Noah L. Schrick
1492657

Pairwise Sequence Alignment with dynamic programming. Use color to indicate your answers and submit R files.

Background. We will implement the recursive rules for constructing the elements of the alignment matrix `F` for two sequences (x and y) using dynamic programming. In the examples, we will use DNA sequences, which have simpler scoring matrices than protein sequences. The inputs are two sequences x and y; a penalty for adding a gap to one sequence (`gap_penalty`); and a match (`match_score`) and mismatch (`mismatch_score`) score that are used to create the substitution matrix S. The outputs are two matrices: the final score matrix (`F`) and the traceback matrix (`T`). For each ij-element of the `F` matrix, the algorithm looks for the path that leads to the maximum score from three directions (from above, from the left, or diagonal back). Whichever of the three paths gives the max then corresponds to an integer (1, 2, or 3) that is placed in the corresponding ij-element of the traceback matrix `T`. Max uses the order when dealing with ties.

$$F_{i,j}=\max\begin{cases} F_{i-1,j-1}+S\left[x_{i-1},y_{j-1}\right] & T_{i,j}=1 \quad (\text{match/mismatch}) \\ F_{i-1,j}+\text{gap}_{\text{penalty}} & T_{i,j}=2 \quad (\text{gap in up seq/from above}) \\ F_{i,j-1}+\text{gap}_{\text{penalty}} & T_{i,j}=3 \quad (\text{gap in side seq/from left}) \end{cases}$$

mis/match

up-gap

left-gap

A. Specifying the input. The algorithm requires two input sequences and choices for scoring matches, substitutions, and gaps. For concreteness we will prototype the code using the two sequences and the scoring information that we did in class. Begin a script with the following example input.

```
##### input
x_str <- "ATAC"   # side sequence
y_str <- "GTGTAC" # top sequence
match_score <- 3
mismatch_score <- -1
gap_penalty <- -4
```

Substitution (S) Matrix. Before creating the F and T matrices, we need to create the substitution matrix, S. This does not involve the recursion algorithm above and does not involve the gap penalty. It is simply a lookup matrix that contains the **match** and **mismatch** score for each pair of letters in the alphabet. Add the following code to your script and run it. This code is similar to the handout during the last class. **1.** Fill in and explain the code. **2.** What is the size of the S matrix? **3.** What does the if-statement check for and what should the two possible outputs be (blanks)?

```
#S-code
dna.letters<-c("A","C","G","T")
num.letters <- length(dna.letters)
S<-data.frame(matrix(0,nrow=num.letters,ncol=num.letters))  # data frame
rownames(S)<-dna.letters; colnames(S)<-dna.letters
for (i in 1:4){
      for (j in 1:4){
            if(dna.letters[i]==dna.letters[j]){

                  S[i,j]← match_score
            }
            else{
                  S[i,j]← mismatch_score
            }
      }
}
```

1.
This code initializes a matrix based on the specified DNA letters with rows and columns named as “A”, “C”, “T”, and “G”.
The code then loops through this matrix, and compares each cell’s row and column name/associated DNA letter.
If the two letters are the same (for instance, the cell is in row “A”, and the cell is in column “A”), then the match score is placed in that cell. If the two letters are not the same (for instance, cell row “A”, but cell column “C”), then the mismatch score is placed in that cell.

2.
The size of the S matrix is 4x4, corresponding to the 4 DNA letters specified.

3.
The if statement checks if the row name and column name match. For instance, if the cell’s row is “G”, the check compares the cell’s row to see if it is also “G”. If they are the same, the match score is placed in the cell. If they are not the same (“G” and “T”), then the mismatch score is placed in the cell.

4. Show how you access the mismatch score for "A" and "T" from the S matrix (using the letters or indices). Note that S is a data frame, so you can use "A" and "T" to access elements.

```
> S["A", "T"]
[1] -1

> match("A", dna.letters)
[1] 1
> match("T", dna.letters)
[1] 4
> S[1,4]
[1] -1
```

B. Initializing the Alignment Score Matrix (F) and Traceback Matrix (T).

Before using the recursive formula, we need to initialize the F and T matrices. **1.** Add the code below to your script, run it and document what it does. **2.** Why is the size of Fmat (and Tmat) larger than the size of x and y by 1? **3.** Why are the first rows and columns of Fmat given the values they are given in the code?

```
#F-code 1
x <- unlist(strsplit(x_str, ""))
y <- unlist(strsplit(y_str, ""))
x.len <- length(x)
y.len <- length(y)
```

```
Fmat<-matrix(0,nrow=x.len+1,ncol=y.len+1)
Tmat<-Fmat # 0's to start

rownames(Fmat)<-c("-",x); colnames(Fmat)<-c("-",y)
rownames(Tmat)<-c("-",x); colnames(Tmat)<-c("-",y)

# create first row and column
Fmat[,1]<- seq(from=0,len=x.len+1,by=-abs(gap_penalty))
Fmat[1,]<- seq(from=0,len=y.len+1,by=-abs(gap_penalty))
Tmat[,1]<- rep(2,x.len+1)  # 2 means align with a gap in the upper seq
Tmat[1,]<- rep(3,y.len+1)  # 3 means align with a gap in the side seq
```

1.
The code starts by extracting each individual letter from each sequence, and counting how many letters there are.
The F matrix is then initialized to "1 plus the x size" by "1 plus the y size", and is filled with zeroes. The T matrix is initialized to the same by just making a copy of the F matrix.
Each matrix has their columns and rows named as a "-" to start, followed by the sequence.
The first row and column of the F matrix begins with a 0, then each subsequent cell in the row takes the previous value, and subtracts the gap penalty from it.
The first T matrix row is filled with 3s, and the first T matrix row is filled with 2s (apart from the first cell, which gets overwritten with a 3).

2.
The matrix sizes are larger than x and y by 1 to account for the initial gap that is added.

3.
The initial corner cell is '0' since the alignment has not yet begun (starts with a gap). For each subsequent row and column, since it is misaligned by a gap, the gap penalty is applied.

C. Building the Alignment Score Matrix (F) and Traceback Matrix (T). The recursive algorithm uses the maximum of three numbers at each step. To see how `max` works, run the following at the command line. **1.** What do `max` and `which.max` do?

```
my.numbers <- c(7,9,-4)
max(my.numbers)
which.max(my.numbers)
```

1.
'max' retrieves the highest value (9 in this example).
'which.max' retrieves the index of the highest value (index of 9 in this example, which is 2)

2. What happens with which.max if `my.numbers <- c(9,9,-4)`, that is, if there is a tie?
It retrieves the first occurrence of the maximum (1, in this example)

Now we can use the recursive equation for F and T. **3.** Complete the code below and add to your script; figure out what goes in the three blanks. **The blanks are based on the recursive equation at the top of the handout**. For the match/mismatch part, S needs to have letters input from sequences x and y, which need to be indexed by i and j.

```
#F-code 2
# Use the recursion formula with the max function
for (i in 2:nrow(Fmat)){
      for (j in 2:ncol(Fmat)){     # use F recursive rules
            test_three_cases <- c(Fmat[i-1, j-1] + S[rownames(Fmat)[i],
colnames(Fmat)[j]],    # 1 mis/match
                                  Fmat[i-1,j]+gap_penalty,    # 2 up-gap
                                  Fmat[i, j-1] + gap_penalty)   # 3 left-gap
            Fmat[i,j]=max(test_three_cases)
            Tmat[i,j]=which.max(test_three_cases)
      }
}
final_score <- Fmat[nrow(Fmat),ncol(Fmat)]
```

4. Show the resulting F and T matrices and the final_score. Verify that it matches what we found in the lecture slides.

```
Fmat
    -    G    T    G    T    A    C
-   0   -4   -8  -12  -16  -20  -24
A  -4   -1   -5   -9  -13  -13  -17
T  -8   -5    2   -2   -6  -10  -14
A -12   -9   -2    1   -3   -3   -7
C -16  -13   -6   -3    0   -4    0
```

```
> Tmat
  - G T G T A C
- 3 3 3 3 3 3 3
A 2 1 1 1 1 1 3
T 2 1 1 3 1 3 1
A 2 1 2 1 1 1 3
C 2 1 2 1 1 1 1
```

```
> final_score
[1] 0
```

5. Use Tmat and follow the rules to create the alignment of the two sequences.

```
> seq_align
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "-"  "-"  "A"  "T"  "A"  "C"
[2,] "G"  "T"  "G"  "T"  "A"  "C"
```

D. Organizing code into functions. As we showed in previous labs, functions are a good way to organize and clean up your code. Use your code for making S, F, and T matrices to write a function called `make.alignment.matrices`. The function should have the following structure. The inputs are the two sequences to align and the scoring parameters. In order to return multiple output variables, we return a `list` data type, which uses names that can be accessed using the $ operator as illustrated later in the handouot.

```
make.alignment.matrices <- function(x_str, y_str, match_score, mismatch_score,
gap_penalty){
       # add this function to script
       #
       # Body: Add S-code and F-code from above
       #
```

```
        final_score <- Fmat[nrow(Fmat),ncol(Fmat)]
        return(list(Fmat=Fmat, Tmat=Tmat, score_out=final_score))
}
```

Use the following code to test your function and the following new sequence pair.

```
# load new input
x_str2 <- "GATTA"  # side sequence
y_str2 <- "GAATTC" # top sequence
match_score <- 2
mismatch_score <- -1
gap_penalty <- -2

align.list2 <- make.alignment.matrices(x_str2, y_str2, match_score,
                                        mismatch_score, gap_penalty)
align.list2$Fmat
align.list2$Tmat
align.list2$score_out
```

1. Show the F, T, and final score. Use the T matrix to create the alignment by hand. Add the following code to the script to create a heatmap of the F matrix. You may need to install gplots.
2. Show the heatmap.

```
install.packages("gplots")
library(gplots)
Fmat2 <- align.list2$Fmat
col = c("black","blue","red","yellow","green")
breaks = seq(min(Fmat2),max(Fmat2),len=length(col)+1)

heatmap.2(Fmat2[-1,-1], dendrogram='none', density.info="none",
          Rowv=FALSE, Colv=FALSE, trace='none',
          breaks = breaks, col = col,
          sepwidth=c(0.01,0.01),
          sepcolor="black",
          colsep=1:ncol(Fmat2),
          rowsep=1:nrow(Fmat2))
```

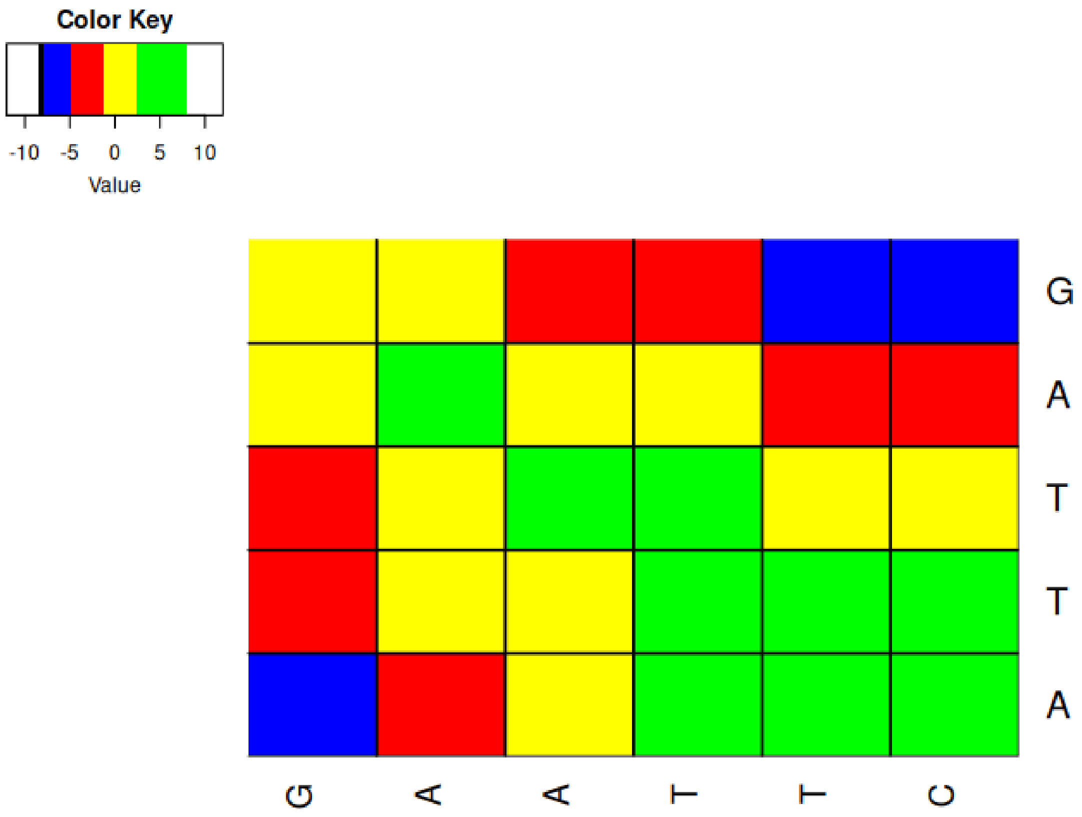
1.

```
> align.list2$Fmat
   -  G  A  A  T   T   C
-  0 -2 -4 -6 -8 -10 -12
G -2  2  0 -2 -4  -6  -8
A -4  0  4  2  0  -2  -4
T -6 -2  2  3  4   2   0
T -8 -4  0  1  5   6   4
A -10 -6 -2  2  3   4   5
```

```
> align.list2$Tmat
  - G A A T T C
- 3 3 3 3 3 3 3
G 2 1 3 3 3 3 3
A 2 2 1 1 3 3 3
T 2 2 2 1 1 1 3
T 2 2 2 1 1 1 3
A 2 2 1 1 2 1 1

> align.list2$score_out
[1] 5
```

2.



E. Traceback Matrix. Add the following traceback function your script. The while loop traverses the T matrix backwards starting at the lower right corner and writes the two-row `alignment` matrix from right to left.

```
show.alignment <- function(x_str,y_str,Tmat){
	################ create the alignment
	# input Tmat and the two sequences: x side seq and y is top seq
	# make character vectors out of the strings
	x<-unlist(strsplit(x_str,""))
```

```
        y<-unlist(strsplit(y_str,""))

        n<-nrow(Tmat) # start at bottom right of Tmat
        m<-ncol(Tmat)
        alignment<-character()
        while( (n+m)!=2 ){
              if (Tmat[n,m]==1){
              # subtract 1 from x and y indices because they are
              # one row/col smaller than Tmat
                    curr_align_col <- rbind(x[n-1],y[m-1])
                    alignment <- cbind(curr_align_col,alignment)
                    n=n-1; m=m-1; # move back diagonally
              }else if(Tmat[n,m]==2){
                    curr_align_col <- rbind(x[n-1],"-") # put gap in top seq
                    alignment <- cbind(curr_align_col,alignment)
                    n=n-1 # move up
              }else{
                    curr_align_col <- rbind("-",y[m-1]) # put gap in side seq
                    alignment <- cbind(curr_align_col,alignment)
                    m=m-1 # move left
              }
        } # end while
        return(alignment)
} # end function
```

1. Use the following code to show the alignment for x_str2 and y_str2. The variable **`align.list2$Tmat`** is the T matrix you computed for these sequences.

```
#alignment <- show.alignment(x_str,y_str,Tmat) # first sequences
alignment2 <- show.alignment(x_str2,y_str2,align.list2$Tmat)
alignment2
# the following prints nicely
write.table(alignment2,row.names=F,col.names=F,quote=F)
```

```
> alignment2 <- show.alignment(x_str2,y_str2,align.list2$Tmat)

> alignment2
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "G"  "-"  "A"  "T"  "T"  "A"
[2,] "G"  "A"  "A"  "T"  "T"  "C"

> write.table(alignment2,row.names=F,col.names=F,quote=F)
G - A T T A
G A A T T C
```

2. Test your alignment functions for the following alignment scenario.

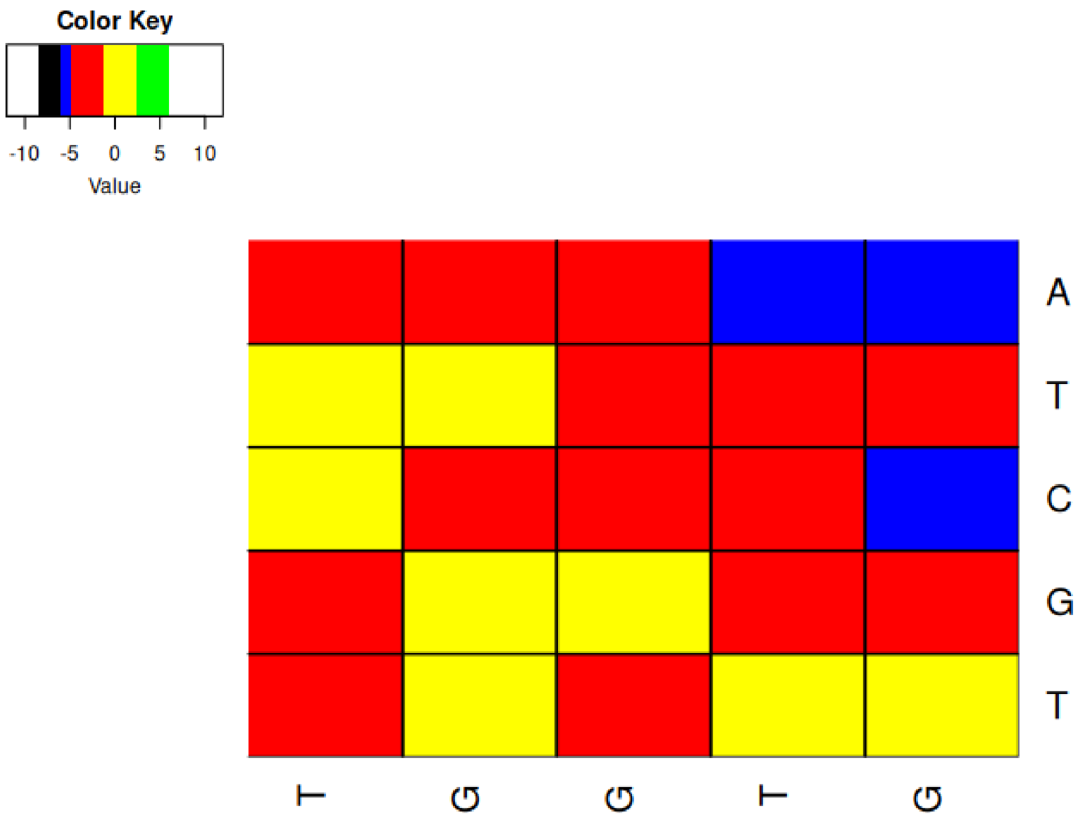
```
x_str3 <- "ATCGT"  # side sequence
y_str3 <- "TGGTG" # top sequence
match_score <- 1
mismatch_score <- -2
gap_penalty <- -1
```

Show F, T, final score, the alignment and heatmap.

```
> align.list3$Fmat
   -  T  G  G  T  G
-  0 -1 -2 -3 -4 -5
A -1 -2 -3 -4 -5 -6
T -2  0 -1 -2 -3 -4
C -3 -1 -2 -3 -4 -5
G -4 -2  0 -1 -2 -3
T -5 -3 -1 -2  0 -1
```

```
> align.list3$Tmat
  - T G G T G
- 3 3 3 3 3 3
A 2 1 1 1 1 1
T 2 1 3 3 1 3
C 2 2 1 1 1 1
G 2 2 1 1 3 1
T 2 1 2 1 1 3
```

```
> align.list3$score_out
[1] -1
```



```
> alignment3
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "A"  "T"  "C"  "G"  "T"  "-"
```

```
[2,] "-"  "T"  "G"  "G"  "T"  "G"
> write.table(alignment3,row.names=F,col.names=F,quote=F)
A T C G T -
- T G G T G
```