**CS423 Lab 4: Sequence Evolution**

**Authors: Caleb Piekstra, Connor Haas**

**Write-up (55 points):**

Create a Word document or pdf file to complete your writeup of this lab and show that you tested your code. Be sure to include your name(s) and lab number in your file.

1. (5 pts) Run your mutationExp program to generate the output file. Using this output file, create a graph with the following information (use MS excel for this part):
   1. One line graph of the observed mutations over time (Hamming distances after each mutation). The x-axis should be number of actual mutations (1 thru 1000) and the y-axis should be the # of observed mutations (H).

***Figure 1****: JC correction and H values over 1000 mutations of a 1000-length DNA sequence*

* 1. On the same graph, put a line graph of the estimated mutations over time (Jukes Cantor after each mutation). The x-axis is the number of actual mutations (1 thru 1000) and the y-axis should be the # of estimated mutations (EST).

**See above.**

* 1. In MS excel, you should be able to use your output file directly. The two lines graphs should be labeled and drawn with a different line type (by color or style).) **Copy and paste** this graph in your write-up labeled as *Figure 1: JK correction and H values over 1000 mutations of a 1000-length DNA sequence*.

**See above.**

1. (1 pt) After 1000 mutations, what is the final value of K? What is the final value of H?

**H: 542**

**K: 961.9013**

1. (2 pts) How well do you think the Jukes Cantor correction works given your results?

**The Jukes Cantor correction appears to work extremely well based on our results. We muted the gene 1000 times randomly, and the final JK correction value was about 962 which is quite close to 1000 and much closer than the hamming distance of 542.**

1. (5 pts) Now, run your mutationExp program for a simulation of 2000 mutations on a sequence of length 1000. Before running the program, you may want to name your output file something else, so you do not lose the output file you created for question 1.
   1. Copy a paste a similar graph to Figure 1 with this data. Label the figure *Figure 2: JK correction and H values over 2000 mutations of a 1000-length DNA sequence*.

***Figure 2****: JC correction and H values over 2000 mutations of a 1000-length DNA sequence*.

1. (1 pt) After 2000 mutations over a 1000-length sequence, what is the final value of K? What is the final value of H?

**H: 675**

**K: 1726.9388**

1. (2 pts) Describe differences in results for 2000 runs versus 1000 runs for original sequences of length 1000?

**The H value increased only slightly, about 100, with the 2000 runs compared to the 1000 runs of the original test. In contrast, the K value increased drastically, almost doubling and is still much closer to the 2000 than the H value. Still, the previous K value was much closer to the 1000 than this value is to the 2000 we know to be true.**

1. (2 pts) Now run the experiment with a randomly generated DNA sequence of 10,000 nucleotides for 2000 runs of mutations. How do K and H differ between sequence with length 1000 nucleotides and 2000 runs? Why do you think H is getting closer to the actual number of mutations in this scenario?

**H: 1720**

**K: 1953.745**

**Compared to the sequence with 1000 nucleotides and 2000 mutation runs, the H value is much more accurate and the K is only slightly more accurate. I think the H is getting closer to the actual number of mutations in this scenario because with more and more nucleotides, the chance of the same nucleotide being mutated back to its original base is much lower.**

1. (5 pts) Come up with a new set of values (length of nucleotides and # of mutations). Before running the experiment, make a prediction as to what you expect for values of H and K. Describe your experiment in your write-up, your prediction, and if you prediction was correct. Include the figure of your results or multiple figures of your results if you ran several versions for your experiment.

**The experiment will be run with 100,000 nucleotides and only 1000 mutations.**

**I predict that with a massive increase in the number of nucleotides and a decrease in the number of mutations, the H value will be very accurate, probably around 980. The K value should also be quite close, likely around 995.**

**Actual Mutations: 1000**

**Predicted H: 980**

**Predicted K: 995**

**Calculated H: 994**

**Calculated K: 1000.646**

***Figure 3:*** *JC correction and H values over 1000 mutations of a 100,000-length DNA sequence*.

1. (5 pts) Run your code for calculating the codon position frequency difference percentages. Using your results, create a bar chart (in MS excel) that has a bar for the first position, the second position, and the third position. Include this chart in your write-up. Label the figure *Figure 3: codon position frequency difference percentages*.

***Figure 4:*** *codon position frequency difference percentages*.

1. (2 pts) What is the most frequently substituted position in the codons? Why might this be the case in terms of biology?

**Position 3 is the most frequently substituted position in the codons. In terms of biology, this might be the case due to the fact that you can have changes in the third position and often times the resulting codon still works for the same amino acid. This can be observed in the RNA to Codon table further down in this write-up.**

1. Answer these questions (not related to the code):
   1. (2 pts) At the end of this document is the amino acid translation table. Refer to this table to answer this question. **Synonymous mutations** are those mutations that change a nucleotide in a coding region, but do not change the amino acid sequence of the translated protein. A **non-synonymous mutation** is one that changes the amino acid in the translated protein. Assume the RNA sequence “CAU” is translated to an amino acid. Provide a codon where one of the bases in “CAU” is changed and it results in a synonymous mutation.

**Before base change**

**CAU – His**

**After base change (same amino acid!)**

**CAC - His**

* 1. (2 pts) Assume the RNA sequence “CAU” is translated to an amino acid. Provide a codon where one of the bases in “CAU” is changed and it results in a non-synonymous mutation.

**Before base change**

**CAU – His**

**After base change (different amino acid!)**

**CAA - Gln**

* 1. (3 pts) In general, do you expect the substitution rate of synonymous or non-synonymous mutations to be higher? Why?

**In general, I expect the substitution rate of synonymous mutations to be higher because position 3 is more likely to change than position 1 or 2 and if just position 3 changes, there is a fairly high likelihood that the resulting sequence can be translated into the same amino acid as before, a synonymous mutation.**

* 1. (3 pts) Even though synonymous mutations do not affect the amino acid in the resulting protein, it can impact the organism. Name one impact that a change in DNA could have even though the translated amino acid is un-altered. (Hint: DNA is two-stranded.)

**A change in DNA could cause a wobble base pair which does not follow Watson-Crick base pair rules.**

(<https://en.wikipedia.org/wiki/Wobble_base_pair>)

* 1. (3 pts) Assume the *wizard* genome has just been sequenced and it is found to have the same 20 amino acids as humans, but the DNA of wizards has just three different nucleotides. How many unique codons (sequence of 3 nucleotides) can be specified for wizards?

**The number of unique codons that can be specified for wizards would be 7 permute 3 which is 210.**

* 1. (2 pts) Would you expect wizards or humans to have more synonymous mutations?

**I would expect wizards to have many more synonymous mutations due to the fact that they still have the same 20 amino acids, so they would have more sequences for the same amino acid.**

* 1. (10 pts) Find information online, in your textbook, or in the library about the PAM and BLOSUM amino acid substitution matrices. These were created by observing mutations in amino acids in proteins.
     1. What does a positive score in PAM signify? A negative score in PAM?

**A positive PAM score signifies that the alignment is more likely due to point accepted mutations. A negative PAM score signifies that the alignment is more likely due to chance.**

* + 1. What does a positive score in BLOSUM signify? A negative score in BLOSUM?

**A positive score in BLOSUM signifies that a substitution is more likely. A negative score in BLOSUM signifies that a substitution is less likely.**

* + 1. Both matrices were derived from protein sequences. Which matrix used more protein examples to create the model?

**The BLOSUM matrix used more protein examples to create the model. The PAM just looks at two at a time, whereas the BLOSUM compares many protein sequences.**

* + 1. Which matrix do you think is more accurate in estimating amino acid substitutions? Why?

**The BLOSUM matrix is likely more accurate due to the fact that it looks at the “bigger picture” by comparing many sequences for each entry.**

* + 1. What references did you use to learn about the matrices?

<https://en.wikipedia.org/wiki/Point_accepted_mutation>

<https://en.wikipedia.org/wiki/BLOSUM>

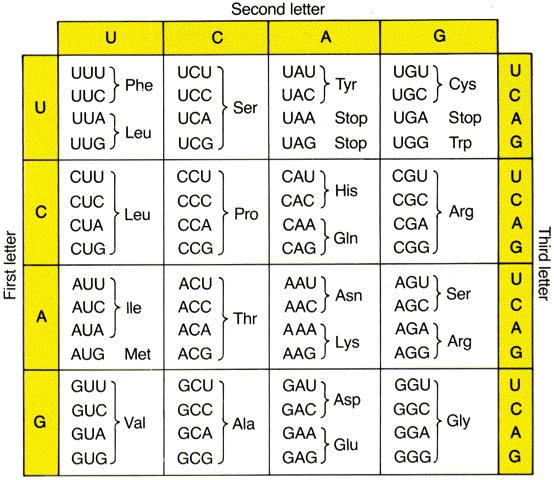


Figure 1: RNA to Codon Table

**Appendix A: Authorship (please include statement in your write-up)**

The code and write-up submitted for this lab were authored by the named person(s) on this lab report. All external sources to BIO/CS423 are cited properly.

**Appendix B: Code (30 points, based on correctness and style):**

Copy and paste the code you wrote for lab 4 here (use Courier New 9-pt font). Also, upload the code files and results files as part of the zip file to Moodle.

#!/usr/bin/python

##### mutationExp.py

##########################################################

# CS423 Lab 4, fall 2015

#

# Caleb Piekstra

# Connor Haas

##########################################################

import random

import math

## generateRandomChar

#

# Returns G C T or A randomly

#

def generateRandomChar():

randomNum = random.random()

if (randomNum < 0.25):

return "G"

elif (randomNum < 0.50):

return "C"

elif (randomNum < 0.75):

return "A"

return "T"

## generateRandomSequence

#

# Given a length for the sequence to generate, 'N',

# returns a sequence of that length containing

# random bases

#

def generateRandomSequence(N):

count = 0

sequence = ""

while (count < N):

sequence = sequence + generateRandomChar()

count = count + 1

return sequence

## mutate

#

# Given a string sequence, picks a random

# nucleotide in the sequence and changes it

# to one of the three other bases, randomly

#

def mutate(seq):

seq = list(seq)

nucForC = "ATG"

nucForG = "ATC"

nucForT = "ACG"

nucForA = "TCG"

randomChosen = int(random.random()\*len(seq))

mutatedChar = seq[randomChosen]

mutationDecide = random.randint(0,2)

if(mutatedChar == "C"):

seq[randomChosen] = nucForC[mutationDecide]

if(mutatedChar == "T"):

seq[randomChosen] = nucForT[mutationDecide]

if(mutatedChar == "G"):

seq[randomChosen] = nucForG[mutationDecide]

if(mutatedChar == "A"):

seq[randomChosen] = nucForA[mutationDecide]

return "".join(seq) # stub (remove when you complete the code)

## hammingDistance

#

# Given two strings, calculates and returns the

# hamming distance between them. (Count of differences)

#

def hammingDistance(s1, s2):

countBad = 0

if(len(s1) != len(s2)):

return -1

for idx, nuc in enumerate(s1):

if nuc != s2[idx]:

countBad += 1

return countBad

## jukesCantor

#

# Given a hamming distance and the length of a sequence,

# calculates and returns the jukes-cantor correction

#

def jukesCantor(H, L):

return (L) \* (-3.0/4.0) \* math.log(1 - (4.0/3.0)\*(H/L))

## experiment

#

# Given a sequence, number of mutations to perform on

# that sequence, and an output file:

# randomly mutates a single nucleotide in the sequence

# as many times as is specified by expLength

# After each mutation, the hamming distance and jukes-

# cantor correction are printed out to the outputFile

#

def experiment(seq, expLength, outputFile):

with open(outputFile, 'w') as out: #######this line is awesome for file IO

mutatedSeq = seq

for i in range(0, expLength):

mutatedSeq = mutate(mutatedSeq)

Ham = hammingDistance(mutatedSeq, seq)

Cheese = jukesCantor(Ham, len(mutatedSeq))

out.write("%d\t%0.4f\r\n" % (Ham,Cheese))

return

########################################################

### End of functions ###################################

########################################################

# Run mutation experiment

s = generateRandomSequence(1000)

experiment(s, 1000, "mutationResultsTest.txt")

#!/usr/bin/python

####### positionBias.py

##########################################################

# CS423 Lab 4, fall 2015

#

# Authors: Caleb Piekstra, Connor Haas

##########################################################

########################################################

# convertFileToSequence -- converts FASTA file to string

# from lab 1

########################################################

def convertFileToSequence(filename):

# read in file

file = open(filename)

# read in first line

header = file.readline()

if (header[0] == '>'):

print("in FASTA format")

else:

print("invalid format")

return

# read in rest of file

sequence = file.read()

# close file

file.close()

# remove all newline characters

sequence = sequence.replace("\n", "")

sequence = sequence.replace("\r", "") # needed to get rid of some newline characters

return sequence

########################################################

# convertFileToSequence -- converts FASTA file to string

# from lab 1

########################################################

def codonPosition(inputFile1, inputFile2):

# convert the contents of each file to a string

sequence1 = convertFileToSequence(inputFile1)

sequence2 = convertFileToSequence(inputFile2)

# determine the total number of codons

numCodons1 = int(len(sequence1) / 3)

numCodons2 = int(len(sequence2) / 3)

# total number of codons

totalNumCodons = numCodons1 if numCodons1 < numCodons2 else numCodons2

# get the shortest sequence

shortestSeq = sequence1 if len(sequence1) < len(sequence2) else sequence2

# Make these floats initially so that when we

# calculate a percentage they are in float form

totalFirstDiffs = 0.0

totalSecondDiffs = 0.0

totalThirdDiffs = 0.0

# Loop through and compare the first, second, and third

# positions in the codons of the two sequcnes, counting

# the number of differences for each position

for i in range (0, len(shortestSeq), 3):

# abort early if the loop reaches a codon that isn't len 3

if (len(shortestSeq[i:i+3]) < 3):

break

# if the first position in the two codons is different,

# increment the total number of differences in position 1

if sequence1[i] != sequence2[i]:

totalFirstDiffs += 1

# if the second position in the two codons is different,

# increment the total number of differences in position 2

if sequence1[i+1] != sequence2[i+1]:

totalSecondDiffs += 1

# if the third position in the two codons is different,

# increment the total number of differences in position 3

if sequence1[i+2] != sequence2[i+2]:

totalThirdDiffs += 1

# calculate the percent frequency of differences for each position

percentFirstDiffFrequency = (totalFirstDiffs / totalNumCodons) \* 100

percentSecondDiffFrequency = (totalSecondDiffs / totalNumCodons) \* 100

percentThirdDiffFrequency = (totalThirdDiffs / totalNumCodons) \* 100

# print out the results

print ("Position 1 Different: %0.2f%%" % percentFirstDiffFrequency)

print ("Position 2 Different: %0.2f%%" % percentSecondDiffFrequency)

print ("Position 3 Different: %0.2f%%" % percentThirdDiffFrequency)

# Run the function using the seq1.txt and seq2.txt files (for the write-up)

codonPosition("seq1.txt", "seq2.txt")