PROJECT REPORT: SUMMER 2017

Sachita Nishal

June 1 - July 8 2017
Population Biology Lab, IISER Pune
Under Supervision of: Dr. Sutirth Dey

OBJECTIVE/RATIONALE:

In this project, I have explored the effects of genetic and epigenetic factors on the evolutionary dynamics of Wright-Fischer populations. Although genetic contribution in the process of evolution is well appreciated [1] until very recent studies, the contribution of epigenome remained rather undermined [2]. In this regard, understanding the process and the consequences of inheritance of epigenetic information across generations is important. It is a component of epigenetics classically defined as- 'The study of mitotically and/or meiotically heritable changes in gene function that cannot be explained by changes in DNA sequence.'[3]. Naturally, the effect of epigenetic inheritance is most prominently understood when phenotypic variations that do not stem from variations in DNA base sequences are transmitted to subsequent generations of cells or organisms [4].

Taking this information, we ran simulations to track the effects of an individual's genome *and* epigenome, on the fitnesses and rates of adaptation of Wright-Fischer populations over the longer term. In this project, 'pure epigenetic variation was simulated, i.e. 'epigenetic variation that is independent of genetic context' [5]. Modelling was carried out to emulate the action of natural selection on these populations. This was based on a model created by Collins *et. al.* [6], that accounted for different mutation rates for genetic and epigenetic mutations, over two fitness landscapes. It was extended to account for two fitness landscapes - single peak and double peak. The rates of adaptation of the generations were observed and plotted against the generations for two cases: (a) monomorphic starting populations (b) polymorphic starting populations.

MATERIALS AND METHODS

A. STRUCTURE OF MODEL

To form the skeleton of the simulations, from Collins *et. al.*, the following basic model was taken:

1. We consider the evolutionary dynamics of a set of variables g1, g2, . . ., gk describing the state of the genetic system (DNA sequence) and a second set of variables e1, e2, . . ., el describing an epigenetic system. For simplicity we use binary variables for each loci and considered k = l =10.
2. We take the mutation rate to be 10^(-6) for each of the genetic variables per generation, and 10^(-4) for each epigenetic variables.
3. The genetic variables specify a phenotype described by some function w_gen(g1, g2,…) (the genetic fitness). Analogously, the epigenetic variables specify a value of the phenotype w_epigen (e1,e2,. . .) (the epigenetic fitness).

4. The fitness corresponding to a particular state of the genetic and epigenetic variables is determined by the maximum value between w_gen and w_epigen, i.e. w = max(w_gen, w_epigen). This is the simplest implementation of the tenet that genetic changes are independent of epigenetic changes and they do not interact to determine fitness. Here we simulated two types of fitness landscapes:

   (i) The single-peak landscape, where w_gen = 1.5 if g1 = g2 = …= gk = 1 (on-peak) and w_gen = 0.1 otherwise (off-peak), and similarly w_epigen = 1.5 on the peak and w_epigen = 0.1 away from it.

   (ii) The double-peak landscape where w_gen = 1.5 if

      a. if g1 = g2 = … gk = 1, or
      b. if g1 = g3 = g5 = g7 = g9 = 1 and g2 = g4 = g6 = g8 = g10 = 0

   and w_gen = 0, otherwise. The same rules apply to w_epigen, with respect to the vales of e1, e2, …. e10.

5. We numerically simulate the evolutionary dynamics using a standard Wright–Fischer dynamics, using a population of N = 1,000 individuals,

Note for Readability of Model:

Here, I have represented the genotypes and epigenotypes as arrays of binary digits throughout the report with, for example, the maximum fitness genotype and epigenotype for the single peak landscape looking like this: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

B. PROCEDURE

The model was coded with the aforementioned features and conditions, and then set to run for 150,000 generations, with 20 such replications for every fitness landscape (i.e. single peak, double peak, multi peak). Such replications were broadly run for two cases: monomorphic starting population (with all individuals having the same genotype and the same epigenotype) and polymorphic starting population (with all individuals having distinct genotypes and epigenotypes).

For every replicate, the the average fitness of generations was plotted against generations. Upon analysis, some characteristic features and trends were observed in the data obtained in this manner. These have been discussed in the section below.

RESULTS AND DISCUSSION

A. MONOMORPHIC STARTING GENERATION

In this set of simulations, the initial population was monomorphic - all individuals had identical genotypes and epigenotypes, and as a consequence, had identical fitness. The genotypes and epigenotypes of all individuals were [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

SINGLE PEAK FITNESS LANDSCAPE

We began with observing Wright-Fischer population dynamics in a single-peak fitness landscape, with the monomorphic starting population. Each replicate ran for 1.5 x 10^5 generations, and finally the average fitnesses of every generation were plotted against the generations for the same. Figure 1.1.1 shows the general evolutionary dynamics for this case:

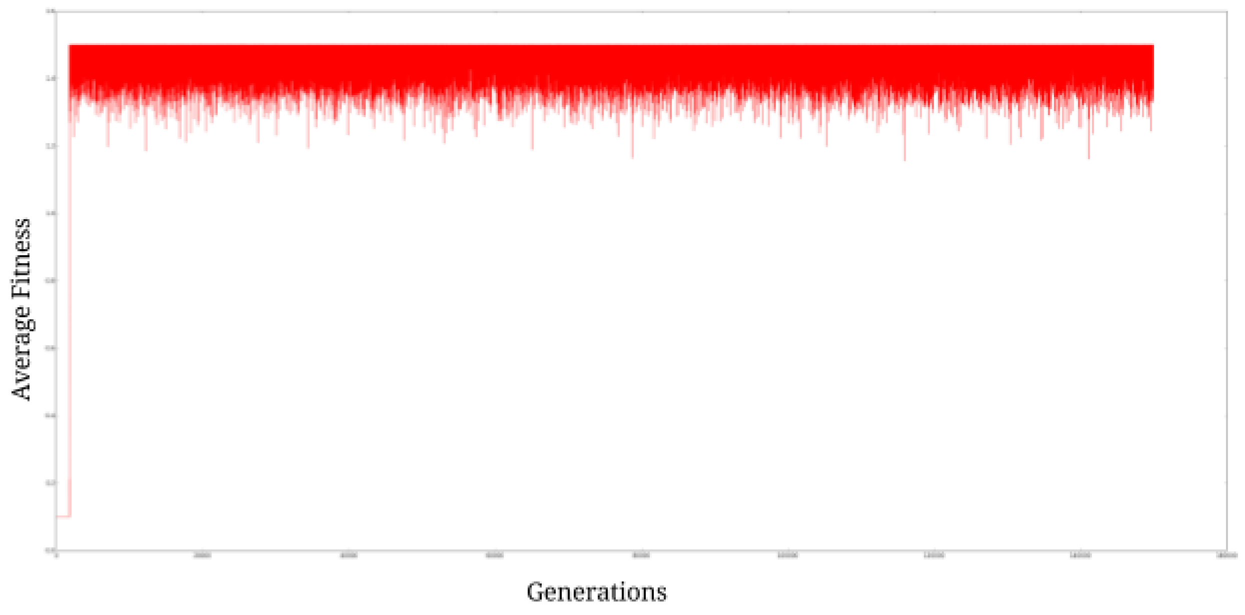

Figure 1.1.1: Change in fitness over generations for a single-peak fitness landscape, starting with monomorphic population.

The same general trend was obtained with all subsequent replicate runs. From the graph above (Figure 1.1.1) two following patterns were observed:

1. The average fitness of the species is very low at the beginning and shows slight fluctuation in the initial generations. Suddenly, however, it drastically increases after a generation. This phenomenon can be attributed to a large number of individuals striking upon a mutation that gives rise to the maximum-fitness genotype or epigenotype. Such a point/generation of sudden escalation was encountered in all these replications of the single peak fitness landscape. I will refer to it as the 'leap generation' in the report further.

   This generation was recorded for all replicates to plot later. However, no discernible trend was observed in the generations at which these fortuitous, beneficial mutations were encountered, as can be seen in Figure 1B.

   This is explained by the fact that the occurrence of mutations in these dynamics is entirely stochastic and would hence not show any patterns - it is entirely probabilistic and random with respect to fitness effects. And hence, there would be no steady trend for generations where beneficial mutations occur so as to elevate the fitness to such a large extent.

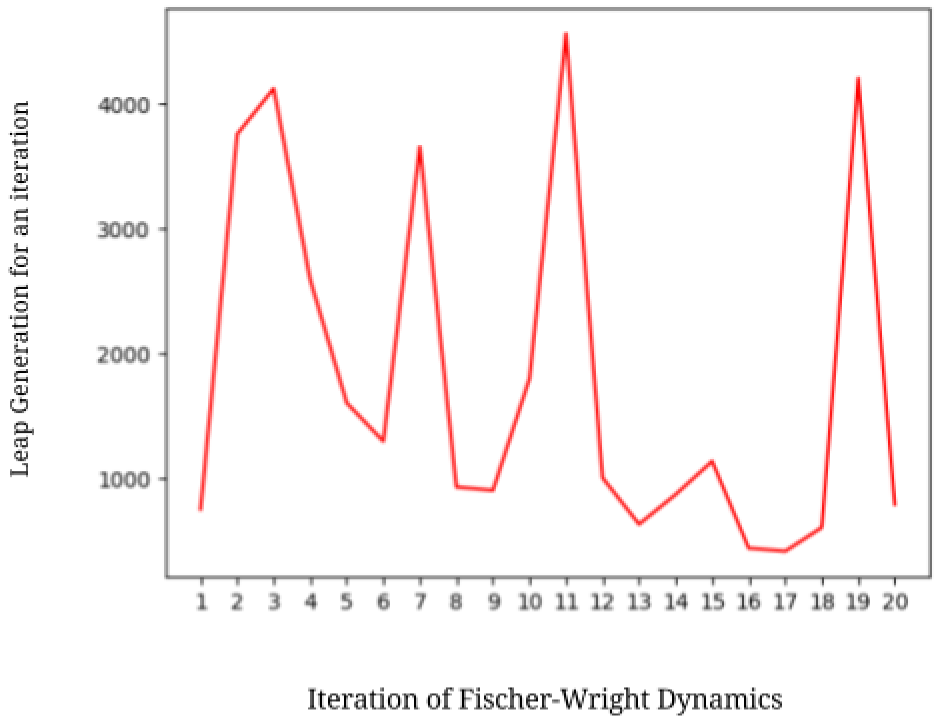

Figure 1.1.2: Leap generations for the iterations of single peak Wirght-Fischer dynamics in monomorphic starting populations

2. After the sudden increase in the average fitness of a generation, the values of average fitness for subsequent generations continue to fluctuate in a certain range and about a mean. A state of equilibrium is attained with respect to the average fitnesses.

   I calculated the geometric mean of the average fitnesses of the last 1000 generations of every replicate to get a value, about which the average fitness values fluctuate in the equilibrium state. I will refer to this value as the 'equilibrium fitness value' henceforth.

   Here follows a graph delineating the values of the geometric mean of the average fitnesses of the last 1000 generations of every iterations, versus the iterations. (Figure 1.1.3)

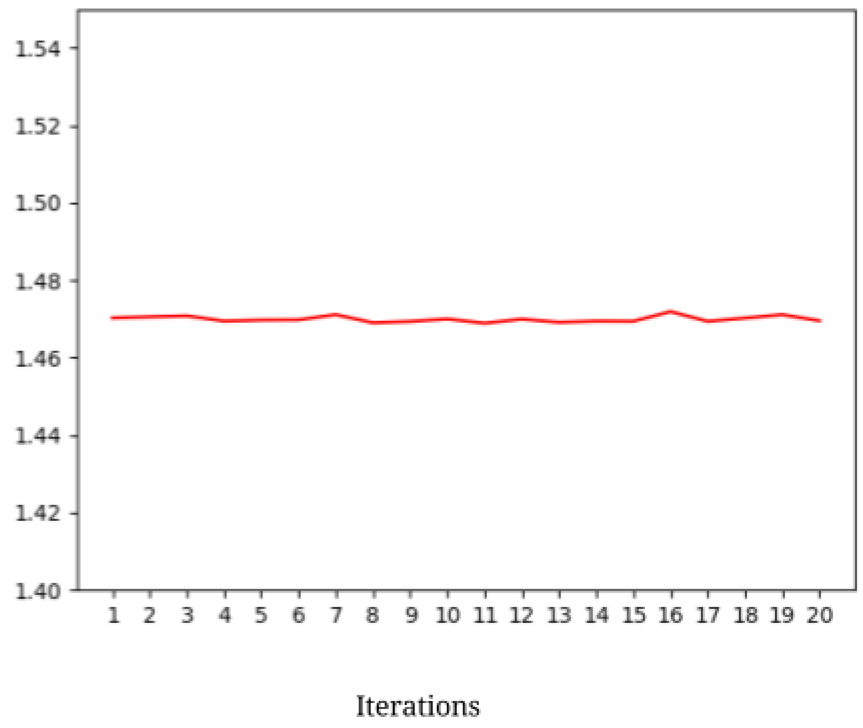

Figure 1.1.3: Change in equilibrium fitness values for 20 Iterations of Wright-Fischer populations

DOUBLE PEAK FITNESS LANDSCAPE

In this case, we observed Wright-Fischer population dynamics in a double-peak fitness landscape, with a monomorphic starting population. Each replicate ran for 1.5 x 10^5 generations, and finally the average fitnesses were plotted against corresponding generations. Figure 1.2.1 shows the general evolutionary dynamics for this case:

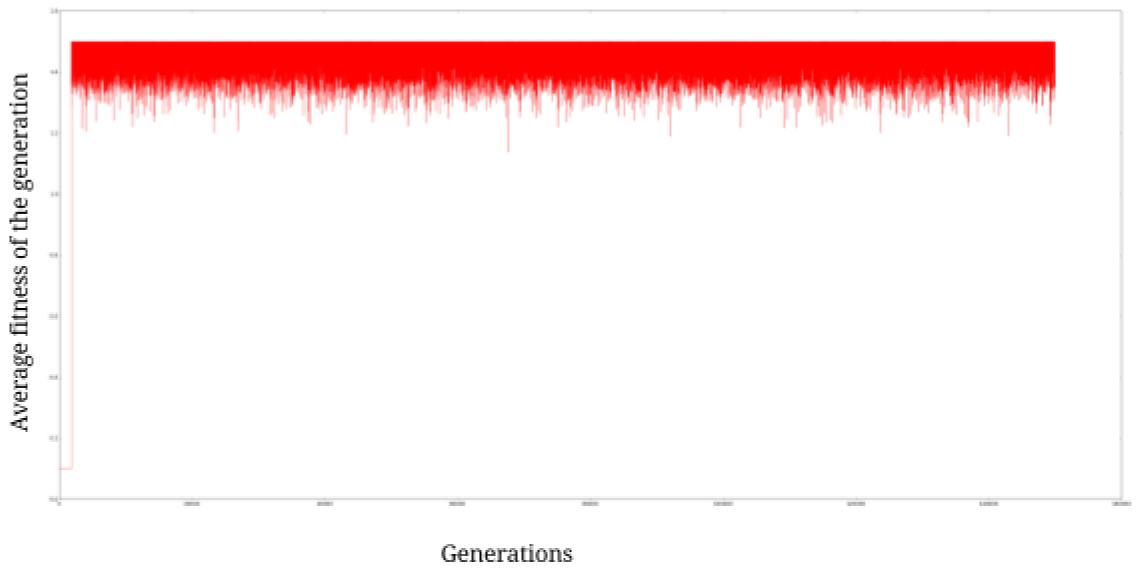

Figure 1.2.1: Change in fitness over generations for a double-peak fitness landscape, starting with monomorphic population.

The general trend observed here is similar to that observed in the case of the single peak fitness landscape.

1. Both models have this feature in common: a drastic increase in average fitness occurs for a single generation (leap generation) in a given iteration, as time progresses. Before this increase, the average fitness values have fluctuated about a point with a very low value. (Figure 1.2.2)

   Once again, however, no discernible trend is observed in this leap generation. The stochastic nature of these dynamics is the reason for this phenomenon.

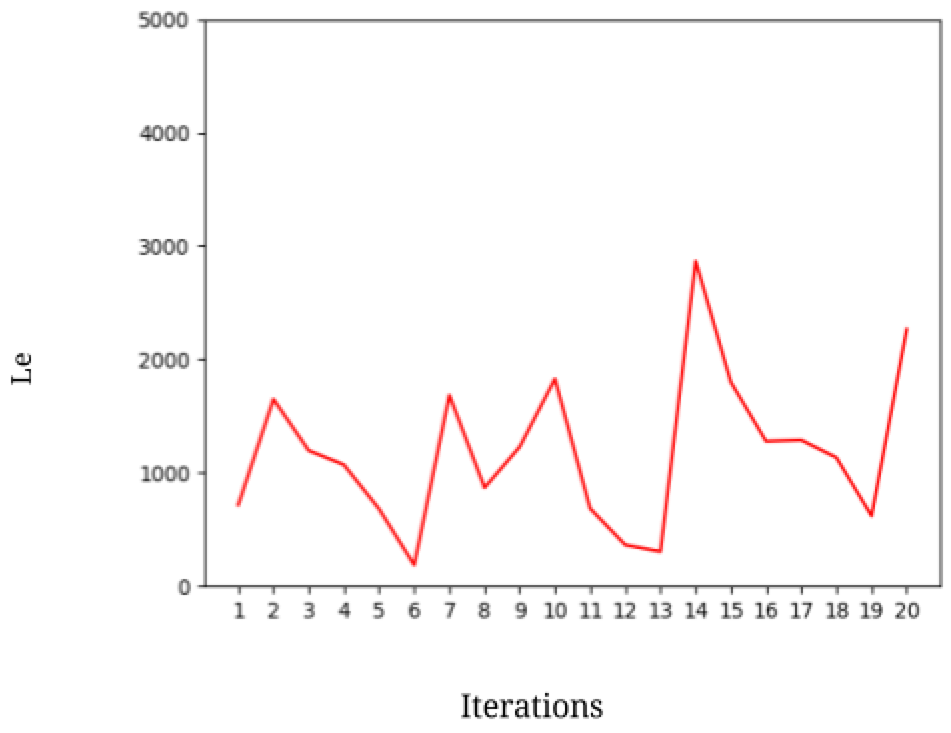

Figure 1.2.2: Leap generations for the iterations of double peak Wirght-Fischer dynamics in monomorphic starting populations

2. The fluctuation of average fitness around a mean, once it has drastically increased as described above, is also observed in the case of the double peak fitness landscape. Once again, the geometric means of the last 1000 generations of every iteration were calculated to use as the equilibrium fitness values about which these values fluctuate, and the below figure shows their plot against the iterations (Figure 1.2.3)

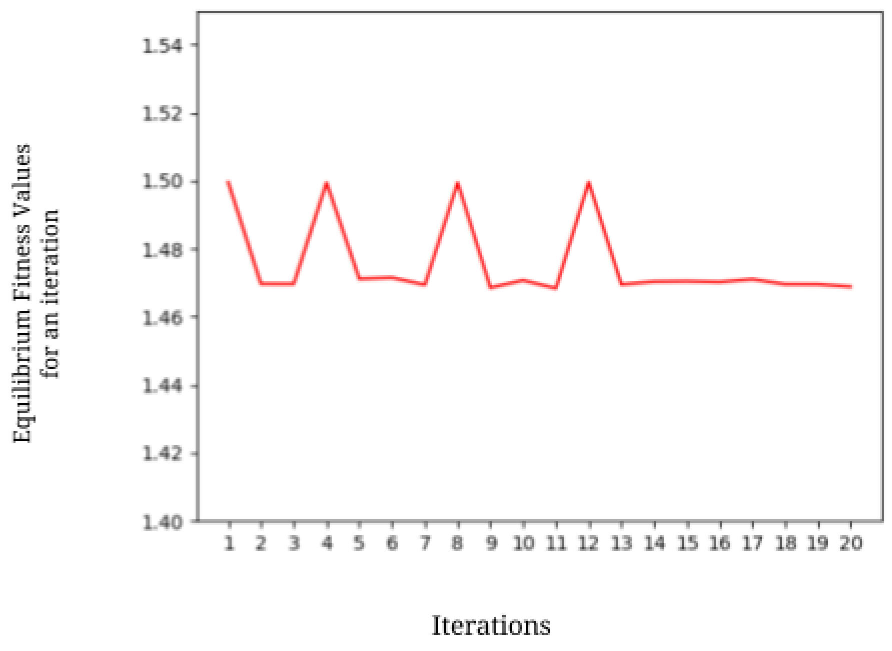

Figure 1.2.3: Change in equilibrium fitness values for 20 Iterations of Wright-Fischer populations

OVERALL ANALYSIS FOR MONOMORPHIC STARTING GENERATIONS

Following the separate simulations for single-peak and double-peak fitness landscapes, I compiled the following graphs to get a more comprehensive idea of the effect of the fitness landscapes on the Wright-Fischer dynamics and evolution of populations.

1. Comparing the arithmetic means of the leap generations (across 20 replicates) for single-peak and double-peak fitness landscapes (Figure 1.3.1)

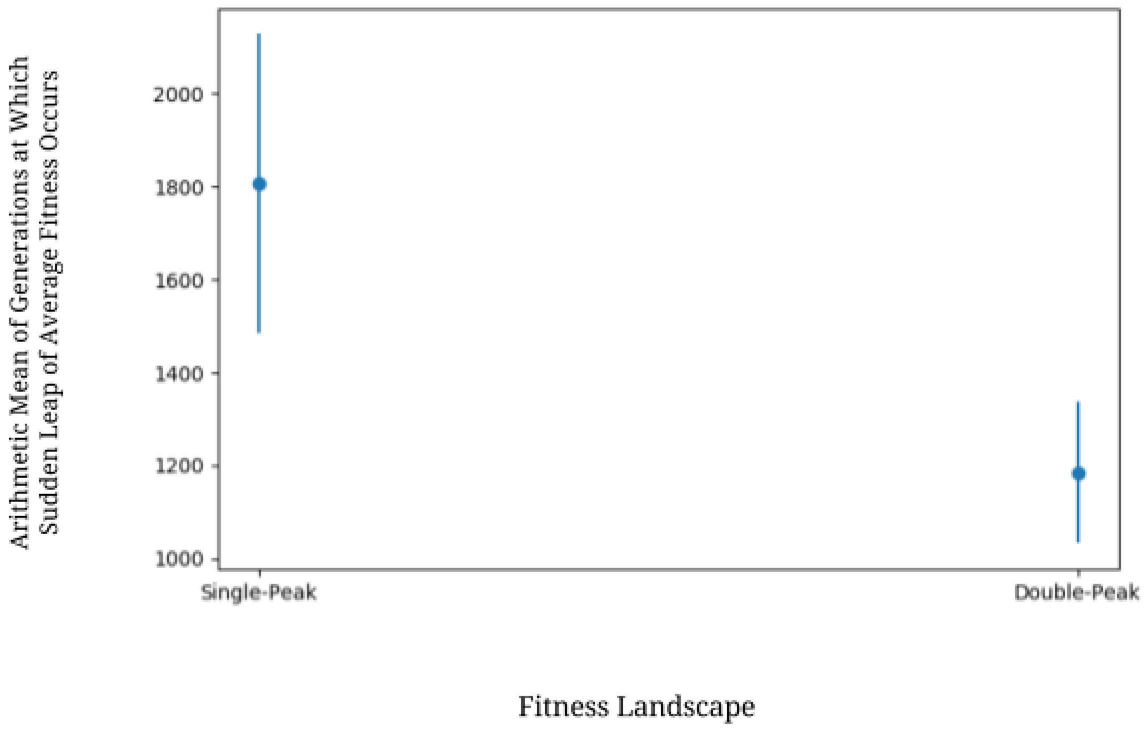

Figure 1.3.1 A comparison of the arithmetic means of leap generations for Single-Peak and Double-Peak fitness landscapes for monomorphic starting generations

One can see the clear distinction in the arithmetic means of generations at which drastic increase of fitness is observed - it tends to fluctuate around a significantly higher value (approximately 1800 generations) for single-peak cases whereas it tends to fluctuate around approximately 1200 generations for double-peak cases.

This drastic difference could be rationalized through the fact that an individual in the double peak fitness landscape has greater probability of achieving the maximum fitness genotype/epigenotype as opposed to one in the single peak fitness landscape. This difference of probability is due to there being two maximum fitness genotypes and epigenotypes in the double peak case, which implies more opportunity to achieve the maximum fitness.

2. Comparing the arithmetic mean of the Equilibrium Fitnes Values (across all replicates) for single-peak and double-peak fitness landscapes. (Figure 1.3.2)

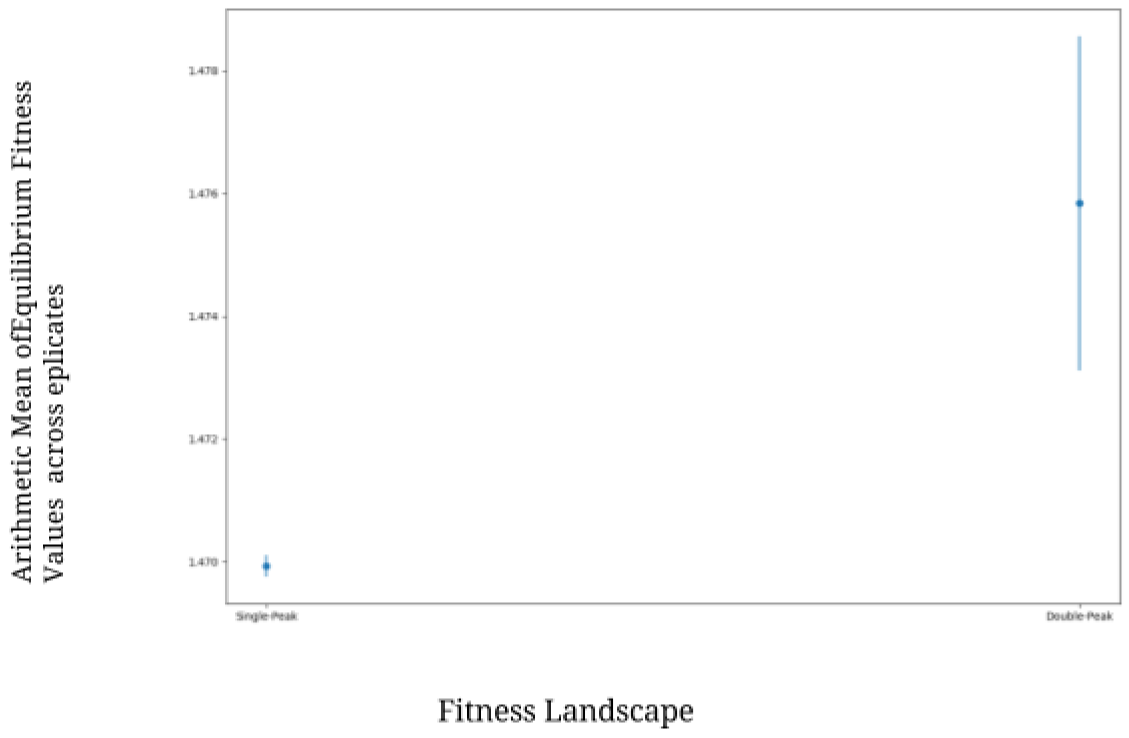

Figure 1.3.2: Comparison of arithmetic means of equilibrium fitness values across all replicates for single peak and double peak fitness landscapes for monomorphic starting generations

B. POLYMORPHIC STARTING GENERATION

In this set of simulations, we considered polymorphic starting populations - all individuals were randomly allotted a genotype and phentoype from the reference set of 1024 genotypes and 1024 epigenotypes. However, cases of maximum fitness (w = 1.5) were not selected in order to have the average fitness of the starting population at 0.1.

SINGLE PEAK FITNESS LANDSCAPE

We began with observing Wright-Fischer population dynamics in a single-peak fitness landscape, with the polymorphic starting population. Each replicate ran for 10^5 generations, and finally the average fitnesses of every generation were plotted against the generations for the same. Figure 2.1.1 shows the general evolutionary dynamics for this case:

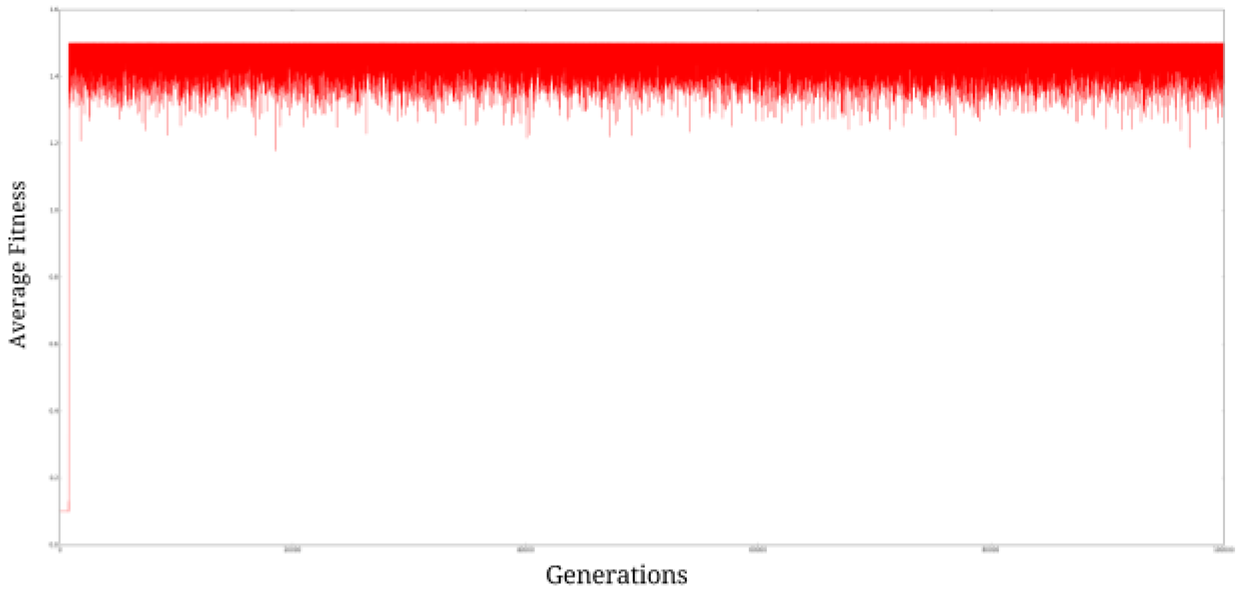

Figure 2.1.1: Change in fitness over generations for a single-peak fitness landscape, starting with polymorphic population.

This general trend was obtained with all such replicates and the same pattern of drastic fitness increase after a certain generation, as observed in cases of monomorphic starting generation, was also observed here. Such a leap generation was encountered in all these replications. However, again, there was no trend in these leap generations, as can be seen in Figure 2.1.2. This can again be attributed to the stochastic nature of these dynamics.

Another observation was the oscillation of the average fitness about a mean once the drastic increase in fitness occurred. However, statistical analysis was not carried out for this aspect.

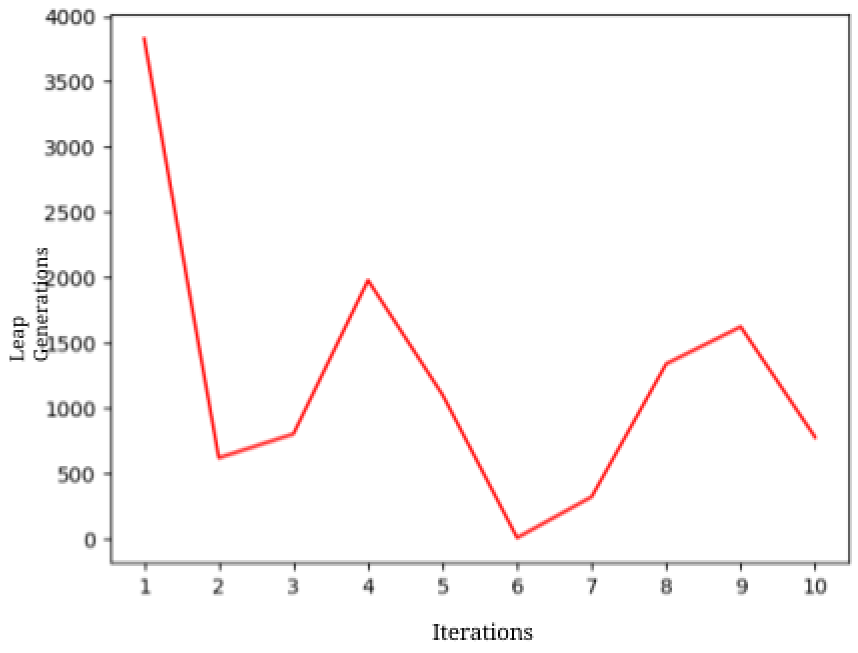

Figure 2.1.2: Leap generations for the iterations of single peak Wirght-Fischer dynamics in polymorphic starting populations

DOUBLE PEAK FITNESS LANDSCAPE

Here, we observed Wright-Fischer population dynamics in a double-peak fitness landscape, with a monomorphic starting population. Each replicate ran for 10^5 generations, and finally the average fitnesses were plotted against corresponding generations. Figure 2.2.1 shows the general evolutionary dynamics for this case:

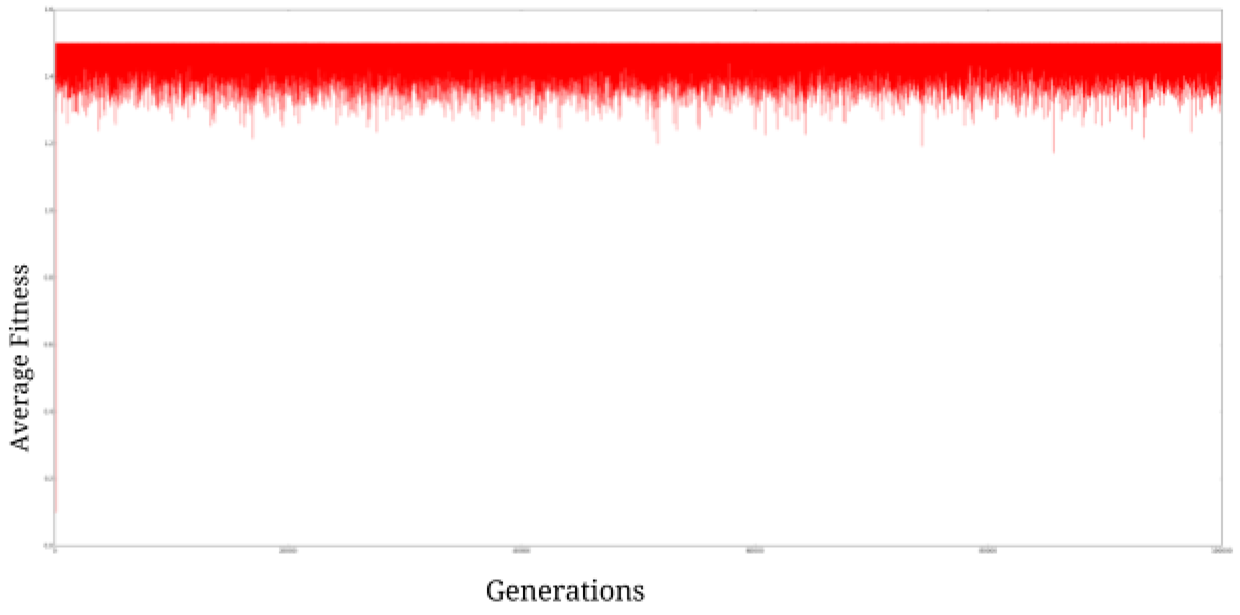

Fig. 2.2.1: Change in fitness over generations for a single-peak fitness landscape, starting with polymorphic population.

The identical trend of the leap in average fitness occurring suddenly can be seen here as well, across all replicates. The following Fig. 2.2.2. is a graph plotting the leap generation against the iteration and there is not distinctive pattern visible once again.

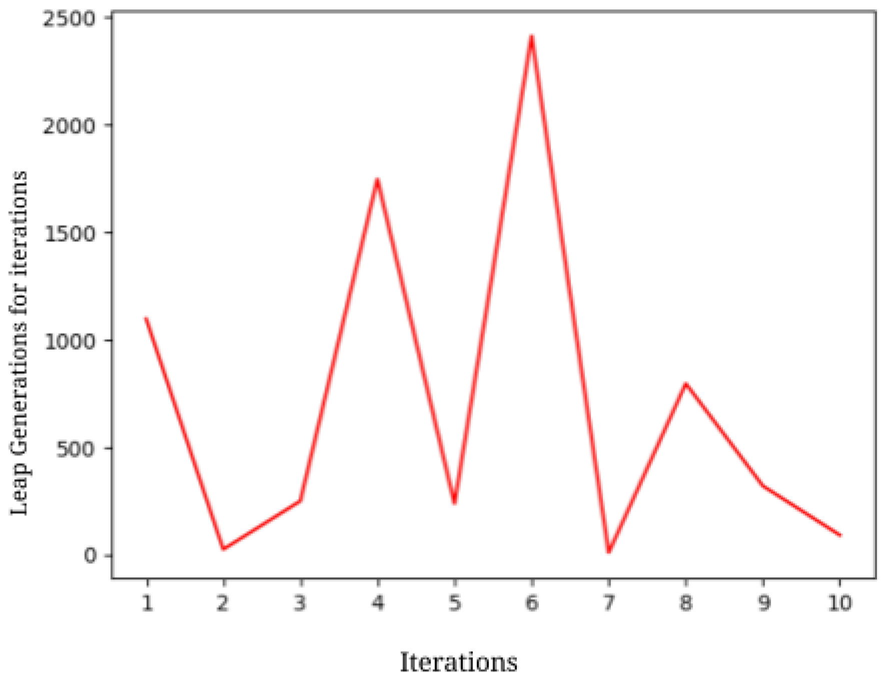

Fig 2.2.2: Leap generations for the iterations of double peak Wright-Fischer dynamics in polymorphic starting populations

OVERALL ANALYSIS FOR POLYMORPHIC STARTING GENERATIONS

Post the single-peak and double-peak analyses for the polymorphic inital populations, I compiled a graph to plot the arithmetic mean of leap generations against the fitness landcapes for the same.

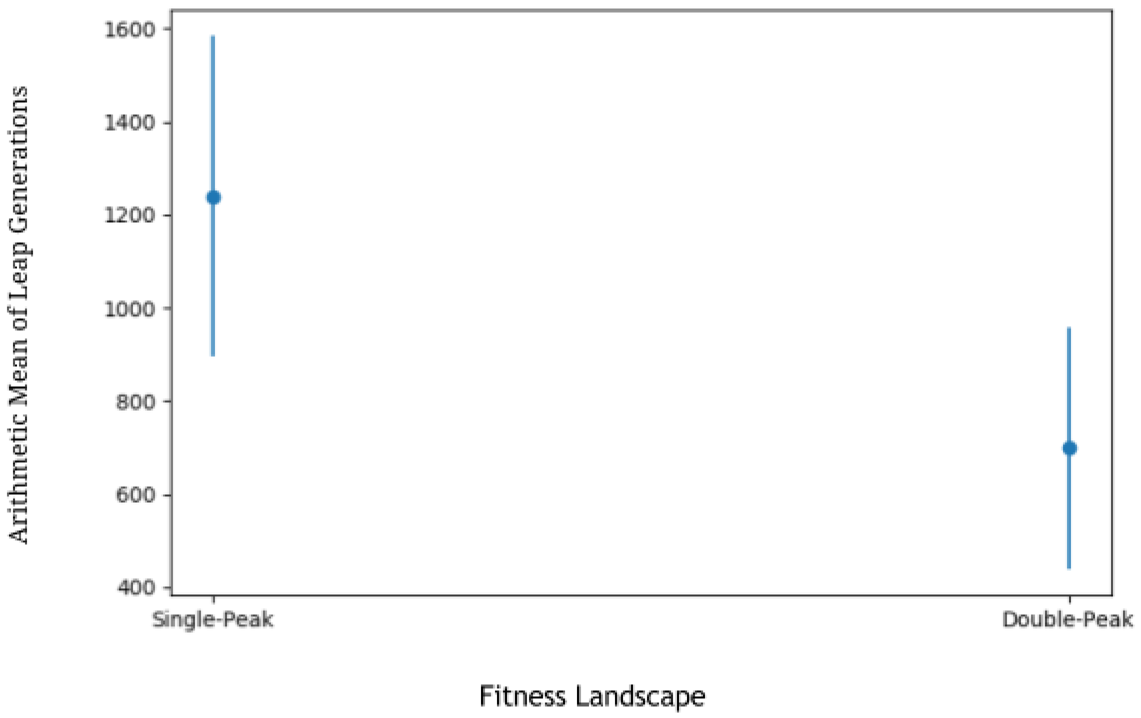

Fig 2.3.1: Arithmetic Mean of leap generations across the replications for single peak and double peak cases

Once again, we observe the immense difference of the arithmetic means of the leap generations. This could once again be explained by the fact that there is more availability of maximum fitness genotypes and epigenotypes in the double peak fitness landscape, as opposed to the single peak fitness landscape.

However, another observation here is that the arithmetic means of generations of fitness leap generally tend to be lower for the polymorphic case, as opposed to the monomorphic case. This can be easily demonstrated upon comparing figures 1.3.1 and 2.3.1

To explain this, I would like to introduce the concept of hamming distances. It is defined as: The number of digit positions in which the corresponding digits of two binary words of the same length are different [7]. When extended to the sequences of alleles at locii, it would work as follows:

For Organsim A having genotype [1, 0, 1, 1, 1, 1, 0, 1, 1, 1], and for Organism B having genotype [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], the hamming distance between their gentoypes would be ‘2’ because except for the difference of binary digits at 2 locii, both are identical.

When extended to the model, once can see that the indiviuals in the first generation of the monomorphic starting population would have a genotypic and epigenotypic hamming distance of ‘10’ with respect to the maximum fitness genotypes and epigenotypes ([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]) given the intial configuration of [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]. These would continue to fluctuate with time as the mutations occurred.

However, the genotypic and epigenotypic hamming distances of the individuals of the polymorphic starting population would vary from the very beginning, and hence on an average, be less than 10. This means it would take, on an average, a shorter time for these population to achieve high-fitness genotypes and epigenotypes. This justifies why, in polymorphic populations, less generations are required to achieve the sudden leap in average fitness.

CONCLUSIONS

1. Populations in single peak and double peak fitness landscapes, when undergoing Fischer-Wright dynamics with the condtions as described in the model, have a specific point in time/generation where their average fitness undergoes a drastic leap upaward. This phenomenon has been observed for cases of monomorphic and polymorphic starting populations.
2. This leap occurs, on an average, earlier for double peak cases in comparison to single peak cases, and earlier for polymorphic populations as compared to monomorphic populations.
3. The average fitness then continues to oscillate about a mean value, which has been verified for the case of monomorphic starting populations. The geometric mean of the last 1000 generations of a particular replication is used to keep track of this ‘average fitness mean’

CITATIONS

1. http://evolution.berkeley.edu/evolibrary/article/evo_17
2. Danchin, Étienne, et al. "Beyond DNA: integrating inclusive inheritance into an extended theory of evolution." *Nature Reviews Genetics* 12.7 (2011): 475-486.

3. Riggs AD, Russo VE, Martienssen RA (1996). *Epigenetic mechanisms of gene regulation*. Plainview, N.Y.: Cold Spring Harbor Laboratory Press. ISBN 0-87969-490-4.
4. Jablonka, Eva, and Gal Raz. "Transgenerational epigenetic inheritance: prevalence, mechanisms, and implications for the study of heredity and evolution." *The Quarterly review of biology* 84.2 (2009): 131-176.
5. Lauria, Massimiliano, et al. "Epigenetic variation, inheritance, and parent-of-origin effects of cytosine methylation in maize (Zea mays)." *Genetics* 196.3 (2014): 653-666.
6. Klironomos, Filippos D., Johannes Berg, and Sinéad Collins. "How epigenetic mutations can affect genetic evolution: model and mechanism." *Bioessays*35.6 (2013): 571-578.
7. https://www.its.bldrdoc.gov/fs-1037/dir-017/_2529.htm

APPENDIX

Code for Monomorphic Simulations - Single Peak

```
from __future__ import print_function, division

import numpy, time, csv

import matplotlib.pyplot as plt

import scipy.stats.mstats


#INITIALIZATIONS


start_time = time.time()

popn = 1000

locii = 10

total_gen = 150000

generations = list(range(1, (total_gen+1)))
```

```
ref_types = []


replicates = 10


for i in range(2**locii): #populating  reference array of reference genotypes and epigenotypes

    binaryRepresentation = list(format(i, 'b'))

    finalRepresentation = [['0']*(locii-len(binaryRepresentation)) + binaryRepresentation]

    ref_types = ref_types + finalRepresentation


ref_types =[[int(j) for j in i] for i in ref_types] #converting the string to int in the
refernce array


def partition(number, part):

    q1, r1 = divmod(number, part)

    indices = [q1 * i + min(i, r1) for i in xrange(part + 1)]

    lens = [indices[i + 1] - indices[i] for i in range(part)]

    return lens


def mutation(allele,freq): #defining mutation function

    temp = numpy.random.rand()

    if temp < freq:

        return(int(not(allele)))

    else:

        return(allele)
```

```
def individual_cycle(gef, offspring, locii, gfitnessref,efitnessref, ref_types):


    gef_transient = [gef]*offspring #generating transient offspring of an indivdual


    for i1 in range(offspring):

        for i2 in range(2): # (0 - genome, 1 - epigenome , 2 - fitness)

            for i3 in range(locii): #iterating through locii


                if i2 == 0: #mutation of genome

                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-6)


                if i2 == 1:#mutation of epigenome

                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-4)


        w_g = gfitnessref[ref_types.index(gef_transient[i1][0])] #to compare genotype of
individual to reference genotypes and get the genetic fitness

        w_e = efitnessref[ref_types.index(gef_transient[i1][1])] #to compare epigenotype of
individual to reference epigenotypes and get the epigenetic fitness

        gef_transient[i1][2] = max(w_g, w_e)


    return gef_transient #array of kids of the individual and their genome, epigenome,
fitnesses


def iterations(num, gef_small, offspring_small,locii,gfitnessref,efitnessref, ref_types):

    return sum([individual_cycle(gef_small[indiv],
offspring_small[indiv],locii,gfitnessref,efitnessref, ref_types) for indiv in range(num)],[])


# ******FOR PARALLEL PROCESSING ***********************
```

```
import pp

ppservers = ()

ncpus = 10

job_server = pp.Server(ncpus, ppservers=ppservers)

cores = job_server.get_ncpus()  # number of cores/workers being used

inputs = partition(popn, cores)  # Job distribution to each cores


# --------------------------------------------------------




gm_lastthousand = [] #array to store the gm of the avg_w of the last thousand generations of
every replicate

maxwdiff_gen = [] #array of the generations where increase in avg_w was max from the prvious
gen, for every iteration

maxwdiff = []


#CREATING the fitness landscape for this replication

gfitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]#populating reference arrays of
FITNESSES of genotpes and epigenotypes by using binary scheme

efitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]


for num in range(replicates):


    #populating gef array of first generation of this replication with monomorphic genome,
epigenome, and the correspong fitness (0.1, in this case)

    gef = [[[0]*locii, [0]*locii, 0.1] for i in range(popn)]

    avg_w = []  # will have the average fitness of each geenration

    w_diff = [0] #array of differences of avg_w of successive generartions
    for count in range(total_gen): #LOOP FOR A GENERATIONS WITHIN A SINGLE REPLICATION


        #SCALING THE FITNESSES AND GETTING ARRAY OF OFFSPRING NUMBER

        w_array = [gef[i][2] for i in range(popn)] #populating an array with only the 
fitnesses of the individuals

        fitness_sum = sum(w_array)

        avg_w += [fitness_sum/popn] #appending to an array of average fitnesses of each 
generation

        scaling_factor = 1 / fitness_sum  # calculating scaling factor for fitness

        w_scaled = list(map(lambda x: x*scaling_factor, w_array)) #multiplying each fitness by 
the scaling factor to update this population fitness array for this generation to use in 
multinomial function

        offspring = numpy.random.multinomial(popn, w_scaled) #generating number of offspring 
of every individual


        #gef_megatransient=iterations(popn,gef, offspring)

        #parallelisation code - to run individual_cycle on parts of the gef array, which are 
groups of individuals distributed to the cores

        job1 = [job_server.submit(iterations, (input, 
gef[sum(inputs[:ii]):sum(inputs[:ii])+input], 
offspring[sum(inputs[:ii]):sum(inputs[:ii])+input], locii, gfitnessref, efitnessref, 
ref_types,), (individual_cycle, mutation,), ("numpy",)) for ii, input in enumerate(inputs)]

        gef = sum([i() for i in job1], []) #summing up the offspring ka gef_transient arrays 
returned by individual_cycle


        #POST PROCESSING OF CODE



    for i in range(len(avg_w)-1): #storing differences of avg_w of successive generations

        w_diff += [avg_w[i+1] - avg_w[i]]
    maxwdiff_gen += [w_diff.index(max(w_diff))] #getting the generation where avg_w increase
was maximum

    maxwdiff += [max(w_diff)]

    gm_lastthousand += [scipy.stats.mstats.gmean(avg_w[149000:])] #storing gm of last thousand
generations' avg_w of current replicate


with open('gm_last_thousand2.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(gm_lastthousand)


with open('maxwdiff_gen2.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(maxwdiff_gen)


with open('maxwdiff2.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(maxwdiff)


plt.figure(figsize=(45, 20))

plt.plot(generations, avg_w, 'r')

plt.savefig('monomorphic, single peak, forwards+backwards2.png', bbox_inches = 'tight')


print("--- %s seconds ---" % (time.time() - start_time))
```

Code for Monomorphic Populations - Double Peak

```
from __future__ import print_function, division

import numpy, time, csv

import matplotlib.pyplot as plt

import scipy.stats.mstats


#INITIALIZATIONS


start_time = time.time()

popn = 1000

locii = 10

total_gen = 150000

generations = list(range(1, (total_gen+1)))


ref_types = []


replicates = 10


for i in range(2**locii): #populating  reference array of reference genotypes and epigenotypes

    binaryRepresentation = list(format(i, 'b'))

    finalRepresentation = [['0']*(locii-len(binaryRepresentation)) + binaryRepresentation]

    ref_types = ref_types + finalRepresentation


ref_types =[[int(j) for j in i] for i in ref_types] #converting the string to int in the
refernce array


def partition(number, part):

    q1, r1 = divmod(number, part)
    indices = [q1 * i + min(i, r1) for i in xrange(part + 1)]

    lens = [indices[i + 1] - indices[i] for i in range(part)]

    return lens


def mutation(allele,freq): #defining mutation function

    temp = numpy.random.rand()

    if temp < freq:

        return(int(not(allele)))

    else:

        return(allele)




def individual_cycle(gef, offspring, locii, gfitnessref,efitnessref, ref_types):



    gef_transient = [gef]*offspring #generating transient offspring of an indivdual



    for i1 in range(offspring):

        for i2 in range(2): # (0 - genome, 1 - epigenome , 2 - fitness)

            for i3 in range(locii): #iterating through locii



                if i2 == 0: #mutation of genome

                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-6)



                if i2 == 1:#mutation of epigenome

                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-4)



        w_g = gfitnessref[ref_types.index(gef_transient[i1][0])] #to compare genotype of
```

```
individual to reference genotypes and get the genetic fitness

        w_e = efitnessref[ref_types.index(gef_transient[i1][1])] #to compare epigenotype of
individual to reference epigenotypes and get the epigenetic fitness

        gef_transient[i1][2] = max(w_g, w_e)


    return gef_transient #array of kids of the individual and their genome, epigenome,
fitnesses


def iterations(num, gef_small, offspring_small,locii,gfitnessref,efitnessref, ref_types):

    return sum([individual_cycle(gef_small[indiv],
offspring_small[indiv],locii,gfitnessref,efitnessref, ref_types) for indiv in range(num)],[])


# ******FOR PARALLEL PROCESSING ************************

import pp

ppservers = ()

ncpus = 10

job_server = pp.Server(ncpus, ppservers=ppservers)

cores = job_server.get_ncpus()  # number of cores/workers being used

inputs = partition(popn, cores)  # Job distribution to each cores


# --------------------------------------------------------




gm_lastthousand = [] #array to store the gm of the avg_w of the last thousand generations of
every replicate

maxwdiff_gen = [] #array of the generations where increase in avg_w was max from the prvious
gen, for every iteration

maxwdiff = []
```

```
#CREATING the fitness landscape for this replication

gfitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]#populating reference arrays of
FITNESSES of genotpes and epigenotypes by using binary scheme

gfitnessref[682] = 1.5

efitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]

efitnessref[682] = 1.5


for num in range(replicates):


    #populating gef array of first generation of this replication with monomorphic genome,
epigenome, and the correspong fitness (0.1, in this case)

    gef = [[[0]*locii, [0]*locii, 0.1] for i in range(popn)]

    avg_w = []  # will have the average fitness of each geenration

    w_diff = [0] #array of differences of avg_w of successive generartions


    for count in range(total_gen): #LOOP FOR A GENERATIONS WITHIN A SINGLE REPLICATION


        #SCALING THE FITNESSES AND GETTING ARRAY OF OFFSPRING NUMBER

        w_array = [gef[i][2] for i in range(popn)] #populating an array with only the
fitnesses of the individuals

        fitness_sum = sum(w_array)

        avg_w += [fitness_sum/popn] #appending to an array of average fitnesses of each
generation

        scaling_factor = 1 / fitness_sum  # calculating scaling factor for fitness

        w_scaled = list(map(lambda x: x*scaling_factor, w_array)) #multiplying each fitness by
the scaling factor to update this population fitness array for this generation to use in
multinomial function

        offspring = numpy.random.multinomial(popn, w_scaled) #generating number of offspring
of every individual
```

```
        #gef_megatransient=iterations(popn,gef, offspring)

        #parallelisation code - to run individual_cycle on parts of the gef array, which are
groups of individuals distributed to the cores

        job1 = [job_server.submit(iterations, (input,
gef[sum(inputs[:ii]):sum(inputs[:ii])+input],
offspring[sum(inputs[:ii]):sum(inputs[:ii])+input], locii, gfitnessref, efitnessref,
ref_types,), (individual_cycle, mutation,), ("numpy",)) for ii, input in enumerate(inputs)]

        gef = sum([i() for i in job1], []) #summing up the offspring ka gef_transient arrays
returned by individual_cycle



        #POST PROCESSING OF CODE




    for i in range(len(avg_w)-1): #storing differences of avg_w of successive generations

        w_diff += [avg_w[i+1] - avg_w[i]]



    maxwdiff_gen += [w_diff.index(max(w_diff))] #getting the generation where avg_w increase
was maximum, for this iteration

    maxwdiff += [max(w_diff)]

    gm_lastthousand += [scipy.stats.mstats.gmean(avg_w[149000:])] #storing gm of last thousand
generations' avg_w of current iteration


with open('gm_last_thousand2.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(gm_lastthousand)


with open('maxwdiff_gen2.csv', 'w') as myfile:

    wr = csv.writer(myfile)
    wr.writerow(maxwdiff_gen)

with open('maxwdiff2.csv', 'w') as myfile:
    wr = csv.writer(myfile)
    wr.writerow(maxwdiff)

plt.figure(figsize=(45, 20))
plt.plot(generations, avg_w, 'r')
plt.savefig('monomorphic, double peak, forwards+backwards2.png', bbox_inches = 'tight')


print("--- %s seconds ---" % (time.time() - start_time))
```

Code for Polymorphic populations - Single Peak

```
from __future__ import print_function, division
import numpy, time, csv
import matplotlib.pyplot as plt
import scipy.stats.mstats

#INITIALIZATIONS

start_time = time.time()
popn = 1000
locii = 10
total_gen = 100000
generations = list(range(1, (total_gen+1)))
```

```
ref_types = []

replicates = 10

for i in range(2**locii): #populating  reference array of reference genotypes and epigenotypes
    binaryRepresentation = list(format(i, 'b'))
    finalRepresentation = [['0']*(locii-len(binaryRepresentation)) + binaryRepresentation]
    ref_types = ref_types + finalRepresentation

ref_types =[[int(j) for j in i] for i in ref_types] #converting the string to int in the
refernce array

def partition(number, part):
    q1, r1 = divmod(number, part)
    indices = [q1 * i + min(i, r1) for i in xrange(part + 1)]
    lens = [indices[i + 1] - indices[i] for i in range(part)]
    return lens

def mutation(allele,freq): #defining mutation function
    temp = numpy.random.rand()
    if temp < freq:
        return(int(not(allele)))
    else:
        return(allele)


def individual_cycle(gef, offspring, locii, gfitnessref,efitnessref, ref_types):
    gef_transient = [gef]*offspring #generating transient offspring of an indivdual

    for i1 in range(offspring):
        for i2 in range(2): # (0 - genome, 1 - epigenome , 2 - fitness)
            for i3 in range(locii): #iterating through locii

                if i2 == 0: #mutation of genome
                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-6)

                if i2 == 1:#mutation of epigenome
                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-4)

        w_g = gfitnessref[ref_types.index(gef_transient[i1][0])] #to compare genotype of
individual to reference genotypes and get the genetic fitness
        w_e = efitnessref[ref_types.index(gef_transient[i1][1])] #to compare epigenotype of
individual to reference epigenotypes and get the epigenetic fitness
        gef_transient[i1][2] = max(w_g, w_e)

    return gef_transient #array of kids of the individual and their genome, epigenome,
fitnesses

def iterations(num, gef_small, offspring_small,locii,gfitnessref,efitnessref, ref_types):
    return sum([individual_cycle(gef_small[indiv],
offspring_small[indiv],locii,gfitnessref,efitnessref, ref_types) for indiv in range(num)],[])

# ******FOR PARALLEL PROCESSING ************************
import pp
```

```
ppservers = ()

ncpus = 10

job_server = pp.Server(ncpus, ppservers=ppservers)

cores = job_server.get_ncpus()  # number of cores/workers being used

inputs = partition(popn, cores)  # Job distribution to each cores


# --------------------------------------------------------




gm_lastthousand = [] #array to store the gm of the avg_w of the last thousand generations of
every replicate

maxwdiff_gen = [] #array of the generations where increase in avg_w was max from the prvious
gen, for every iteration

maxwdiff = []


#CREATING the fitness landscape for the single peak

gfitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]#populating reference arrays of
FITNESSES of genotpes and epigenotypes by using binary scheme

efitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]


#getting the common initial polymorphic population

#gef_init = numpy.loadtxt('init_gen.csv')

dummy=[]

with open('init_gen.csv', 'rb') as csvfile:

    gef_init= (csv.reader(csvfile,delimiter=','))

    for i in gef_init:

        for j in i:

            #print (type(eval(j)))
            dummy+=list(eval(j))


gef_init=numpy.array(dummy[:])

gef_init=gef_init.reshape(1000, 3)

print(len(gef_init))




for i1 in range(popn): #makingchanges to fitness of the individual on the basis of genome and
epigenome

    gef_init[i1][2] = max(gfitnessref[ref_types.index(gef_init[i1][0])],
efitnessref[ref_types.index(gef_init[i1][1])])


for num in range(replicates):


    #populating gef array of first generation of this replication with monomorphic genome,
epigenome, and the correspong fitness (0.1, in this case)

    gef = gef_init

    avg_w = []  # will have the average fitness of each geenration

    w_diff = [0] #array of differences of avg_w of successive generartions


    for count in range(total_gen): #LOOP FOR A GENERATIONS WITHIN A SINGLE REPLICATION


        #SCALING THE FITNESSES AND GETTING ARRAY OF OFFSPRING NUMBER

        w_array = [gef[i][2] for i in range(popn)] #populating an array with only the
fitnesses of the individuals

        fitness_sum = sum(w_array)

        avg_w += [fitness_sum/popn] #appending to an array of average fitnesses of each
generation

        scaling_factor = 1 / fitness_sum  # calculating scaling factor for fitness
        w_scaled = list(map(lambda x: x*scaling_factor, w_array)) #multiplying each fitness by
the scaling factor to update this population fitness array for this generation to use in
multinomial function

        offspring = numpy.random.multinomial(popn, w_scaled) #generating number of offspring
of every individual



        #gef_megatransient=iterations(popn,gef, offspring)

        #parallelisation code - to run individual_cycle on parts of the gef array, which are
groups of individuals distributed to the cores

        job1 = [job_server.submit(iterations, (input,
gef[sum(inputs[:ii]):sum(inputs[:ii])+input],
offspring[sum(inputs[:ii]):sum(inputs[:ii])+input], locii, gfitnessref, efitnessref,
ref_types,), (individual_cycle, mutation,), ("numpy",)) for ii, input in enumerate(inputs)]

        gef = sum([i() for i in job1], []) #summing up the offspring ka gef_transient arrays
returned by individual_cycle



        #POST PROCESSING OF CODE





    for i in range(len(avg_w)-1): #storing differences of avg_w of successive generations

        w_diff += [avg_w[i+1] - avg_w[i]]



    maxwdiff_gen += [w_diff.index(max(w_diff))] #getting the generation where avg_w increase
was maximum

    maxwdiff += [max(w_diff)]

    gm_lastthousand += [scipy.stats.mstats.gmean(avg_w[149000:])] #storing gm of last thousand
generations' avg_w of current replicate



with open('gm_last_thousand_single.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(gm_lastthousand)
```

```
with open('maxwdiff_gen_single.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(maxwdiff_gen)


with open('maxwdiff_single.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(maxwdiff)


plt.figure(figsize=(45, 20))

plt.plot(generations, avg_w, 'r')

plt.savefig('polymorphic, single peak, forwards+backwards.png', bbox_inches = 'tight')

print("--- %s seconds ---" % (time.time() - start_time))
```

Code for Polymorphic populations - Double Peak

```
from __future__ import print_function, division

import numpy, time, csv

import matplotlib.pyplot as plt

import scipy.stats.mstats


#INITIALIZATIONS


start_time = time.time()

popn = 1000

locii = 10

total_gen = 100000

generations = list(range(1, (total_gen+1)))
```

```
ref_types = []


replicates = 10


for i in range(2**locii): #populating  reference array of reference genotypes and epigenotypes

    binaryRepresentation = list(format(i, 'b'))

    finalRepresentation = [['0']*(locii-len(binaryRepresentation)) + binaryRepresentation]

    ref_types = ref_types + finalRepresentation


ref_types =[[int(j) for j in i] for i in ref_types] #converting the string to int in the
refernce array


def partition(number, part):

    q1, r1 = divmod(number, part)

    indices = [q1 * i + min(i, r1) for i in xrange(part + 1)]

    lens = [indices[i + 1] - indices[i] for i in range(part)]

    return lens


def mutation(allele,freq): #defining mutation function

    temp = numpy.random.rand()

    if temp < freq:

        return(int(not(allele)))

    else:

        return(allele)
```

```
def individual_cycle(gef, offspring, locii, gfitnessref,efitnessref, ref_types):

    gef_transient = [gef]*offspring #generating transient offspring of an indivdual

    for i1 in range(offspring):
        for i2 in range(2): # (0 - genome, 1 - epigenome , 2 - fitness)
            for i3 in range(locii): #iterating through locii

                if i2 == 0: #mutation of genome
                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-6)

                if i2 == 1:#mutation of epigenome
                    gef_transient[i1][i2][i3] = mutation(gef_transient[i1][i2][i3], 10**-4)

        w_g = gfitnessref[ref_types.index(gef_transient[i1][0])] #to compare genotype of
individual to reference genotypes and get the genetic fitness
        w_e = efitnessref[ref_types.index(gef_transient[i1][1])] #to compare epigenotype of
individual to reference epigenotypes and get the epigenetic fitness
        gef_transient[i1][2] = max(w_g, w_e)

    return gef_transient #array of kids of the individual and their genome, epigenome,
fitnesses

def iterations(num, gef_small, offspring_small,locii,gfitnessref,efitnessref, ref_types):
    return sum([individual_cycle(gef_small[indiv],
offspring_small[indiv],locii,gfitnessref,efitnessref, ref_types) for indiv in range(num)],[])

# ******FOR PARALLEL PROCESSING ************************
```

```
import pp

ppservers = ()

ncpus = 10

job_server = pp.Server(ncpus, ppservers=ppservers)

cores = job_server.get_ncpus()  # number of cores/workers being used

inputs = partition(popn, cores)  # Job distribution to each cores


# --------------------------------------------------------




gm_lastthousand = [] #array to store the gm of the avg_w of the last thousand generations of
every replicate

maxwdiff_gen = [] #array of the generations where increase in avg_w was max from the prvious
gen, for every iteration

maxwdiff = []


#CREATING the fitness landscape for this replication

gfitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]#populating reference arrays of
FITNESSES of genotpes and epigenotypes by using binary scheme

gfitnessref[682] = 1.5

efitnessref = [0.1 for i in range((2**locii)-1)] + [1.5]

efitnessref[682] = 1.5


#getting the common initial polymorphic population

#gef_init = numpy.loadtxt('init_gen.csv')

dummy=[]

with open('init_gen.csv', 'rb') as csvfile:

    gef_init= (csv.reader(csvfile,delimiter=','))
    for i in gef_init:

        for j in i:

            #print (type(eval(j)))

            dummy+=list(eval(j))


gef_init=numpy.array(dummy[:])

gef_init=gef_init.reshape(1000, 3)


for i1 in range(popn): #makingchanges to fitness of the individual on the basis of genome and
epigenome

    gef_init[i1][2] = max(gfitnessref[ref_types.index(gef_init[i1][0])],
efitnessref[ref_types.index(gef_init[i1][1])])


for num in range(replicates):


    #populating gef array of first generation of this replication with monomorphic genome,
epigenome, and the correspong fitness (0.1, in this case)

    gef = gef_init

    avg_w = []  # will have the average fitness of each geenration

    w_diff = [0] #array of differences of avg_w of successive generartions


    for count in range(total_gen): #LOOP FOR A GENERATIONS WITHIN A SINGLE REPLICATION


        #SCALING THE FITNESSES AND GETTING ARRAY OF OFFSPRING NUMBER

        w_array = [gef[i][2] for i in range(popn)] #populating an array with only the
fitnesses of the individuals

        fitness_sum = sum(w_array)

        avg_w += [fitness_sum/popn] #appending to an array of average fitnesses of each
generation
```

```
        scaling_factor = 1 / fitness_sum  # calculating scaling factor for fitness

        w_scaled = list(map(lambda x: x*scaling_factor, w_array)) #multiplying each fitness by
the scaling factor to update this population fitness array for this generation to use in
multinomial function

        offspring = numpy.random.multinomial(popn, w_scaled) #generating number of offspring
of every individual



        #gef_megatransient=iterations(popn,gef, offspring)

        #parallelisation code - to run individual_cycle on parts of the gef array, which are
groups of individuals distributed to the cores

        job1 = [job_server.submit(iterations, (input,
gef[sum(inputs[:ii]):sum(inputs[:ii])+input],
offspring[sum(inputs[:ii]):sum(inputs[:ii])+input], locii, gfitnessref, efitnessref,
ref_types,), (individual_cycle, mutation,), ("numpy",)) for ii, input in enumerate(inputs)]

        gef = sum([i() for i in job1], []) #summing up the offspring ka gef_transient arrays
returned by individual_cycle



        #POST PROCESSING OF CODE





    for i in range(len(avg_w)-1): #storing differences of avg_w of successive generations

        w_diff += [avg_w[i+1] - avg_w[i]]



    maxwdiff_gen += [w_diff.index(max(w_diff))] #getting the generation where avg_w increase
was maximum, for this iteration

    maxwdiff += [max(w_diff)]

    gm_lastthousand += [scipy.stats.mstats.gmean(avg_w[149000:])] #storing gm of last thousand
generations' avg_w of current iteration



with open('gm_last_thousand_double.csv', 'w') as myfile:

    wr = csv.writer(myfile)
    wr.writerow(gm_lastthousand)


with open('maxwdiff_gen_double.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(maxwdiff_gen)


with open('maxwdiff_double.csv', 'w') as myfile:

    wr = csv.writer(myfile)

    wr.writerow(maxwdiff)


plt.figure(figsize=(45, 20))

plt.plot(generations, avg_w, 'r')

plt.savefig('polymorphic, double peak, forwards+backwards2.png', bbox_inches = 'tight')



print("--- %s seconds ---" % (time.time() - start_time))
```