Solutions for Problem Set 1

Written by: Anita Kulkarni and Benjamin Good
(last updated on February 24, 2021)

Sample code is provided at the end of the document.

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

Part (a)

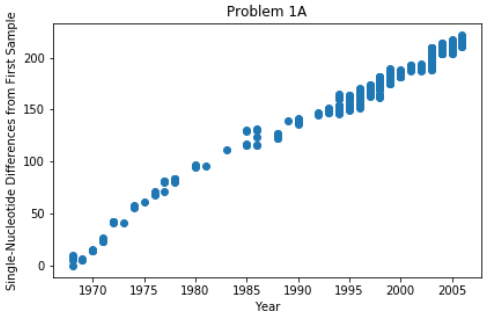


Figure 1: Number of single nucleotide differences between first HA gene sample (A/Aichi/2/1968) and others as a function of sampling year.

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

Part (b)

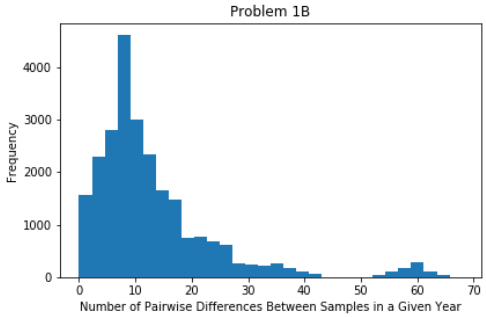


Figure 2: Distribution of number of genetic differences between all pairs of strains from the same year, aggregated across all years.

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:

$$\langle M_T \rangle = E\left[\sum_{t=1}^{T} 2^{T-t} m(t)\right] = \sum_{t=1}^{T} E[2^{T-t} m(t)] = \sum_{t=1}^{T} 2^{T-t} E[m(t)] = \sum_{t=1}^{T} 2^{T-t}\theta(t)$$

$$= \sum_{t=1}^{T} 2^{T-t}\mu N_0 2^t = \mu N_0 2^T \sum_{t=1}^{T} 1 = T \cdot N_0\mu \cdot 2^T$$

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 $\theta(t)$:

$$\mathrm{Var}(M_T) = \mathrm{Var}\left(\sum_{t=1}^{T} 2^{T-t} m(t)\right) = \sum_{t=1}^{T} \mathrm{Var}(2^{T-t} m(t)) = \sum_{t=1}^{T} 2^{2T-2t}\mathrm{Var}(m(t)) = \sum_{t=1}^{T} 2^{2T-2t}\theta(t)$$

$$= \sum_{t=1}^{T} 2^{2T-2t}\mu N_0 2^t = \mu N_0 2^{2T} \sum_{t=1}^{T} 2^{-t} = \mu N_0 4^T (1 - 2^{-T}) = \mu N_0 2^T (2^T - 1) \approx N_0\mu \cdot (2^T)^2$$

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 distingiush 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 $\mathrm{Var}(\overline{M}_T)$:

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

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 $\mathrm{Var}(M_T)$ 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

$$nN_0\mu 2^T \geq 1 \tag{1}$$

(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

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

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

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

The coefficient of variation is therefore given by

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

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

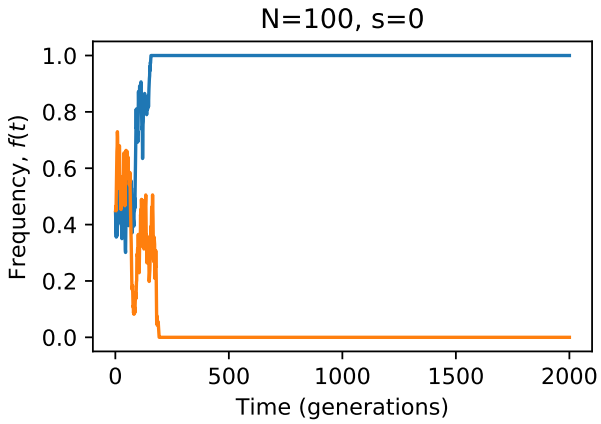
$$F = 1 + \frac{1 - 2^{-T}}{T} \approx 1 + \frac{1}{T}$$

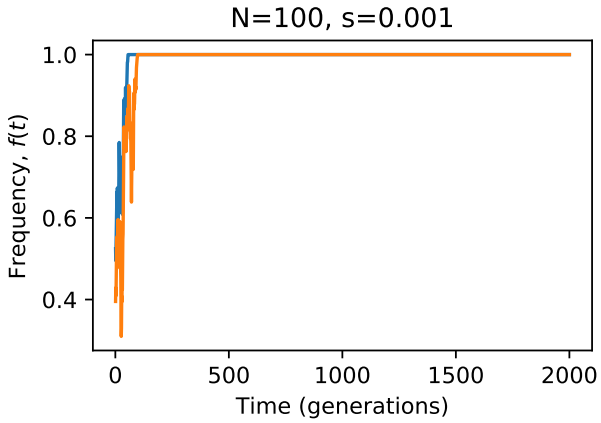
which approaches the Poisson limit of $F \approx 1$ when $T \gg 1$. For example, for a dilution factor of $2^T = 100$, we have $T \approx 6.7$ and $F \approx 1.14$ – a relatively small deviation from the Poisson approximation we used in class.

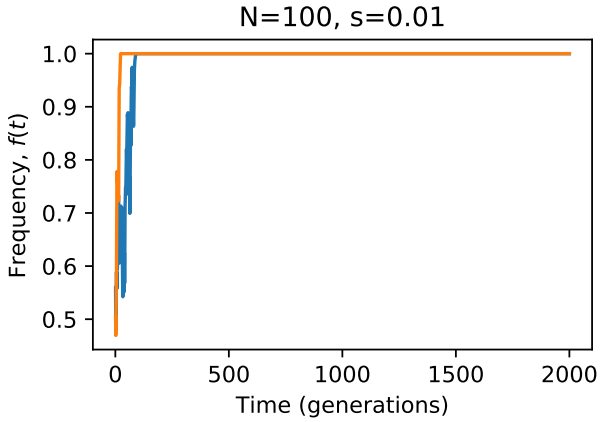
Problem 3:

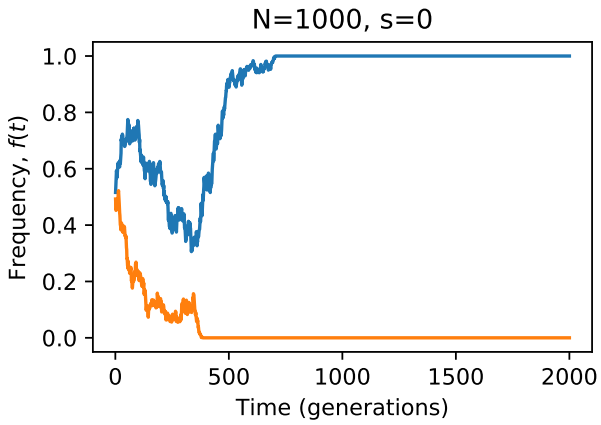
Part (a)

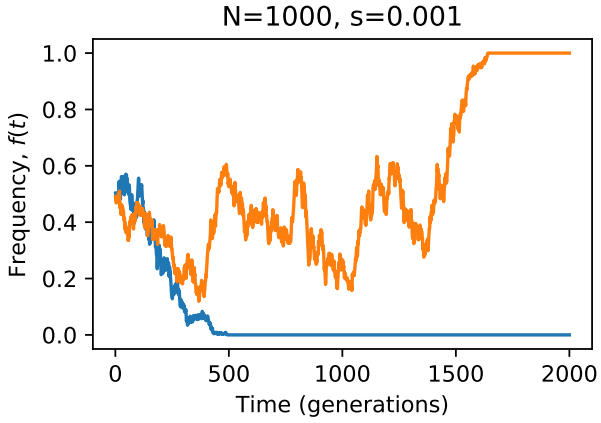
Two replicate simulations are shown for each of the parameter combinations below:

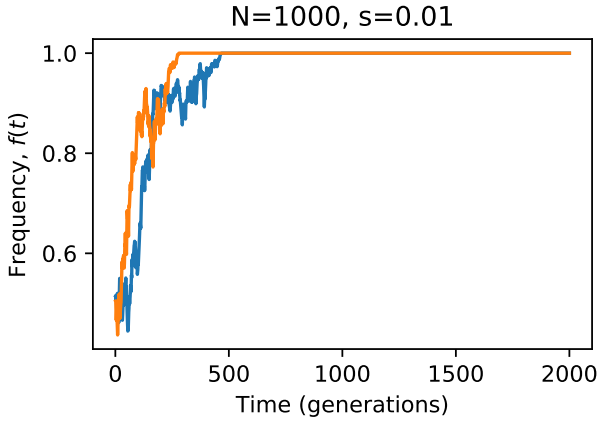


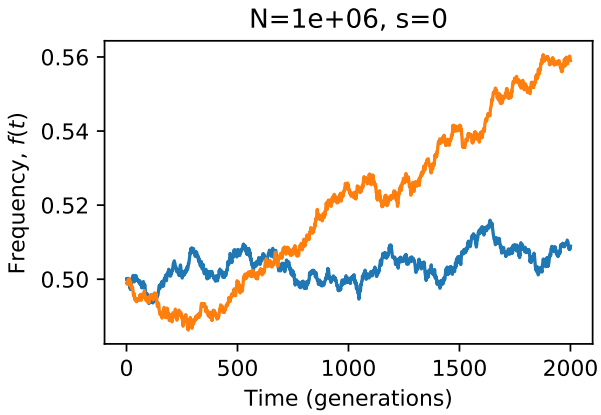


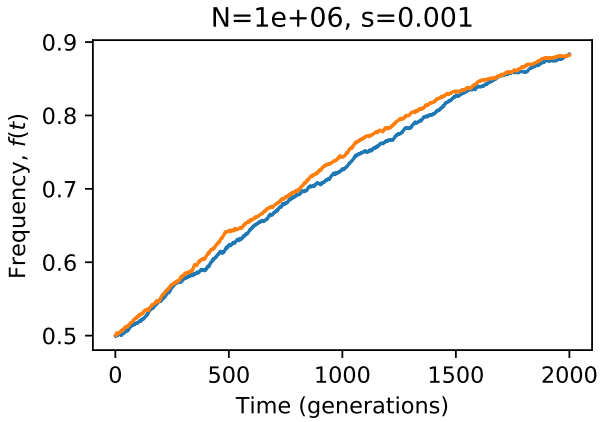


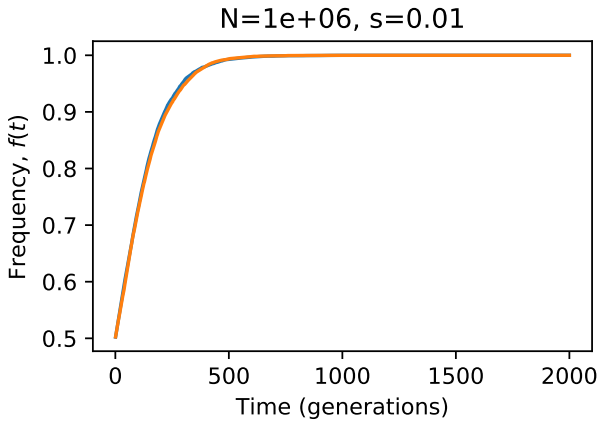

N=1000, s=0
Frequency, f(t)
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
Time (generations)


N=1000, s=0.001
Frequency, f(t)
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
Time (generations)


N=1000, s=0.01
Frequency, f(t)
1.0
0.8
0.6
0
500
1000
1500
2000
Time (generations)

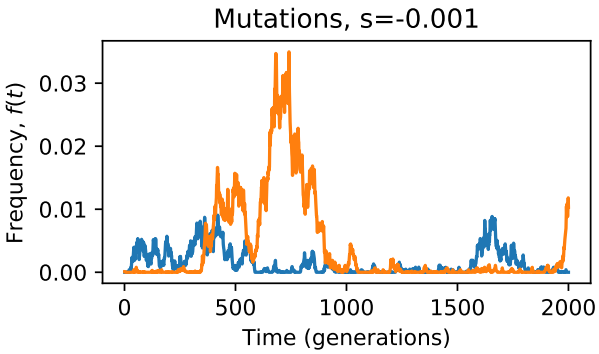


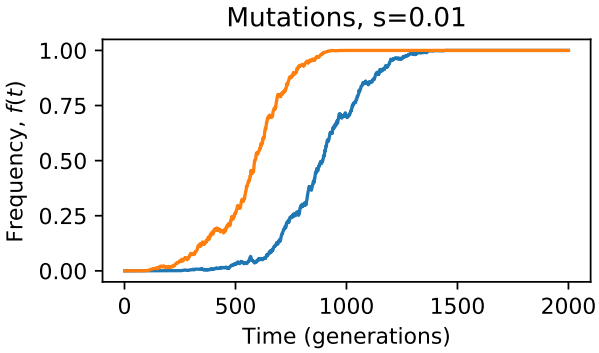




Part (b)

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 $\mu = 10^{-5}$:

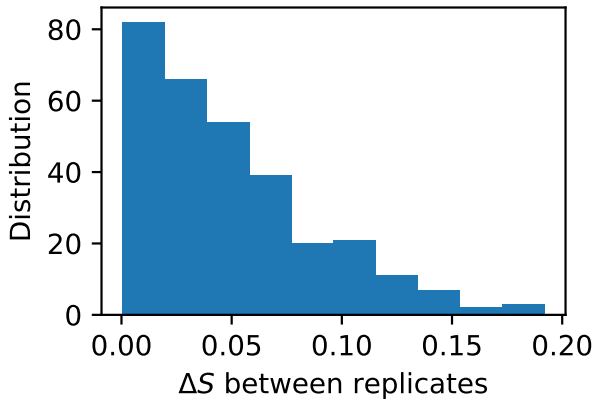

Mutations, s=-0.001
Frequency, f(t)
0.03
0.02
0.01
0.00
0
500
1000
1500
2000
Time (generations)


Mutations, s=0.01
Frequency, f(t)
1.00
0.75
0.50
0.25
0.00
0
500
1000
1500
2000
Time (generations)

Problem 4:

Part (a)

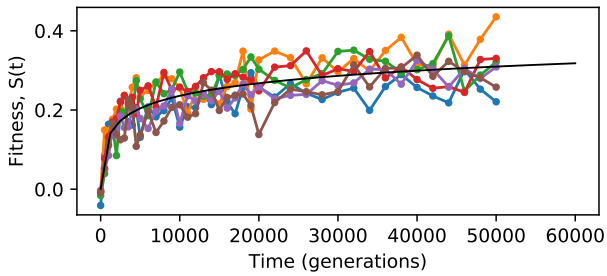
The observed distribution of differences between replicate fitness measurements is shown below:



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:

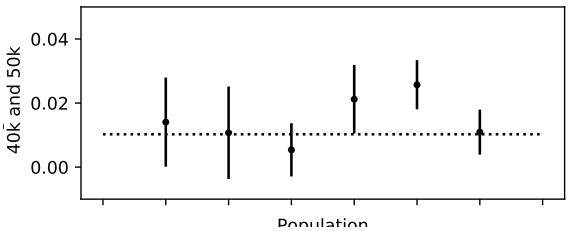


Part (c)

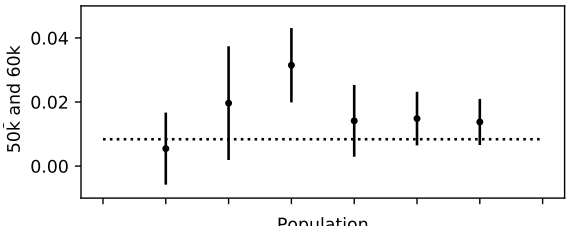
The predicted trajectory, $X(t) = X_c \log(1 + v_0 t/X_c)$ is shown in the solid black line in the plot in part (b) for $X_c \approx 4.6 \times 10^{-2}$ and $v_0 = 7.7 \times 10^{-4}$. It seems to be roughly consistent with the data. The predicted fitness gain between generation $40,000$ and $50,000$ 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 popluations 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 $s_i \to s_i' = s_i + c$:

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

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

$$\frac{f_k(t)}{f_0(t)} = \frac{f_k(0)e^{s_k t}}{f_0(0)e^{s_0 t}} = \frac{f_k(0)}{f_0(0)}e^{s_k t} \implies \frac{N_k(t)}{N_0(t)} = \frac{N_k(0)}{N_0(0)}e^{s_k t} \implies \frac{N_k(t_2)}{N_0(t_2)}\frac{N_0(t_1)}{N_k(t_1)} = e^{s_k(t_2-t_1)}$$

$$\implies s_k = \frac{1}{t_2 - t_1}\log\left(\frac{N_k(t_2)N_0(t_1)}{N_0(t_2)N_k(t_1)}\right)$$

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):

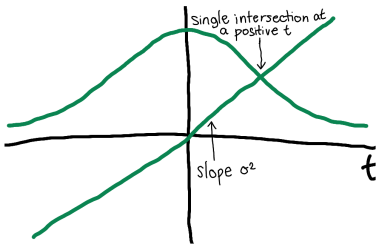
$$\langle f_k(t) \rangle \approx \frac{\frac{1-f_0}{K} e^{s_k t}}{f_0 + (1 - f_0) e^{\frac{1}{2}\sigma^2 t^2}}$$

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:

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

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

$\Rightarrow$ @ steady state, Eq 1 becomes:

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

(i.e., r^* & τ_g are <u>independent</u> 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 $r(c^*) = \delta$)

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

$\Rightarrow$ steady state pop size is <u>independent</u> 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:

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

substituting into Eq. 1, we obtain:

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

$\Rightarrow$ solve w/ integrating factors:

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

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

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

(relaxes to steady state @ rate δ)

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

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

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

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

adiabatic approx:

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

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

$\Downarrow$ 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

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

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

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

$\Rightarrow$ $N(t) = VC_{in}$ @ all times!

Then defining $f \equiv \frac{n_{mut}}{N} = \frac{n_{mut}}{VC_{in}}$ (implies $n_{wt} = \frac{1-f}{VC_{in}}$)
we have:

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

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

$\Rightarrow$ @ 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, $\tau_g = 1/r^* = 1/\delta$,
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

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

(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):

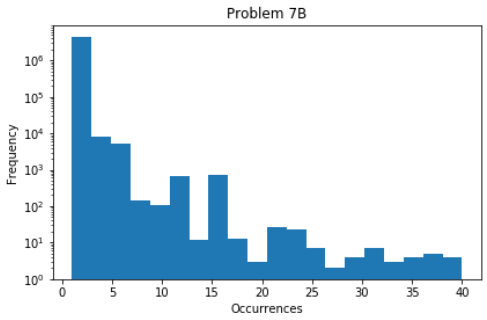


Figure 3: 20-mer occurrence distribution in *E. coli* genome.

99.2% of unique 20-mers appear only once (as a fraction of all unique 20-mers), and 97.4% of 20-mers in the E. coli genome appear exactly once. This shows that most (specifically 97.4%) sites in the E. coli genome can be uniquely identified by a 20 bp sequence.

Part (c)

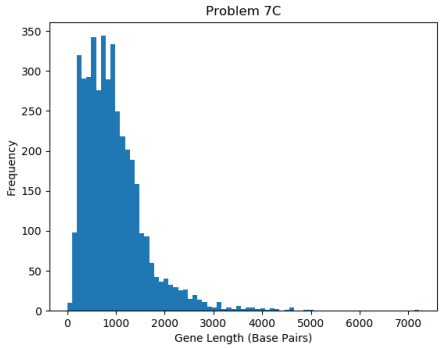


Figure 4: Distribution of gene lengths in *E. coli* genome.

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) is 3,059,233, nonsense mutations is 404,289, and missense mutations is 8,587,451. Answers may vary somewhat depending on certain ambiguities in the definitions of missense and nonsense mutations, differing levels of knowledge within the class about how DNA is read, etc.

Sample code for Problem Set 1

```
# Code for Problem 1 of Problem Set 1

# -*- coding: utf-8 -*-
"""
Created on Tue Jan 21 01:05:31 2020

@author: Anita Kulkarni
"""

import matplotlib.pyplot as plt

f = open("../data_files/problem_set_data/influenza_HA_dna_sequences.fasta", "r")
data = f.readlines()
sequences = []  # sequences is a list of tuples (year integer, DNA sequence string)

for i in range(0, len(data), 2):  # lines alternate between label (year, location, et
    year = int(data[i][-5:-1])
    seq = data[i+1][:-1]
    sequences.append((year, seq))

ref_seq = sequences[0][1]  # first sequence (Aichi, 1968) is reference sequence
print(len(ref_seq))

sequences.sort(key=lambda tup: tup[0])  # sort list of tuples by year

def compare_seq(s0, s1):  # number of differences between two sequences
    diff = max((len(s0),len(s1))) - min((len(s0),len(s1)))  # difference in length
    for i in range(min((len(s0), len(s1)))):
        if s0[i] != s1[i]:
            diff = diff + 1
    return diff

num_differences = []
years = []
for i in range(len(sequences)):
    num_differences.append(compare_seq(ref_seq, sequences[i][1]))
    years.append(sequences[i][0])
plt.plot(years, num_differences, 'o')
plt.xlabel("Year")
plt.ylabel("Single-Nucleotide Differences from First Sample")
plt.title("Problem 1A")
plt.tight_layout()
```

```
plt.savefig("AP237_PS1_Problem1A.png")
plt.show()

unique_years = list(set(years))
unique_years.sort()
# number of sequences for each year
year_counts = [years.count(unique_years[y]) for y in range(len(unique_years))]

pointer = 0
pairwise_diffs = []
for i in range(len(unique_years)):
    year_sequences = []
    for j in range(year_counts[i]):
        year_sequences.append(sequences[pointer+j][1])
    pointer = pointer + year_counts[i]
    for s in range(year_counts[i]):
        if year_counts[i]-s >= 1:
            for S in range(s+1, year_counts[i]):
                pairwise_diffs.append(compare_seq(year_sequences[s], year_sequences[S
plt.hist(pairwise_diffs, bins=30)
plt.xlabel("Number of Pairwise Differences Between Samples in a Given Year")
plt.ylabel("Frequency")
plt.title("Problem 1B")
plt.tight_layout()
plt.savefig("AP237_PS1_Problem1B.png")
plt.show()
```

```
### Code for Problem 3 on Problem Set 1

import numpy
import pylab
from numpy.random import poisson
from math import exp

# Code for running basic simulation loop
def run_simulation(N,s,mu,f0,tmax):

    ts = []
    fs = []

    W = exp(s)

    f = f0
    for t in xrange(1,tmax+1):

                # growth of mutant          # mutations from wildtype
        N2 = poisson(N*f*W/(1-f+W*f)) + poisson(N*mu*(1-f)/(1-f+W*f))

        N1 = poisson( N*(1-mu)*(1-f)/(1-f+W*f))

        f = N2*1.0/(N1+N2)

        fs.append(f)
        ts.append(t)

    return ts,fs


# Part a
f0 = 0.5
Ns = [1e02,1e03,1e06]
ss = [0,1e-03,1e-02]
mu = 0

current_idx = 0
for N in Ns:
    for s in ss:
        current_idx+=1
        tmax = 2000

        pylab.figure(figsize=(4,2.5))
        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')
```

```
### Code for Problem 4 on Problem Set 1

import numpy
import pylab
from math import log
import sys


# Part A

# Load data from file
file = open("../data_files/LTEE_ancestor_fitness_assays.txt","r")
file.readline() # ignore header
records = []
for line in file:
    items = line.split(",")
    population = items[0].strip()
    t = float(items[1])

    NE0 = float(items[2]) # evolved strain counts at time 0
    NA0 = float(items[3]) # ancestor strain counts at time 0
    NEF = float(items[4]) # evolved strain counts at time 1
    NAF = float(items[5]) # ancestor strain counts at time 1

    records.append((population,t,NE0,NA0,NEF,NAF))


fitness_data_map = {}
dt = numpy.log2(100.0) # these measurements were carried out with 100-fold dilution
# Collate by population and timepoint
for population,t,NE0,NA0,NEF,NAF in records:

    if population not in fitness_data_map:
        fitness_data_map[population] = {}

    if t not in fitness_data_map[population]:

        fitness_data_map[population][t] = []

    # Calculate fitness:

    s = 1.0/dt * log(NEF/NAF/(NE0/NA0))

    fitness_data_map[population][t].append(s)

```

```
# Calculate differences between replicate measurements
deltas = []
for population in sorted(fitness_data_map):
    for t in sorted(fitness_data_map[population]):
        ss = fitness_data_map[population][t]

        # Look at all distinct pairs of replicates
        for i in xrange(0,len(ss)):
            for j in xrange(i+1,len(ss)):
                delta = numpy.fabs(ss[i]-ss[j])
                deltas.append(delta)

pylab.figure(figsize=(3,2))
pylab.xlabel('$\Delta S$ between replicates')
pylab.ylabel('Distribution')
pylab.hist(deltas)
pylab.savefig('problem_4_a.pdf',bbox_inches='tight')

# Part B and C
pylab.figure(figsize=(5,2))
pylab.xlabel('Time (generations)')
pylab.ylabel('Fitness, S(t)')
for population in sorted(fitness_data_map):
    ts = []
    savgs = []
    for t in sorted(fitness_data_map[population]):
        ss = numpy.array(fitness_data_map[population][t])
        savg = ss.mean()
        ts.append(t)
        savgs.append(savg)


    pylab.plot(ts,savgs,'.-')

theory_ts = numpy.linspace(1,60000)
Xc = 4.6e-02
v0 = 7.7e-04
theory_ss = Xc*numpy.log(1+v0*theory_ts/Xc)
pylab.plot(theory_ts,theory_ss,'k-',linewidth=1)
pylab.savefig('problem_4_bc.pdf',bbox_inches='tight')

sys.stdout.write("Predicted gain from 40k to 50k is: %g\n" % ( Xc*numpy.log(1+v0*5e04

# Part D
# 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     NE0 = float(items[2]) # evolved strain counts at time 0
102     NA0 = 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,NE0,NA0,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,NE0,NA0,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/(NE0/NA0))
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))
```

```
# Code for Problem 7 of Problem Set 1

# -*- coding: utf-8 -*-
"""
Created on Tue Jan 14 23:17:40 2020

@author: Anita Kulkarni
"""

import numpy as np
import matplotlib.pyplot as plt

f1 = open("../data_files/ecoli_reference_genome.fasta", "r")
f1.readline()
genome = f1.readline()
f1.close()
f2 = open("../data_files/ecoli_genes.txt", "r")
genes = f2.readlines()
del(genes[0])
for i in range(len(genes)):
    genes[i] = genes[i].split(", ")
    del(genes[i][0])
    genes[i][0] = int(genes[i][0])-1
    genes[i][1] = int(genes[i][1])
    if genes[i][2][0] == 'f':
        genes[i][2] = 1
    else:
        genes[i][2] = 0
f2.close()

len_genome = len(genome)
print(len_genome)
letter_counts = [0,0,0,0] #A, T, C, G
for i in range(len_genome):
    if genome[i] == 'A':
        letter_counts[0] = letter_counts[0] + 1
    elif genome[i] == 'T':
        letter_counts[1] = letter_counts[1] + 1
    elif genome[i] == 'C':
        letter_counts[2] = letter_counts[2] + 1
    elif genome[i] == 'G':
        letter_counts[3] = letter_counts[3] + 1
letter_counts = np.array(letter_counts)
letter_frequencies = letter_counts/np.sum(letter_counts)
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
104 syn = -sum(gene_lengths) # don't double-count replacements of same nucleotide
105 non = 0
106 mis = 0
107 nuc = ['A','T','C','G']
108 for i in range(len(genes)):
109
110     forward_gene = genome[genes[i][0]:genes[i][1]]
111     reversed_gene = genome[genes[i][1]:genes[i][0]:-1]
112
113     #print len(forward_gene)
114     #print len(reversed_gene)
115     if genes[i][2] == 1: # non-reversed gene
116         gene = forward_gene
117     else: # reversed gene
118         gene = reversed_gene
119     for j in range(0, len(gene), 3):
120         codon = gene[j:j+3]
121         aa = codons[codon]
122         for n in nuc:
123             new_codon = ''.join((n, gene[j+1], gene[j+2]))
124             new_aa = codons[new_codon]
125             if new_aa == aa:
126                 syn = syn + 1
127             elif new_aa == 'STOP':
128                 non = non + 1
129             else:
130                 mis = mis + 1
131             new_codon = ''.join((gene[j], n, gene[j+2]))
132             new_aa = codons[new_codon]
133             if new_aa == aa:
134                 syn = syn + 1
135             elif new_aa == 'STOP':
```

```
136                 non = non + 1
137             else:
138                 mis = mis + 1
139             new_codon = ''.join((gene[j], gene[j+1], n))
140             new_aa = codons[new_codon]
141             if new_aa == aa:
142                 syn = syn + 1
143             elif new_aa == 'STOP':
144                 non = non + 1
145             else:
146                 mis = mis + 1
147 print(syn)
148 print(non)
149 print(mis)
```