

Solutions for Problem Set 1

Written by: Anita Kulkarni and Benjamin Good
(last updated on January 28, 2021)

Sample code is provided at the end of the document.

Problem 1: Molecular evolution and genetic diversity in the influenza virus

Part (a)

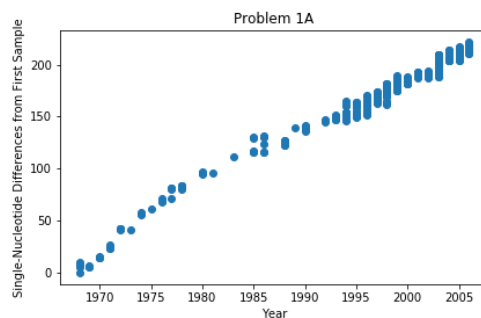


Figure 1: Number of single nucleotide differences between first HA gene sample (A/Aichi/2/1968) and others as a function of sampling year.

Approximately ~ 200 (more like 210-220) differences accumulated over ~ 40 years. This corresponds to roughly 12-13% of the HA gene.

Part (b)

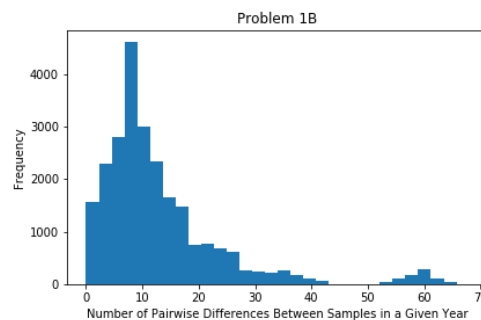


Figure 2: Distribution of number of genetic differences between all pairs of strains from the same year, aggregated across all years.

Most co-circulating strains vary at about 10 sites on average. Comparing this to the slope from part (a), we see that this corresponds to a turnover time of ≈ 2 -3 years.

Problem 2: The Luria-Delbrück experiment

Part (a)

At $t = 0$, there are N_0 individuals without any antibiotic resistance, and at each subsequent time step t we assume that all $N(t)$ individuals divide, thus doubling the population ($2N(t)$ daughter cells). During this process, all daughter cells have an equal probability μ of acquiring a mutation; thus, the mean number of new mutations produced in generation t is

$$\theta(t) = \mu N_0 2^t$$

(Note that this formula is only valid for $1 \leq t \leq T$.)

Part (b)

If a mutation arises at generation t , then it will have $T - t$ generations to leave descendants by growing exponentially. Thus,

$$n(t) = 2^{T-t}$$

The total number of descendants left by all the mutants that arise at time t is $m(t)n(t)$; thus,

$$M_T = \sum_{t=1}^T m(t)n(t) = \sum_{t=1}^T 2^{T-t}m(t)$$

Part (c)

Mean of M_T :

$$\begin{aligned} \langle M_T \rangle &= E \left[\sum_{t=1}^T 2^{T-t}m(t) \right] = \sum_{t=1}^T E[2^{T-t}m(t)] = \sum_{t=1}^T 2^{T-t}E[m(t)] = \sum_{t=1}^T 2^{T-t}\theta(t) \\ &= \sum_{t=1}^T 2^{T-t}\mu N_0 2^t = \mu N_0 2^T \sum_{t=1}^T 1 = T \cdot N_0 \mu \cdot 2^T \end{aligned}$$

Use a similar approach for the variance (i.e. properties of the variance of linear combinations of independent random variables, as each time point is independent), noting that since $m(t)$ is Poisson distributed its variance will be $\theta(t)$:

$$\begin{aligned} \text{Var}(M_T) &= \text{Var} \left(\sum_{t=1}^T 2^{T-t}m(t) \right) = \sum_{t=1}^T \text{Var}(2^{T-t}m(t)) = \sum_{t=1}^T 2^{2T-2t}\text{Var}(m(t)) = \sum_{t=1}^T 2^{2T-2t}\theta(t) \\ &= \sum_{t=1}^T 2^{2T-2t}\mu N_0 2^t = \mu N_0 2^{2T} \sum_{t=1}^T 2^{-t} = \mu N_0 4^T (1 - 2^{-T}) = \mu N_0 2^T (2^T - 1) \approx N_0 \mu \cdot (2^T)^2 \end{aligned}$$

The Fano factor is thus

$$F = \frac{2^T - 1}{T} \approx \frac{2^T}{T}$$

F which is larger than the Poisson limit by a factor of $2^T/T \gg 1$.

This suggests that we should be able to distinguish between the induction and mutation hypotheses by calculating this Fano factor from the observed data. If the variance in the number of colonies seen is much larger than the mean (i.e. there are a few plates with very many colonies and the rest have no or very few colonies), then the mutation hypothesis would be supported. If the mean and variance are similar (i.e. almost all of the plates have just a few colonies), then the induction hypothesis would be supported.

Part (d)

First, calculate $\langle \overline{M}_T \rangle$:

$$\langle \overline{M}_T \rangle = \frac{1}{n} \sum_{i=1}^n \langle M_{T,i} \rangle = \frac{n}{n} \mu N_0 T 2^T = T \cdot N_0 \mu \cdot 2^T$$

Next, calculate $\text{Var}(\overline{M}_T)$:

$$\text{Var}(\overline{M}_T) = \text{Var}\left(\frac{1}{n} \sum_{i=1}^n M_{T,i}\right) = \frac{1}{n^2} \text{Var}\left(\sum_{i=1}^n M_{T,i}\right) = \frac{1}{n} \text{Var}(M_T) = \frac{1}{n} \cdot N_0 \mu \cdot 2^T (2^T - 1)$$

After some algebra, we get that the coefficient of variation is

$$c_V = \frac{1}{T} \sqrt{\frac{1 - 2^{-T}}{n \mu N_0}} \approx \frac{1}{T \sqrt{n N_0 \mu}}$$

Setting this equal to ϵ and solving for n , we get that we need

$$n \approx \frac{1 - 2^{-T}}{N_0 \mu \epsilon^2 T^2} \approx \frac{1}{N_0 \mu \epsilon^2 T^2}$$

independent experiments to get $O(\epsilon)$ relative error. When $N_0 \mu \ll 1$, the CV and n get very large; lots of experiments are needed to precisely estimate $\langle M_T \rangle$ and $\text{Var}(M_T)$ as the mutation rate gets small.

Part (e)

Calculate $\theta_{<}(t|n)$, or the total number of mutations among all n populations expected to arise before generation t :

$$n \sum_{j=1}^{t-1} \theta(j) = n \sum_{j=1}^{t-1} N_0 \mu 2^j = n N_0 \mu \sum_{j=1}^{t-1} 2^j = n N_0 \mu (2^t - 2)$$

Set this equal to 1 to find t^* :

$$n N_0 \mu (2^{t^*} - 2) = 1 \implies 2^{t^*} = \frac{1}{n N_0 \mu} + 2 \implies t^* = \log_2 \left(\frac{1}{n N_0 \mu} + 2 \right)$$

By definition, this critical time t^* only makes sense when $t^* \leq T$, which requires that

$$nN_0\mu 2^T \geq 1 \quad (1)$$

(in other words, we should typically expect to have at least one mutation in one of the replicates by the end of the experiment)

Using these expressions, we find that the typical mean is given by

$$\langle \overline{M}_T \rangle_{\text{typ}} = \langle M_T \rangle_{\text{typ}} = \sum_{t=1}^T 2^{T-t} \hat{\theta}(t|n) = \sum_{t=t^*}^T 2^{T-t} \theta(t) = N_0\mu 2^T (T - t^* + 1)$$

and the typical variance is

$$\text{Var}(\overline{M}_T)_{\text{typ}} = \frac{1}{n} \text{Var}(M_T)_{\text{typ}} = \frac{1}{n} \sum_{t=t^*}^T 2^{2T-2t} N_0\mu 2^t = \frac{1}{n} N_0\mu 2^T [2^{T-t^*+1} - 1]$$

both of which depend on t^* only through the compound parameter

$$T - t^* + 1 = \log_2 \left(\frac{2 \cdot nN_0\mu 2^T}{1 + 2nN_0\mu} \right) \quad (2)$$

The coefficient of variation is therefore given by

$$c_V = \frac{\sqrt{\text{Var}(\overline{M}_T)_{\text{typ}}}}{\langle \overline{M}_T \rangle_{\text{typ}}} = \sqrt{\frac{2^{T-t^*+1} - 1}{(T - t^* + 1)^2 n N_0\mu 2^T}} = \frac{\sqrt{\frac{2}{1+2nN_0\mu} - \frac{1}{nN_0\mu 2^T}}}{\log_2 \left(\frac{2 \cdot nN_0\mu 2^T}{1+2nN_0\mu} \right)}$$

which no longer blows up for small $nN_0\mu$, since we have assumed that $nN_0\mu 2^T$ is always greater than 1. We note, however, that the coefficient of variation does not decay as $n^{-1/2}$ as we would expect from the central limit theorem ($\sim n^{-1/2}$), but instead displays a much slower logarithmic decay.

Part (f)

We know that $M'_T \sim \text{Poisson} \left(N_0 \frac{M_T}{N_T} \right) = \text{Poisson} \left(\frac{M_T}{2^T} \right) \equiv \text{Poisson}(R)$ (we're defining a new random variable $R \equiv M_T/2^T$). Then:

$$\langle M'_T \rangle = E[\text{Poisson}(R)] = E[R] = \frac{1}{2^T} E[M_T] = \frac{1}{2^T} N_0\mu T 2^T = N_0\mu T$$

To find the variance of M'_T , use the formula $\text{Var}(X) = E[\text{Var}(X|Y=y)] + \text{Var}(E[X|Y=y])$:

$$\text{Var}(M'_T|R) = R(=r) \implies E[R] = N_0\mu T$$

$$E[M'_T|R] = R(=r) \implies \text{Var}(R) = \frac{1}{4^T} \text{Var}(M_T) = N_0\mu(1 - 2^{-T})$$

$$\implies \text{Var}(M'_T) = N_0\mu(T + 1 - 2^{-T})$$

So

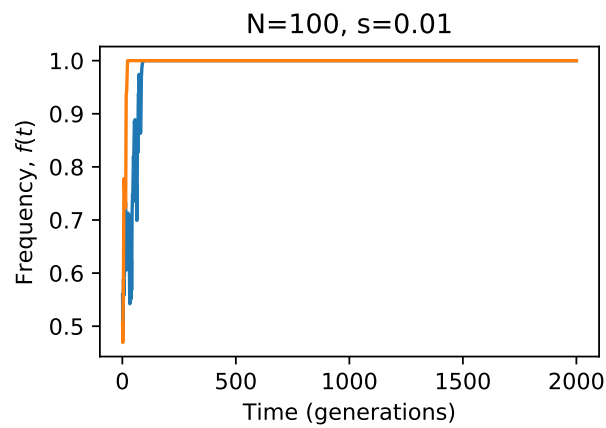
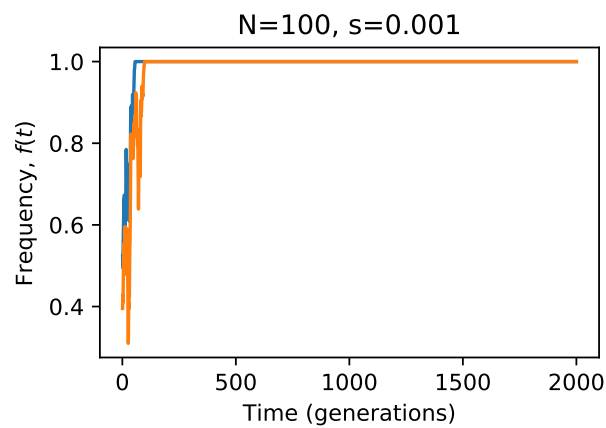
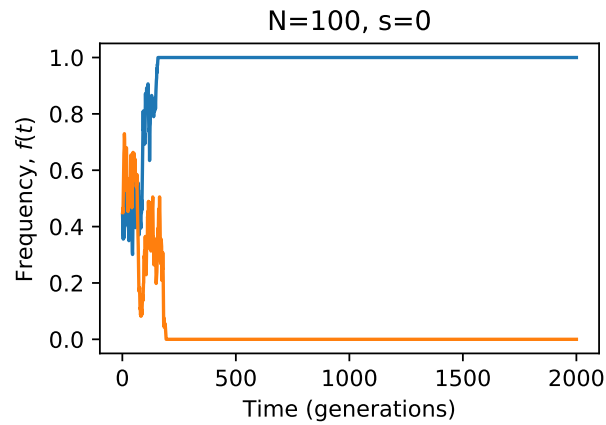
$$F = 1 + \frac{1 - 2^{-T}}{T} \approx 1 + \frac{1}{T}$$

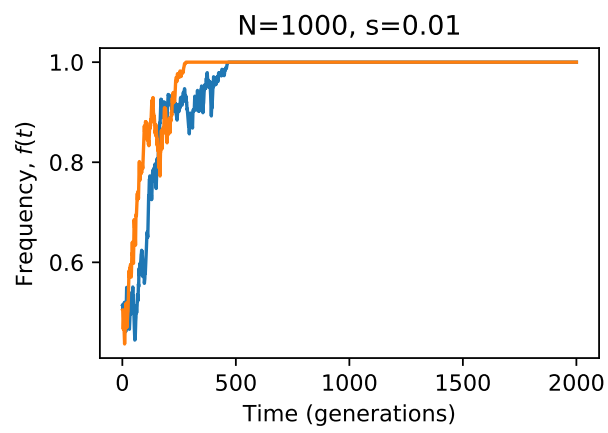
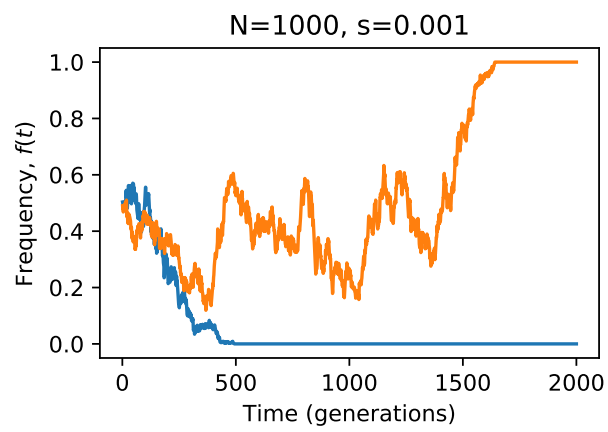
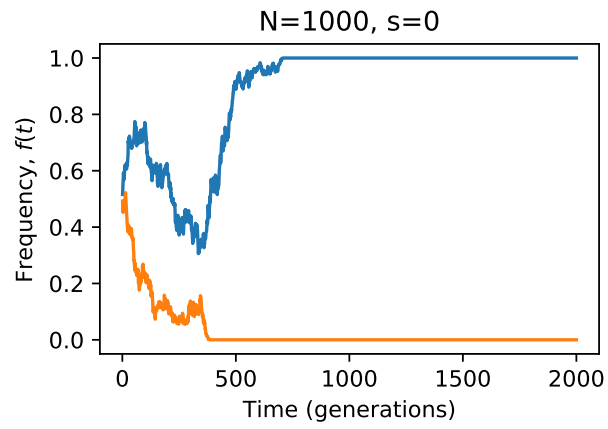
which approaches the Poisson limit of $F \approx 1$ when $T \gg 1$. For example, for a dilution factor of $2^T = 100$, we have $T \approx 6.7$ and $F \approx 1.14$ – a relatively small deviation from the Poisson approximation we used in class.

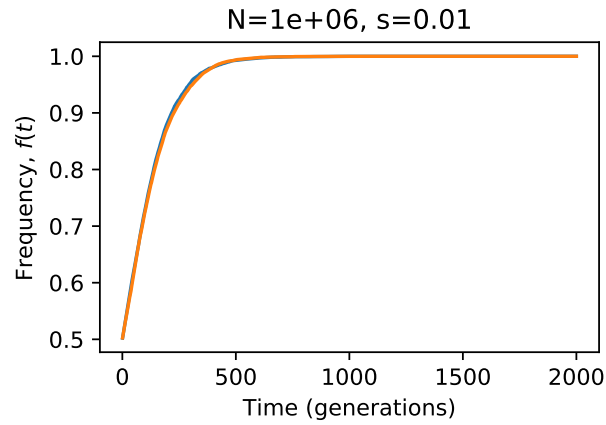
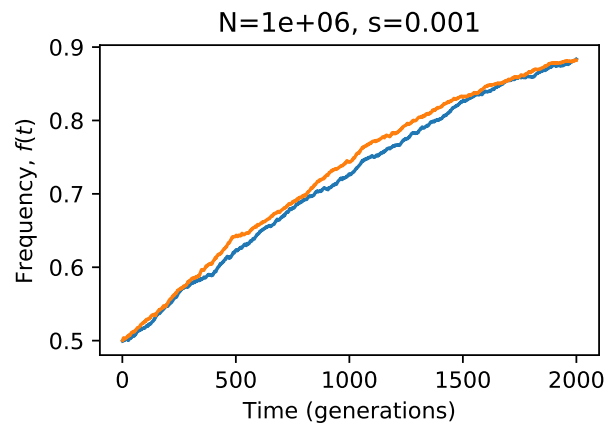
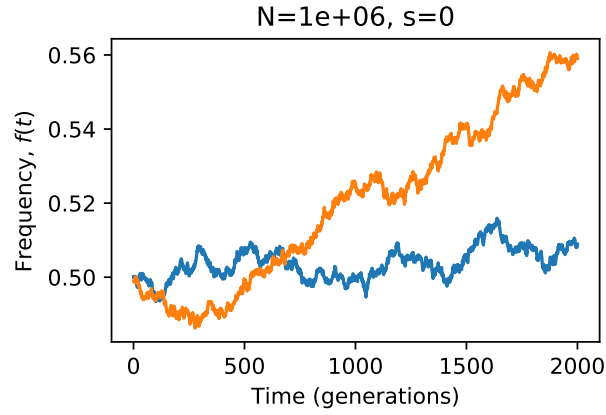
Problem 3:

Part (a)

Two replicate simulations are shown for each of the parameter combinations below:

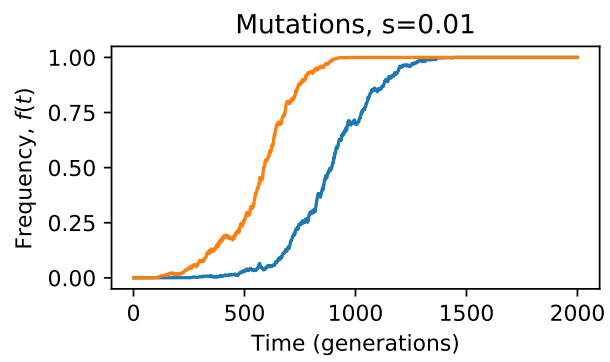
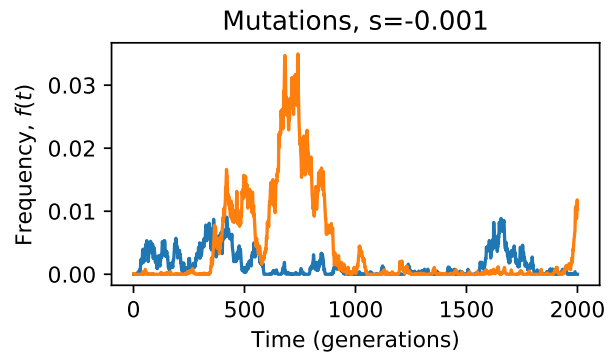






Part (b)

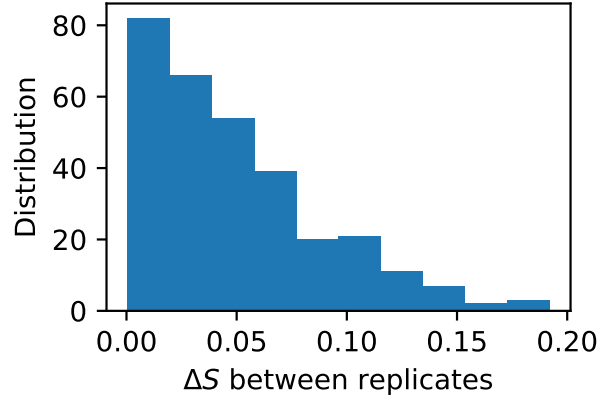
Two replicate simulations with de novo mutations are shown for two different selection coefficients below, with the other parameters fixed at $N = 10^4$ and $\mu = 10^{-5}$:



Problem 4:

Part (a)

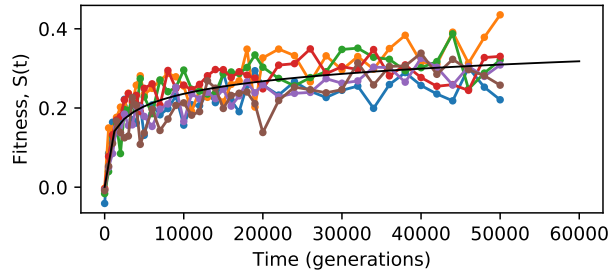
The observed distribution of differences between replicate fitness measurements is shown below:



This shows that the typical errors are on the order of $\sigma_S \sim 5\%$.

Part (b)

The estimated fitness trajectories for each population are shown in the colored lines below:

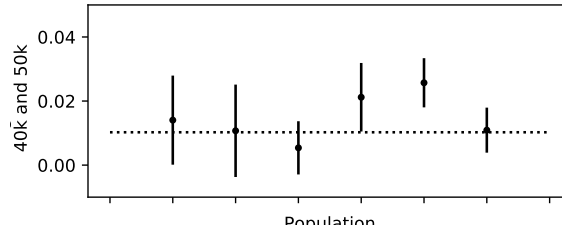


Part (c)

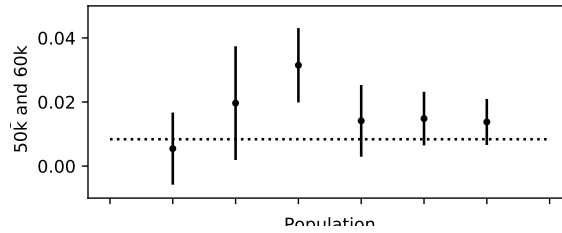
The predicted trajectory, $X(t) = X_c \log(1 + v_0 t / X_c)$ is shown in the solid black line in the plot in part (b) for $X_c \approx 4.6 \times 10^{-2}$ and $v_0 = 7.7 \times 10^{-4}$. It seems to be roughly consistent with the data. The predicted fitness gain between generation 40,000 and 50,000 is $\Delta X \approx 0.01$, which is well within the range of uncertainty on the individual fitness measurements above.

Part (d)

Using the more highly replicated fitness assays in the provided file, the fitness gains between generation 40,000 and 50,000 are shown for different replicate populations below:



The error bars denote ± 2 standard errors, and the dashed line indicates the prediction from the theoretical model in Part (c). An analogous figure for generation 50,000 to 60,000 is shown below:



In both cases, the error bars exclude 0 for most of the replicate populations, suggesting that fitness is still improving in the Lenski experiment. (We note however, that the statistical support for the predicted fitness trajectory is much weaker.)

Problem 5: Pooled fitness assay

Part (a)

Given that each cycle is of length Δt and each strain grows as $N_k(t) = N_k(0)e^{s_k t}$,

$$f_k(\Delta t) = \frac{N_k(\Delta t)}{\sum_{i=1}^K N_i(\Delta t)} = \frac{N_k(0)e^{s_k \Delta t}}{\sum_{i=1}^K N_i(0)e^{s_i \Delta t}} = \frac{\frac{1}{\sum_{i=1}^K N_i(0)} e^{s_k \Delta t} N_k(0)}{\frac{1}{\sum_{j=1}^K N_j(0)} \sum_{i=1}^K N_i(0)e^{s_i \Delta t}} = \frac{f_k(0)e^{s_k \Delta t}}{\sum_{i=1}^K f_i(0)e^{s_i \Delta t}}$$

Part (b)

If we neglect noise, the frequencies calculated in part a will be conserved during the dilution step and the next growth phase will be deterministic; the calculation will be akin to that of part a once again. The calculation is also not dependent on completing a full growth phase Δt . It is thus easy to see that

$$f_k(2 \text{ cycles}) = \frac{f_k(0)e^{2s_k \Delta t}}{\sum_{i=1}^K f_i(0)e^{2s_i \Delta t}}$$

$$f_k(n \text{ cycles}) = \frac{f_k(0)e^{ns_k \Delta t}}{\sum_{i=1}^K f_i(0)e^{ns_i \Delta t}}$$

$$f_k(t) = \frac{f_k(0)e^{s_k t}}{\sum_{i=1}^K f_i(0)e^{s_i t}}$$

Part (c)

Shift all $s_i \rightarrow s'_i = s_i + c$:

$$f'_k(t) = \frac{f_k(0)e^{(s_k+c)t}}{\sum_{i=1}^K f_i(0)e^{(s_i+c)t}} = \frac{f_k(0)e^{s_k t} e^{ct}}{e^{ct} \sum_{i=1}^K f_i(0)e^{s_i t}} = f_k(t)$$

Only knowing $f_k(t)$ (tracking strain frequencies over time) is not enough to determine the absolute values of s_k .

Part (d)

Let strain 0 be the wildtype with $s_0 = 0$. Then

$$\begin{aligned} \frac{f_k(t)}{f_0(t)} &= \frac{f_k(0)e^{s_k t}}{f_0(0)e^{s_0 t}} = \frac{f_k(0)}{f_0(0)} e^{s_k t} \implies \frac{N_k(t)}{N_0(t)} = \frac{N_k(0)}{N_0(0)} e^{s_k t} \implies \frac{N_k(t_2)}{N_0(t_2)} \frac{N_0(t_1)}{N_k(t_1)} = e^{s_k(t_2-t_1)} \\ \implies s_k &= \frac{1}{t_2 - t_1} \log \left(\frac{N_k(t_2)N_0(t_1)}{N_0(t_2)N_k(t_1)} \right) \end{aligned}$$

Part (e)

The initial frequency of the wildtype (assume $s_0 = 0$) is f_0 and the initial frequencies of the rest of the strains are $(1 - f_0)/K$ (since K is very large, it is safe to have K in the denominator instead of $K - 1$). Plug these into our formula for f_k :

$$f_k(t) = \frac{\frac{1-f_0}{K} e^{s_k t}}{f_0 + \frac{1-f_0}{K} \sum_{i=1}^K e^{s_i t}}$$

Now we want to find the “average” frequency trajectory for the trajectory of the focal strain k . Since K is very large, by the central limit theorem, $\frac{1}{K} \sum_{i=1}^K e^{s_i t}$ can safely be approximated by the mean $\langle e^{s_i t} \rangle = e^{\frac{1}{2} \sigma^2 t^2}$ (technically we would need to pull out $e^{s_k t}$ from the sum since presumably s_k is fixed/known, but this shouldn't matter much when K is very large):

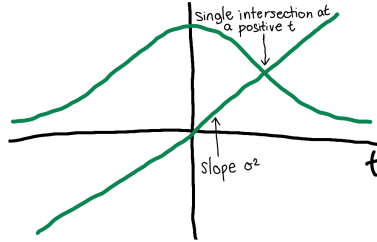
$$\langle f_k(t) \rangle \approx \frac{\frac{1-f_0}{K} e^{s_k t}}{f_0 + (1-f_0) e^{\frac{1}{2} \sigma^2 t^2}}$$

Check if this is monotonic by looking for maxima/minima:

$$\frac{\partial \langle f \rangle}{\partial t} = \frac{1-f_0}{K} \frac{1}{\left(f_0 e^{s_k t} + (1-f_0) e^{\frac{1}{2} \sigma^2 t^2 - s_k t} \right)^2} \left(-s_k f_0 e^{-s_k t} + (1-f_0)(\sigma^2 t - s_k) e^{\frac{1}{2} \sigma^2 t^2 - s_k t} \right) = 0$$

$$\implies -s_k f_0 + (1-f_0)(\sigma^2 t - s_k) e^{\frac{1}{2} \sigma^2 t^2} = 0 \implies \sigma^2 t = s_k \left(\frac{f_0}{1-f_0} e^{-\frac{1}{2} \sigma^2 t^2} + 1 \right)$$

If $s_k > 0$ and $t > 0$, then both sides of the equation are positive and we get the following situation:



So the frequency trajectory has a maximum at the t^* that solves the following equation:

$$t^* = \frac{s_k}{\sigma^2} \left(\frac{f_0}{1-f_0} e^{-\frac{1}{2} \sigma^2 t^{*2}} + 1 \right)$$

Problem 6: chemostat (in theory)

a.) From basic equations:

$$\frac{dn}{dt} = r(c)n - \delta n = [r(c) - \delta]n \quad (1)$$

$$\frac{dc}{dt} = \delta c_{in} - \delta c - \frac{r(c)n}{V} \quad (2)$$

\Rightarrow @ steady state, Eq 1 becomes:

$$0 = [r(c^*) - \delta]n^* \Rightarrow \boxed{r(c^*) = \delta}$$

(i.e., r^* & τ_y are independent of c_{in} !)

b.) @ steady state, Eq 2 becomes:

$$0 = \delta c_{in} - \delta c^* - \frac{r(c^*)n^*}{V} = \delta c_{in} - \delta c^* - \frac{\delta n^*}{V}$$

(since $r(c^*) = \delta$)

$$\Rightarrow n^* = V(c_{in} - c^*) \approx Vc_{in} \text{ (when } c^* \ll c_{in})$$

\Rightarrow steady state pop size is independent of $r(c)$!

\Rightarrow this happens because concentration c^* adjusts until $r(c^*) = \delta$, regardless of $r(c)$.

c.) In "adiabatic limit", Eq 2 becomes:

$$0 \approx \delta Vc_{in} - r(c)n \Rightarrow r(c)n = \delta Vc_{in} = \text{const.}$$

substituting into Eq. 1, we obtain:

$$\boxed{\frac{dn}{dt} = \delta Vc_{in} - \delta n} \quad (\text{only depends on } n \checkmark)$$

\Rightarrow solve w/ integrating factors:

$$\frac{dn}{dt} + \delta n = \delta Vc_{in} \Rightarrow \frac{d}{dt} \left[n e^{\delta t} \right] = \delta Vc_{in} e^{\delta t}$$

$$\Rightarrow n(t)e^{\delta t} - n(0) = Vc_{in}(e^{\delta t} - 1)$$

$$\Rightarrow \boxed{n(t) = n(0)e^{-\delta t} + Vc_{in}(1 - e^{-\delta t})}$$

(relaxes to steady state @ rate δ)

d.) Now we have 2 strains, basic eqs become:

$$(3) \quad \frac{dn_{wt}}{dt} = r(c)n_{wt} - \delta n_{wt}$$

$$(4) \quad \frac{dn_{mut}}{dt} = (1+s)r(c)n_{mut} - \delta n_{mut}$$

$$(5) \quad \frac{dc}{dt} = \delta c_{in} - \delta c - \frac{1}{V} \left[r(c)n_{wt} + (1+s)r(c)n_{mut} \right]$$

adiabatic
approx:

$$0 = \delta Vc_{in} - r(c)n_{wt} - (1+s)r(c)n_{mut}$$

$$\Rightarrow r(c) = \frac{\delta V c_{in}}{n_{wt} + (1+s)n_{mut}}$$

\Downarrow substituting into Eqs 3+4

$$\frac{dn_{wt}}{dt} = \frac{\delta V c_{in} n_{wt}}{n_{wt} + (1+s)n_{mut}} - \delta n_{wt}$$

$$\frac{dn_{mut}}{dt} = \frac{\delta V c_{in} (1+s)n_{mut}}{n_{wt} + (1+s)n_{mut}} - \delta n_{mut}$$

\Downarrow adding together

$$\frac{d(n_{wt} + n_{mut})}{dt} = \delta V c_{in} (1) - \delta [n_{wt} + n_{mut}]$$

$$\Rightarrow \boxed{\frac{dN}{dt} = \delta V c_{in} - \delta N} \quad (\text{same as for single strain above})$$

(e) If $N(0) = n^* = V_{cin}$, then $\frac{dN}{dt} = 0$

$\Rightarrow N(t) = V_{cin}$ @ all times!

Then defining $f \equiv \frac{n_{mut}}{N} = \frac{n_{mut}}{V_{cin}}$ (implies $n_{wt} = \frac{1-f}{V_{cin}}$)

we have:

$$V_{cin} \frac{df}{dt} = \frac{\delta V_{cin} (1+s)f}{(1-f) + (1+s)f} - \delta V_{cin} f$$

$$\rightarrow \frac{1}{\delta} \frac{df}{dt} = \frac{(1+s)f}{1+sf} - f = \frac{sf(1-f)}{1+sf}$$

\Rightarrow @ lowest order in s , $\frac{df}{dt} \approx s\delta f(1-f)$ ($s \ll 1$)

which is the same as serial dilution model w/ " s " = $s\delta$

(if time is measured in generations, $\tau_g = \frac{1}{r^*} = \frac{1}{s\delta}$,
then s is identical to serial dilution model)

f) Probability that cell falls in dilution volume is:

$$\Pr[\text{diluted}] = \frac{V_d}{V} = \frac{\delta V \Delta t}{V} = \delta \Delta t$$

\Rightarrow since cells diluted independently, we have

$$\text{Var}(n_{\text{mut}}^{\text{diluted}}) \approx n_{\text{mut}} \delta \Delta t \quad (\text{same for WT})$$

(similar in form to serial dilution model)

Problem 7: The *E. coli* genome

Part (a)

The genome is 4,629,812 bp long. The relative fractions are all roughly the same:

- A: 24.64%
- T: 24.59%
- C: 25.42%
- G: 25.35%

Part (b)

20-mer occurrence distribution (this gives a general idea; note the log scale):

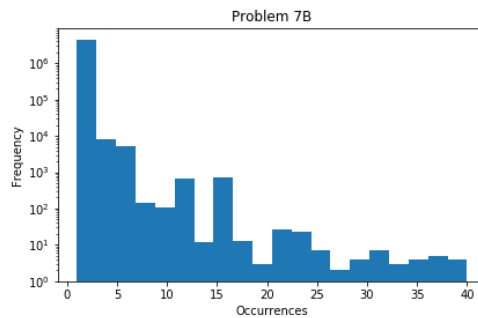


Figure 3: 20-mer occurrence distribution in *E. coli* genome.

99.2% of 20-mers appear only once - this is the fraction of sites in the *E. coli* genome that can be uniquely identified by a 20 bp sequence.

Part (c)

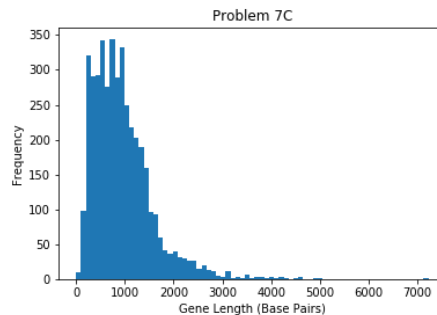


Figure 4: Distribution of gene lengths in *E. coli* genome.

4,217 genes account for 86.76% of the total genome length, and 52.15% of genes are transcribed in the reverse direction.

Part (d)

The number of possible synonymous mutations (in the coding region) are 2,783,982, nonsense mutations are 383,918, and missense mutations are 8,883,073.

Sample code for Problem Set 1

```
1 # Code for Problem 1 of Problem Set 1
2
3 # -*- coding: utf-8 -*-
4 """
5 Created on Tue Jan 21 01:05:31 2020
6
7 @author: Anita Kulkarni
8 """
9
10 import matplotlib.pyplot as plt
11
12 f = open("../data_files/problem_set_data/influenza_HA_dna_sequences.fasta", "r")
13 data = f.readlines()
14 sequences = [] # sequences is a list of tuples (year integer, DNA sequence string)
15
16 for i in range(0, len(data), 2): # lines alternate between label (year, location, et
17     year = int(data[i][-5:-1])
18     seq = data[i+1][: -1]
19     sequences.append((year, seq))
20
21 ref_seq = sequences[0][1] # first sequence (Aichi, 1968) is reference sequence
22 print(len(ref_seq))
23
24 sequences.sort(key=lambda tup: tup[0]) # sort list of tuples by year
25
26 def compare_seq(s0, s1): # number of differences between two sequences
27     diff = max((len(s0), len(s1))) - min((len(s0), len(s1))) # difference in length
28     for i in range(min((len(s0), len(s1)))):
29         if s0[i] != s1[i]:
30             diff = diff + 1
31     return diff
32
33 num_differences = []
34 years = []
35 for i in range(len(sequences)):
36     num_differences.append(compare_seq(ref_seq, sequences[i][1]))
37     years.append(sequences[i][0])
38 plt.plot(years, num_differences, 'o')
39 plt.xlabel("Year")
40 plt.ylabel("Single-Nucleotide Differences from First Sample")
41 plt.title("Problem 1A")
42 plt.tight_layout()
```

```

43 plt.savefig("AP237_PS1_Problem1A.png")
44 plt.show()
45
46 unique_years = list(set(years))
47 unique_years.sort()
48 # number of sequences for each year
49 year_counts = [years.count(unique_years[y]) for y in range(len(unique_years))]
50
51 pointer = 0
52 pairwise_diffs = []
53 for i in range(len(unique_years)):
54     year_sequences = []
55     for j in range(year_counts[i]):
56         year_sequences.append(sequences[pointer+j][1])
57     pointer = pointer + year_counts[i]
58     for s in range(year_counts[i]):
59         if year_counts[i]-s >= 1:
60             for S in range(s+1, year_counts[i]):
61                 pairwise_diffs.append(compare_seq(year_sequences[s], year_sequences[S]))
62 plt.hist(pairwise_diffs, bins=30)
63 plt.xlabel("Number of Pairwise Differences Between Samples in a Given Year")
64 plt.ylabel("Frequency")
65 plt.title("Problem 1B")
66 plt.tight_layout()
67 plt.savefig("AP237_PS1_Problem1B.png")
68 plt.show()

```

```

1  ### Code for Problem 3 on Problem Set 1
2
3  import numpy
4  import pylab
5  from numpy.random import poisson
6  from math import exp
7
8  # Code for running basic simulation loop
9  def run_simulation(N,s,mu,f0,tmax):
10
11      ts = []
12      fs = []
13
14      W = exp(s)
15
16      f = f0
17      for t in xrange(1,tmax+1):
18
19          # growth of mutant          # mutations from wildtype
20          N2 = poisson(N*f*W/(1-f+W*f)) + poisson(N*mu*(1-f)/(1-f+W*f))
21
22          N1 = poisson( N*(1-mu)*(1-f)/(1-f+W*f))
23
24          f = N2*1.0/(N1+N2)
25
26          fs.append(f)
27          ts.append(t)
28
29      return ts,fs
30
31
32  # Part a
33  f0 = 0.5
34  Ns = [1e02,1e03,1e06]
35  ss = [0,1e-03,1e-02]
36  mu = 0
37
38  current_idx = 0
39  for N in Ns:
40      for s in ss:
41          current_idx+=1
42          tmax = 2000
43
44          pylab.figure(figsize=(4,2.5))
45          pylab.xlabel('Time (generations)')

```

```

46     pylab.ylabel('Frequency, $f(t)$')
47     pylab.title('N=%g, s=%g' % (N,s))
48     for i in xrange(0,2):
49         ts,fs = run_simulation(N,s,mu,f0,tmax)
50         pylab.plot(ts,fs,'-')
51
52     pylab.savefig('problem_3_a_%d.pdf' % current_idx,bbox_inches='tight')
53
54 # Part b
55 # Now add mutations
56 f0 = 0
57 N = 1e04
58 mu = 1e-05
59 ss = [-1e-03,1e-02]
60
61 current_idx = 0
62 if True:
63     for s in ss:
64         current_idx+=1
65         tmax = 2000
66
67         pylab.figure(figsize=(4,2))
68         pylab.xlabel('Time (generations)')
69         pylab.ylabel('Frequency, $f(t)$')
70         pylab.title('Mutations, s=%g' % (s))
71         for i in xrange(0,2):
72             ts,fs = run_simulation(N,s,mu,f0,tmax)
73             pylab.plot(ts,fs,'-')
74
75     pylab.savefig('problem_3_b_%d.pdf' % current_idx,bbox_inches='tight')

```



```

1  ### Code for Problem 4 on Problem Set 1
2
3  import numpy
4  import pylab
5  from math import log
6  import sys
7
8
9  # Part A
10
11 # Load data from file
12 file = open("../data_files/LTEE_ancestor_fitness_assays.txt","r")
13 file.readline() # ignore header
14 records = []
15 for line in file:
16     items = line.split(",")
17     population = items[0].strip()
18     t = float(items[1])
19
20     NEO = float(items[2]) # evolved strain counts at time 0
21     NAO = float(items[3]) # ancestor strain counts at time 0
22     NEF = float(items[4]) # evolved strain counts at time 1
23     NAF = float(items[5]) # ancestor strain counts at time 1
24
25     records.append((population,t,NEO,NAO,NEF,NAF))
26
27
28 fitness_data_map = {}
29 dt = numpy.log2(100.0) # these measurements were carried out with 100-fold dilution
30 # Collate by population and timepoint
31 for population,t,NEO,NAO,NEF,NAF in records:
32
33     if population not in fitness_data_map:
34         fitness_data_map[population] = {}
35
36     if t not in fitness_data_map[population]:
37
38         fitness_data_map[population][t] = []
39
40     # Calculate fitness:
41
42     s = 1.0/dt * log(NEF/NAF/(NEO/NAO))
43
44     fitness_data_map[population][t].append(s)
45

```

```

46 # Calculate differences between replicate measurements
47 deltas = []
48 for population in sorted(fitness_data_map):
49     for t in sorted(fitness_data_map[population]):
50         ss = fitness_data_map[population][t]
51
52         # Look at all distinct pairs of replicates
53         for i in xrange(0,len(ss)):
54             for j in xrange(i+1,len(ss)):
55                 delta = numpy.fabs(ss[i]-ss[j])
56                 deltas.append(delta)
57
58 pylab.figure(figsize=(3,2))
59 pylab.xlabel('$\Delta S$ between replicates')
60 pylab.ylabel('Distribution')
61 pylab.hist(deltas)
62 pylab.savefig('problem_4_a.pdf',bbox_inches='tight')
63
64 # Part B and C
65 pylab.figure(figsize=(5,2))
66 pylab.xlabel('Time (generations)')
67 pylab.ylabel('Fitness, S(t)')
68 for population in sorted(fitness_data_map):
69     ts = []
70     savgs = []
71     for t in sorted(fitness_data_map[population]):
72         ss = numpy.array(fitness_data_map[population][t])
73         savg = ss.mean()
74         ts.append(t)
75         savgs.append(savg)
76
77     pylab.plot(ts,savgs,'.-')
78
79
80 theory_ts = numpy.linspace(1,60000)
81 Xc = 4.6e-02
82 v0 = 7.7e-04
83 theory_ss = Xc*numpy.log(1+v0*theory_ts/Xc)
84 pylab.plot(theory_ts,theory_ss,'k-',linewidth=1)
85 pylab.savefig('problem_4_bc.pdf',bbox_inches='tight')
86
87 sys.stdout.write("Predicted gain from 40k to 50k is: %g\n" % ( Xc*numpy.log(1+v0*5e04
88
89 # Part D
90 # Load other data file

```

```

91
92 # Load data from file
93 file = open("../data_files/LTEE_40k_fitness_assays.txt","r")
94 file.readline() # ignore header
95 records = []
96 for line in file:
97     items = line.split(",")
98     population = items[0].strip()
99     t = float(items[1])
100
101     NEO = float(items[2]) # evolved strain counts at time 0
102     NAO = float(items[3]) # ancestor strain counts at time 0
103     NEF = float(items[4]) # evolved strain counts at time 1
104     NAF = float(items[5]) # ancestor strain counts at time 1
105
106     records.append((population,t,NEO,NAO,NEF,NAF))
107
108
109 fitness_data_map = {}
110 dt = 3*numpy.log2(100.0) # these measurements were carried out over 3 days
111 # Collate by population and timepoint
112 for population,t,NEO,NAO,NEF,NAF in records:
113
114     if population not in fitness_data_map:
115         fitness_data_map[population] = {}
116
117     if t not in fitness_data_map[population]:
118
119         fitness_data_map[population][t] = []
120
121     # Calculate fitness:
122
123     s = 1.0/dt * log(NEF/NAF/(NEO/NAO))
124
125     fitness_data_map[population][t].append(s)
126
127 for min_t,max_t in [(4e04,5e04),(5e04,6e04)]:
128     pylab.figure(figsize=(5,2))
129     pylab.ylabel('Fitness gain between \n %dk and %dk' % (min_t/1000,max_t/1000))
130     pylab.xlabel('Population')
131     pylab.ylim([-0.01,0.05])
132     current_idx = 0
133     for population in sorted(fitness_data_map):
134         current_idx+=1
135

```

```

136     ss = numpy.array(fitness_data_map[population][min_t])
137     s0 = ss.mean()
138     ds0 = ss.std()/(len(ss)*1.0)**0.5
139
140     ss = numpy.array(fitness_data_map[population][max_t])
141     sf = ss.mean()
142     dsf = ss.std()/(len(ss)*1.0)**0.5
143
144     s = sf-s0
145     ds = (ds0**2+dsf**2)**0.5
146
147     pylab.plot([current_idx],[s],'k.')
148     pylab.plot([current_idx,current_idx],[s-2*ds,s+2*ds],'k-')
149
150
151     # theory line
152     predicted_ds = Xc*numpy.log(1+v0*max_t/Xc)-Xc*numpy.log(1+v0*min_t/Xc)
153
154     pylab.plot([0,current_idx+1],[predicted_ds,predicted_ds],'k:')
155     pylab.gca().set_xticklabels([])
156     pylab.savefig('problem_4_d_%dk.pdf' % (max_t/1000))

```

```

1 # Code for Problem 7 of Problem Set 1
2
3 # -*- coding: utf-8 -*-
4 """
5 Created on Tue Jan 14 23:17:40 2020
6
7 @author: Anita Kulkarni
8 """
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12
13 f1 = open("../data_files/problem_set_data/ecoli_reference_genome.fasta", "r")
14 f1.readline()
15 genome = f1.readline()
16 f1.close()
17 f2 = open("../data_files/problem_set_data/ecoli_genes.txt", "r")
18 genes = f2.readlines()
19 del(genes[0])
20 for i in range(len(genes)):
21     genes[i] = genes[i].split(", ")
22     del(genes[i][0])
23     genes[i][0] = int(genes[i][0])-1
24     genes[i][1] = int(genes[i][1])
25     if genes[i][2][0] == 'f':
26         genes[i][2] = 1
27     else:
28         genes[i][2] = 0
29 f2.close()
30
31 len_genome = len(genome)
32 print(len_genome)
33 letter_counts = [0,0,0,0] #A, T, C, G
34 for i in range(len_genome):
35     if genome[i] == 'A':
36         letter_counts[0] = letter_counts[0] + 1
37     elif genome[i] == 'T':
38         letter_counts[1] = letter_counts[1] + 1
39     elif genome[i] == 'C':
40         letter_counts[2] = letter_counts[2] + 1
41     elif genome[i] == 'G':
42         letter_counts[3] = letter_counts[3] + 1
43 letter_counts = np.array(letter_counts)
44 letter_frequencies = letter_counts/np.sum(letter_counts)
45 print(letter_frequencies)

```

```

46
47 twenty_mers = {}
48 for i in range(0, len_genome-20):
49     seq = genome[i:i+20]
50     if seq in twenty_mers:
51         twenty_mers[seq] = twenty_mers[seq] + 1
52     else:
53         twenty_mers[seq] = 1
54 twenty_mer_frequencies = list(twenty_mers.values())
55 occ_list = list(set(twenty_mer_frequencies))
56 occ_list.sort()
57 twenty_mer_freq_dist = []
58 for val in occ_list:
59     twenty_mer_freq_dist.append(twenty_mer_frequencies.count(val))
60 print(occ_list)
61 print(twenty_mer_freq_dist)
62 print(twenty_mer_freq_dist[0]/sum(twenty_mer_freq_dist))
63
64 num_genes = len(genes)
65 print(num_genes)
66 gene_lengths = []
67 num_reverse = 0
68 for i in range(len(genes)):
69     gene_lengths.append(genes[i][1]-genes[i][0])
70     if genes[i][2] == 0:
71         num_reverse = num_reverse + 1
72 print(sum(gene_lengths))
73 print(sum(gene_lengths)/len(genome))
74 print(num_reverse/num_genes)
75 gene_lengths_unique = list(set(gene_lengths))
76 gene_lengths_unique.sort()
77 gene_lengths_freq_dist = []
78 for val in gene_lengths_unique:
79     gene_lengths_freq_dist.append(gene_lengths.count(val))
80 plt.hist(gene_lengths, bins=np.arange(0, 99*int(gene_lengths_unique[len(gene_lengths_
81 plt.xlabel("Gene Length (Base Pairs)")
82 plt.ylabel("Frequency")
83 plt.title("Problem 7C")
84 plt.savefig("AP237_PS1_Problem7C.png")
85 plt.show()
86
87 codons = {'TTT':'F', 'TCT':'S', 'TAT':'Y', 'TGT':'C',
88           'TTC':'F', 'TCC':'S', 'TAC':'Y', 'TGC':'C',
89           'TTA':'L', 'TCA':'S', 'TAA':'STOP', 'TGA':'STOP',
90           'TTG':'L', 'TCG':'S', 'TAG':'STOP', 'TGG':'W',

```

```

91         'CTT':'L', 'CCT':'P', 'CAT':'H', 'CGT':'R',
92         'CTC':'L', 'CCC':'P', 'CAC':'H', 'CGC':'R',
93         'CTA':'L', 'CCA':'P', 'CAA':'Q', 'CGA':'R',
94         'CTG':'L', 'CCG':'P', 'CAG':'Q', 'CGG':'R',
95         'ATT':'I', 'ACT':'T', 'AAT':'N', 'AGT':'S',
96         'ATC':'I', 'ACC':'T', 'AAC':'N', 'AGC':'S',
97         'ATA':'I', 'ACA':'T', 'AAA':'K', 'AGA':'R',
98         'ATG':'M', 'ACG':'T', 'AAG':'K', 'AGG':'R',
99         'GTT':'V', 'GCT':'A', 'GAT':'D', 'GGT':'G',
100        'GTC':'V', 'GCC':'A', 'GAC':'D', 'GGC':'G',
101        'GTA':'V', 'GCA':'A', 'GAA':'E', 'GGA':'G',
102        'GTG':'V', 'GCG':'A', 'GAG':'E', 'GGG':'G'}
103 syn_noncoding = 3*(len_genome-sum(gene_lengths))
104 syn = -sum(gene_lengths)
105 non = 0
106 mis = 0
107 nuc = ['A','T','C','G']
108 for i in range(len(genes)):
109     gene = genome[genes[i][0]:genes[i][1]]
110     for j in range(0, len(gene), 3):
111         codon = gene[j:j+3]
112         aa = codons[codon]
113         for n in nuc:
114             new_codon = ''.join((n, gene[j+1], gene[j+2]))
115             new_aa = codons[new_codon]
116             if new_aa == aa:
117                 syn = syn + 1
118             elif new_aa == 'STOP':
119                 non = non + 1
120             else:
121                 mis = mis + 1
122             new_codon = ''.join((gene[j], n, gene[j+2]))
123             new_aa = codons[new_codon]
124             if new_aa == aa:
125                 syn = syn + 1
126             elif new_aa == 'STOP':
127                 non = non + 1
128             else:
129                 mis = mis + 1
130             new_codon = ''.join((gene[j], gene[j+1], n))
131             new_aa = codons[new_codon]
132             if new_aa == aa:
133                 syn = syn + 1
134             elif new_aa == 'STOP':
135                 non = non + 1

```

```
136             else:
137                 mis = mis + 1
138 print(syn)
139 print(non)
140 print(mis)
141 print(syn_noncoding)
```