Pam Simulation homework (assigned April 10\textsuperscript{th} dues April 20\textsuperscript{th})

1. Simple mutation simulation

You job is to implement a very simplified model of evolution that will operate on either an amino acid sequence or a nucleotide sequence. For each scenario outlined below you will generate an ancestral sequence comprised of 1000 characters and then allow that sequence to evolve for the appropriate number of generations. At the end of the specified number of generations count the number of differences between the mutated string and the original. Store this value in a file. For each scenario follow the same procedure 500 times. Then use R to plot the data that you generated. Add a horizontal line at the mean, and two dotted horizontal lines two standard deviations away from the mean. In addition, generate a boxplot of the data (ask me for details). Send me the graphs that you generate and all the programs (in whichever language(s) that you used) to generate the data and the graphs. In this problem it is important that a mutation at a specific site result in a change in the sequence. Assume that any character is as likely to be generated as any other:

Scenarios:

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>amino acid</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>amino acid</td>
<td>1000</td>
</tr>
<tr>
<td>C</td>
<td>amino acid</td>
<td>2500</td>
</tr>
<tr>
<td>D</td>
<td>nucleotide</td>
<td>10</td>
</tr>
<tr>
<td>E</td>
<td>nucleotide</td>
<td>100</td>
</tr>
<tr>
<td>F</td>
<td>nucleotide</td>
<td>1000</td>
</tr>
</tbody>
</table>
2. Using R to simulate evolution

For this next problem we are going to use a more sophisticated evolution process, but I’ll walk you through the steps. Along the way you should learn a little bit more about importing and manipulating data in R. You’re going to need an amino acid sequence to mutate. Use this one:

```
AKLDAKKLAAKLGVKLGLGKLALGKAAGLAAKLKLAKLKLAKLAKLKLKLKLKLKLKLKL
ATKALLAKKLALLKGLAAALLLALTGLAKKLKLKLKLVTGLAKKKGAKGGTAKGSATKAAAG
KKAGLGLGGLLLGLKGLKTAAGVGLAKDQSKLAAALLKGGLGGKLAGLGAGGGLVGLGGGGL
AAAVLKSALLLLLTAGGGGLGGKLLTLGLGKLAKAVAGGAKLKLAKLAKLKLKTVLG
KLGLALAAALGKLLIKLGVKKGKGLGGVKAAGLGLGGAKKAKKLKLALKLGLALGLGALG
LLVAKKTALVAGALLGGLGAAKKGKGKAGGKALGTLALLVAAAKGAKLSGAGKLGAAGLI
LGAKLLLAGSAAKAGGKAKAKLKKAAAKGGLGGGKGKAGVALGAAALLAKGAAAKGLLGALLL
GKKAAGGEGAGGLKARALLAGGLGKLALLLGKLGLKGKLKKLGGKLAGGKLKLGLAGTR
KGGLAGLKLGLGGLGAGKAGAAALKGGAKGAGVKAASLKLKLGGKLGLLAAKKGAPGS
KGKAIATKAGKLLALKAKATGKKAKGGGLGGLAGGKLVALAGGLKLAGGAKALSGKAKKKG
KALKKLLGKSGAKGLAKKLKLKLGGLAKLGLGGLGLGAAKLVALAALTLKLLKALKAGVLKLA
KLKLKLKLGKGLKGGLKGLTTAGGKAKLKAAGKVCGLGAAAGKLGGKAKAALKKLAKLKGLTGA
KLGAAGLGLKGLGLALGLALAGGKLAAALGKGALANALGLGGLKVLAKLAKNAAGAG
GRKLGKLKGAAGKGGKKGGKGALVAKLAKLAKALGKLAKALGKLAKLAKLAKLAKLAKLAL
AKKKGAGGGLGDGLGLALGGLALAAQGLKLAAAGGLLAALLAALAKKGLVAKGVLGGKLALAGGK
KAKLGKSGLGAGGLGGAAEATGAK
```

Steps:

Instead of assuming that ANY amino acid is as likely as any other use the transition matrix underlying the PAM1 model. Let’s import that array into R:

b. Select this portion of the table and copy to the clipboard (it is important to copy EXACTLY what is shown below):

```
    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
B  2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
C  3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
D  4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
E  5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
F  6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
G  7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
H  8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
I  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
J 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
K 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
L 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
M 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
N 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
O 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
P 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Q 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
R 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
S 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
T 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
U 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
V 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
W 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
X 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
Y 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
Z 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
```

c. Now in R type this (NOTE: don’t type the caret):
> tmp = scan(what="character")

d. When you hit [enter], you’ll see something like this: 1:

e. Now paste the data in and hit [enter] a couple of times

f. By default, R will use white space to delimit columns. This means you’ll be missing two entries in the first row. Fix that like this:
> tmp=c(",",",",tmp)

g. Now we’ll make it into a table. First we give it dimensions 22 by 21. That doesn’t seem to match with what we copied into the clipboard, but matrix entries in R go vertically by default... in other words, the vector [1,2,3,4] is equivalent to:
```
1 3
2 4
```

This means that the data we entered has the rows and columns backwards of what we would like. We fix this with the ‘t’ command (t is for transpose):
> dim(tmp)=c(22,21); tmp=t(tmp)

Now we have a text table which is ALMOST correct. Let’s extract the amino acid names:
> acids=tmp[1,3:22]
> acids
[1] "A" "R" "N" "D" "C" "Q" "E" "G" "H" "I" "L" "K" "M" "F" "P" "S" "T" "W" "Y" "V"

h. Next we’ll store the data that we want (finally) and make it pretty:
> PAM1=as.integer(tmp[2:21,3:22])
> dim(PAM1)=c(20,20)
> rownames(PAM1)=acids
> colnames(PAM1)=acids
i. Now that we have a relative count, let’s normalize it:
   > row.sum=apply(PAM1,1,sum)
   > M=PAM1/row.sum

M is our Markov chain transition matrix. We’re going to use it to evolve the sequence I gave you up
above. Loops in R (at least prior to version 2.8—and possibly even now) are horribly expensive... We’re
going to need to be a little bit devious to make R do the mutation fairly efficiently.

   i. copy and paste the string into R... assign it to a variable called str
   ii. Turn it into a character vector:
       ancestor=unlist(strsplit(str,split=''))
   iii. Now mutate it using the matrix ‘M’:
       mut=sapply(ancestor,function(c){sample(acids,1,p=M[c,])})
   iv. And count the differences:
       sum(mut!=ancestor)
   v. Now let’s do it 100 times and see how many similarities we get:
       n=100
       mut=sapply(ancestor,function(c){sample(acids,1,p=M[c,])})
       for(i in 2:n){
           mut=sapply(mut,function(c){sample(acids,1,p=M[c,])})
       }
       sum(mut==ancestor)
   vi. If we had a lot of memory, and weren’t very worried about space, we could be tricky
       and use sapply again.

For this problem, there is nothing to turn in

3. More sophisticated evolution

Using the code provided in Question 2, repeat scenarios A-C from question 1 but with 1/10 the number
of generations. Email me your graphs. Note: In this question you will ALWAYS start with same
ancestral sequence (the one I provided in Question 2).