Noah L. Schrick

1492657

Lab 8. Pairwise Sequence Alignment I. EMBOSS Server and For Loops. Reminder: paste your color-coded responses and output in the attached Word lab handout, re-saved with your name in the file name. You should also save your commands in an R script with a similar file name with .R extension. Submit your report and any scripts.

A. EMBOSS pairwise alignment server and influenza. Segment 4 of influenza genomes encodes the hemagglutinin (HA) protein, which is responsible for allowing the virus to bind to different host cells. The right mutations are needed in HA to allow the virus to infect different hosts. Use the EBI website below to search for the segment 4 (HA) DNA sequences of the two following strains: the A/California/07/2009(H1N1) strain (FJ969540) and the A/Brisbane/59/2007(H1N1) strain (CY030230).

http://www.ebi.ac.uk/

Paste the two sequences into EMBOSS (link below) to globally align the sequences. Choose Nucleotide (because you are aligning DNA) under the Needle algorithm (because you want to perform global alignment). First use the default score settings. Click submit to run.

http://www.ebi.ac.uk/Tools/emboss/

1. Report the gap penalty, total score, identity, similarity and gap percentages. Note that these two sequences are mostly coding (CDS), so we expect more mismatches than insertions/deletions (gaps). 2. With this in mind, redo the global alignment with a larger gap open penalty (try 50.) What is different about the gaps in the middle of this new alignment? Both alignment scores are optimal for the given score parameters, but which might be the better alignment and why biologically? Hint: you expect more mismatches/mutations than gaps because flu virus has a high mutation rate. When gaps do occur, you expect them to occur together in multiples of three (length of a codon). Otherwise, the mutations would be disruptive to downstream translation.

1.
Gap Penalty: 10
Total Score: 4961.5
Identity: 1322/1789 (73.9%)
Similarity: 1324/1789 (74.0%)
Gap Percentage: 116/1789 (6.5%)

2.
Gap Penalty: 50
Total Score: 6925.0
Identity: 1292/1764 (73.2%)
Similarity: 1292/1764 (73.2%)
Gap Percentage: 66/1764 (3.7%)

With a gap open penalty of 50, there are gaps at the beginning and end of the alignment that match the penalty of 10 case, but there is only one gap in the middle of the alignment. The gap

in the middle of the alignment in the gap penalty of 50 case is of length 3.
For the gap penalty of 10, there are gaps at the beginning and end of the alignment, but there are numerous gaps throughout the alignment. Of the gaps in the middle, many of these are not of a length that is a multiple of 3: seven are of length 2, one is of length 4, and one is of length 5.

Given what we know about these DNA sequences of influenza strains, the gap penalty of 50 is the better choice. With a gap length trending around 3 (the length of a codon), there will be fewer issues with downstream translation. In the gap penalty of 10 case, with the numerous gaps that are not multiples of 3, there are likely issues with downstream translation.

Save the fasta files for the two DNA sequences. Use `seqinr's read.fasta` to load the California strain (FJ969540.fa) into R. 3. Use the code below with `seqinr translate` to convert the DNA sequence into protein (amino acid sequence). Show the resulting amino acid sequence frequency table. Need `library(seqinr)`.

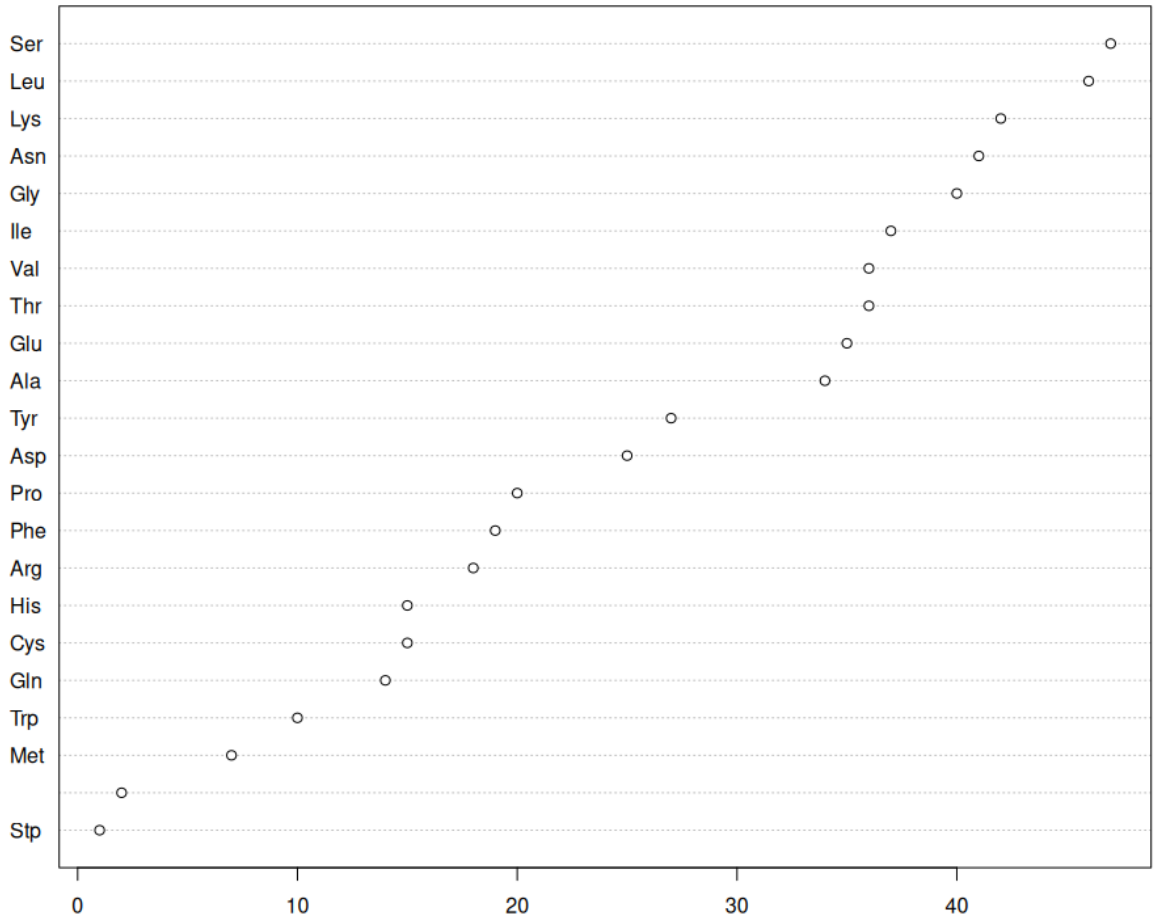
```
h1n1.Cali.aa.vec<-seqinr::translate(h1n1.Cali.dna.vec)
h1n1.Cali.aa.table<-table(h1n1.Cali.aa.vec) # aa count table
```

```
> h1n1.Cali.aa.table
h1n1.Cali.aa.vec
 *  A  C  D  E  F  G  H  I  K  L  M  N  P  Q  R  S  T  V  W  X  Y
 1 34 15 25 35 19 40 15 37 42 46  7 41 20 14 18 47 36 36 10  2 27
```

4. Use the code below with `table` and `order` to create a `dotchart` of the amino acid frequencies for this flu protein. What are the 3 most frequent amino acids used? Which of those 3 are most expected (hint: google the mapping table of 64-codons to 20-amino acids).

```
# sort the table (smallest to largest)
h1n1.Cali.aa.sortedtable <-h1n1.Cali.aa.table[order(h1n1.Cali.aa.table)]
# convert the AA table names to 3-letter code
# You can ignore the warning or remove the offending letters from
# the table
names(h1n1.Cali.aa.sortedtable)<-aaa(names(h1n1.Cali.aa.sortedtable))
dotchart(h1n1.Cali.aa.sortedtable)
```

4.
The 3 most frequent amino acids are Ser, Leu, and Lys.
Ser and Leu are most expected, since they each have 6 total possibilities.



Repeat for the Brisbane sequence. Recall the pairwise alignment diagram from EMBOSS. 5. Describe the ends of the alignments. What does the beginning of the alignments tell you about the location of the start codon (ATG) for the Brisbane sequence (CY030230)? 6. With the location of the start codon in mind, repeat the above translation steps (repeat part 3) for the Brisbane sequence, making sure to use the correct `rm_index` below because translation of `h1n1.Bris.dna.vec` does not begin with the first index of the fasta (there is some extra DNA at the beginning before the start codon). The following code will remove nucleotides 1 through `rm_index` (you need to figure out the index from the alignment diagram by counting).

```
h1n1.Bris.aa.vec <- translate(h1n1.Bris.dna.vec[-seq(1,rm_index)])
```

Show your protein sequence. It should start with M. This code will make it look nice.

```
paste(h1n1.Bris.aa.vec,collapse="",sep="")
```

5.
The beginning of the alignment starts with a large gap of length 32, where the California strain is padded. This gap is due to the Brisbane sequence's start code (ATG) being embedded within the sequence, and not appearing at the start of the sequence itself.
There is also a large gap at the end of the alignment, where the California strain is again padded. The Brisbane sequence is greater in length by 60, totaling at 1761, where the California sequence has a total length of 1701.

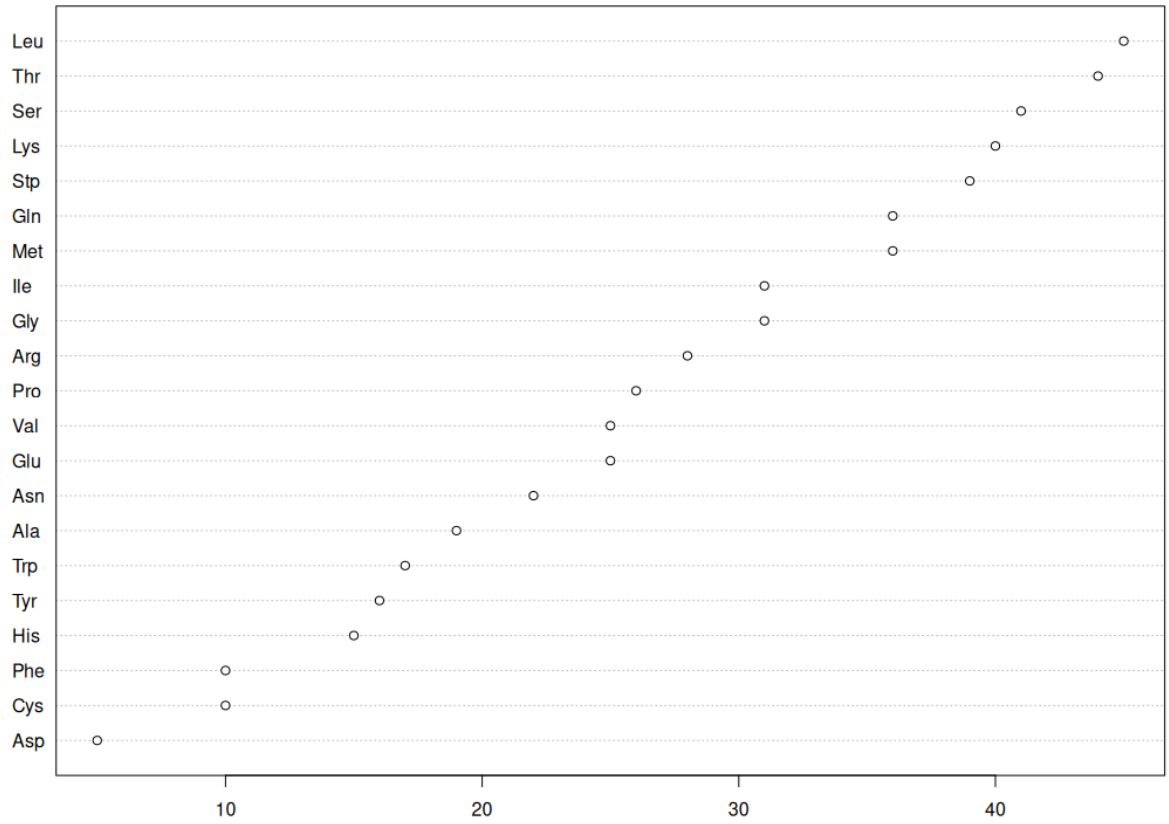
6 .
The start codon is ATG, which translates to Methionine (abbreviated as "M"). From counting, we identify that the index is 27. The code below automatically locates this, and removes it using match:

```
h1n1.Bris.dna.vec <- fasta2vec("CY030230.1.fasta")
h1n1.Bris.aa.vec <- seqinr::translate(h1n1.Bris.dna.vec)
h1n1.Bris.aa.vec <- h1n1.Bris.aa.vec[-seq(1, match('M', h1n1.Bris.aa.vec)-1)]

h1n1.Bris.aa.table<-table(h1n1.Bris.aa.vec) # aa count table
h1n1.Bris.aa.sortedtable <-h1n1.Bris.aa.table[order(h1n1.Bris.aa.table)]

names(h1n1.Bris.aa.sortedtable)<-aaa(names(h1n1.Bris.aa.sortedtable))
dotchart(h1n1.Bris.aa.sortedtable)
paste(h1n1.Bris.aa.vec,collapse="",sep="")
```

```
> paste(h1n1.Bris.aa.vec,collapse="",sep="")
[1]
"MQTQYV*ATMLTTRPTLLTQYLKRM*Q*HTLSTCLRTVTMENYVY*KE*PHYNWVIAALPGGS*E
TQNANY*FPRSHGPTL*KNQILRMEHVTQGISLTMRN*GSN*VQYLHLRGSKYSPKKAHGPTTP*
PECQHHAPIMGKAVFTEICYG*RGRMVCTQT*ASPMQTTKKKKSLYYGVFITRQT*VTKRPSIIQ
KMLMSL*CLHIIAENSPQK*PKDPK*EIKKEESITTGLCLNPGIQ*YLRQMEI**RQDMLSH*VEALD
QESSTQMHQWINVMRSAKHLREL*TAVFLSRTYTQSQ*ESVQSMSGVQN*GWLQD*GTSHPFN
PEVCLEPLPVSLKGGGLEW*MVGMVIIIRMSKDLAMLQIKKAHKMPLMGLQTR*IL*LRK*TLNSQ
QWAKNSTNWKEGWKT*IKKLMMGL*TFGHIMQNCWFYWKMKGLWISMTPM*RICMRK*KAS*R
IMLKK*EMGVLNSITSVTMNAWRV*RMELMTIQNIPKNQS*TGRKLME*NWNQWESIRFWRSTQ
QSPVLWFFWSPWGQSASGCVPMGLYSVEYASKTRISEI*GKT"
```



B. Warmup nested for loop recursion problem for next lab. Write the following code in an R script for finding the total number of paths on a grid with m row **edges** and n column **edges**. Paths start at the upper left corner and end at the lower right corner. Paths are only allowed to move right or down. Note: because n and m are the number of edges and R matrices start indexing at 1, not 0, we need to make the matrix size and the ending of the for loops be m+1 and n+1. 1. Complete the code where indicated and write comments describing what the lines do. The recursive rule for number of paths is $P_{i,j} = P_{i-1,j} + P_{i,j-1}$.

```
m<-3; n<-6     # these will be part of the function header
path_matrix <- matrix(1, nrow=m+1, ncol=n+1) # init a matrix of size rows+1
and cols+1 (add space for gap) with all 1s
path_matrix # shows output, comment out/remove from function
for (i in seq(2,m+1)){
      for (j in seq(2,n+1)){
            path_matrix[i,j] <- path_matrix[i-1,j] + path_matrix[i,j-1] # the
current cell in the matrix is equal to the current row but adjacent left
column, plus the current column but adjacent left row
```

```
      }
}
path.matrix # in function, below return the last row and col element
```

```
> path_matrix
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    1    2    3    4    5    6    7
[3,]    1    3    6   10   15   21   28
[4,]    1    4   10   20   35   56   84
```

```
> path_matrix[m+1, n+1]
[1] 84
```

2. Turn the above code into a function called `calc.num.paths` with inputs n and m, and return value the [n+1, m+1] element of `path.matrix`. 3. Compare your output for (n,m) = (5,5), (5,6), and (10,10) with the closed form solution: P(n,m)=(n+m)!/(n!m!). Note: in R, n! is factorial(n).

2.

```
calc.num.paths <- function(n,m){
  path_matrix <- matrix(1, nrow=m+1, ncol=n+1)
  for (i in seq(2,m+1)){
    for (j in seq(2, n+1)){
      path_matrix[i,j] <- path_matrix[i-1,j] + path_matrix[i,j-1]
    }
  }
  path_matrix[m+1,n+1]
}
```

3.

```
> m <- 5 # row edges
> n <- 5 # col edges
> calc.num.paths(n,m)
[1] 252
> factorial(n+m)/(factorial(n)*factorial(m))
[1] 252
```

```
> m <- 5 # row edges
> n <- 6 # col edges
> calc.num.paths(n,m)
[1] 462
> factorial(n+m)/(factorial(n)*factorial(m))
[1] 462
```

```
> m <- 10 # row edges
> n <- 10 # col edges
> calc.num.paths(n,m)
[1] 184756
```

```
> factorial(n+m)/(factorial(n)*factorial(m))
[1] 184756
```

The function returns the same result as the closed form solution.

Continued…

C. Dinucleotide Signals. Codons are triplet (trinucleotide) signals in DNA for translating to amino acids. We have also seen quadruplet TATA palindrome signals for starting translation. There are also doublet CG (dinucleotide) signals. When many CG's are strung together, you get a CpG island, which can suppress the transcription/expression of a gene. The following function creates a series of sliding-window CG content likelihoods from an input fasta vector. The likelihood is the ratio of the observed CG frequency compared to the expected frequency (the product of C and G frequencies). Put this function in your script and comment what the lines do.

```
calc.sliding.cpg <- function(fastaVec, slideWin){
      # allocate memory for odds ratio output
      cg_dinuc_oddsRatio <- double()
      n<-length(fastaVec)
      for (i in 1:(n-slideWin)){
            # 1. what is [i:(i+slideWin)] doing below?
            # 2. what is count(...,2) doing below? Use ?count help.
            dinucs<-count(fastaVec[i:(i+slideWin)],2)
            # 3. dinucs is a table with names, what is ["cg"] doing?
            cg_dinuc_count <- dinucs["cg"]
            # 4. for a given step, what does nucs return below?
            nucs<-count(fastaVec[i:(i+slideWin)],1)
            # 5. why is (nucs["c"]*nucs["g"]) in the denominator?
            cg_dinuc_oddsRatio[i] <-
                                    cg_dinuc_count/(nucs["c"]*nucs["g"])
      }
      return(cg_dinuc_oddsRatio) # returns vector
}
```

1. Array slicing: this particular portion looks only at the vector indices from 'i' to 'i+slideWin'. For a snapshot example, if i=5 and slideWin=10, then this would only look at the vector values at index 5 to 15.

2. This obtains the dinucleotide counts/appearances. For a snapshot example, if the vector slice being examined was "ATGAA", then this would count the number of times "AA" appeared, "AC" appeared, "AG", "AT", CA", "CC"… etc. For this example, "AA" appears once, "AT" appears once, "GA" appears once, and "TG" appears once.

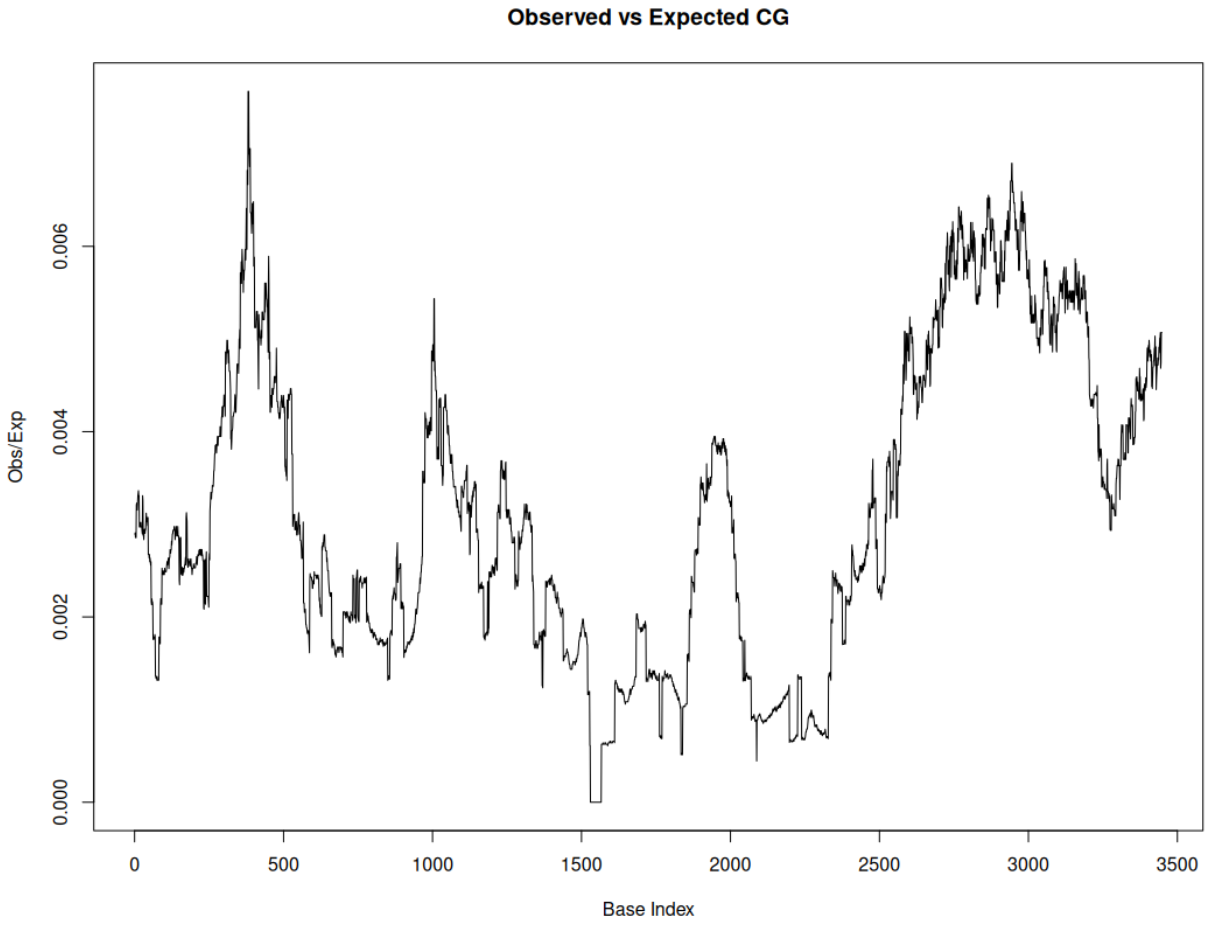
3. This retrieves the number of times the dinucleotide "CG" appeared in the given segment.

4. This obtains a count of just the nucleotides. It would return the number of times “A” appeared, “C” appeared, “G” appeared, and “T” appeared.

5. The base nucleotides will appear more often than their pairing. This obtains the number of times the pairing appears per times that the individual components appear.

6. Apply your cpg function (code example below) to the APOE fasta file from a previous lab and show the plot. **Note**: when reading the fasta file, you will need to split the string into a vector of characters because the function above uses a vector as input. Where might there be CpG islands?

```
cpg_vec <- calc.sliding.cpg(apoe.fasta.vec,150)
plot(cpg_vec,type="l",main="Observed vs Expected CG",xlab="Base Index", ylab="Obs/Exp")
```



There might be CpG islands at base index 500, 1100, and in the general area of 2500-3350.