In this homework, we will discuss evolutionary models using the T92 model of DNA evolution. This model accounts for both transition/transversion and GC bias.

	A	G	C	T
A	-	$\kappa g/2$	$g/2$	$(1-g)/2$
T	$\kappa(1-g)/2$	-	$g/2$	$(1-g)/2$
C	$(1-g)/2$	$g/2$	-	$\kappa(1-g)/2$
G	$(1-g)/2$	$g/2$	$\kappa g/2$	-

Table 1: T92 model: g indicates the G+C proportion and κ is the transition/transversion bias. It is assumed that $p_A = p_T = (1-g)/2$ and $p_C = p_G = g/2$

1 T92 model of DNA evolution

1. Show that the Jukes-Cantor model is nested in the T92 model by showing the constraints it imposes on the T92 parameters.
2. Determine if T92 has a stationary distribution.
3. Find the stationary distribution (guess what you think it should be, then prove the guess is correct).
4. Show that this model is time reversible.
5. What is the overall rate of substitution?
6. Find the discrete time transition probability matrix embedded in this model.

2 Tree likelihood

The goal of this problem is to build a tree from 4 DNA sequences. Using the T92 model, estimate the branch lengths and topology.

```
A  CACGCAGATCAGCACGCCCGGAGTCAAAAACACGTACCAAGCCACGCCTCCCGTCTCACTTCTGCAATTTACATC
B  GTGACTCGAAACCTGCCCCTGACTCAAAAACACCTACCCACTCAACCGATGCGCCTCTCGTCTGCATTGCTCGGC
C  GACGCGGTACAGCTGGCGCGGAGACAAAATCAGGTACCATGAGACGGCTCCCGCCTCGCTTCTGCCACTCACATC
D  GTTACCCCAAACGTGCGCCTGACGCAAAAACAACCTGCCAGTCAACCCATGCGCCTGTGGTCTCCAATTGACGTG
```

1. Find the likelihood function for a given tree, branch lengths, and T92 parameters.
2. Write an R function for this likelihood.
3. Maximize the likelihood over κ, g, and branchlengths for each topology, (*e.g.* with `optim()`). Do the κ and g values you calculated make sense (no need to go too deep here, just a quick reality check)?
4. Plot each of your trees along with branch lengths (you may use the `plot_tree` function from below). Interpret the branch lengths.

Skeleton code for the project is included below. If you use any of the code below, be sure you understand it.

```r
# You will need this for handling trees
library(ape)
# You will need this for matrix exponentiation
library(Matrix)

# I will be using this base ordering globally
alphabet <- c('A', 'C', 'G', 'T')

# Read unrooted newick tree into a phylo object
load_tree <- function(newick_tree){
  unroot(ape::read.tree(text=newick_tree))
}

# Build a T92 Q matrix, where
# g := G+C%
# k := transition/transversion bias
make_Q <- function(g,k){
  mat <- matrix(c(
  # A           C       G       T
    0         , g/2   , g/2   , k*(1-g)/2 , # A
    (1-g)/2   , 0     , k*g/2 , (1-g)/2   , # C
    (1-g)/2   , k*g/2 , 0     , (1-g)/2   , # G
    k*(1-g)/2 , g/2   , g/2   , 0           # T
  ),
    byrow=TRUE,
    dimnames=list(alphabet, alphabet),
    ncol=4,
    nrow=4
  )
  diag(mat) <- -1*colSums(mat)
  mat
}
# Build a Q matrix:
(Q <- make_Q(0.5, k=1))
# And you can exponentiate a Q matrix to get the P matrix, for example:
Matrix::expm(Q * 0.1)

logsum <- function(x){
  # adapted from http://andrewgelman.com/2016/06/11/log-sum-of-exponentials/
  logsum2 <- function(a,b){
    max(a,b) + log(exp(a - max(a,b)) + exp(b - max(a,b)))
  }
  Reduce(logsum2, x[-1], x[1])
}

likelihood <- function(b, k, g, X, tree){
  # TODO: finish this function

  # FYI: The edge map stored by `phylo` objects will be of use when you compute
  # the likelihoods for all possible trees.
}

# This function requires recursive calls to optim
```

```r
optimize_tree <- function(tree, X){
  likelihood_wrapper <- function(par, ...){
    likelihood(g=par[1], k=par[2], b=par[-(1:2)], ...)
  }
  # optimize for branch lengths
  result <- optim(
    par     = c(0.5, 1, tree$edge.length),
    fn      = likelihood_wrapper,
    X       = X,
    control = list(fnscale=-1),
    tree    = tree
  )
  tree$edge.length <- result$par[-(1:2)]
  list(
    L = result$value,
    g = result$par[1],
    k = result$par[2],
    tree = tree
  )
}

# read in alignment as a matrix
X <- matrix(unlist(strsplit(c(
  "CACGCAGATCAGCACGCCCGGAGTCAAAAACACGTACCAAGCCACGCCTCCCGTCTCACTTCTGCAATTTACATC",
  "GTGACTCGAAACCTGCCCCTGACTCAAAAACACCTACCCACTCAACCGATGCGCCTCTCGTCTGCATTGCTCGGC",
  "GACGCGGTACAGCTGGCGCGGAGACAAAATCAGGTACCATGAGACGGCTCCCGCCTCGCTTCTGCCACTCACATC",
  "GTTACCCCAAACGTGCGCCTGACGCAAAAACAACCTGCCAGTCAACCCATGCGCCTGTGGTCTCCAATTGACGTG"
  ), "")), nrow=4, byrow=TRUE)
# give the sequences the names A, B, C, D
rownames(X) <- LETTERS[1:4]

trees <- list() # TODO: make a list of all permutations of the tree

# FYI: With my likelihood function (which I didn't try to optimize), this code
# runs in a few minutes, if your code is taking much longer, tean you are
# probably doing something wrong.
optimal_trees <- lapply(trees, function(tree) optimize_tree(tree, X=X))

# Plot an unrooted tree with visible node labels and branch lengths. Also add
# labels for inferred parameters.
# x is the list returned from optimize_tree:
# list(
#   L    :: tree likelihood
#   g    :: G+C proportion
#   k    :: transition/transversion bias
#   tree :: phylo object
# )
plot_tree <- function(x){
  plot(
    x$tree,
    main = sprintf("L=%.5f g=%.5f k=%.5f", x$L, x$g, x$k),
    show.node.label=TRUE,
    type='unrooted'
```

```
  )
  edgelabels(signif(x$tree$edge.length, 5), cex=0.7)
}

plot_tree(optimal_trees[[1]])
plot_tree(optimal_trees[[2]])
# etc, etc, etc
```

```
# Here is a tree example:
library(ape)

# Newick formatted tree with made-up branch lengths (specified by ':<number>')
#   ((A:0.1,B:0.2)Y1,(C:0.25,D:0.15)Y2:0.22);
#      A   B           C    D
#   0.1 \ / 0.2     0.25 \ / 0.15
#      Y1 -------------- Y2
#            0.22 (after unrooting)
newick_tree <- "((A:0.1,B:0.2)Y1:0,(C:0.25,D:0.15)Y2:0.22);"
tree <- load_tree(newick_tree)
plot(tree, show.node.label=TRUE, type='unrooted')

# In phylo objects, the leafs are always indexed first, so indices 1-4
# correspond to the leafs.
# Here is the edge map:
tree$edge
```

3 BLAST – Independence

The purpose of this question is to investigate the assumptions made by BLAST. Understanding these assumptions is very important when interpreting BLAST results.

BLAST assumes site-independence across a sequence. This is certainly not the case for biological sequences. The goal of this question is to perform an ad hoc test of how much this assumption might influence our results.

There is no biochemical mechanism that could reverse the order of amino acids in a coding sequence (just reversing the nucleotide order creates a radically different amino acid sequence). So a reversed protein can be seen as a unique (but subtly dependent) draw from random protein distribution for the given species. Extremely low entropy intervals in a protein may retain sequence similarity upon reversal (e.g. "DDDDDDDD" or "ADADADADADAD"). Comparing forward and reverse protein sequences has been used to test algorithms, such as the 'tantan' repeat masker (Frith, 2010).

Find the 'reverse-blast.tab' file on Canvas. It contains the BLAST results from a search of 1000 randomly selected and reversed proteins against the full proteome. These BLAST searches are ungapped. You shouldn't usually use ungapped BLAST in practice, but it simplifies the interpretation for this homework. I have provided the code to load the file below:

```
require(magrittr, quietly = T)
require(dplyr, quietly = T, warn.conflicts = F)
resultr <- read.table('reverse-blast.tab') %>%
    dplyr::group_by(V1) %>%
    dplyr::filter(V3 == max(V3)) %>%
    dplyr::ungroup(V1) %>%
```

```
    dplyr::select(score=V3, evalue=V4, qseq=V5, sseq=V6)

# NOTE: BLAST will not report a hit with an E-value greater than 10. Thus our
# data are truncated.
nrow(resultr) == 1000

## [1] FALSE
```

1. Calculate a p-value for each result and make a histogram of the result. Interpret this histogram. Is it what you expected it to be? What would you expect it to look like if all the BLAST assumptions held?

2. After adjusting for multiple testing (e.g. with 'p.adjust'), how many of the reversed sequences appear to be statistically significant? How many would you expect to be significant given BLAST assumptions held?

4 BLAST – Homogeneity

Another assumption of BLAST is that every sequence is described by the same matrix. We can test this compositional assumption by randomizing the amino acid order of each protein. This preserves the composition, but removes true homology and forces site-independence.

The BLAST data can be loaded in the same way as before.

```
require(magrittr, quietly = T)
require(dplyr, quietly = T, warn.conflicts = F)
resultp <- read.table('permuted-blast.tab') %>%
    dplyr::group_by(V1) %>%
    dplyr::filter(V3 == max(V3)) %>%
    dplyr::ungroup(V1) %>%
    dplyr::select(score=V3, evalue=V4, qseq=V5, sseq=V6)
```

1. Calculate a p-value for each result and make a histogram of the result. Interpret this histogram. Is it what you expected it to be? What would you expect it to look like if all the BLAST assumptions held?

2. After adjusting for multiple testing (e.g. with 'p.adjust'), how many of the reversed sequences appear to be statistically significant? How many would you expect to be significant given BLAST assumptions held?

5 Appendix

5.1 References

Martin Frith (2010) "A new repeat-masking method enables specific detection of homologous sequences."

5.2 Code

The code below includes the data prep and blast commands. The 'at.faa' file is the *Arabidopsis thaliana* proteome (TAIR10 annotation). I use 'smof' to prepare the sequence (`https://github/arendsee/smof`).

You don't need to look at or run any of this code, since I give you results to you in Canvas files.

```
# Randomly sample 1000 protein sequences from at.faa.
# The random seed is set to 42 for reproducibility.
smof sample --seed 42 -n 1000 at.faa |
    # Remove extraneous header info
    smof clean -s |
    # Randomize the order of amino acids in each protein, but preserve the
    # location of the first residue (since it is biologically constrained).
    smof permute -s 1 > permuted.fa

smof sample --seed 42 -n 1000 at.faa |
    smof clean -s |
    # Reverse the order of residues in each protein
    smof reverse > reverse.fa
```

Prepare blast databases

```
makeblastdb -dbtype prot -in at.faa
```

Run each BLAST (assuming you are a wimp with only 2 cores)

```
# Run an ungapped BLAST search against
# Uses the default matrix, BLOSUM62

blastp -db at.faa -query reverse.fa                 \
  -outfmt '6 qseqid sseqid bitscore evalue gaps'    \
  -ungapped -comp_based_stats F                     \
  -max_target_seqs 1 > reverse-blast.tab

blastp -db at.faa -query permuted.fa                \
  -outfmt '6 qseqid sseqid bitscore evalue gaps'    \
  -ungapped -comp_based_stats F                     \
  -max_target_seqs 1 > permuted-blast.tab
```