Altschul and Gish 1996 - Table 1, Single Script Approach

Nathan Brouwer

10/29/2019

Introduction

The code below can be used to replicate Table 1 on page 465 of Altschul and Gish (1996) "Local Alignment Statistics." This table is the result of a **computational experiment** to understand the statistical basis for **E values** and **P values** of **un-gapped local alignments** like those used in **BLAST searches**.

Each row of the table represents the result of 10,000 simulations. I refer to each row of the table as a subexperiment. During each iteration of a subexperiment, two separate but equal length random sequences of amino acids were generated. The sequences were then run through a local alignment algorithm and scored using a BLOSUM 62 scoring matrix. The score (s) and length of the alignment was then stored. Once all 10,000 iterations were finished the mean length of the alignments (l) was calculated. The distribution of scores was then analyzed using statistical methods which can estimate the parameters mu and lambda of the Gumbel **extreme value distribution** (EVD). The parameters mu and lambda were then plugged into the following equation to solve for the parameter K

Plain text version of equation:

mu = (ln(m x n x K))/lambda

Rendered version of equation.

$$\mu = \frac{\ln(m * n * K)}{\lambda}$$

This script below takes a very basic and inefficient approach to replicating this table. **Part 0** reproduces the original table. **Part 1** of the script outlines a basic workflow for a single iteration of the experiment (a single alignment). **Part 2** demonstrates how to use a for() to carry out the necessary computations for a full run of simulations necessary to create one row of the table. **Part 3** demonstrates how to calculate the numbers that appears in the table, using the first row as an example. **Part 4** contains additional copies of the code to run the computations for *all* of the lines of the table. **Part 5** processes the data produced by Part 4 to calculate mu and lambda. **Part 6** compiles the final table and calculates K.

Other scripts in this project folder take a more advanced approach to this problem.

Part 0: The original table

```r
mn          <- c(191,245,314,403,518,665,854,1097,1408,1808,2322,2981)
l.orig      <- c(22.6 ,25.8 ,29.4, 32.4,36.3,40.3,43.9,48.1,51.6,55.1,59.1,63.5)
ln.n.m.     <- c(10.25, 10.78,11.30,11.83,12.35,12.87,13.39,13.91,14.43,14.94,15.45,15.96)
u.orig      <- c(26.45,28.31,30.21,32.04,33.92,35.94,37.84,39.75,41.71,43.54,45.53,47.32)
lambda.orig <- c(0.298,0.286,0.282,0.275,0.279,0.273,0.272,0.275,0.268,0.271,0.267,0.270)
K.orig      <- c(0.073,0.055,0.051,0.041,0.048,0.041,0.040,0.046,0.036,0.041,0.035,0.040)

table1 <- data.frame(mn = mn,
                      l = l.orig,
             ln.nm = round(log(mn*mn),3),
             ln.n.m.,
             u = u.orig,
             lambda = lambda.orig,
             K = K.orig)

log(245*245)

table1.NLB
```

Part 1: Basic alignment workflow

Below I outline the basic steps needed to replicate Table 1 of Altchul and Gish (1996).

Preliminaries: Packages / libraries

Two packages are needed for this simulation experiment:

1. Biostrings: for pairwiseAlignment() function
2. extRemes: For calculating parameters related to the Gumbel extreme value distribution.

```r
library(Biostrings)

## Install extRemes if necessary
# install.packages("extRemes")

library( extRemes)
```

Polypeptide simulation

Simulation of polypeptides is done using the sample() function. This requires a vector of possible amino acids letters and their probabilities of occurring.

Possible amino acids

Make a vector of all letters that represent amino acid

```r
# 1 letter codes
aa1 <- c("A", "C", "D", "E", "F", "G", "H",
         "I", "K", "L", "M", "N", "P", "G",
         "R", "S", "T", "V", "W", "Y")


aa3 <- c("Ala","Cys","Asp","Glu","Phe","Gly","His","Ile","Lys",
  "Leu","Met","Asn","Pro","Gln","Arg","Ser","Thr","Val",
  "Trp","Tyr")
```

Example: Randomly select a single amino acid with equal frequency

```r
sample(x = aa1, size = 1, replace = T)
```

Example: Make a vector of 200 amino acids (eg, a simulated polypeptide)

```r
pp.length <- 20
sample(x = aa1, size = pp.length, replace = T)
```

Amino acid frequencies (Robinson & Robinson 1991)

Amino acid frequencies from Robinson and Robinson (1991) “Distribution of glutamine and asparagine residues and their near neighbors in peptides and proteins” PNAS. (R & R 1991). These frequencies are used by Altschul and Gish (1996) for their simulated polypeptides.

Number of each amino acid reported in (R & R 1991):

```r
aa.count <- c(35155,8669,24161,28354,17367,
              33229,9906,23161,25872,40625,
              10101,20212,23435,19208,23105,
              32070,26311,29012,5990,14488)
```

Check frequency; should sum to 450431

```r
aa.total <- sum(aa.count)
aa.total
```

Setting up amino acid relative frequencies

Create dataframe of amino acid frequencies (Note: stringsAsFactors = F prevents a default R behavior which is annoying)

```r
robinson.aafreq <- data.frame(aa3,
                              aa1,
                              aa.count = aa.count,
                              stringsAsFactors = F)
```

Calculate **relative frequency** of each amino acid:

```r
robinson.aafreq$aa.freq <- robinson.aafreq$aa.count/aa.total
```

Check that relative frequency sums to 1

```r
sum(robinson.aafreq$aa.freq)
```

Example simulation experiment workflow

The following code walks through the basic workflow of the simulation for a random sequence of length n = 191.

Random sequence 1

Create first sequence

```r
seq1 <- sample(x = robinson.aafreq$aa1,
               size = 191,
               replace = TRUE,
               prob = robinson.aafreq$aa.freq)
```

Turn sequence into string

```r
seq1 <- paste(seq1, sep = "",
               collapse = "")
```

Random sequence 2

```r
# create sequence
seq2 <- sample(x = robinson.aafreq$aa1,
                  size = 2981,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)

# turn into a string
seq2 <- paste(seq2, sep = "",collapse = "")
```

Align simulated sequences

pairwiseAlignment() from Biostrings is used to do a local alignment of the two random sequences.

```r
alignment.i <- pairwiseAlignment(pattern = seq1,
                                 subject = seq2,
                    type              = "local",
                    substitutionMatrix = "BLOSUM62",
                    gapOpening         = 12,
                    gapExtension       = 1)
```

Look at the alignment; note that for long alignments this can be abridged with a "…" in the middle

```r
alignment.i
```

Process alignment

Once and alignment has been generated it needs to be processed

Alignment score (s): Score based on specified scoring matrix and indel parameters. This is accessed using the function score().

```r
score.i         <- score(alignment.i)
score.i
```

Alignment length: Extract full alignment sequence as a string with Biostrings::toString()

```r
alignment.seq.i <- toString(alignment.i)
alignment.seq.i
```

Determine the actual length of the alignment with base R nchar() function

```r
length.i        <- nchar(alignment.seq.i)
length.i
```

The score and length then need to be stored in a dataframe and the above process repeated several thousand times.

Part 2: Implement one sub-experiment as loop

In this sections the basic computations of the experiment are iterated in order to generate the date for one row of the table.

Run experiment for n = m = 191 amino acids

The first row of Table 1, page 465 indicates that simulated sequences of 191 amino acids were investigated. **In the original simulation 10000 iterations were carried out; we'll do just 1000.**

for() loop for 191 amino acids

```r
# create dataframe to store scores and lengths
random.scores.191 <- data.frame(interation = 1:1000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 191,
                         seq.length2 = 191)

# for loop

for(i in 1:1000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size = 191,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                 size = 191,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                 collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                   type               = "local",
                   substitutionMatrix = "BLOSUM62",
                   gapOpening         = 12,
                   gapExtension       = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.191$score.i[i]  <- score.i
 random.scores.191$length.i[i] <- length.i

}
```

The results of one full set of simulations look like this:

```r
head(random.scores.191)

summary(random.scores.191)
```

Part 3: Data analysis

Sample analysis and exploration of n = m = 191 data

Distribution of scores

The scores produced by the simulation are similar in shape to an extreme value distribution (EVD)

```r
random.scores.191$score.i
```

We can make the graph a bit smoother by setting breaks = 20, which determines the number of bars in the histogram

```r
hist(random.scores.191$score.i,
     breaks = 20)
```

If we want we can make R give each score its own bar on the histogram. The code below does this, but is a bit dense and I won't explain it

```r
x <- range(random.scores.191$score.i)
bins <- x[2]-x[1]+1
hist(random.scores.191$score.i,
     breaks = bins)
```

Determining the mode

The Gumbel extreme value distribution has two parameters: mu and lambda.

mu is defined as the highest point of the distribution. This is similar to the **mode** of the distribution. We can therefore approximate mu by calculate the mode. We can get the mode using the table() function, followed by some processing

Make a table of the scores

```r
table.i <- table(random.scores.191$score.i)
table.i
```

I can use which.max() to figure out which element of the table has the highest value.

```r
i.max <- which.max(table.i)
table.i[i.max]
```

The scores are actually scored as the names of the table elements. I can get them using names()

```r
mode.i <- names(table.i[i.max])
mode.i
```

Names are character data so I use as.numeric() to turn it into a numeric value

```r
mode.i <- as.numeric(mode.i)
mode.i
```

I can now make a histogram with the mode shown. abline() with v = mode.i puts a line at the mode.

```r
hist(random.scores.191$score.i,
     breaks = bins)
abline(v = mode.i, col = 2, lwd = 4, lty = 2)
```

Estimating mu and lambda - explanation

Statistical distributions are described by equations. For example, the normal distribution is described by two parameters, the mean and standard deviation. These equations can be complex-looking, but luckily R has all the math already set up within functions.

The Gumbel EVD is described by an equation with two parameters: mu and lambda. Mu is approximately the mode, but we want a precise value. There's no easy way to get an approximate value for lambda. To get these parameters we'll feed the results of our simulation to a sophisticated statistical algorithm the will figure out which values of lambda and mu best fit the data.

There are several ways to do the math underlying estimation of mu and lambda; Altschul and Gish use the **method of moments**; in R its easiest to use **maximum likelihood estimation** which is called **MLE** for short.

Estimation of mu and lambda is done with the function fevd(). The fevd() function will doing some complex math for us very quickly. Basically what its doing is figuring which parameters, if plugged in to a simulation, would result in data that is most similar to our observed data.

We can simulate random evd data like this (don't worry about specific.) "location" refers to mu, so we'll put our best estimate of mu, the mode. scale is equal to 1/lambda. I know lambda is often around 0.27, so I"ll put in values on either side of that

I'll simulate 3 sets of values, one set for each value of lambda. I'll simulate 1000 replications each.

First, the scale parameters I'll use

```r
scale1 <- round(1/0.135, 4)
scale2 <- round(1/0.27,  4)
scale3 <- round(1/0.405, 4)
```

Now I'll run my simulations

```r
# number of simulated values
ns <- 1000

# simulate data with scale1
revd.scale1 <-  revd(n = ns, loc = mode.i, scale = scale1, type = "GEV")

# simulate data with scale2
revd.scale2 <-  revd(n = ns, loc = mode.i, scale = scale2, type = "GEV")

# simulate data with scale3
revd.scale3 <-  revd(n = ns, loc = mode.i, scale = scale3, type = "GEV")
```

Now I'll plot all 3 of these and compare to the original data. I'll make a 4 x 4 grid, with the original data in the upper left. This code is a bit dense, don't worry about details

First I'll set some stuff up for the plot. Don't worry about these details

```r
all.data <- c(revd.scale1,revd.scale2,revd.scale3)
xlims <- c(min(all.data)-1, max(all.data)+1)
breakz <- seq(xlims[1],xlims[2], length.out = xlims[2]-xlims[1])
```

For reference, I'm going to plot the minimum and maximum of the original data on each plot

```r
max.i <- max(random.scores.191$score.i)
min.i <- min(random.scores.191$score.i)
```

Now I'm going to plot the original data. This is the data from my for() loop above. The mode is in red, and min and max are in green. Again, this is data from actual alignment.s

```r
par(mfrow = c(2,2), mar = c(2,1,1,2))
hist(random.scores.191$score.i,
     breaks = breakz,
     main = "orig alignment data",
     xlim = xlims)
abline(v = mode.i, col = 2, lwd = 4, lty = 2)
abline(v = max.i, col = 3, lwd = 1, lty = 2)
abline(v = min.i, col = 3, lwd = 1, lty = 2)
```

Now I'm going to plot the data for the first set of data simulated from the evd. No alignments were involved in this - this is just using the equation for an EVD via the revd() function ("random extreme value distribution") to simulate data for a given value of mu and lambda. Recall that for mu I used the mode of the original alignment data. The green lines are the min and max of the original alignment data. Note that the data simulated from an EVD goes much higher and much lower than the min and max of the alignment data.

```r
hist(revd.scale1,
     breaks = breakz,
     main = paste("scale = ",scale1), xlim = xlims)
abline(v = max.i, col = 3, lwd = 1, lty = 2)
abline(v = min.i, col = 3, lwd = 1, lty = 2)
```

Now the second set of data simulated from the EVD. This is actually pretty close, with values going just a bit above and bit below the min and max.

```r
hist(revd.scale2,
     breaks = breakz,
     main = paste("scale = ",scale2), xlim = xlims)
abline(v = max.i, col = 3, lwd = 1, lty = 2)
abline(v = min.i, col = 3, lwd = 1, lty = 2)
```

Finally the third set of simulated data. This isn't too bad for the minimum, but the value fall pretty short of the maximum.

```r
hist(revd.scale3,
     breaks = breakz,
     main = paste("scale = ",scale3), xlim = xlims)
abline(v = max.i, col = 3, lwd = 1, lty = 2)
abline(v = min.i, col = 3, lwd = 1, lty = 2)
```

Estimating mu and lambda in R

Let's have R estimate mu and lambda for real for us. This is done easily like this:

```r
fit.gumbel <- fevd(random.scores.191$score.i,
                     type = "Gumbel",
                     method = "MLE")
```

(For the brave, a good outline of the math is here:https://stats.stackexchange.com/questions/71197/usable-estimators-for-parameters-in-gumbel-distribution; this requires some familiarity with the concept of likelihoods).

The summary() function gets important results for the model

```r
summary(fit.gumbel)
```

What we're looking for is under "Estimated parameters". mu = Location, and lambda = 1/scale If scale = 3.48, 1/3.48 = 0.2873563

We can save the summary output to an object

```r
summary.gumbel <- summary(fit.gumbel)
```

With some trial and error I figured out where location (mu) and scale (1/lambda) were stored and saved them to their own object.

```r
mu.param  <- summary.gumbel$par[1]
scale.param <- summary.gumbel$par[2]
lambda.parm  <- 1/scale.param
```

I can plot the original data represented as a smooth curve (black) and compare it to the modeled data (blue dashed). The lines are similar to each other so I there is good agreement.

```r
par(mfrow = c(1,1))
plot(fit.gumbel, type = "density")
abline(v = loc.param, col = 2, lwd = 4, lty = 2)
```

I can also simulate data based on the mu and scale (1/lambda) from our model

```r
revd.from.model <-  revd(n = ns, loc = mu.param, scale = 1/lambda.parm, type = "GEV")
```

Now I'll compare this to the observed data

```r
# observed data
par(mfrow = c(2,1), mar = c(2,1,1,2))
hist(random.scores.191$score.i,
     breaks = breakz,
     main = "orig alignment data",
     xlim = xlims)
abline(v = mode.i, col = 2, lwd = 4, lty = 2)
abline(v = max.i, col = 3, lwd = 1, lty = 2)
abline(v = min.i, col = 3, lwd = 1, lty = 2)

# simulated data from fitted model
hist(revd.from.model,
     breaks = breakz,
     main = "orig alignment data",
     xlim = xlims)
abline(v = max.i, col = 3, lwd = 1, lty = 2)
abline(v = min.i, col = 3, lwd = 1, lty = 2)
```

Estimating K

Key to E values and P values for alignments are lambda and K. From the extreme value distribution of scores we can get lambda and mu and need to calculate K.

Plain text version of equation:

mu = (ln(m x n x K))/lambda

Rendered version of equation.

$$\mu = \frac{\ln(m * n * K)}{\lambda}$$

We can re-arrange this to be

$$\lambda * \mu = \ln(m * n * K)$$

And then

$$\exp(\lambda * \mu) = \exp(\ln(m * n * K))$$

Which simplifies to

$$\exp(\lambda * \mu) = m * n * K$$

and then

$$\frac{exp(\lambda * \mu)}{m * n} = K$$

For ease let's flip that around

$$K = \frac{exp(\lambda * \mu)}{m * n}$$

We could also write it like this

$$K = \frac{e^{\lambda * \mu}}{m * n}$$

In R we can therefore get K like this (recall that m = n = 191, the length of the simulated sequences)

```r
m <- 191
n <- 191

K <- (exp(lambda.parm*mu.param))/(m*n)
K
```

An equivalent expression is worked out below, though I haven't derived it:

```r
exp(lambda.parm*mu.param - log(m*n))
```

Summary Part 3

Once we've simulated data we can make a histogram of all of the alignment scores. We can then use R functions to analyze the distribution of scores and estimate mu and lambda. From mu and lambda we can calculate K. In the first row of Table 1 Altchul and Gish (1996) report, for 10000 simulated sequences of n = m = 191:

- mu = 26.45
- lambda = 0.298
- K = 0.073

Using R we get

```
mu.param
lambda.parm
K
```

These values are close. The most likely reason for them not being closer is that we ran 1/10 as many simulations as the original paper, though there could be subtle numerical differences in parts of our procedure that are different than Altchul and ish.

Part 4: Simulate whole experiment

In this section of the script we generate the raw score data for each row of Table 1. I've copy, pasted, and modified by hand the code for each row.

The original table ran simulations for these sequence lengths:

191, 245, 314, 403, 518, 665, 854, 1097, 1408, 1808, 2322, 2981

The code is set for 2000 iterations; the original paper did 10000

I'm curious how long this will take, so I'll record when I start the simulations

```
sim.start <- Sys.time()
```

for() loop for 191 amino acids

```
# create dataframe to store scores and lengths
random.scores.191 <- data.frame(interation = 1:100000,
                          score.i = NA,
                          length.i = NA,
                          seq.length1 = 191,
                          seq.length2 = 191)

# for loop

for(i in 1:100000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size = 191,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                  size = 191,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                  collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                    type              = "local",
                    substitutionMatrix = "BLOSUM62",
                    gapOpening        = 12,
                    gapExtension      = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.191$score.i[i]  <- score.i
 random.scores.191$length.i[i] <- length.i

}
```

for() loop for 245 amino acids

```
# create dataframe to store scores
random.scores.245 <- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 245,
                         seq.length2 = 245)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                  size = 245,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                  collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                  size = 245,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
```

```r
                  collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                    type              = "local",
                    substitutionMatrix = "BLOSUM62",
                    gapOpening        = 12,
                    gapExtension      = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.245$score.i[i]  <- score.i
 random.scores.245$length.i[i] <- length.i

}
```

for() loop for 314 amino acids

```r
# create dataframe to store scores
random.scores.314 <- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 314,
                         seq.length2 = 314)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                  size = 314,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                  collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                  size = 314,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                  collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                     type                = "local",
                     substitutionMatrix = "BLOSUM62",
                     gapOpening         = 12,
                     gapExtension       = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.314$score.i[i]  <- score.i
 random.scores.314$length.i[i] <- length.i

}
```

for() loop for 403 amino acids

```
# create dataframe to store scores
random.scores.403 <- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 403,
                         seq.length2 = 403)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size = 403,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                 size = 403,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                 collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                     type                = "local",
                     substitutionMatrix = "BLOSUM62",
                     gapOpening         = 12,
                     gapExtension       = 1)
```

```r
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.403$score.i[i]  <- score.i
 random.scores.403$length.i[i] <- length.i

}
```

for() loop for 518 amino acids

```r
# create dataframe to store scores
random.scores.518<- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 518,
                         seq.length2 = 518)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size = 518,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                 size = 518,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                 collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                   type               = "local",
                   substitutionMatrix = "BLOSUM62",
                   gapOpening         = 12,
                   gapExtension       = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.518$score.i[i]  <- score.i
 random.scores.518$length.i[i] <- length.i

}
```

for() loop for 665 amino acids

```r
# create dataframe to store scores
random.scores.665<- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 665,
                         seq.length2 = 665)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size = 665,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                 size = 665,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                 collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                   type                = "local",
                   substitutionMatrix = "BLOSUM62",
                   gapOpening          = 12,
                   gapExtension        = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.665$score.i[i]  <- score.i
 random.scores.665$length.i[i] <- length.i

}
```

for() loop for 854 amino acids

```r
# create dataframe to store scores
random.scores.854<- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 854,
                         seq.length2 = 854)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size = 854,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                 size = 854,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                 collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                  subject = seq2,
                   type              = "local",
                   substitutionMatrix = "BLOSUM62",
                   gapOpening        = 12,
                   gapExtension      = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.854$score.i[i]  <- score.i
 random.scores.854$length.i[i] <- length.i

}
```

for() loop for 1097 amino acids

```
# create dataframe to store scores
random.scores.1097 <- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 1097,
                         seq.length2 = 1097)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                  size = 1097,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                  collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                  size = 1097,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                  collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                    subject = seq2,
                    type                = "local",
                    substitutionMatrix = "BLOSUM62",
                    gapOpening          = 12,
                    gapExtension        = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.1097$score.i[i]  <- score.i
 random.scores.1097$length.i[i] <- length.i

}
```

for() loop for 1408 amino acids

```r
# create dataframe to store scores
random.scores.1408 <- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 1408,
                         seq.length2 = 1408)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                  size = 1408,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                  collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                  size = 1408,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                  collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                  subject = seq2,
                    type               = "local",
                    substitutionMatrix = "BLOSUM62",
                    gapOpening         = 12,
                    gapExtension       = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.1408$score.i[i]  <- score.i
 random.scores.1408$length.i[i] <- length.i

}
```

for() loop for 1808 amino acids

```r
# create dataframe to store scores
random.scores.1808 <- data.frame(interation = 1:10000,
                             score.i = NA,
                             length.i = NA,
                             seq.length1 = 1808,
                             seq.length2 = 1808)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                  size = 1808,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                  collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                  size = 1808,
                  replace = TRUE,
                  prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                  collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                    type               = "local",
                    substitutionMatrix = "BLOSUM62",
                    gapOpening         = 12,
                    gapExtension       = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.1808$score.i[i]  <- score.i
 random.scores.1808$length.i[i] <- length.i

}
```

for() loop for 2322 amino acids

```r
# create dataframe to store scores
random.scores.2322 <- data.frame(interation = 1:10000,
                             score.i = NA,
                             length.i = NA,
                             seq.length1 = 2322,
                             seq.length2 = 2322)
```

```r

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size =  2322,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                 size = 2322,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                 collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                   type               = "local",
                   substitutionMatrix = "BLOSUM62",
                   gapOpening         = 12,
                   gapExtension       = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.2322$score.i[i]  <- score.i
 random.scores.2322$length.i[i] <- length.i

}
```

for() loop for 2981 amino acids

```r
# create dataframe to store scores
random.scores.2981 <- data.frame(interation = 1:10000,
                         score.i = NA,
                         length.i = NA,
                         seq.length1 = 2981,
                         seq.length2 = 2981)

# for loop

for(i in 1:10000){
 seq1 <- sample(x = robinson.aafreq$aa1,
                 size = 2981,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq1 <- paste(seq1, sep = "",
                 collapse = "")

 seq2 <- sample(x = robinson.aafreq$aa1,
                 size = 2981,
                 replace = TRUE,
                 prob = robinson.aafreq$aa.freq)
 seq2 <- paste(seq2, sep = "",
                 collapse = "")

 alignment.i <- pairwiseAlignment(pattern = seq1,
                                   subject = seq2,
                    type               = "local",
                    substitutionMatrix = "BLOSUM62",
                    gapOpening         = 12,
                    gapExtension       = 1)
 score.i         <- score(alignment.i)

 # cat("Iteration ", i)
 # cat("score ", score.i)
 # cat("\n")

 alignment.seq.i <- toString(alignment.i)

 length.i        <- nchar(alignment.seq.i)


 random.scores.2981$score.i[i]  <- score.i
 random.scores.2981$length.i[i] <- length.i

}
```

Part 5: Analysis of all data

Output from simulations

All data objects produced by each for loop

random.scores.191 random.scores.245 random.scores.314 random.scores.403 random.scores.518 random.scores.665 random.scores.854 random.scores.1097 random.scores.1408 random.scores.1808 random.scores.2322 random.scores.2981

Get mean length of alignments

```r
l.191 <- mean(random.scores.191$length.i)
l.245 <- mean(random.scores.245$length.i)
l.314 <- mean(random.scores.314$length.i)
l.403 <- mean(random.scores.403$length.i)
```

```r
l.518 <- mean(random.scores.518$length.i)
l.665 <- mean(random.scores.665$length.i)
l.854 <- mean(random.scores.854$length.i)
l.1097 <- mean(random.scores.1097$length.i)
l.1408 <- mean(random.scores.1408$length.i)
l.1808 <- mean(random.scores.1808$length.i)
l.2322 <- mean(random.scores.2322$length.i)
l.2981 <- mean(random.scores.2981$length.i)
```

Fit Gumbel extreme value distribution model to each set of output

```r
library(extRemes)
fit.gumbel.191 <- fevd(random.scores.191$score.i,
                   type = "Gumbel",
                   method = "MLE")


fit.gumbel.245 <- fevd(random.scores.245$score.i,
                   type = "Gumbel",
                   method = "MLE")


fit.gumbel.314 <- fevd(random.scores.314$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.403 <- fevd(random.scores.403$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.518 <- fevd(random.scores.518$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.665 <- fevd(random.scores.665$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.854 <- fevd(random.scores.854$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.1097 <- fevd(random.scores.1097$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.1408 <- fevd(random.scores.1408$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.1808 <- fevd(random.scores.1808$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.2322 <- fevd(random.scores.2322$score.i,
                   type = "Gumbel",
                   method = "MLE")

fit.gumbel.2981 <- fevd(random.scores.2981$score.i,
                   type = "Gumbel",
                   method = "MLE")
```

Get summary out from each model

All of the output from all of the models

fit.gumbel.191 fit.gumbel.245 fit.gumbel.314 fit.gumbel.403 fit.gumbel.518 fit.gumbel.665 fit.gumbel.854 fit.gumbel.1097 fit.gumbel.1408 fit.gumbel.1808 fit.gumbel.2322 fit.gumbel.2981

```r
summary.gumbel.191 <- summary(fit.gumbel.191)
loc.param.191 <- summary.gumbel.191$par[1]
scale.param.191  <- 1/summary.gumbel.191$par[2]

summary.gumbel.245 <- summary(fit.gumbel.245)
loc.param.245 <- summary.gumbel.245$par[1]
scale.param.245  <- 1/summary.gumbel.245$par[2]

summary.gumbel.314 <- summary(fit.gumbel.314)
loc.param.314 <- summary.gumbel.314$par[1]
scale.param.314  <- 1/summary.gumbel.314$par[2]

summary.gumbel.403 <- summary(fit.gumbel.403)
loc.param.403 <- summary.gumbel.403$par[1]
scale.param.403  <- 1/summary.gumbel.403$par[2]

summary.gumbel.518 <- summary(fit.gumbel.518)
loc.param.518 <- summary.gumbel.518$par[1]
scale.param.518  <- 1/summary.gumbel.518$par[2]

summary.gumbel.665 <- summary(fit.gumbel.665)
loc.param.665 <- summary.gumbel.665$par[1]
scale.param.665  <- 1/summary.gumbel.665$par[2]

summary.gumbel.854 <- summary(fit.gumbel.854)
loc.param.854 <- summary.gumbel.854$par[1]
scale.param.854  <- 1/summary.gumbel.854$par[2]

summary.gumbel.1097 <- summary(fit.gumbel.1097)
loc.param.1097  <- summary.gumbel.1097$par[1]
scale.param.1097  <- 1/summary.gumbel.1097$par[2]

summary.gumbel.1408 <- summary(fit.gumbel.1408)
loc.param.1408 <- summary.gumbel.1408$par[1]
scale.param.1408  <- 1/summary.gumbel.1408$par[2]

```

```
summary.gumbel.1808 <- summary(fit.gumbel.1808)
loc.param.1808 <- summary.gumbel.1808$par[1]
scale.param.1808  <- 1/summary.gumbel.1808$par[2]


summary.gumbel.2322 <- summary(fit.gumbel.2322)
loc.param.2322 <- summary.gumbel.2322$par[1]
scale.param.2322  <- 1/summary.gumbel.2322$par[2]


summary.gumbel.2981<- summary(fit.gumbel.2981)
loc.param.2981 <- summary.gumbel.2981$par[1]
scale.param.2981  <- 1/summary.gumbel.2981$par[2]
```

Part 6: Compile table

```
mn <- c(191,245,314,403,518,
        665,
        854,
        1097,
        1408,1808,2322,
        2981
        )

lenght.l <- c(l.191 ,l.245 ,l.314 ,l.403 ,l.518 ,
              l.665,
              l.854 ,l.1097 ,
              l.1408 ,l.1808 ,l.2322 ,
              l.2981
              )

location.u <- c(loc.param.191,
                loc.param.245,
                loc.param.314,
                loc.param.403,
                loc.param.518,
                loc.param.665,
                loc.param.854,
                loc.param.1097,
                loc.param.1408,
                loc.param.1808,
                loc.param.2322,
                loc.param.2981
                )

scale.lambda <- c(scale.param.191,
                  scale.param.245,
                  scale.param.314,
                  scale.param.403,
                  scale.param.518,
                  scale.param.665,
                  scale.param.854,
```

```r
                   scale.param.1097,
                   scale.param.1408,
                   scale.param.1808,
                   scale.param.2322,
                   scale.param.2981
                   )



table1.NLB <- data.frame(mn = mn,
                    l = lenght.l,
           ln.mn = log(mn*mn),
           mu = location.u,
           lambda =scale.lambda
           )
```

Calculate K

```r
table1.NLB$K <-  exp((location.u*scale.lambda) - log(mn*mn))
```

Look at table

```r
head(table1.NLB)
```

Round off columns

```r
# table1.NLB$ln.mn  <- round(table1.NLB$ln.mn, 3)
# table1.NLB$lenght.l     <- round(table1.NLB$l, 2)
# table1.NLB$mu     <- round(table1.NLB$mu, 2)
# table1.NLB$lambda <- round(table1.NLB$lambda, 3)
# table1.NLB$K      <- round(table1.NLB$K, 3)
```

```r
plot(table1$mn ~ table1.NLB$mn)
abline(a = 0, b = 1)

plot(table1$l ~ table1.NLB$l)
abline(a = 0, b = 1)
cor(table1$l , table1.NLB$l)

plot(table1$u ~ table1.NLB$mu,
     xlab = "NLB mu (u)",
     ylab = "Original mu (u)")
abline(a = 0, b = 1)
cor(table1$u , table1.NLB$mu)
cbind(table1$mn,table1$u , table1.NLB$mu)
table1$u - table1.NLB$mu





plot(table1$lambda ~ table1.NLB$lambda,
     xlab = "NLB lambda",
     ylab = "Original lambda")
abline(a = 0, b = 1)
cor(table1$lambda , table1.NLB$lambda)
```

```r
cbind(table1$mn,table1$lambda , table1.NLB$lambda)
table1$lambda - table1.NLB$lambda


plot(table1$K ~ table1.NLB$K,
     xlab = "NLB K",
     ylab = "Original K")
abline(a = 0, b = 1)
cor(table1$K , table1.NLB$K)
cbind(table1$mn,table1$K, table1.NLB$K)
table1$K - table1.NLB$K
```

```r
plot(table1$lambda ~ table1$u, main = "Original")
plot(table1.NLB$lambda ~ table1.NLB$mu, main = "NLB")


plot(table1$l ~ table1$u)
plot(table1.NLB$l ~ table1.NLB$mu)


plot(table1$u ~ table1$K)
plot(table1.NLB$mu ~ table1.NLB$K)
```

```r
summary(lm(table1$u ~ log(table1$mn)))
coefs <- coef(lm(table1$u ~ table1$ln.nm))
lambda.regression <- 1/coefs[2]
exp(coefs[1]*lambda.regression)
```

```r
plot(table1.NLB$mu ~ log(table1.NLB$mn))
cor(table1.NLB$mu , log(table1.NLB$mn))
summary(lm(table1.NLB$mu ~ log(table1.NLB$mn)))
coefs <- coef(lm(table1.NLB$mu ~ table1.NLB$ln.mn))

lambda.regression <- 1/coefs[2]
exp(coefs[1]*lambda.regression)
```

I'm curious how long this will take, so I'll record when the simulations stop

```r
sim.stop <- Sys.time()

sim.start
sim.stop
```

Method of moments versus MLE

The original paper used the "method of moments" statistical approach to estimate the parameters. The extRemes package do this method, but EnvStats does. Both EnvStats and extReme do maximum likelihood estimation (MLE), so I can confirm that my results are dependent on the package I use, which is a good idea since estimating parameters of extreme value distributions is not an everyday statistical task and - most importantly - one I am not very familiar with.

```r
#install.packages("EnvStats")
library(EnvStats)
library(extRemes)
```

```r

mme.191  <- eevd(random.scores.191$score.i, method = "mme")
mmue.191 <- eevd(random.scores.191$score.i, method = "mmue")
pwme.191 <- eevd(random.scores.191$score.i, method = "pwme")

fevd.191 <- fit.gumbel.191 <- fevd(random.scores.191$score.i,
                   type = "Gumbel",
                   method = "MLE")
```

How many replications needed?

```r
n.rows <- nrow(random.scores.191)
cummulative.params <- data.frame(i = 1:n.rows,
                                 loc = rep(NA,n.rows),
                                 scale = rep(NA,n.rows))

i.s <- seq(from = 100, to = nrow(random.scores.191),by = 100)

for(i in i.s){
  fit.gumbel.191 <- fevd(random.scores.191$score.i[1:i],
                   type = "Gumbel",
                   method = "MLE")

loc.param.191.i <- fit.gumbel.191$results$par[1]
scale.param.191.i  <- 1/fit.gumbel.191$results$par[2]

cummulative.params$loc[i] <- loc.param.191.i
cummulative.params$scale[i] <- scale.param.191.i
}

plot(loc ~ i, data = cummulative.params)
abline(h = 26.45)
abline(v = 10000)

plot(scale ~ i, data = cummulative.params)
abline(h = 0.298)
abline(v = 10000)

write.csv(cummulative.params, file = "cumulative_params.csv")
```

How many replications needed?

```r
cummulative.params <- data.frame(i = 1:n.rows,
                                 loc = rep(NA,n.rows),
                                 scale = rep(NA,n.rows))
for(i in 100:nrow(random.scores.2981)){
  fit.gumbel.2981 <- fevd(random.scores.2981$score.i[1:i],
                   type = "Gumbel",
                   method = "MLE")

loc.param.2981.i <- fit.gumbel.2981$results$par[1]
scale.param.2981.i  <- 1/fit.gumbel.2981$results$par[2]

cummulative.params$loc[i] <- loc.param.2981.i
cummulative.params$scale[i] <- scale.param.2981.i
}

plot(loc ~ i, data = cummulative.params)
abline(h = 26.45)

plot(scale ~ i, data = cummulative.params)
abline(h = 0.298)

write.csv(cummulative.params, file = "cumulative_params.csv")
```