 4.3. Allelic frequency when females are sexually selected.



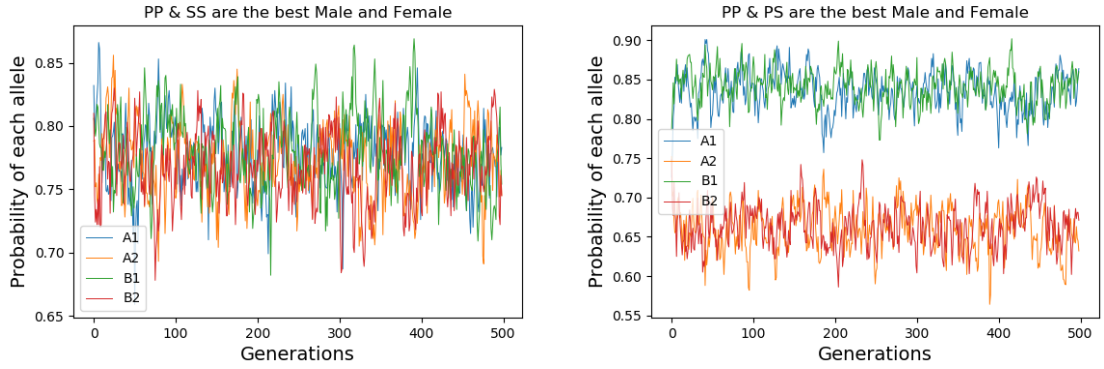
(a) P(SS) = 0.4 & others 0.04 **(b)** P(SS) = 0.23 & all others 0.07

Figure 4.4. Genotypic frequency when females are sexually selected.

We observe that the population swiftly reaches fixation and **SS** becomes the most probable genotype. The results that we got from males and females show us that any arbitrary distinction between males and females are meaningless in this model. However, it is not to say that our model lacks sexual selection. Instead, it incorporates female resistance and male stimulus in terms of the probability distributions.

4.3 Both sexes are sexually selected

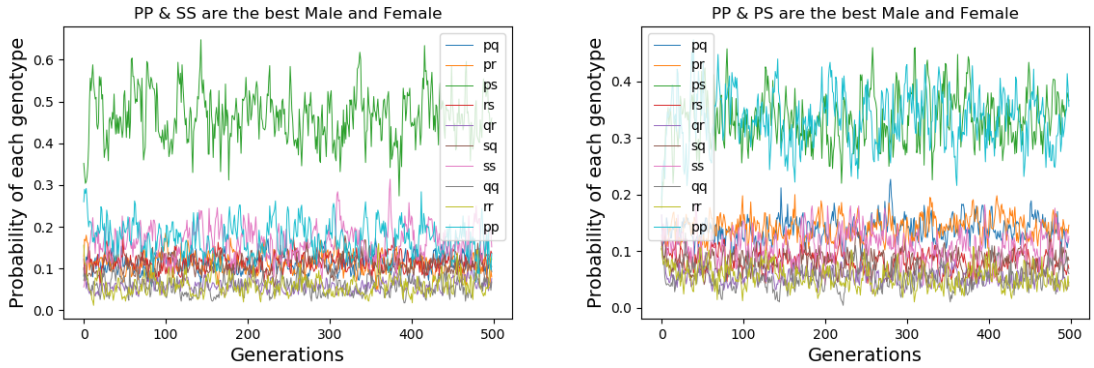
Here, we have **PP** as the most desirable male and **SS** or **PS** as the most desirable female. All individuals prefer the best mate possible. We explore two scenarios, one where the males and females choose conflicting haplotype and one where they share a preference for a particular haplotype.



(a) P(PP) = P(SS) = 0.4 **(b)** P(SS) = 0.4 & P(PS) = 0.4

Figure 4.5. Allelic frequency when both are sexually selected.

In figure 4.5a, we observe that when male and female preferences conflict, there is no clear dominant allele in terms of frequency. However, when they share a preferred haplotype 4.5, it becomes common in the population.



(a) P(PP) = P(SS) = 0.4 **(b)** P(SS) = 0.4 & P(PS) = 0.4

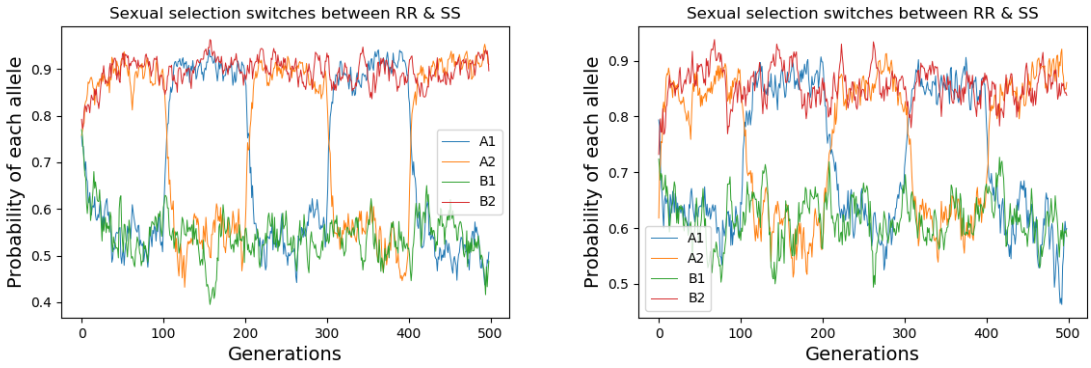
Figure 4.6. Genotypic frequency when both are sexually selected.

Figure 4.6a shows us that when the ideal male is **PP** and the ideal female is **SS**, it is their hybrid genotype **PS** that becomes frequent in the population followed by **PP** and **SS**. However, as shown in figure 4.6b, when the ideal male **PP** and ideal female

PS share a common haplotype **P**, both of them become the common genotypes in the population, followed by other genotypes that share **P** and **S** haplotypes.

4.4 One sex is sexually selected with switching

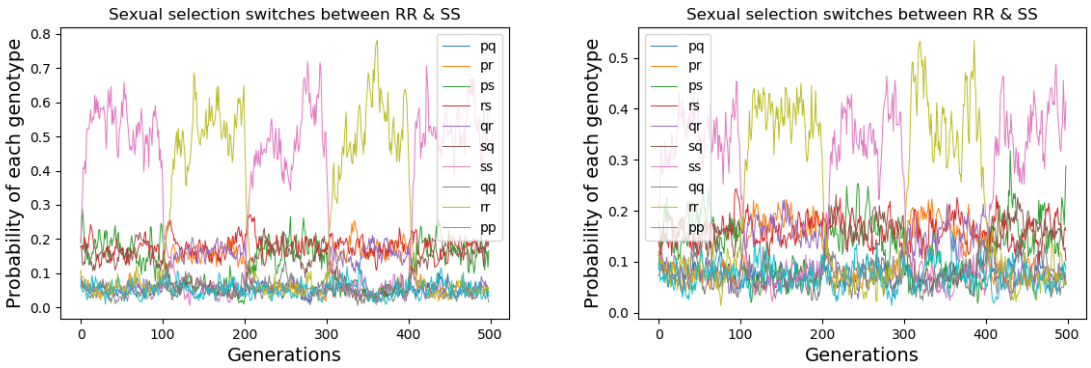
Now, that we have covered some common scenarios, we can begin to test the robustness of the model. So, here we see a modified version of the model where one sex is selected using our sexual selection algorithm and the other is chosen randomly. However, in this case, we change the ideal genotype from **RR** to **SS** and back and forth and observe how the genotypic and allelic frequencies change. There are two cases, one where the selection is strong and one where the selection is weak.



(a) P(SS) = P(RR) = 0.4 **(b)** P(SS) = P(RR) = 0.25

Figure 4.7. Allelic frequency when one parent is selected with switching.

From figures 4.7a and 4.7b, we notice that apart from the alleles that **RR** and **SS** shared, all others oscillated in frequencies with the change in the ideal genotype.



(a) P(SS) = P(RR) = 0.4 **(b)** P(SS) = P(RR) = 0.25

Figure 4.8. Genotypic frequency when one parent is selected with switching.

Here, we notice that whenever the ideal genotype changes, the frequencies reach fixation fast. Apart from that, the genotype that share haplotypes with the ideal genotype also changes in frequencies but the change in frequency is smaller.

4.5 Conclusion

In all cases of sexual selection, the best genotype increases in frequency. Therefore, if we measure the average of a quantitative trait value of males (stimulus) and females (resistance), it increases over generations. This is similar to runaway selection as the attractive trait starts to dominate the population. However, we reach fixation due to our inbuilt carrying capacity inherent in the code.

The framework developed can be used for many more scenarios dealing with explicit genetics including deletion, mutation and other biological processes. Therefore, these models serve as a foundation for further explorations.

Chapter 5

Annex: Python Codes and Explanations

All the source codes along with detailed explanations are available on Github `https://github.com/kriashks/expgen`. If you wish to work with the codes, you can fork the repository from Git. However, for the sake of completion, I shall venture to explain the Python codes used in this thesis.

5.1 Fundamental Codes

These codes are used in almost all the following models. Therefore, it pays to learn what these code snippets do.

5.1.1 Libraries Required

In this snippet, we import all the necessary libraries.

```
1 import numpy as np # To work with arrays
2 import matplotlib.pyplot as plt # To plot the final results
3 # To use the pseudorandom number generator
4 import random
5 # To create a simulation for mating
6 from itertools import product
```

5.1.2 To count the frequencies

To plot the frequencies of each genotype or allele, we find the number of time each genotype or allele occurs in a generation and then divide this frequency with the total number of individuals per generation. To find the frequency, we use the builtin **count** function and to find the population size, we use the **len** function. After finding the ratio of each genotype or allele, we add them to a list using the **append** function so that we can plot this list to see its change over generations.

```
#   Define these empty lists outside the main loop
nextgen = []; all_children = []
fpq=[]; fpr=[]; fps=[]; fqr=[]; fsq=[]
frs=[]; fss=[]; frr=[]; fqq=[]; fpp=[]
fa1=[]; fa2=[]; fb1=[]; fb2=[]

#   Inside the main loop, add these counters after mating.
        leng=len(gen0)

        fpq.append(((gen0.count('pq')+gen0.count('qp'))/leng))
        fpr.append(((gen0.count('pr')+gen0.count('rp'))/leng))
        fps.append(((gen0.count('ps')+gen0.count('ps'))/leng))
        frs.append(((gen0.count('rs')+gen0.count('sr'))/leng))
        fqr.append(((gen0.count('qr')+gen0.count('rq'))/leng))
        fsq.append(((gen0.count('sq')+gen0.count('qs'))/leng))
        fss.append(((gen0.count('ss')+gen0.count('ss'))/leng))
        fqq.append(((gen0.count('qq')+gen0.count('qq'))/leng))
        frr.append(((gen0.count('rr')+gen0.count('rr'))/leng))
        fpp.append(((gen0.count('pp')+gen0.count('pp'))/leng))

        A1 = gen0.count('pq') + gen0.count('qp') + gen0.count('rp') +
             gen0.count('pr') +  gen0.count('sp') + gen0.count('sr')+
             gen0.count('ps') + gen0.count('rs') + gen0.count('qr') +
             gen0.count('rq') + gen0.count('pp') + gen0.count('rr')

        A2 = gen0.count('pq') + gen0.count('qp')  + gen0.count('qr') +
             gen0.count('rq')  + gen0.count('qs') + gen0.count('sq') +
             gen0.count('rs') + gen0.count('sr')  + gen0.count('ps') +
             gen0.count('sp') + gen0.count('qq') + gen0.count('ss')

        B1 = gen0.count('pq') + gen0.count('qp') + gen0.count('pr') +
             gen0.count('rp') + gen0.count('ps') + gen0.count('sp') +
             gen0.count('qr') + gen0.count('rq') + gen0.count('qs') +
             gen0.count('sq') + gen0.count('pp') + gen0.count('qq')

        B2 = gen0.count('pr') + gen0.count('rp') + gen0.count('qr') +
```

```
            gen0.count('rq') + gen0.count('rs') + gen0.count('sr') +
            gen0.count('ps') + gen0.count('sp') + gen0.count('qs') +
            gen0.count('sq') + gen0.count('rr') + gen0.count('ss')
        fa1.append((A1/leng))
        fa2.append((A2/leng))
        fb1.append((B1/leng))
        fb2.append((B2/leng))
```

5.1.3 To plot the frequencies

After finding the frequencies, we use the **plot** function from **matplotlib** to plot these in a graph.

```
# Plot each genotype
plt.plot(fpq, label='pq', linewidth=0.7)
plt.plot(fpr, label='pr', linewidth=0.7)
plt.plot(fps, label='ps', linewidth=0.7)
plt.plot(frs, label='rs', linewidth=0.7)
plt.plot(fqr, label='qr', linewidth=0.7)
plt.plot(fsq, label='sq', linewidth=0.7)
plt.plot(fss, label='ss', linewidth=0.7)
plt.plot(fqq, label='qq', linewidth=0.7)
plt.plot(frr, label='rr', linewidth=0.7)
plt.plot(fpp, label='pp', linewidth=0.7)

# Plot each allele
plt.plot(fa1, label='A1', linewidth=0.7)
plt.plot(fa2, label='A2', linewidth=0.7)
plt.plot(fb1, label='B1', linewidth=0.7)
plt.plot(fb2, label='B2', linewidth=0.7)

# Output Customisations
plt.xlabel('Generations', fontsize=14)
plt.ylabel('Probability of each genotype/allele', fontsize=14)
#plt.ylabel('Probability of each genotype/allele')
plt.title("Title")
plt.legend()
plt.savefig('title.png',dpi=100)
plt.show()
```

5.1.4 To create a parent generation

Each time we use a model, the first thing we do is to create the parent (initial) generation. From the list of all possible genetics, we chose a certain number of individuals randomly using **random.sample** from the **random** library and add them to a list using the list extension function **extend()**. This list becomes the new parent generation.

```
gen0=[]
#A1B1=p, A2B1=q, A1B2=r, A2B2=s

# All possible genetics
l=['pq','pr','ps','qr','qs','rs','pp','rr','ss','qq']

#p = int(input('print parent generation size: '))
for k in range(0,200):
        a = random.sample(l, 1)
        gen0.extend([''.join(c) for c in a])
        # join selections as ['','','','']
#print ('gen0 = ',gen0)
```

5.1.5 To implement sexual selection

Given below are the functions used to select both male and female parents using sexual selection. To select the parents using sexual preference, we define a certain probability for each genotype. Whenever we choose parents for the next generation, these functions are called as you can see in the next sections. These functions find the frequency of each genotype and create two lists in such a way that there is a one to one map between the probability of a certain genotype being selected to that genotype.

```
def zero(n, d):
    return n / d if d else 0

def finddad(gen0,n):

        count_pq = gen0.count('pq')
        count_qp = gen0.count('qp')
        count_pr = gen0.count('pr')
```

```
        count_rp = gen0.count('rp')
        count_ps = gen0.count('ps')
        count_sp = gen0.count('sp')
        count_qs = gen0.count('qs')
        count_rq = gen0.count('rq')
        count_qr = gen0.count('qr')
        count_sq = gen0.count('sq')
        count_sr = gen0.count('sr')
        count_rs = gen0.count('rs')
        count_pp = gen0.count('pp')
        count_qq = gen0.count('qq')
        count_rr = gen0.count('rr')
        count_ss = gen0.count('ss')
        gen0.sort()
        probdad_pq = 0.04
        probdad_qp = 0.04
        probdad_pr = 0.04
        probdad_rp = 0.04
        probdad_ps = 0.04
        probdad_sp = 0.04
        probdad_qr = 0.04
        probdad_rq = 0.04
        probdad_qs = 0.04
        probdad_sq = 0.04
        probdad_rs = 0.04
        probdad_sr = 0.04
        probdad_pp = 0.4
        probdad_qq = 0.04
        probdad_rr = 0.04
        probdad_ss = 0.04
        probdad =   [zero(probdad_pp,count_pp)]*count_pp +
                    [zero(probdad_pq,count_pq)]*count_pq +
                    [zero(probdad_pr,count_pr)]*count_pr +
                    [zero(probdad_ps,count_ps)]*count_ps +
                    [zero(probdad_qp,count_qp)]*count_qp +
                    [zero(probdad_qq,count_qq)]*count_qq +
                    [zero(probdad_qr,count_qr)]*count_qr +
                    [zero(probdad_qs,count_qs)]*count_qs +
                    [zero(probdad_rp,count_rp)]*count_rp +
                    [zero(probdad_rq,count_rq)]*count_rq +
                    [zero(probdad_rr,count_rr)]*count_rr +
                    [zero(probdad_rs,count_rs)]*count_rs +
                    [zero(probdad_sp,count_sp)]*count_sp +
                    [zero(probdad_sq,count_sq)]*count_sq +
                    [zero(probdad_sr,count_sr)]*count_sr +
                    [zero(probdad_ss,count_ss)]*count_ss
        #pdad = [ '%.3f' % elem for elem in probdad ]
        probdad = np.array(probdad)
```

```
57          probdad /= probdad.sum()
58          p = list(np.random.choice(gen0,n,p=probdad,replace=False))
59
60          return p
61
62  def findmum(gen0,n):
63
64          count_pq = gen0.count('pq')
65          count_qp = gen0.count('qp')
66          count_pr = gen0.count('pr')
67          count_rp = gen0.count('rp')
68          count_ps = gen0.count('ps')
69          count_sp = gen0.count('sp')
70          count_qs = gen0.count('qs')
71          count_rq = gen0.count('rq')
72          count_qr = gen0.count('qr')
73          count_sq = gen0.count('sq')
74          count_sr = gen0.count('sr')
75          count_rs = gen0.count('rs')
76          count_pp = gen0.count('pp')
77          count_qq = gen0.count('qq')
78          count_rr = gen0.count('rr')
79          count_ss = gen0.count('ss')
80          gen0.sort()
81          probmum_pq = 0.04
82          probmum_qp = 0.04
83          probmum_pr = 0.04
84          probmum_rp = 0.04
85          probmum_ps = 0.2
86          probmum_sp = 0.2
87          probmum_qr = 0.04
88          probmum_rq = 0.04
89          probmum_qs = 0.04
90          probmum_sq = 0.04
91          probmum_rs = 0.04
92          probmum_sr = 0.04
93          probmum_pp = 0.04
94          probmum_qq = 0.04
95          probmum_rr = 0.04
96          probmum_ss = 0.04
97          probmum =   [zero(probmum_pp,count_pp)]*count_pp +
98                      [zero(probmum_pq,count_pq)]*count_pq +
99                      [zero(probmum_pr,count_pr)]*count_pr +
100                     [zero(probmum_ps,count_ps)]*count_ps +
101                     [zero(probmum_qp,count_qp)]*count_qp +
102                     [zero(probmum_qq,count_qq)]*count_qq +
103                     [zero(probmum_qr,count_qr)]*count_qr +
104                     [zero(probmum_qs,count_qs)]*count_qs +
```

```
                    [zero(probmum_rp,count_rp)]*count_rp +
                    [zero(probmum_rq,count_rq)]*count_rq +
                    [zero(probmum_rr,count_rr)]*count_rr +
                    [zero(probmum_rs,count_rs)]*count_rs +
                    [zero(probmum_sp,count_sp)]*count_sp +
                    [zero(probmum_sq,count_sq)]*count_sq +
                    [zero(probmum_sr,count_sr)]*count_sr +
                    [zero(probmum_ss,count_ss)]*count_ss
        #pmum = [ '%.3f' % elem for elem in pmum ]
        probmum = np.array(probmum)
        probmum /= probmum.sum()
        p = list(np.random.choice(gen0,n,p=probmum,replace=False))

        return p
```

5.2 Gavrilets' Model

Given below is the code used to simulate S. Gavrilets' model given in his paper. We follow the algorithm given the chapter 2; using his equations with different initial values.

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

xx = [0.0, 5.0, 10.0, 15.0, 0.0, 5.0, 10.0]
yy = [0.0, 0.0, 5.0, 10.0, 5.0, 10.0, 15.0]
k = 0.0
i=0
while (i<7):
    x=xx[i]
    y=yy[i]

    xdash = x
    ydash = y

    tx = []
    ty = []

    n = 0
    while (n<100000):

        tx.append(xdash)
        ty.append(ydash)
```

```
        #avgx = np.mean(tx)
        #avgy = np.mean(ty)
        z = ydash - xdash
        p = 0.5*np.tanh((0.2*z) + 1)
        k = np.cosh(1+(0.2*z))
        pdash = 0.1/(k*k)
        dx = 0.25*pdash*(p - 0.2)
        dy = 0.05*pdash
        xdash = x + dx
        ydash = y + dy
        x = xdash; y = ydash
        n = n + 0.5

    i=i+1
    plt.plot(tx,ty)
    plt.xlim((0,25))
    plt.ylim((0,25))
plt.xlabel('female resistance x', fontsize=14)
plt.ylabel('male stimulus y', fontsize=14)
#plt.ylabel('Probability of each genotype')
plt.title("Gavrilets Runaway Sexual Selection")
plt.savefig('gavri.png',dpi=150)
plt.show()
```

5.3 Random Selection Model

Given below is the Python code for generating a random selection model as discussed in Chapter 3. Note that this code only has the main loop. One must add all the other fundamental elements for the complete output. For more information, kindly visit `https://github.com/kriashks/expgen`.

```
for j in range(0,40):
# overall loop/ The number of generations.
        for i in range(0,20):
        #This loop randomly chooses pairs from the existing
        #generation and creates a list of possible progenies.
        #It then creates a certain number of children.

        # randomly choosing partners from gen0
                p1,p2 = np.random.choice(gen0,2,replace=False)
                #print (p1,p2)

                # crossing to get first filial generation #
```

```
                # product('ABCD', 'xy') --> Ax Ay Bx By Cx Cy Dx Dy #
                  children = product(p1, p2)
                  # joining the array as ['','','',''] #
                  all_children.extend(['',join(c) for c in children])

                  # append nextgen for the next generation to cross and
                  # choose a certain number of children from every cross
                  nextgen.extend(np.random.choice(all_children,10))
                  #print 'all_children=', all_children
                  #print 'next generation=', nextgen
                  all_children=[]
                  gen0.pop(gen0.index(p1))
                  gen0.pop(gen0.index(p2))
                  #print 'gen0 after pop=', gen0
                  if len(gen0) == 0:
                          break
          # replace parent generation with filial generation
          gen0 = nextgen
          #print(gen0)
```

5.4 One Sex is Sexually Selected

Given below is the Python code for generating a sexual selection model where only one of the parents are selected via sexual selection and the other parent is chosen randomly, as discussed in Chapter 4. Note that this code only has the main loop. One must add all the other fundamental elements for the complete output. For more information, kindly visit `https://github.com/kriashks/expgen`.

```
for j in range(1,500):
# overall loop/ The number of generations.
          #print ('Generation number: ',j)
          p1 = list(np.random.choice(gen0,100,replace=False))
          p2 = findmum(gen0,100)

          for i in range(50):
          #This loop randomly chooses pairs from the existing
          #generation and creates a list of possible progenies.
          #It then creates a certain number of children.
          # randomly choosing partners from gen0
                  dad = p1[i]
                  mom = p2[i]

```

```
                # crossing to get first filial generation
                children = product(dad, mom)
                # product('ABCD', 'xy') --> Ax Ay Bx By Cx Cy Dx Dy

                all_children.extend([''.join(c) for c in children])
                # joining the array as ['','','',''] #

                # append nextgen for the next generation to cross and
                # choose a certain number of children from every cross
                nextgen.extend(np.random.choice(all_children,10))
                all_children=[]
                p1.pop(p1.index(dad))
                p2.pop(p2.index(mom))
                if len(p1) == 0:
                        break
                if len(p2) == 0:
                        break
        # replace parent generation with filial generation
        if len(nextgen) > 1000:
        # limited the sample size of the generation by 1000
                nextgen = list(np.random.choice(gen0,1000))

        gen0 = nextgen
```

5.5 Both sexes are sexually selected

Given below is the Python code for generating a sexual selection model where both the parents are selected via sexual selection, as discussed in Chapter 4. Note that this code only has the main loop. One must add all the other fundamental elements for the complete output. For more information, kindly visit `https://github.com/kriashks/expgen`.

```
for j in range(1,500):
# overall loop/ The number of generations.
        #print ('Generation number: ',j)
        p1 = finddad(gen0,100)
        p2 = findmum(gen0,100)

        for i in range(50):
        #This loop randomly chooses pairs from the existing
        #generation and creates a list of possible progenies.
        #It then creates a certain number of children.
```

```
                # randomly choosing partners from gen0
                dad = p1[i]
                mom = p2[i]

                # crossing to get first filial generation #
                children = product(dad, mom)
                # product('ABCD', 'xy') --> Ax Ay Bx By Cx Cy Dx Dy #

                all_children.extend([''.join(c) for c in children])
                # joining the array as ['','','',''] #

                # append nextgen for the next generation to cross and
                # choose a certain number of children from every cross
                nextgen.extend(np.random.choice(all_children,10))
                all_children=[]
                p1.pop(p1.index(dad))
                p2.pop(p2.index(mom))
                if len(p1) == 0:
                        break
                if len(p2) == 0:
                        break
        # replace parent generation with filial generation
        if len(nextgen) > 1000:
        # limited the sample size of the generation by 1000
                nextgen = list(np.random.choice(gen0,1000))

        gen0 = nextgen
        leng=len(gen0)
```

5.6 One sex is sexually selected with switching

Given below is the Python code for generating a sexual selection model where only one of the parents are selected via sexual selection and the other parent is chosen randomly, as discussed in Chapter 4. In addition, in this model, the preferred genotype changes over a certain period. Note that this code only has the main loop. One must add all the other fundamental elements for the complete output. For more information, kindly visit `https://github.com/kriashks/expgen`.

```
for j in range(1,500):
# overall loop/ The number of generations.
        #print ('Generation number: ',j)
        p1 = list(np.random.choice(gen0,100,replace=False))
```

```
5
6           if (counter<100):
7                   p2 = findmum1(gen0,100)
8           if (99<counter<200):
9                   p2 = findmum2(gen0,100)
10          counter=counter+1
11          if (counter==200):
12                  counter=0
13
14          for i in range(50):
15          #This loop randomly chooses pairs from the existing
16          #generation and creates a list of possible progenies.
17          #It then creates a certain number of children.
18                  # randomly choosing partners from gen0
19                  dad = p1[i]
20                  mom = p2[i]
21
22                  # crossing to get first filial generation #
23                  children = product(dad, mom)
24                  # product('ABCD', 'xy') --> Ax Ay Bx By Cx Cy Dx Dy #
25
26                  all_children.extend([''.join(c) for c in children])
27                  # joining the array as ['','','',''] #
28
29                  # append nextgen for the next generation to cross and
30                  # choose a certain number of children from every cross
31                  nextgen.extend(np.random.choice(all_children,10))
32                  all_children=[]
33                  p1.pop(p1.index(dad))
34                  p2.pop(p2.index(mom))
35                  if len(p1) == 0:
36                          break
37                  if len(p2) == 0:
38                          break
39          # replace parent generation with filial generation
40          if len(nextgen) > 1000:
41          # limited the sample size of the generation by 1000
42                  nextgen = list(np.random.choice(gen0,1000))
43
44          gen0 = nextgen
```

Bibliography

[Gavrilets 01] S. Gavrilets, G. Arnqvist & U. Friberg. *The evolution of female mate choice by sexual conflict.* Proceedings of the Royal Society B: Biological Sciences, 2001.

[Gavrilets 03] Sergey Gavrilets. *Perspective: Models of speciation - What have we learned in 40 years?*, 2003.

[Gavrilets 14a] Sergey Gavrilets. *Is sexual conflict an "Engine of speciation"?* Cold Spring Harbor Perspectives in Biology, 2014.

[Gavrilets 14b] Sergey Gavrilets. *Models of speciation: Where are we now?*, 2014.