Population Genetics: Assignment 2

University of Cambridge

Henrik Åhl

April 24, 2017

Preface

This is an assignment report in connection to the *Population Genetics* module in the Computational Biology course at the University of Cambridge, Lent term 2017. All related code is as of April 24, 2017 available through a Github repository by contacting hpa22@cam.ac.uk.

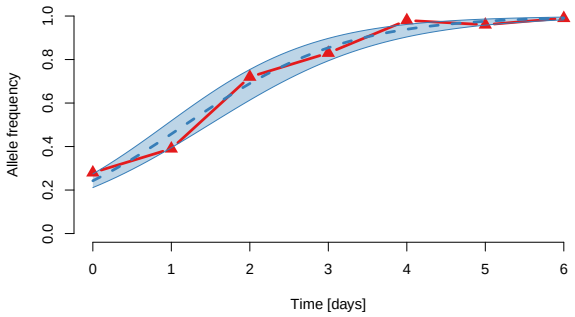

Figure 1: Allele development as given by data (red) and model. The shaded area denotes the 95 % confidence interval of the optimisation.

Exercises

1 Under the population size of $N \sim 10^7$, we get that the variance will be relatively low, and therefore not be a determining factor for the development of the allele frequency. We thus get the equation

$$q(t) = q_0 \frac{e^{\sigma t}}{1 - q_0 + q_0 e^{\sigma t}}$$

as an estimator for the development of q. Our likelihood function for a model $\mathcal{M}$ can then be describes as

$$\mathcal{L} = \sum_k \log \frac{N!}{n_A(t_k)! n_a(t_k)!} q_A(t_k)^{n_A(t_k)} q_a(t_k)^{n_a(t_k)},$$

where the n's are our observations, and the q's our model predictions, compared to some number of total observations N, which in our case is the number of reads. We optimise the log-likelihood using L-BFGS-B optimisation the `R` function `stats4::mle`, having set q bounded at 0 and 1.

The results of the optimisation can be seen in fig. 1, where the shaded area signifies the range of the standard error. Compare this to our found optimal values of $q_0 = 0.24 \pm 0.03$ and $\sigma = 0.97 \pm 0.08$

Figure 2 shows the negative log-likelihood surface for σ and q. Due to the growth rate of the log-likelihood, some configurations give rise to numerical errors, as shown by the white surface in the figure. Note in particular the darker shaded area, which signifies the most likely parameter configurations.

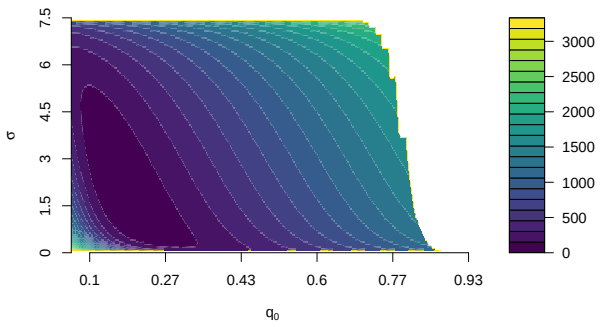

Figure 2: Contour plot of the negative log-likelihood surface for different parameter configurations. Note how our optimised parameters indeed fall inside the optimum.

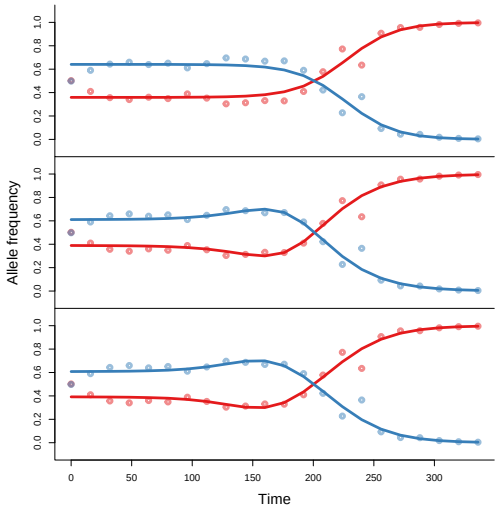

Figure 3: Fitted curves under models with one, two, or three mutations respectively.

2 We download the *Optimist* package according to instructions. For all of our investigations, we use a seed value of 2. For our simulations with this, we get a log-likelihood of -92.5714, without any variance.

When instead perform a more thorough optimisation, we get the results seen in fig. 3 for our three cases. We can visually note that two and three mutations seem to fit better with the data. When observing the average log-likelihoods, we can also note that increasing from two to three mutations does not seem to produce better results, hinting at redundancy in the number of free parameters. Looking at the AIC and BIC scores in fig. 4, we see that two beneficial mutations indeed give us the best model for the data.

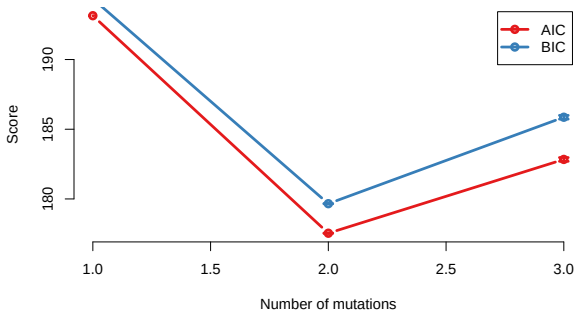

Figure 4: AIC and BIC scores, with errorbars corresponding to the standard error, for the 10 optimisations. Two mutations consistently performs the best, with significance.

However, we must be aware of that because we have such little data, with few data points that have been manually acquired. We therefore have a high uncertainty in the data itself that must be considered. Possibly, this is what causes the majority of our observed variance, since the population size suggests that the fluctuations should be minor. With the error present, it might prove useful to fit a model with drift in order to account for the additional variability. Directly attempting to attain estimates of the acquisition errors would of course also be preferable, in case there is no possibility of performing repeated trials.

Antibiotic resistance can the achieved through many different ways, and depends largely on the type of organism, the population size, and the generation length. One example could be the accumulation of single-nucleotide polymorphisms which in various ways can affect the function of the antibiotic. A common inactivation method for several antibiotics is enzymatic inactivation through hydrolysis, which can occur due to SNP's increasing the likelihood of such events taking place [1]. Several antibiotics, such as quinolone cipofloxacin, targets parts of the bacterial genome itself in order to prevent replication. Like an-

tibiotics themselves can be target indirectly through mutations, resistance can also arise due to mutations altering the targets of the antibiotics [2]. A third option for mutations to be beneficial is by having them alter the membrane selectivity such that the antibiotics are unable to reach their molecular targets [3].

All these changes induce a selective pressure towards keeping the mutation within the population. These can thereafter progress through direct inheritance, as well as thorugh horizontal transfer. It might also be that the antibiotics wipe out most of the population aside for the select few who have acquired beneficial mutations, whereafter these instead can replicate successfully and establish a completely new population consisting primarily of resistant cells [1].

3 Using the data provided, we get the resulting values of $D = -0.12, n_{abba} = 25$ and $n_{baba} = 32$. We clearly have a slightly higher number of `ABBA` than `BABA`, and therefore some inferred introgression between `A` and `C`.

Calculating a p-value for the D-value observed, we get a result of 0.43, saying we cannot reject the null-hypothesis. However, this assumes that the sites are independent of each other, which we know they are not, and we must go to additional lengths to ensure ourselves of the significance. In order to do this, we calculate the allele frequencies, whose first few rows indeed correspond with the example supplied, apart from an apparent typo in the instructions.

When instead using the allele frequencies, we now get a result of $D = -0.19$, i.e. higher than before now that more data is incorporated. Likewise, our p-value is significantly lower, reaching $p = 0.0017278$ In this scenario, our results are indeed significant, and we can reject the null-hypothesis.

Given the figure for the linkage disequilibirium between populations `A` and `B`, we see that there is indeed significant short-range interactions in both of the populations. We look at the distributions between minimum distances for our SNPs, we get fig. 5 and see that the distribution pans of quickly after a distance of ca 20000. Most SNPs are thus in this high-correlated zone that we see before the bend in our instruction figure.

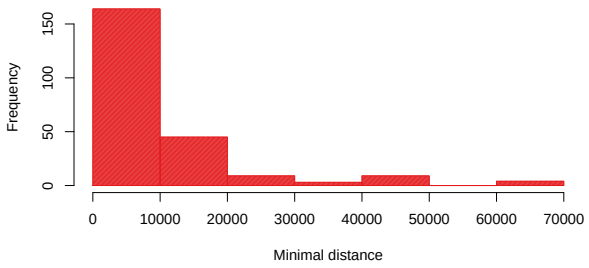

Figure 5: Minimum neighbour distance distribution. Apparently, most SNPs are within 20000-30000 basepairs of their closest neighbour.

We want too choose the jackknife when the correlation has panned out, and thus take it to be at a distance of 30000. Separating our data into chunks of this size and perform the analysis, we now get a p-value of 0.0062384, slightly worse than the leave-one-out cross-validation case.

In the last figure of the assignment, population `B` appears to be affiliated with a slightly higher short-interaction density, and lower intermediate-interaction density. Since we know that populations which diverged long ago should display high short-range correlation, we can infer that `B` diverged from `C` earlier than `A`. Comparing to above, where we note introgression between `A` and `C`, the results can indeed be said to be consistent.

References

[1] JL Martinez and F Baquero. Mutation frequencies and antibiotic resistance. *Antimicrobial agents and chemotherapy*, 44(7):1771–1777, 2000.

[2] Que Chi Truong, JC Nguyen Van, David Shlaes, Laurent Gutmann, and Nicole J Moreau. A novel, double mutation in dna gyrase a of escherichia coli conferring resistance to quinolone antibiotics. *Antimicrobial Agents and Chemotherapy*, 41(1):85–90, 1997.

[3] Anne H Delcour. Outer membrane permeability and antibiotic resistance. *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics*, 1794(5): 808–816, 2009.

A Code

```r
setwd("~/compbio/src/assignments/pga2/code/")
library(RColorBrewer)
palette(brewer.pal(n = 8, name = "Set1"))
library(scales)
library(stats4)
library(viridis)

# Parameters / data
N = 100
timepoints = 0:6
data = data.frame(time = 0:6, n = c(28, 39, 72, 83, 98, 96, 99) / N)
x = data[, "time"]
xs = seq(0, 6, 0.1)

# Fraction development over time
q = function(q.init, sigma, t) {
  q.init * exp(sigma * t) / (1 - q.init + q.init * exp(sigma * t))
}

# Calculate the log-likelihood
nas = data[,"n"]*N
prefact = factorial(N)/(sapply(nas, factorial)*sapply(N-nas, factorial))
log.lik = function(q.init, sigma){
  qas = q(q.init, sigma, timepoints)
  # nas = qas*N
  ln.p = log(prefact*qas**(nas)*((1-qas)**(N-nas)))
  -sum(ln.p)
}

fit = mle(minuslogl = log.lik, start = list(q.init = .5, sigma = .5), method="L-BFGS-B",
    lower = 0, upper=c(1, Inf))
coef = coef(summary(fit))

# Calculate upper and lower bound of estimate
yhigh = q(coef[1, 1] + coef[1, 2], coef[2, 1] + coef[2, 2], xs)
ylow = q(coef[1, 1] - coef[1, 2], coef[2, 1] - coef[2, 2], xs)

# Plot it all
plot(1, type="n", axes=T, xlab="Time [days]", ylab="Allele frequency", ylim=c(0,100/N), xlim
    = c(0,6), bty = "n")
polygon(c(xs, rev(xs)), c(yhigh, rev(ylow)),
          col = alpha(2, .33), border = F, lty = 2, lwd = 2)
lines(x, data[,"n"], type = "b", pch = 17, cex = 1.5, lwd = 3, col = 1)
lines(xs, q(coef[1, 1], coef[2, 1], xs), col = 2, lwd = 3, lty = 2)

# Plot borders
lines(xs, ylow, col = alpha(2, 1), lwd = 1)
lines(xs, yhigh, col = alpha(2, 1), lwd = 1)

#########################################################################
### Image plot
#########################################################################

qvals = seq(0, 1, 0.001)
svals = seq(0, 7.5, 0.05)

likelihoods = sapply(qvals, function(x) sapply(svals, function(y) log.lik(x, y)))

filled.contour((likelihoods),   xlab = expression(q[0]),   ylab = expression(sigma),
               col = viridis(17),
               axes = F
)
axis(1, at = seq(0, .75, 0.15), labels = round(seq(.1, 1, 1/.75 * .25/2), 2))
axis(2, at = seq(0, 1, 0.2), labels = seq(0, 7.5, 7.5 * 0.2))
axis(4, at = seq(0, .9, 0.15), labels = seq(0, 3000, 500), las =2, line= -2)
```

```
64
65
66 ##################################################
67 ### SIMULATED ANNEALING PART
68 ##################################################
69 # Parameters
70 # generations = 300000
71 # temp        = 1e1
72 # temp.rate   = 1 - 1e-4
73 #
74 # change.state = function(q.init, sigma){
75 #   # if(runif(1) > .5){
76 #     q.init = q.init + rnorm(1, sd = 0.025)
77 #     q.init = max(q.init, 0)
78 #     q.init = min(q.init, 1)
79 #   # }else{
80 #     sigma = sigma + rnorm(1, sd = 0.025)
81 #     sigma = max(sigma, 0)
82 #     sigma = min(sigma, 1)
83 #   # }
84 #   list(q=q.init, s=sigma, ll=log.lik(q.init, sigma))
85 # }
86 #
87 # state = change.state(.5, .5)
88 # for(iter in 1:generations){
89 #   # Suggest update
90 #   new.state = change.state(state[[1]], state[[2]])
91 #
92 #   # Is it better?
93 #   if (new.state[[3]] <= state[[3]]) {
94 #     state = new.state
95 #   }
96 #   # Should we still go there?
97 #   else if (runif(1) < exp((state[[3]] - new.state[[3]]) / temp)) {
98 #     state = new.state
99 #   }
100 #   # Change temperature
101 #   temp = temp * temp.rate
102 #
103 #   # Print current result
104 #   if (iter %% 10000 == 0){
105 #     cat(iter, "\t", temp, "\t", state[[1]], "\t", state[[2]], "\t",  state[[3]], "\n")
106 #   }
107 # }
```

../code/exc1.R

```
1 setwd("~/compbio/src/assignments/pga2/code/")
2 .libPaths("/home/henrik/R/x86_64-pc-linux-gnu-library/3.3/")
3 library("RColorBrewer")
4 library(scales) #imports alpha
5 palette(brewer.pal(n = 8, name = "Set1"))
6 line.size = 3
7 lw.s = 3
8
9
10 # 1) ./analyse_first_multi <seed> <no mut> <no optimis> <input file>
11 # setwd("~/.local/optimist")
12 # no.mut = 1
13 # system(paste("./analyse_first_multi 2", no.mut, "10 Colour_freq.out"))
14 # system("rm Transfer_data.out")
15 # system(paste0("head -n ", 5 + no.mut, " Transfer_data_multi.out > Transfer_data.out"))
16 # system("./analyse_sys_multi 2 Colour_freq.out Transfer_data.out")
17 # system("cp Real_frequencies.out ~/compbio/src/assignments/pga2/data/Real_frequencies1.out")
18 # system("cp Model_frequencies.out ~/compbio/src/assignments/pga2/data/Model_frequencies1.out
     ")
19
20 # no.mut = 2
```

```
# system(paste("./analyse_first_multi 2", no.mut, "10 Colour_freq.out"))
# system("rm Transfer_data.out")
# system(paste0("head -n ", 5 + no.mut, " Transfer_data_multi.out > Transfer_data.out"))
# system("./analyse_sys_multi 2 Colour_freq.out Transfer_data.out")
# system("cp Real_frequencies.out ~/compbio/src/assignments/pga2/data/Real_frequencies2.out")
# system("cp Model_frequencies.out ~/compbio/src/assignments/pga2/data/Model_frequencies2.out
    ")
#
## 1) ./analyse_first_multi <seed> <no mut> <no optimis> <input file>
# no.mut = 3
# system(paste("./analyse_first_multi 2", no.mut, "10 Colour_freq.out"))
# system("rm Transfer_data.out")
# system(paste0("head -n ", 5 + no.mut, " Transfer_data_multi.out > Transfer_data.out"))
# system("./analyse_sys_multi 2 Colour_freq.out Transfer_data.out")
# system("cp Real_frequencies.out ~/compbio/src/assignments/pga2/data/Real_frequencies3.out")
# system("cp Model_frequencies.out ~/compbio/src/assignments/pga2/data/Model_frequencies3.out
    ")
# setwd("~/compbio/src/assignments/pga2/code/")

# ######################################
# ### PLOT
# ######################################
#
d1 = read.table("../data/Model_frequencies1.out")
d2 = read.table("../data/Real_frequencies1.out")
d3 = read.table("../data/Model_frequencies2.out")
d4 = read.table("../data/Real_frequencies2.out")
d5 = read.table("../data/Model_frequencies3.out")
d6 = read.table("../data/Real_frequencies3.out")
#

# Log-likelihoods produced.
L1 = c(
    -92.5714,-92.5714,-92.5714,-92.5714,-92.5714,-92.5714,-92.5714,-92.5714,-92.5714,-92.5714)

L2 = c
    (-81.7473,-81.7689,-81.75,-81.7561,-81.8105,-81.7567,-81.7759,-81.7451,-81.7841,-81.8126
    )
L3 = c
    (-81.1746,-81.7301,-81.0976,-81.4593,-81.3007,-81.3259,-81.7453,-81.356,-81.5501,-81.4329)


AIC = function(params, log.L){
  2*params - 2*log.L
}

BIC = function(n, k, log.L){
  log(n)*k - 2*log.L
}

Ls = rbind(L1, L2, L3)
means  = apply(Ls, 1, mean)
stds = apply(Ls, 1, function(x) sd(x)/sqrt(length(x)))

aics = AIC(1:3, means)
bics = BIC(10, 1:3, means)
aics.err.up = AIC(1:3, means + stds)
aics.err.do = AIC(1:3, means - stds)
bics.err.up = BIC(10, 1:3, means + stds)
bics.err.do = BIC(10, 1:3, means - stds)

# PLOT
par(mfrow=c(3,1))
plot(d1[,1], d1[,2], col = 1, type = "l", pch = 16, ylim=0:1, lwd = line.size, lty = 1)
points(d1[,1], d1[,3], col = 2, type = "l", pch = 17, lwd = line.size, lty = 1)
lines(d2[,1], d2[,2], lwd = line.size, type = "p", col = alpha(1, .5))
lines(d2[,1], d2[,3], lwd = line.size, type = "p", col = alpha(2, .5))
```

```
80
81 plot(d3[,1], d3[,2], col = 1, type = "l", pch = 16, ylim=0:1, lwd = line.size, lty = 1)
82 points(d3[,1], d3[,3], col = 2, type = "l", pch = 17, lwd = line.size, lty = 1)
83 lines(d4[,1], d4[,2], lwd = line.size, type = "p", col = alpha(1, .5))
84 lines(d4[,1], d4[,3], lwd = line.size, type = "p", col = alpha(2, .5))
85
86 plot(d5[,1], d5[,2], col = 1, type = "l", pch = 16, ylim=0:1, lwd = line.size, lty = 1)
87 points(d5[,1], d5[,3], col = 2, type = "l", pch = 17, lwd = line.size, lty = 1)
88 lines(d6[,1], d6[,2], lwd = line.size, type = "p", col = alpha(1, .5))
89 lines(d6[,1], d6[,3], lwd = line.size, type = "p", col = alpha(2, .5))
90
91
92
93 par(mfrow=c(1,1))
94 plot(1:3, aics, col = 1, type = "b", lwd = lw.s, xlab = "", ylab = "")
95 lines(1:3, bics, col = 2, type = "b", lwd = lw.s)
96 arrows(1:3, aics.err.up, 1:3, aics.err.do, length = 0.05, angle = 90, code = 3, col = "1",
       lwd = lw.s)
97 arrows(1:3, bics.err.up, 1:3, bics.err.do, length = 0.05, angle = 90, code = 3, col = "2",
       lwd = lw.s)
98 mtext(side=2, line = 3, las = 3, "Score")
99 mtext(side=1, line = 3, las = 1, "Number of mutations")
100 legend("topright", col =1:2, lty = 1, pch = 21, c("AIC", "BIC"), inset = c(0.02,0.02), lwd =
       lw.s)
101 # 2 beneficial mutations!
```

../code/exc2.R

```
1 setwd("~/compbio/src/assignments/pga2/code/")
2 library(RColorBrewer)
3 library(scales) #imports alpha
4 library(stats)
5 palette(brewer.pal(n = 8, name = "Set1"))
6 line.size = 2
7
8 data = read.csv("../data/four_population_sequencing_data.csv")
9 data.firstind = data[, which(grepl("0", colnames(data)))] # get first individual
10
11 get.D = function(test.data, frac = FALSE) {
12   if (!frac) {
13     n.abba = sum(apply(test.data, 1, function(x) x[1] == 0 &
14                          x[1] != x[2] &
15                          x[2] == x[3] &
16                          x[3] != x[4])) # 0 = ancestral
17     n.baba = sum(apply(test.data, 1, function(x) x[1] == 1 &
18                          x[1] != x[2] &
19                          x[2] != x[3] &
20                          x[3] != x[4]))
21     D = (n.abba - n.baba) / (n.abba + n.baba)
22   } else{
23     As = test.data[, which(grepl("A_", colnames(test.data)))]
24     Bs = test.data[, which(grepl("B_", colnames(test.data)))]
25     Cs = test.data[, which(grepl("C_", colnames(test.data)))]
26     Ds = test.data[, which(grepl("D_", colnames(test.data)))]
27     As = apply(As, 1, mean); Bs = apply(Bs, 1, mean); Cs = apply(Cs, 1, mean); Ds = Ds
28     ddd = cbind(As, Bs, Cs, Ds)
29     n1 = sum(apply(ddd, 1, function(x) (1 - x[1]) * x[2] * x[3] * (1 - x[4])))
30     n2 = sum(apply(ddd, 1, function(x) (x[1]) * (1 - x[2]) * x[3] * (1 - x[4])))
31     D = (n1 - n2) / (n1 + n2)
32   }
33   list(D=D, n.abba=n.abba, n.baba=n.baba)
34 }
35 first.data = get.D(data.firstind)
36 D = first.data[[1]]
37 n.abba = first.data[[2]]
38 n.baba = first.data[[3]]
39
40 abba.baba.test = function(data, n.folds = 10, frac = FALSE, Dval) {
```

```
  folds.i = sample(rep(1:n.folds, length.out = nrow(data)))
  Ds = vector(mode = "numeric", length = n.folds)
  for (ii in 1:n.folds) {
    test.i = which(folds.i == ii)
    train = data[-test.i,]
    Ds[ii] = get.D(train, frac)[[1]]
  }
  Z = Dval / sqrt(n.folds * var(Ds))
  p = 1 - pnorm(abs(Z))
  p
}

# Binom
p.binom = binom.test(n.abba, n.abba + n.baba, 0.5)

# Using fractions instead
D.frac = get.D(data, frac = TRUE)[[1]]
p = abba.baba.test(data.firstind, n.folds = nrow(data.firstind), frac = FALSE, Dval = D)
p.frac = abba.baba.test(data, n.folds = nrow(data), frac = TRUE, Dval = D.frac)

# Histogram
ll = c()
for(ii in 2:(nrow(data)-1)){
  ll = c(ll, min(data[ii,1] - data[ii-1, 1], data[ii+1,1] - data[ii,1]))
}
ll = c(ll, data[2,1] - data[1,1])
ll = c(ll, data[nrow(data),1] - data[nrow(data)-1,1])
hist(ll, breaks = 7, col = 1, density = 100, xlab = "Minimal distance", main = "")

# Perform jack-blockknife
jack.blockknife = 30000
data[,1] = data[,1] / jack.blockknife
blocks = as.integer(data[,1]) + 1
frac = TRUE
n.folds = length(unique(blocks))#max(blocks)

Ds = vector(mode = "numeric", length = n.folds)
count = 1
for (ii in sort(unique(blocks))) {
  test.i = which(blocks == ii)
  train = data[-test.i, ]
  Ds[count] = get.D(train, frac)[[1]]
  count = count + 1
}
Z.jack = D.frac / sqrt(n.folds * var(Ds, na.rm = T))
p.jack = 1 - pnorm(abs(Z.jack))
p.jack
```

../code/exc3.R