CS 369 Assignment 2 2016

Due Friday May 13 8:30 pm

In this assignment, you will simulate pairs of sequences according to a simple evolutionary model, study some properties of the simulated sequences and implement a pairwise alignment algorithm to align simulated sequences.

Write your code in Python and present your code embedded in a report in a Jupyter Notebook. (Ask me first and have a very good reason if you want to use a different language.)

The sequences referred to in this assignment are all assumed to be DNA sequences with four bases, $\{A, C, G, T\}$. Your programs should read and write such sequences, though it may be easier internally to translate the bases to the integers, $\{1, 2, 3, 4\}$ for example.

Remember that the most important thing when writing scientific programs is that they are accurate and are seen to be accurate. This means that you need to test the code thoroughly and write well-structured, commented code that others can understand.

In your report, include explanations of what you are doing and describe any testing you have done to ensure accuracy (feel free to include test code).

Submit your Jupyter notebook to `https://adb.auckland.ac.nz/` by 8:30 pm on the due date.

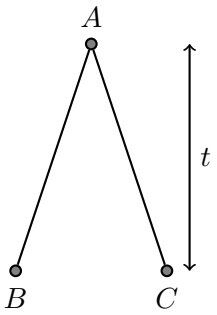
1. *[8 marks total]* The Jukes-Cantor model of DNA sequence evolution is simple: each site mutates at rate μ and when a mutation occurs, a new base is chosen uniformly at random from the four possible bases, $\{A, C, G, T\}$. If we ignore mutations from base X to base X, the mutation rate is $\frac{3}{4}\mu$. All sites mutate independently of each other.

   Thus we observe mutations at a site after an exponentially distributed waiting time with rate $\frac{3}{4}\mu$. At a mutation, choose from the 3 possible bases to mutate to with equal probability.

   A sequence that has evolved over time according to the Jukes-Cantor model has each base equally likely to occur at each site.

   (a) *[4 marks]* Write a method that simulates pairs of sequences that have diverged from a recent common ancestor t time units ago. Assume that evolution has occurred according to the Jukes Cantor model. The distribution for the sequence of the most recent common ancestor is uniform over the four possible bases at each site. The method should take sequence length, time t and mutation rate μ as inputs. It should return the ancestral sequence (A in the figure) and the descendant sequences (B and C in the figure).

   Simulate a pair of sequences of length 50 with $\mu = 0.01$ and $t = 10$. Print the resulting sequences along with the ancestral sequence. Report the number of sites at which each sequence differs from the ancestral sequence and from its sibling sequence.



Sequence A is the most recent common ancestor of sequences B and C.
The time since B split from C is t time units.

(b) *[2 marks]* Explain why you would expect the number of mutations that occur on a tree to be Poisson distributed with parameter $2tL\frac{3}{4}\mu$, where L is the sequence length. Simulate 1000 pairs of sibling sequences of length 1000 with $\mu = 0.01$ and $t = 25$. For each simulated pair, count the number of sites at which they differ from each other. Report the mean and variance of the number of differing sites. Is this number Poisson distributed with parameter $2tL\frac{3}{4}\mu$? Explain why or why not.

(c) *[2 marks]* Simulate a pair of sibling sequences, B and C, of length 10000 with $t = 10$ and $\mu = 0.03$ and use them to calculate the probabilities p_{ab} described in Section 16.3 of the notes. Assume that $p_{ab} = p_{ba}$.

Your estimates for p_{ab} should be close to the theoretical values

$$p_{ab} = \begin{cases} \frac{1}{4} + \frac{3}{4}\exp(-2t\mu) & \text{if } a = b \\ \frac{1}{4} - \frac{1}{4}\exp(-2t\mu) & \text{if } a \neq b. \end{cases}$$

2. *[5 marks total]*

(a) *[4 marks]* Extend your sequence simulation method from Problem 1a by implementing a simple model of insertions and deletions as follows.

First, simulate the ancestral sequence of length L and mutate it over time t with mutation rate μ according to the simulator 1a.

Then simulate insertions: draw a Poisson random variate, h_I, with rate $Lt\mu/10$ giving the number of insertions. For each insertion, select a site uniformly at random from the current length of the sequence and insert a uniform random sequences of length 3 immediately after the selected site. Note that the sequence length changes after each insertion.

Then simulate deletions: draw a Poisson random variate, h_D, with parameter $Lt\mu/10$ giving the number of deletions (use the L you used for the ancestral sequence). For each deletion, select a site uniformly at random from the current length of the sequence and delete this site and the following two sites (or until the end of the sequence if there are not two sites that follow).

For example, simulate sequences B and C of length 7 using your method from 1a to get

```
B: ATGGGTT
C: ATCAGTT
```

Then simulate 1 insertion in B and no insertions in C to get

```
B: ATCATGGGTT
C: AT---CAGTT
```

and simulate 1 deletion in C and no deletions in B to get

```
B: ATCATGGGTT
C: AT---C---T
```

The final length of B is 10 while C is of length 4. Remember that the gaps we show in sequence C here are not part of the sequence, they are just there so that the homologous sites in B and C are aligned.

Demonstrate your simulator by simulating a pair of sibling sequences, B and C, with mutation rate $\mu = 0.01$ from a common ancestor of length $L = 50$, $t = 20$ time units ago. Display B and C in their true alignment with gaps as appropriate where insertions and deletions have occurred. If, by chance, there are no insertions or deletions in your example, run it again.

(b) *[1 mark]* A better model of insertions and deletions might have insertions and deletions occurring at exponentially distributed intervals with per site rate $\mu/10$ (so that a sequence of length k would experience insertions at rate $k\mu/10$ and deletions at rate $k\mu/10$). If this model were run for time t on a sequence of initial length L, explain why it would not produce a Poisson distributed number of insertions and deletions with parameter $2Lt\mu/10$.

3. *[7 marks total]*

(a) *[4 marks]* Implement the overlap alignment algorithm we discussed in lectures. As input, it should take two sequences, a score matrix and a gap penalty. It is sufficient that your algorithm produces a single alignment with the optimal score (rather than all possible alignments with the optimal score).

To aid your answer, the pseudocode given below (taken from Wikipedia) for the global alignment algorithm may help.

(b) *[2 marks]* To demonstrate that your algorithm is correct, align the first 35 bases of B to the last 35 bases of C, where B and C are the ones you generated in Problem 2a.

For example, if your sequences B and C were

```
B: ATCAGTTTGGAGAGGCCAGCTAAATCGCTGATGGTCACTGCCTCGTGCTT
C: AGGAATGTTCAGCTAAAACGCTGATGGCCACTGACTTGTGATT
```

you would select the first 35 bases of B and the last 35 bases of C to get

```
B': ATCAGTTTGGAGAGGCCAGCTAAATCGCTGATGGT
C': TCAGCTAAAACGCTGATGGCCACTGACTTGTGATT
```

which you then align with your overlap alignment algorithm using the score matrix

$$s(x, y) = \begin{cases} 2, & \text{if } x = y \\ -2, & \text{if } x \neq y \end{cases}$$

and a gap penalty of $d = -3$.

(c) *[1 mark]* Repeat Q3b using gap penalties of $d = -4, -2$ and -1 and display the results. In your opinion, which value of d (including -3) gives the best result and why?

Pseudocode for global alignment

```
% make the F matrix
for i=0 to length(A)
  F(i,0) <- d*i
for j=0 to length(B)
  F(0,j) <- d*j
for i=1 to length(A)
  for j=1 to length(B)
  {
    Match <- F(i-1,j-1) + S(Ai, Bj)
    Delete <- F(i-1, j) + d
    Insert <- F(i, j-1) + d
    F(i,j) <- max(Match, Insert, Delete)
  }


% backtrack and form alignment
AlignmentA <- ""
AlignmentB <- ""
i <- length(A)
j <- length(B)
while (i > 0 or j > 0)
{
  if (i > 0 and j > 0 and F(i,j) == F(i-1,j-1) + S(Ai, Bj))
  {
    AlignmentA <- Ai + AlignmentA
    AlignmentB <- Bj + AlignmentB
    i <- i - 1
    j <- j - 1
  }
  else if (i > 0 and F(i,j) == F(i-1,j) + d)
  {
    AlignmentA <- Ai + AlignmentA
    AlignmentB <- "-" + AlignmentB
    i <- i - 1
  }
  else (j > 0 and F(i,j) == F(i,j-1) + d)
  {
    AlignmentA <- "-" + AlignmentA
    AlignmentB <- Bj + AlignmentB
    j <- j - 1
  }
}
```