[Re] Local alignment statistics - Compare to original table

Nathan Brouwer

10/29/2019

Introduction

Every run of a simulation will present different results. Also, I am coding up my version of this simulation based just on the original article. In the following code I generate plots telling me about how close I got to reproducing the original paper and whether 10000 simulations is enough to get a stable estimate of lambda and mu. Also, I used a different statistical approach

Load data

Load original version of the table

```
table1 <- read.csv(file = "table1.csv")
```

Load my version of the table

```
table1.NLB <- read.csv(file = "table1NLB.csv")
```

Load raw data from simulatoins

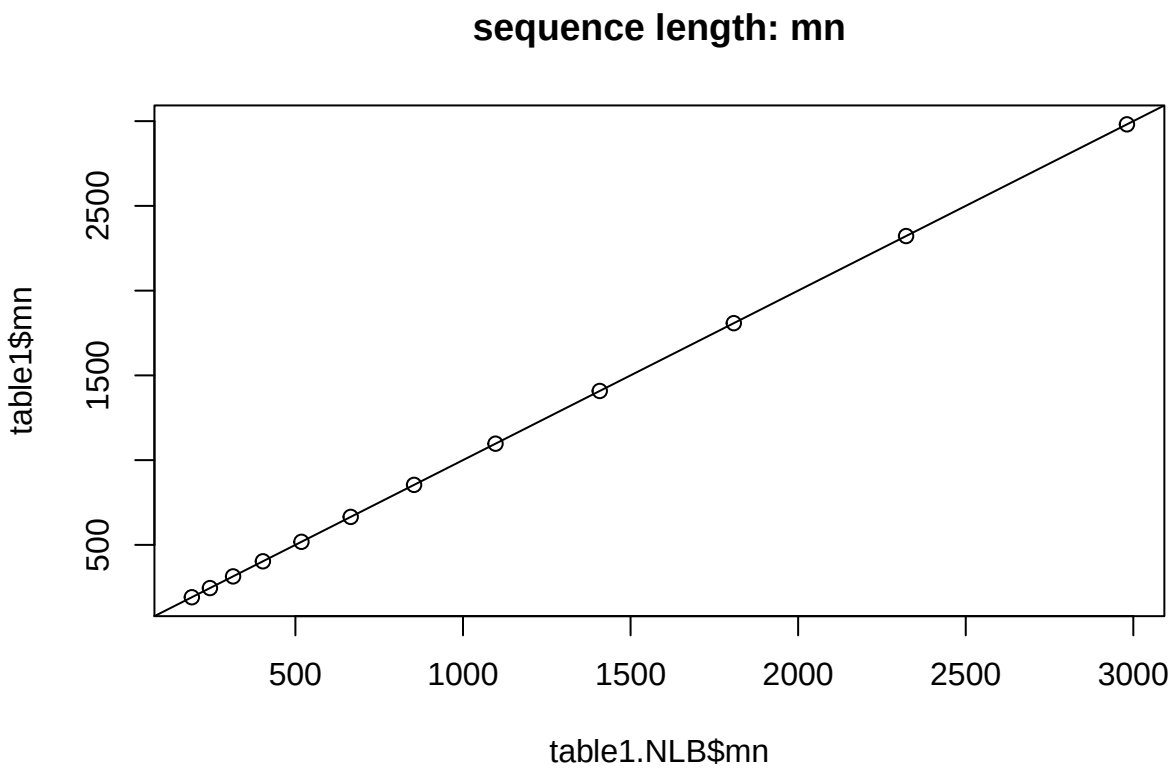
```
load("~/1_R/git/blaststats/full_experiment.RData")
```

Make diagnostic plots

Plot the sequence lengths (m,n). THis better work! But always good to make sure a silly typo didn't enter in..

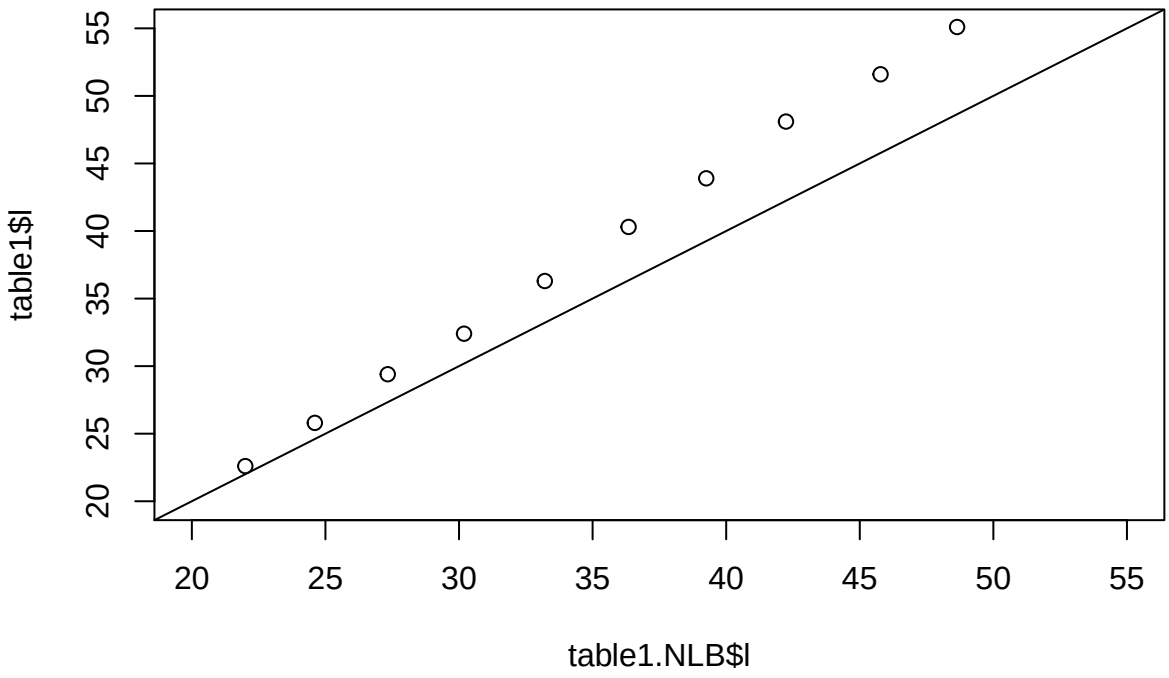
The diagonal line is called a 1:1 and indicates a perfect correlation.

```
plot(table1$mn ~ table1.NLB$mn, main = "sequence length: mn")
abline(a = 0, b = 1)
```



Plot *alignment* length. This is troubling! Even though highly correlated the points don't fall on the line. WOrse, there's **drift**, with the mean lenghts of the sequences being close on the short end but the difference grows.

```
plot(table1$l ~ table1.NLB$l, xlim = c(20,55),ylim=c(20,55))
abline(a = 0, b = 1)
```

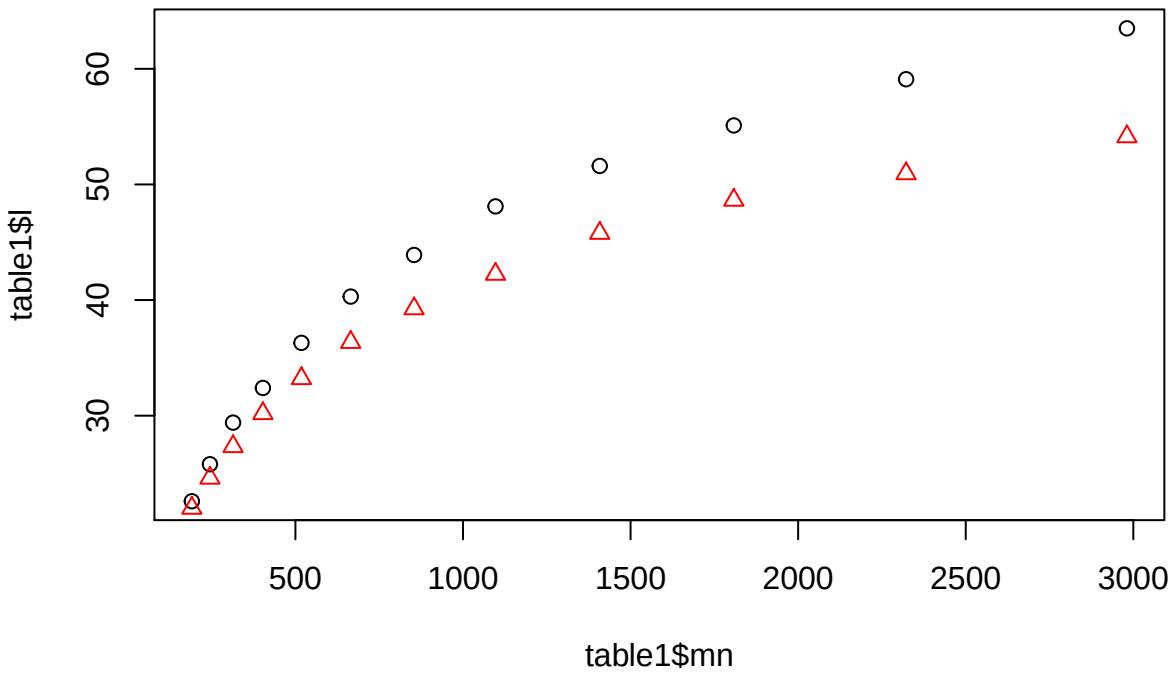



```
cor(table1$l , table1.NLB$l)
```

```
## [1] 0.9994748
```

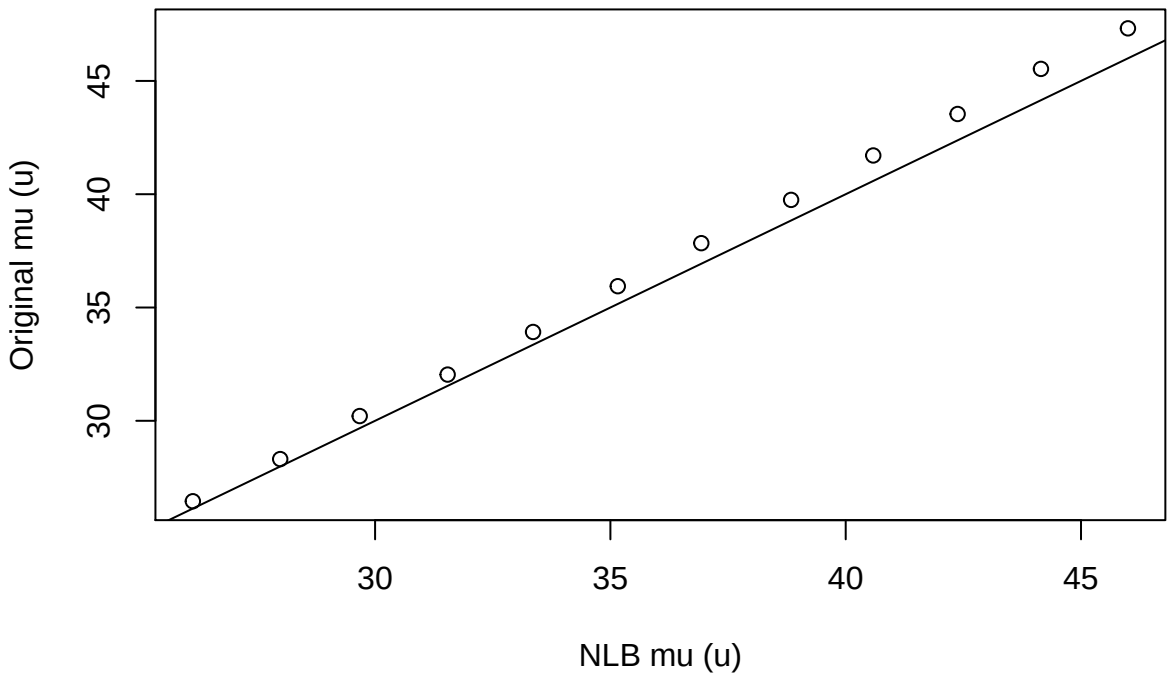
Another way to look at this is to plot sequence length against alignment length. Both of these plots indicate that they were able to build alignments that were somewhat longer than those which I'm building. I'm not sure why this is!

```
#plot original data - black circles
plot(table1$l ~ table1$mn)

#plot my data - read triangels
points(table1.NLB$l ~ table1.NLB$mn, col = 2, pch =2)
```



Compare the other paramters. First mu (I call it mu, on their table it u). Same thing, there's drift.

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



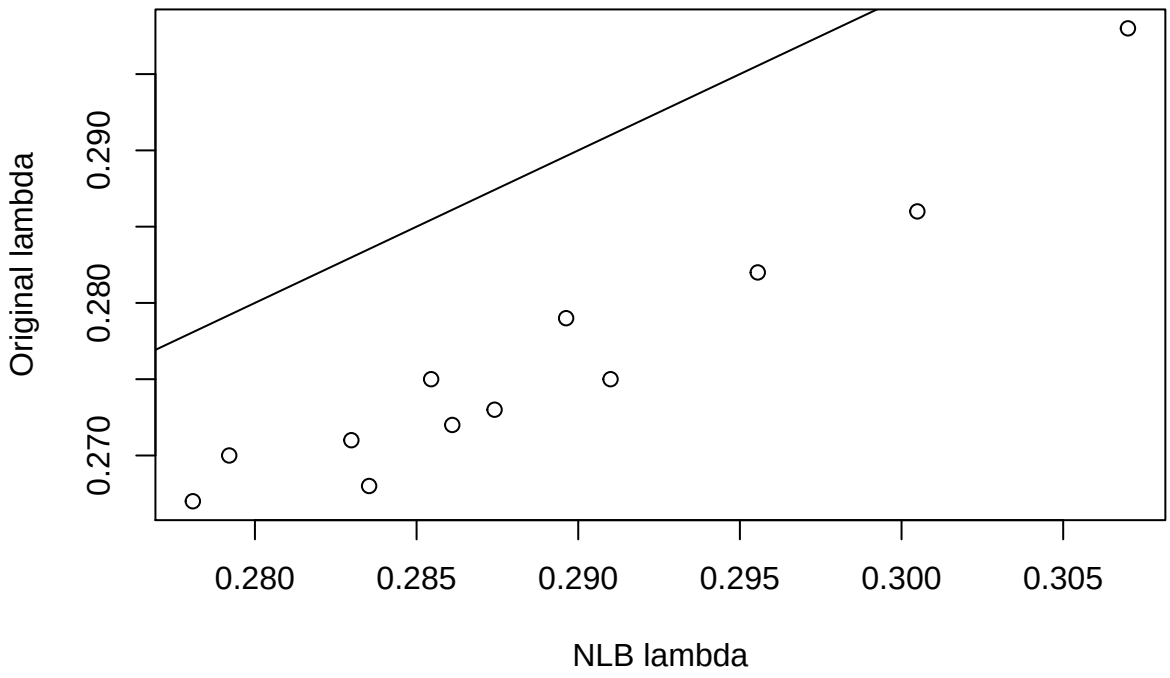
I'll buid a quick table to compare the values. There's between a 1 and 3% differene. Not horrible, but not sure why its happening.

```
cbind(table1$mn,
      table1$u ,
      table1.NLB$mu,
      (table1$u - table1.NLB$mu)/table1$u*100)
```

```
##       [,1]  [,2]     [,3]     [,4]
##  [1,]  191 26.45 26.12708 1.220866
##  [2,]  245 28.31 27.98365 1.152786
##  [3,]  314 30.21 29.67285 1.778039
##  [4,]  403 32.04 31.54146 1.555988
##  [5,]  518 33.92 33.35689 1.660106
##  [6,]  665 35.94 35.15688 2.178974
##  [7,]  854 37.84 36.93168 2.400417
##  [8,] 1097 39.75 38.84004 2.289214
##  [9,] 1408 41.71 40.58527 2.696556
## [10,] 1808 43.54 42.37834 2.668034
## [11,] 2322 45.53 44.14878 3.033647
## [12,] 2981 47.32 45.99767 2.794443
```

Now lambda - this is kinda ugly!

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



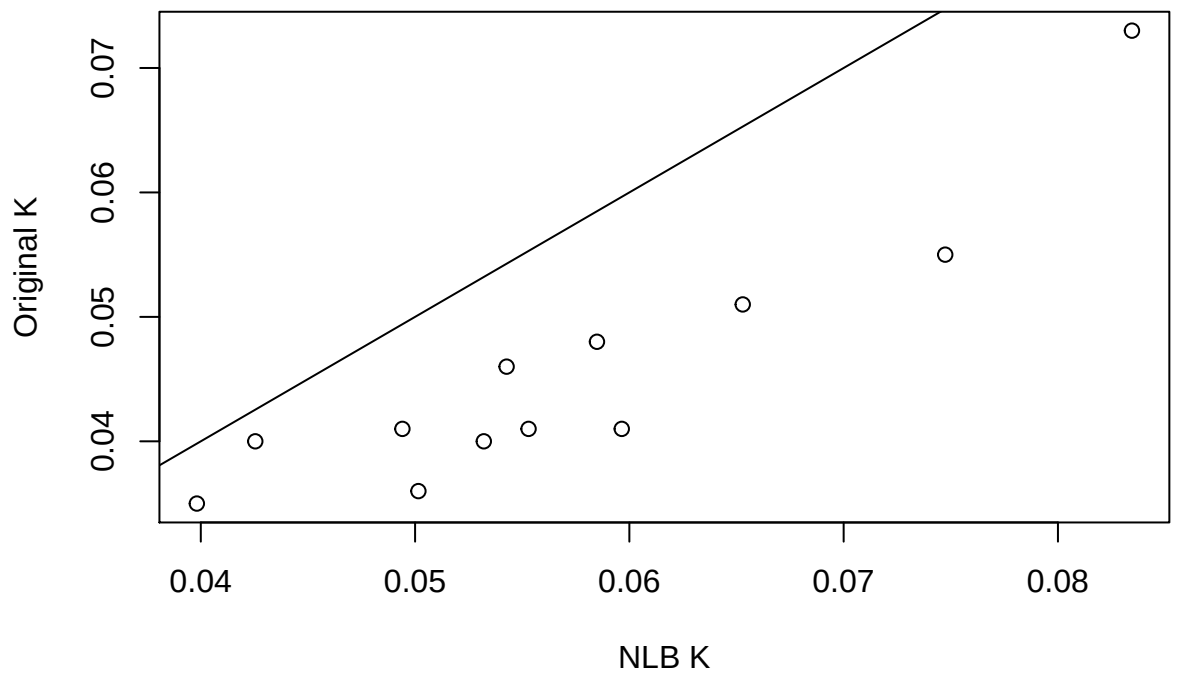

```
cor(table1$lambda , table1.NLB$lambda)
cbind(table1$mn,table1$lambda , table1.NLB$lambda)
table1$lambda - table1.NLB$lambda
```

```
## [1] 0.9610492
##       [,1]  [,2]      [,3]
##  [1,]  191 0.298 0.3070028
##  [2,]  245 0.286 0.3004887
##  [3,]  314 0.282 0.2955524
##  [4,]  403 0.275 0.2909991
##  [5,]  518 0.279 0.2896259
##  [6,]  665 0.273 0.2874119
##  [7,]  854 0.272 0.2861039
##  [8,] 1097 0.275 0.2854506
##  [9,] 1408 0.268 0.2835316
## [10,] 1808 0.271 0.2829821
## [11,] 2322 0.267 0.2780796
## [12,] 2981 0.270 0.2792049
##  [1] -0.009002775 -0.014488661 -0.013552359 -0.015999118 -0.010625867
##  [6] -0.014411887 -0.014103860 -0.010450583 -0.015531567 -0.011982117
## [11] -0.011079607 -0.009204876
```

K is pretty bad too.

```
plot(table1$K ~ table1.NLB$K,
     xlab = "NLB K",
     ylab = "Original K")
```

```r
abline(a = 0, b = 1)
```




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

```
## [1] 0.9151261
##       [,1]  [,2]       [,3]
##  [1,]  191 0.073 0.08345393
##  [2,]  245 0.055 0.07473939
##  [3,]  314 0.051 0.06529082
##  [4,]  403 0.041 0.05964555
##  [5,]  518 0.048 0.05848812
##  [6,]  665 0.041 0.05529516
##  [7,]  854 0.040 0.05320699
##  [8,] 1097 0.046 0.05427142
##  [9,] 1408 0.036 0.05015451
## [10,] 1808 0.041 0.04940808
## [11,] 2322 0.035 0.03981575
## [12,] 2981 0.040 0.04254238
##  [1] -0.010453931 -0.019739391 -0.014290823 -0.018645554 -0.010488116
##  [6] -0.014295165 -0.013206989 -0.008271416 -0.014154507 -0.008408083
## [11] -0.004815749 -0.002542378
```

Upshot - somethings wrong. I need to check that errors in older versions that I identified actually got fixed!.

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 not. 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.

THe follwoing code fits a bunch of variations in the estimatation method, both MLE from two

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

```
##
## Attaching package: 'EnvStats'
```

```
## The following objects are masked from 'package:stats':
##
##     predict, predict.lm
```

```
## The following object is masked from 'package:base':
##
##     print.default
```

```
library(extRemes)
```

```
## Loading required package: Lmoments
```

```
## Loading required package: distillery
```

```
##
## Attaching package: 'extRemes'
```

```
## The following objects are masked from 'package:EnvStats':
##
##     devd, pevd, qevd, revd
```

```
## The following objects are masked from 'package:stats':
##
##     qqnorm, qqplot
```

```
# methods of moments - EnvStats
mme.191  <- eevd(random.scores.191$score.i, method = "mme")  # methods of moments
mmue.191 <- eevd(random.scores.191$score.i, method = "mmue") # method of moments, unbiased estimator of
pwme.191.a <- eevd(random.scores.191$score.i, method = "pwme") # method of moments, probability-weighte
pwme.191.b <- eevd(random.scores.191$score.i, method = "pwme",
                   pwme.method = "plotting.position") # method of moments, probability-weighted vs 2

# max like - EnvStats
mle.191  <- eevd(random.scores.191$score.i, method = "mle")  # mle

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

I'll eyeball the comparison between statistcal methods. They are very close for mu but scale (1/lambda) varys a lot more. There defintely appears to be a difference in between method of moments and max like.

```r
rbind(eevd.mme  = mme.191$parameters,
      eevd.mmue = mmue.191$parameters,
      eevd.pwme.a = pwme.191.a$parameters,
      eevd.pwme.b = pwme.191.b$parameters,
      eevd.mle = mle.191$parameters,
      fevd.mle = fevd.191$results$par)
```

```
##             location    scale
## eevd.mme    26.11266 3.303690
## eevd.mmue   26.11256 3.303855
## eevd.pwme.a 26.12827 3.276643
## eevd.pwme.b 26.12776 3.277528
## eevd.mle    26.12708 3.257298
## fevd.mle    26.12708 3.257300
```

How many replications needed?

While not relevant to understanding why my results are a bit off, its important to think about how many iterations of the experiment are necessary to get good estimtes of mu, lambda and k. As is, it take about 5 hours to run all these simulations. Maybe its possible to use fewer.

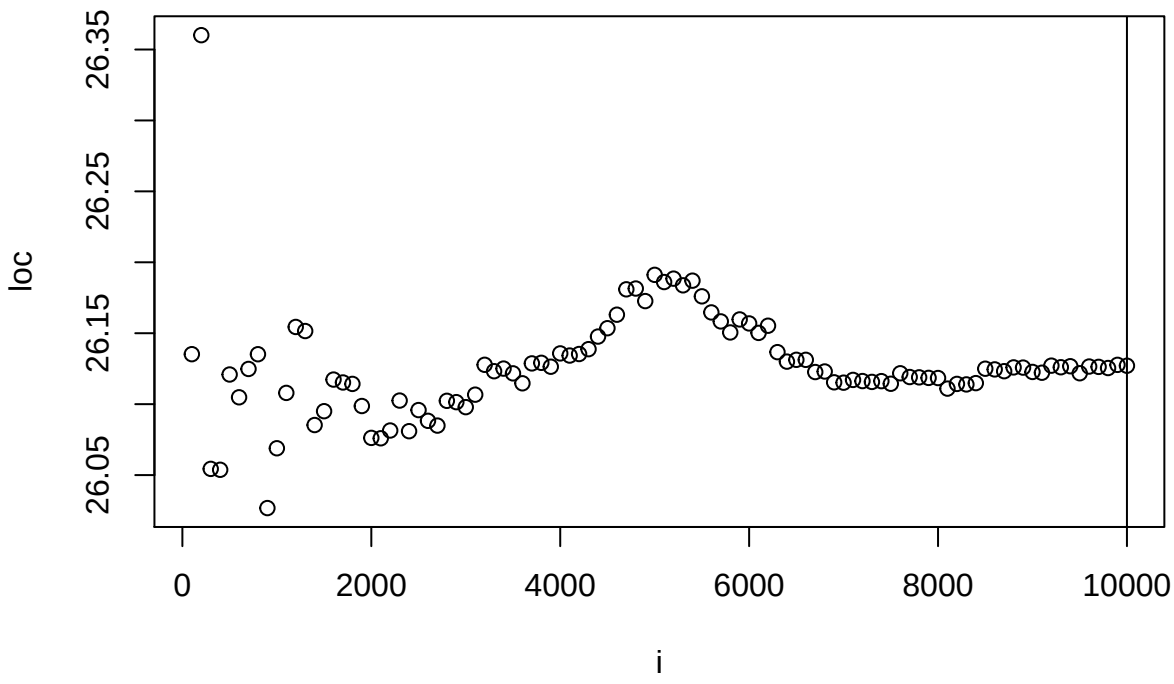
In the code take the data for m = n = 191 and fit an extreme value distribution model at a range of sample sizes, from 100 to 10,000

```r
# how many rows in output?
n.rows <- nrow(random.scores.191)


# sequence of values - I don't want to do this for all 10,000 rows
## I'll go by multiples of 100
i.s <- seq(from = 100, to = nrow(random.scores.191),by = 100)

#data frame for storage
cummulative.params <- data.frame(i = i.s,
                                 loc = rep(NA,length(i.s)),
                                 scale = rep(NA,length(i.s)))

for(i in 1:length(i.s)){
  i.s.working <- i.s[i]
  fit.gumbel.191 <- fevd(random.scores.191$score.i[1:i.s.working],
                   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 location parameter againts the number of samples. It gets pretty level by 10, though there does appear to be some drift upward.

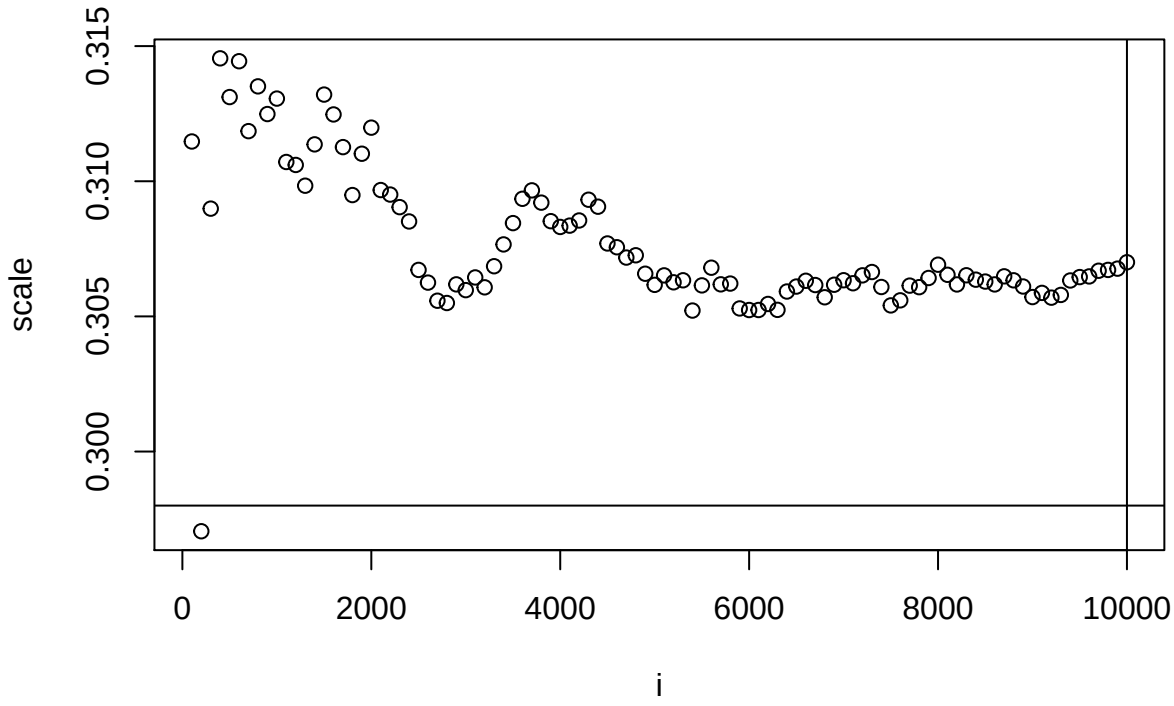
```r
plot(loc ~ i, data = cummulative.params)
abline(h = 26.45)
```

```
abline(v = 10000)
```



Now the scale parameter. This remains noisier

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



How many replications needed? m = n = 2981

```
# how many rows in output?
n.rows <- nrow(random.scores.2981)


# sequence of values - I don't want to do this for all 10,000 rows
## I'll go by multiples of 100
i.s <- seq(from = 100, to = nrow(random.scores.2981),by = 100)

#data frame for storage
cummulative.params <- data.frame(i = i.s,
                                 loc = rep(NA,length(i.s)),
                                 scale = rep(NA,length(i.s)))

for(i in 1:length(i.s)){
  i.s.working <- i.s[i]
  fit.gumbel.191 <- fevd(random.scores.2981$score.i[1:i.s.working],
                   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]

```

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


par(mfrow = c(2,1))
plot(loc ~ i, data = cummulative.params)
abline(h = 26.45)

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