**CS423 Lab 3: Randomness and Probability**

**Translating to Protein Sequences, Finding Potential Genes, and Detecting Motifs**

**Due:** Wednesday, Sept 23, 2015 [beginning of class]

**Purpose:** This lab continues to introduce you to programming in Python by building tools to analyze genomic data. These tools are somewhat general and will give you more practice analyzing DNA sequences.

**Directions:** Find a partner with whom you have not worked this semester (again, the partnership between a person with expertise in biology and a person with expertise in computing). You should take turns as the “driver” and “navigator” while pair programming, swapping roles about every 10 minutes. All submitted work should be typed and include your name(s). Submit your code and write-up (one per pair) *electronically* to Moodle.

**Lab Exercises:**

Download the files for lab 3. Open lab3.py in IDLE. Put your names in comments at the top. Note that this file imports the random module in order to use random number generator functions.

import random

**Random Sequences**: Below is example python code that shows one way to generate a random sequence of 100 DNA nucleotides.

sequence = ""

count = 0

while (count < 100):

randomNum = random.random()

if (randomNum < 0.25):

sequence = sequence + "A"

elif (randomNum < 0.50):

sequence = sequence + "C"

elif (randomNum < 0.75):

sequence = sequence + "G"

else:

sequence = sequence + "T"

count = count + 1

1. **Random sequence generation**. Create code for a function called generateRandomDNA that does the following:

a. Takes two parameters: seqLength that dictates the length of the sequence to be generated and filename as a string which is the name of an output file.

b. Produces a DNA sequence with length seqLength and with DNA composition so that the GC content matches that of the **yeast genome**. A and T should be generated at a rate of 31% each and C and G should be generated at a rate of 19% each. (Instead of the 25% each distribution as the code above does)

c. The resulting sequence should be written to the output file in FASTA format (with > Random DNA Sequence on the top line and the sequence printed 60 characters per row on subsequent lines). The mod operator % is handy for checking to see if you should insert a newline (hint: if the count % 60 is equal to 0, then you write a newline to the file).

*Switch the driver and navigator roles.*

2. **Translation:** It is useful to translate DNA sequences to their corresponding amino acid sequences. Write a function called translate that does the following:

a. Takes two parameters: inputfile of a FASTA filename as a parameter and outputfile as the filename for an output file.

b. Uses the convertFileToSequence function to convert the input file sequence to a string.

c. Translates each DNA codon (3 nucleotides) to the corresponding amino acid (using its single character name). Your function should translate the entire DNA sequence (3 nucleotides at a time) to the single letter amino acid sequence, starting with the first 3 nucleotides listed. You do not need to find a start codon for this exercise. Commented out code in the starter file should help you. If the number of DNA bases is not a multiple of 3, translate what you can. In other words, there may be 1 or 2 DNA bases at the end that are not translated.

d. Open the output file for writing and write “> inputfilename translated to amino acids” as the first line. After that, write the amino acid sequence to the output file, using the single letter codes for amino acids. Write the amino acid sequence, so there are 60 characters per line.

e. Here is an example of the first three lines of a properly formatted output file:

> ORF.txt translated to amino acids

MCMQECCDKCFWHAKWDWMWDWETPCQCWCNSRQSWTWCWEAPLPYWKWFVSDGCDQDQK

QWNCWSNCCVPVIWWDWSRCWCWWQKCVGWCLWNRKRWYPDCVSLPWERYDCWRVCHNDK

*Switch the driver and navigator roles.*

3. **Finding Potential Genes (Open reading frames)**: Now, you will write a function called findORFs that takes a DNA sequence and finds all open reading frames (ORFs). This function should:

a. Have two parameters: inputfile, the name of the input file in FASTA format, and outputfile, the name of the file where the ORFs will be written

b. An open reading frame (ORF) must start with the codon “ATG” and end, in frame, with one of the stop codons: “TAA”, “TAG”, or “TGA”

For example, “ATGCATCGTAGCTAG” is an ORF since the “TAG” at the end is in frame. However, “ATGGATCTAG” is not an ORF since the “TAG” is out of frame.

c. The function should find all ORFs. Note that there might be overlapping ORFs in the sequence in different reading frames and your function should find them all. Hint: the find method may be handy. You may need an outer loop to find the start codon and an inner loop to find the stop codon that goes with this start codon.

d. The function should write each ORF to an output file by writing > ORF <position index>) to the file, writing a newline, and then writing the DNA sequence of each ORF on its own line, with a max of 60 characters per line. An example of what an output file might look is as follows:

> ORF 35

ATGGGGATATAA

> ORF 48

ATGAACCATTTTATATAG

> ORF 1003

ATGACATATGATGGGGGGGGGTTAATTTGA

*Switch the driver and navigator roles.*

4. **Detecting Motifs**: In the first lab, you found instances of TATA boxes – these are regulatory regions in DNA. In computing, we refer to these sequences as motifs. Motifs are highly conserved DNA sequences and may serve as signals for gene regulation, so they are important to study. We can use a weight matrix to describe the probability of each nucleotide at each position in the TATA box motif.

1 2 3 4 5 6

A .04 .90 0.0 .95 .66 .97

C .10 .01 0.0 0.0 .01 0.0

G .03 .01 0.0 0.0 .01 0.03

T .83 .08 1.0 .05 .32 0.0

This weight matrix shows that most common nucleotide in the first position is T, but A, C, and G might occur. Notice that the 3rd position must be a T in the TATA box motif.

The probability that a sequence corresponds to a motif is the product of the frequency of each nucleotide in the given sequence as determined by the weight matrix.

For example, the probability that TATAAA corresponds to a TATA box is:

P(TATAAA) = .83 \* .90 \* 1.0 \* .95 \* .66 \* .97 = 0.45

Let’s look at a sequence that is **not** likely to be a TATA box:

P(GGGGGG) = .03 \* .01 \* 0.0 \* 0.0 \* .01 \* .03 = 0.0

Write a python function called TATAProb that does the following:

a. Takes a hexamer (a length 6 nucleotide sequence) hex as a string parameter

b. Checks that hex is a valid DNA sequence of length 6. If not, return 0.0

c. Calculates and returns the probability of the hexamer as a TATA box

(using the weight matrix above)

(Hint: You may want to store the probabilities in a 2D array or 4 one-dimensional

arrays for easy access.)

*Switch the driver and navigator roles.*

5. Write a python function called printAllHex that does the following:

a. Takes a filename inputfile of a FASTA file as the first parameter

b. Takes a filename outputfile for an output file as the second parameter

c. Converts the sequence in the file to a string (use convertFileToSequence)

d. For each hexamer (start with positions 0 through 5, then do positions 1 through 6, etc.), print its starting index, the hexamer, and its TATAProb to the output filename given as a parameter. Call your TATAProb function to calculate the probability of each hexamer. If you need to convert a number to a string for printing, you may use str(num) in python. The format per line in the output file should be (tabs are \t):

0 ATGTGT 0.0

1 TGTGTA 0.0

*Switch the driver and navigator roles.*

6. When trying to decide how likely a hexamer is actually a TATA box, it would be useful to know how likely the hexamer occurs by random chance in the genome. Some sequences that look like TATA boxes might be a by-product of random chance or simply the result of mutations in the genome, but have no biologic function. Because some genomes have low GC content, the AT content is higher and the chances of a random TATA box sequence appearing in the genome will be higher just due to the background distribution of the genome. So, now, let’s look at the weight matrix for a genome with a low GC content of 38% (yeast):

1 2 3 4 5 6

A .31 .31 .31 .31 .31 .31

C .19 .19 .19 .19 .19 .19

G .19 .19 .19 .19 .19 .19

T .31 .31 .31 .31 .31 .31

The probability of TATAAA occurring by chance, just due to the background distribution of the yeast genome, is:

P(TATAAA) = .31 \* .31 \* .31 \* .31 \* .31 \* .31 = 8.875 x 10-4 = .0008875

Write a python function called chanceHexProbYeast that:

a. takes a hexamer hex as a string parameter

b. if hex is not length 6 or does not contain valid DNA characters, return 0.0

c. calculates and returns the probability of hex occurring by chance in the yeast genome using the distribution weight matrix above. For example, P(GAAATC) = .19\*.31\*.31\*.31\*.31\*.19. You should not need to store the matrix, since the probabilities are defined for each nucleotide irrespective of position within the hexamer.

*Switch the driver and navigator roles.*

7. So, now we can calculate the **likelihood ratio** of a hexamer X being a TATA box:

Likelihood = ProbTATA(X) / ProbCHANCE(X)

What this means is if hexamer X better fits the TATA box weight matrix than the background weight matrix, the numerator will be bigger than the denominator and the likelihood ratio will be greater than 1.0. If hexamer X better fits the background distribution than the TATA box distribution, the denominator will be bigger than the numerator and the likelihood will be less than 1.0 for X.

Write a python function called likelyTATA that:

a. takes a FASTA file inputfile as a parameter and an output file outputfile as a parameter.

b. The function should identify hexamers with likelihood ratios greater than 1.0 using the background distribution of the yeast genome. Call functions that you wrote for parts 4 and 6. For each hexamer with likelihood ratio greater than 1.0, print its position, the hexamer, and the likelihood ratio to the output file (with each entry on its own line, separated by tabs, similar to #6 above).

Note that in class, we created the **log likelihood ratio** profile for motifs, so the calculations involved addition rather than multiplication. To speed the computation, doing the log likelihood profile would be faster for the computer. However, the numbers can get really small (negative infinity), so for the purpose of this lab, Tammy chose for you to stick to multiplication since taking log of 0 causes arithmetic errors.

**Write-Up:**

To verify that you tested your program, create a file containing the information below. This file may be a MS Word file or a pdf file. Include your name(s) and lab number in the file.

**Testing and Evaluation (30 points):**

1. (3 points) Test your generateRandomDNA function by creating a sequence with 500 random nucleotides and storing it to output file random500.txt. Include this file in your zip file for submission.
2. (3 points) Run the translate function on the file ORF.txt. Name the output file for the parameter for this function ORF\_AA.txt. Include this file in your zip file for submission.
3. (12 points) Test the findORFs function on yeast chromosome # 7 (yeast7.txt). Answer the following questions (you might need to write some more python code to help you answer them):
   1. How many ORFs are in yeast chromosome #7?
   2. How many ORFs in yeast chromosome #7 have at least 200 but no more than 3000 nucleotides?
   3. Generate a random sequence of 1090946 nucleotides (using exercise 1) and create a FASTA file with this random sequence by inserting > at the top of the file (name the file randomYeast7.txt). How many ORFs are in this random sequence?
   4. How many ORFs are there in randomYeast7.txt?
   5. How many ORFs in your random sequence have at least 200 but no more than 3000 nucleotides?
   6. Which has more ORFs? (yeast #7 or the random sequence). What is the explanation for the difference in number of ORFs?
4. (2 points) Run TATAProb on the following hexamers: “TATAAA”, “AATAAA”, “CATAAA”. “GGAAAA” What are the probabilities that these are TATA boxes?
5. (2 points) Run chanceHexProbYeast on the following hexamers: “TATAAA”, “AATAAA”, “CATAAA”, “GGAAAA”. What are the probabilities that these occur by chance in yeast?
6. (4 points) Run printAllHex on ORF.txt. Name the output file prob6output.txt and submit it in your zip file.
7. (4 points) Run likelyTATA on ORF.txt. Name the output file prob7output.txt and submit it in your zip file.

**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 (55 points, based on correctness and style):**

Copy and paste the code you wrote for lab 3 here (use Courier or fixed width font size 9). Also, include your lab3.py file in the zip file you submit to Moodle. Be sure that your code is properly commented and each function/exercise can be easily found. Please include the code you wrote to answer the lab questions at the bottom of the script.