Population Genetics: Assignment 1

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.

Exercises

1 – Measurement of variance

A

Table 1: Solution to exercise 1a

	Selected	¬Selected	Total [%]
p^2	0.18	0.12	0.14
$2pq^2$	0.49	0.45	0.47
q^2	0.32	0.44	0.39

B The heterozygosity is the frequency of the middle row in table 1.

C We calculate $F_{ST} = 1 - \frac{2p_S q_S}{2p_T q_T} = 0.00836$. A low value in this would correspond to a situation where no difference in heterozygosity is prevalent between the subpopulations. In contrast, a high value would mean that the populations are completely segregated, with the respective alleles in each subpopulation being fixed. Indeed, this mirrors what we see in the statistics above.

2 – Modelling fitness in a diploid system

A The Hardy-Weinberg proportions simply correpsond to the combinatorial probability of achieving a certain setup of alleles. Like before, it is thus simply p^2 for genotype AA, $2p(1-p)$ for Aa and aA (assuming they are equivalent), and $(1-p)^2$ for aa.

B The mean fitness is given by evaluating the different fractions which are affiliated with a certain fitness. Retaining the algebraic form of our fitness values before calculating, we get the following expression and subsequent result:

$$\begin{aligned}\bar{f} &= (0.9a + 0.1)p^2 + 2(0.9b + 0.1)p(1-p) \\ &\quad + c(1-p)^2 = \\ &= 0.84\end{aligned}$$

C Reducing our expression above further, we can choose to differentiate with respect to the probability. There is only extremum, which is a maximum, since the second derivative evaluates to < 0.

$$\begin{aligned}\bar{f} &= 0.82p^2 + 1.82p(1-p) + 0.7(1-p)^2 \\ \frac{d\bar{f}}{dp} &= 1.64p + 1.82(1-2p) - 1.4(1-p) \overset{!}{=} 0 \\ p^* &= 0.7 \\ \bar{f}_{p^*} &= 0.85\end{aligned}$$

D We transform our problem accordingly:

Table 2: Solution to exercise 2d

Genotype	Required form	Transformed form
AA	$1 + 2\sigma$	1.17
Aa	$1 + 2h\sigma$	$1.29b + 0.14$
aa	1	1

With our transformed values, solving for the unknowns gives us $\sigma = 0.086$ and $h = 7.5b - 5$. We can thereafter investigate for which values in our equation governing the rate of change is negative, which we require for fixation in the q_A

underdominance case. Relabelling q as p, we get r.o.c. is given by

$$\frac{dq}{dt} = 2\sigma p(1-p)(p+h(1-2p))$$

where $p \in [0,1]$. Only the last factor will therefore affect the sign of the derivative. Inserting our values reduces the informative part to

$$p + (7.5b-5)(1-2p) \overset{!}{\leq} 0$$

where we now want to find the values fulfilling this. Solving for our parameters gives us that this occurs in the two regions

$$b > \frac{11}{15},\ p \geq \frac{5(3b-2)}{2(15b-11)}$$
$$b < \frac{11}{15},\ p \leq \frac{5(3b-2)}{2(15b-11)}.$$

However, we are only interested in the over- and underdominance cases for q_A, i.e. for $h > 1$ and $h < 0$ respectively. Only in the latter case are we required to know the sign of the derivative. Our equation of $h = 7.5b - 5$, we therefore get that we have fixation of q_a when $b > 0.8$ and $b < \frac{11}{15},\ p \leq \frac{5(3b-2)}{2(15b-11)}$, where the first corresponds to overdominance for q_A, and the latter underdominance.

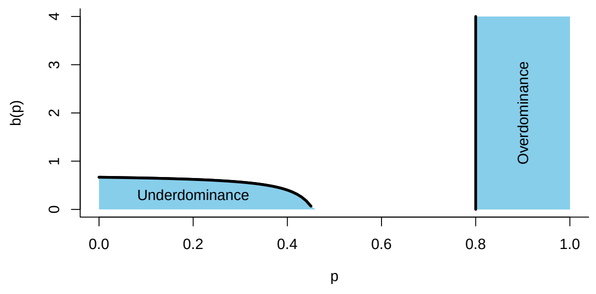

Figure 1: Fixation diagram for the allele q_a. The allele will fixate for values within the shaded areas, where under-/overdominance is given for q_A.

3 – Dynamics of allele frequency change

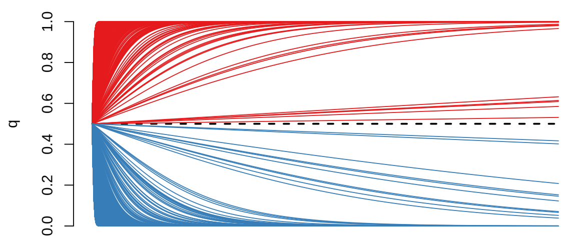

Figure 2: Evolution trajectories for different values of the selection rate σ, with the popualation initalised at $q = 0.5$. Naturally, negative values induce a negative pressure towards the allele, and it is as a consequence eradicated. Similarly, positive selection fixates the allele.

A μ, σ and N denote the mutation rate, the selection rate for a given allele, and the population size.

B Under selective pressure, the allele is bound to either fixate or to simply die out. Whichever effect happens depends on the sign of σ, i.e. for $\sigma > 0$, the allele will eventually fixate with $q_i^1 = 1$. In the contrasting case, the other allele will do the same. We can see this in fig. 2, where we have separated randomly drawn positive values of σ above in red, and correspondingly all negative values in blue.

C With no selective pressure, and mutations frequently producing either allele, as well as under no genetic drift, we will get the effect outlined in fig. 3, namely that the population will trend towards an equilibrium with an even split between the fractions. How fast this effect happens depends inherently on the mutation rate, which is signified in the figure by colour.

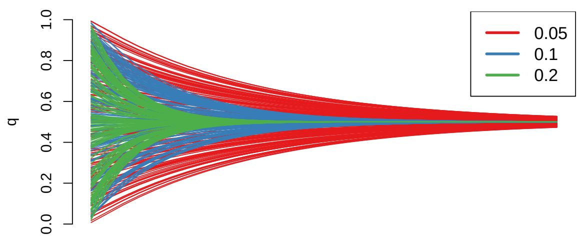

Figure 3: Population dynamics without any selective pressure. The mutations will instead force the population to its stable equilibrium, with the rate of convergence depending on the mutation rate.

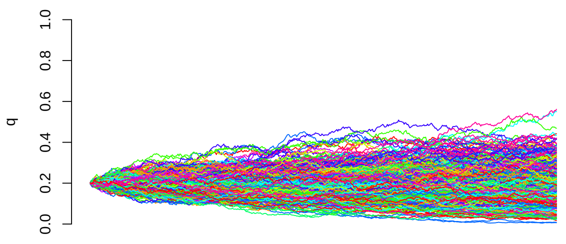

Figure 4: Curves for 500 drift-driven simulations. Note how the population spreads out uniformly around its initial condition.

D Figure 4 shows 500 drift-driven simulations initialised at $q = 0.2$, as simulated under Stratonovich integration (using Milstein's method). As we can see, the population of simulations on the large spreads out uniformly due to no pressure in going towards either end. We can argue about how likely it is on the grand scale that a single simulation will reach an allele fraction of $q = 1$ by considering Kimura's equation first. Since the overall expression reduces to $\dfrac{\partial p}{\partial t} = 0$ for $x = 1$, the probability for all initial conditions is some constant over time, which must depend on the initial condition. Reducing the problem to a case of fewer individuals, it is easy to see that the probability of the allele fixating is equal to the initial frequency under no selective pressure. This is also what we see when we simulate our stochastic system sufficiently many times (data not shown, although hinted at in fig. 4). Another way to consider the problem is to imagine a setting where we have N different alleles all with equal probability of fixating. At some point, on of them will have done so. Under no drift, the distribution of alleles for every generation can be traced back to the previous one, where the inheritance will depend directly on the prevalence of it. Reducing this all back to the origion, it is clear that the initial distribution will be equal to the probability of fixation.

4 – Time-dependent selection

A We separate our expression into two parts, and use induction to reason our way to the final answer. In particular, we have

$$
\begin{aligned}
x(t') &= \frac{x_0 e^{\sigma_1 t'}}{1 - x_0 + x_0 e^{\sigma_1 t'}} \\
x(t > t') &= \frac{x_{t'} e^{\sigma_2 (t-t')}}{1 - x_{t'} + x_{t'} e^{\sigma_2 (t-t')}} = \\
&= \frac{x_0 e^{\sigma_2 (t-t') + \sigma_1 t'}}{(1 - x_0 + x_0 e^{\sigma_1 t'})} \times \\
&\times \frac{1}{\left(1 - \dfrac{x_0 e^{\sigma_1 t'} + x_0 e^{\sigma_1 t' + \sigma_2 (t-t')}}{1 - x_0 + x_0 e^{\sigma_1 t'}}\right)} = \\
&= \frac{x_0 e^{\sigma_1 t' + \sigma_2 (t-t')}}{1 - x_0 + x_0 e^{\sigma_1 t' + \sigma_2 (t-t')}}
\end{aligned}
$$

for some arbitrary intervals where our σ's are separable. Since we can repeat this process for any number of intervals, our corresponding exponent will equal a sum over the $\sigma_i \cdot t_i$ products, which in the limit of our timesteps trending towards zero is equivalent to our sought-after Riemann sum, i.e. integral, over the range.

B It is not clear whether the variable x corresponds to the allele frequency, or if it simply serves as an abstract, continuous genotype representation. Nevertheless, we choose to impose bounds such that the variable is restricted within the range $[0, 1]$.

As we can see in fig. 5, the population of smaller size is affiliated with a much larger variability, as is to be expected from the expression defining the probability of a fixation. In contrast, the larger population steadily reaches the optimum and does not sway further on. As given in the formula, the larger population size simply keeps the probability of fixation down. Far away from the optimum, the regression in fitness caused by a non-beneficial fixation is not relatively as bad, which is why we some some variance in the beginning.

Because of the rigid optimum, the accumulated fitness variations mirror precisely the fraction trajectory. Because the optimum is static, we see a direct mirroring in the change in fitness from the change in x.

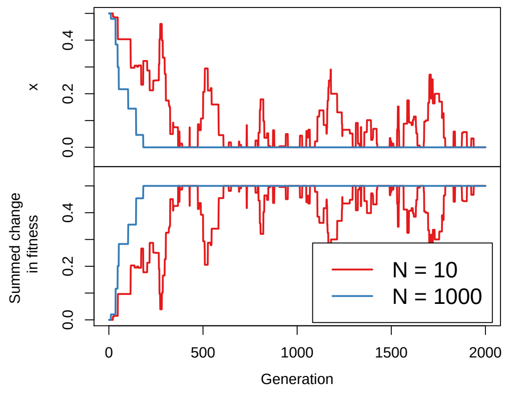

Figure 5: Change in x

C Inducing oscillations, we see very different dynamics in the system than before. Figure 6 shows how the two populations try to adapt to the changing fitness. As the accumulated fitness change shows, the larger population has a higher trend-curve, which again shows the signs of lesser variance. Since the smaller population more oftenly fixates in the 'wrong' direction, the cumulative trend will be sligthly lower, as we observe. We also note how both populations are able to adapt decently, although the larger population does so in a more stable manner.

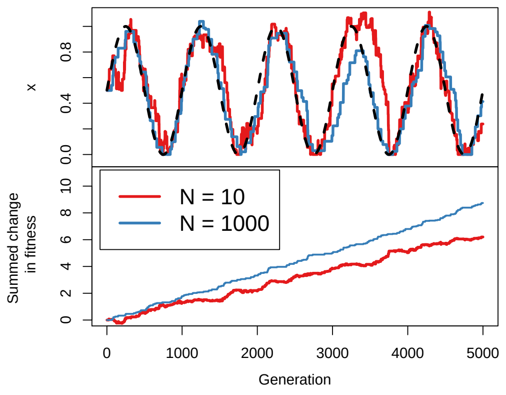

Figure 6: Dynamics for the same scenario as fig. 5, but with a sinusoidally changing optimum around 0.5. Note how both populations are able to adapt to the changing fitness, but with different local variance.

D Figures 7 and 8 show thr dynamics when the rate of the oscillations increase, with exemplary curves and the mean over 100 simulations respectively. Like in the previous case, the smaller population is affiliated with larger fluctuations, which shows similarly to previously in the cumulative change (fig. 8). Nevertheless, both populations are generally able to adapt to the fluctuations, although the smaller population size still has a higher tendency to shift in the wrong direction. As the mean trend in particular shows, however, the increase in oscillatory rate pushes the populations towards the mean of the fitness value. The rapid change of the optimum is simply too much for the populations in order for them to adapt in time, which becomes the more apparent the higher the oscillations. Still, we again see the general trend in the larger population size being able to follow the optimum better in general. Even though the smaller population size in theory allows it to "jump past" the optimum, it does not prove a sufficiently competetive trait in this setting.

However, a system under varying oscillatory rate will never completely pan out and average to the center, as this depends on the phase of the curve. In some cases the two will be in phase in such a way that the x variable is driven towards the ends. This nevertheless depends inherently on the development of λ, and other oscillating systems might circumvent this. Still, we see the a *trend* towards the mean of the oscillations, which is indeed telling of the overall dynamics. In a biological system, we would likely see that same general trend, with a bias towards staying close to the average, given that we would have a fitness landscape which depends on one variable. In reality, this is of course rarely the case, and movement on the fitness landscape happens in much more complex ways, through means of for example neutral mutations, which are unaccounted for here. We can nevertheless still infer that there would be a pressure towards *adaptability*, which would prevent the population from reaching its optimum, and instead making it transverse intermediate state that allow for rapidly climbing towards new optima.

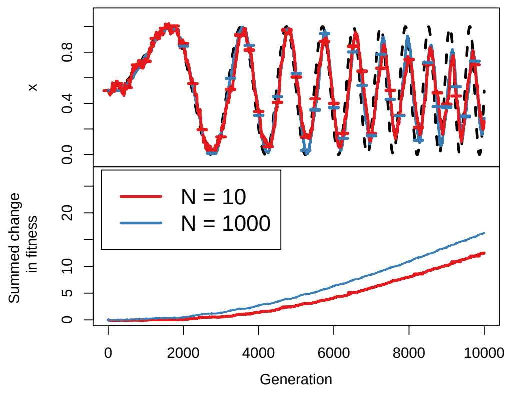

Figure 8: Mean and standard error for 1000 simulations in the same setting as in fig. 7. Note how a slight trend in the adaptability is apparent, with the smaller population consistently being less able to reach an adequate fitness.

Bacteria cultures could be an example of a population type which have assumed this fast-paced adaptive approach, with large population numbers being able to react to rapid changes in the environment (and hence fitness landscape). In contrast, a smaller population size would make the system a lot more susceptible to changes in the environment, which might prove useful in some cases, but also detrimental to the population as a whole.

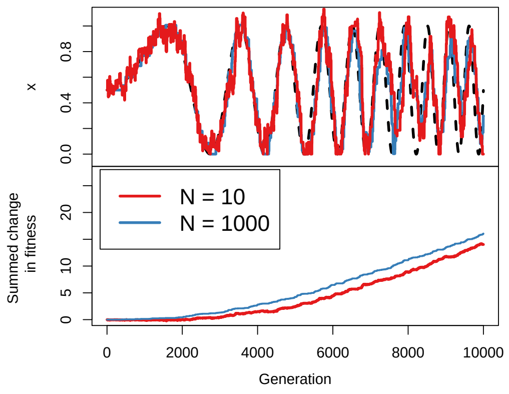

Figure 7: λ ranging linearly between 0.001 and 0 over the 10000 generations.

A Code

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

sigma = 0#0.0
mu = 0.0
xi = 0
h = 0.1
N = 10
vol = 0
start.t = 0
end.t = 500
init.q = 0.5
no.iterations = (end.t - start.t) / h
no.replicates = 1000

derivs = function(t, q) {
  sigma * q * (1 - q) + mu * (1 - 2 * q)
}

euler = function(t, q, h) {
  q + h * derivs(t, q)
}

milstein = function(t, q, h) {
  vol = sqrt(abs(q * (1 - q) / N))
  q.pred = q + derivs(t, q) * h + 1.0 * sqrt(h)
  q + derivs(t, q) * h + 1.0 * rnorm(1) * vol * sqrt(h)
}

run.func = function(n, m) {
  qs = vector(mode = "numeric", end.t)
  qs[1] = init.q
  q = qs[1]
  sigma <<- runif(1, min = n, max = m)
  last.t = 0
  for (ii in 2:no.iterations) {
    q = euler(t, q, h)
    q = min(1, q)
    q = max(0, q)
    if (ii * h > last.t) {
      last.t = last.t + 1
      qs[last.t] = q
    }
  }
  return(qs)
}

qs1 = replicate(no.replicates, run.func(0, 1))
qs2 = replicate(no.replicates, run.func(-1, 0))

par(mfrow=c(1,1), mai = c(.5,.8,.5,.5))
plot(
  qs1[, 1],
  type = "l",
  col = 1,
  bty = "n",
  ylab = "q",
  xlab = "", xaxt = "n", ylim = c(0,1)
)
lines(1:end.t, rep(init.q, end.t - start.t), col = "black", lwd = 2, lty = 2 )
out = apply(qs1[, -1], 2, function(x)
  lines(x, col = 1))
out = apply(qs2[, -1], 2, function(x)
  lines(x, col = 2))
```

../code/stoch.R

```r
setwd("~/compbio/src/assignments/pga2/code/")
library(RColorBrewer)
palette(brewer.pal(n = 8, name = "Set1"))
library(scales)
library(stats4)
library(viridis)
sigma = 0
mu = 0.2
xi = 0
h = 0.1
N = 100
vol = 0
start.t = 0
end.t = 30
init.q = 0
no.iterations = (end.t - start.t) / h
no.replicates = 150

derivs = function(t, q) {
  sigma * q * (1 - q) + mu * (1 - 2 * q)
}

euler = function(t, q, h) {
  q + h * derivs(t, q)
}

milstein = function(t, q, h) {
  vol = sqrt(abs(q * (1 - q) / N))
  q.pred = q + derivs(t, q) * h + 1.0 * sqrt(h)
  q + derivs(t, q) * h + 1.0 * rnorm(1) * vol * sqrt(h)
}

run.func = function(mut) {
  qs = vector(mode = "numeric", end.t - start.t)
  qs[1] = runif(1) #init.q
  q = qs[1]
  mu <<- mut#.2 #runif(1)
  last.t = 0
  for (ii in 2:no.iterations) {
    q = euler(t, q, h)
    q = min(1, q)
    q = max(0, q)
    if (ii * h > last.t) {
      last.t = last.t + 1
      qs[last.t] = q
    }
  }
  return(qs)
}

qs1 = replicate(no.replicates, run.func(.05))
qs2 = replicate(no.replicates, run.func(.1))
qs3 = replicate(no.replicates, run.func(.2))

par(mfrow=c(1,1), mai = c(.5,.8,.5,.5))
plot(
  qs1[, 1],
  type = "l",
  col = 1,
  bty = "n",
  ylab = "q",
  xlab = "", xaxt = "n", ylim = c(0,1)
)
# lines(1:end.t, rep(init.q, end.t - start.t), col = "black", lwd = 2, lty = 2 )
out = apply(qs1[, -1], 2, function(x)
  lines(x, col = 1))
out = apply(qs2[, -1], 2, function(x)
  lines(x, col = 2))
out = apply(qs3[, -1], 2, function(x)
  lines(x, col = 3))
legend("topright", c("0.05", "0.1", "0.2"), col = c(1, 2,3), lty=c(1,1,1), cex = 1.2, lwd =
    3)

# print(count / no.replicates)
```

../code/stochc.R

```
setwd("~/compbio/src/assignments/pga2/code/")
library(RColorBrewer)
palette(brewer.pal(n = 8, name = "Set1"))
library(scales)
library(stats4)
library(viridis)
sigma = 0
mu = 0.0
h = 0.1
N = 10000
vol = 0
start.t = 0
end.t = 500
init.q = 0.2
no.iterations = (end.t - start.t) / h
no.replicates = 500

derivs = function(t, q) {
  sigma * q * (1 - q) + mu * (1 - 2 * q)
}

euler = function(t, q, h) {
  q + h * derivs(t, q)
}

milstein = function(t, q, h) {
  vol = sqrt(abs(q * (1 - q) / N))
  q.pred = q + derivs(t, q) * h + 1.0 * sqrt(h)
  q + derivs(t, q) * h + 1.0 * rnorm(1) * vol * sqrt(h)
}

run.func = function() {
  qs = vector(mode = "numeric", end.t - start.t)
  qs[1] = init.q
  q = qs[1]
  last.t = 0
  for (ii in 2:no.iterations) {
    q = milstein(t, q, h)
    q = min(1, q)
    q = max(0, q)
    if (ii * h > last.t) {
      last.t = last.t + 1
      qs[last.t] = q
    }
  }
  return(qs)
}

qs1 = replicate(no.replicates, run.func())
qs2 = replicate(no.replicates, run.func())
qs3 = replicate(no.replicates, run.func())

par(mfrow=c(1,1), mai = c(.5,.8,.5,.5))
```

```
plot(
  qs1[, 1],
  type = "l",
  col = sample(rainbow(10)),
  bty = "n",
  ylab = "q",
  xlab = "", xaxt = "n", ylim = c(0,1)
)
# lines(1:end.t, rep(init.q, end.t - start.t), col = "black", lwd = 2, lty = 2 )
out = apply(qs1[, -1], 2, function(x)
  lines(x, col = sample(rainbow(10))))
```

../code/stochd.R

```
#!/usr/bin/env Rscript
setwd("~/compbio/src/assignments/pga1/code/")
library(RColorBrewer)
library(scales) #imports alpha
library(stats)
palette(brewer.pal(n = 8, name = "Set1"))
lw.s = 3

# Params
x.init = 0.5
no.generations = 2000
pop.size = 0
no.replicates = 1
lambda = 0.001

# Data
x = vector(mode = "numeric", no.generations)
sum.df = vector(mode = "numeric", no.generations)
x[1] = x.init

# Run!
run.sim = function(poppy.size) {
  target = 0
  for (t in 2:no.generations) {
    #target = sin(2 * pi * lambda * t) / 2 + 1 / 2
    dx = runif(1, min = -0.1, max = 0.1)
    if (x[t - 1] + dx < 0) {
      if (x[t - 1] == 0) {
        x[t] = x[t - 1]
        sum.df[t] = sum.df[t - 1]
        next
      }
      dx = -x[t - 1]
    }

    df = -(abs(x[t - 1] + dx - target) - abs(x[t - 1] - target))
    p.fix = (1 - exp(-2 * df)) / (1 - exp(-2 * poppy.size * df))

    if (runif(1) < p.fix) {
      x[t] = x[t - 1] + dx
      sum.df[t] = sum.df[t - 1] + df
    } else {
      x[t] = x[t - 1]
      sum.df[t] = sum.df[t - 1]
    }
  }
  return(list(x, sum.df))
}

data10 = replicate(no.replicates, run.sim(10))
data10k = replicate(no.replicates, run.sim(1000))
xs.10  = matrix(unlist(data10[1,]), ncol = no.replicates, byrow = FALSE)
fs.10  = matrix(unlist(data10[2,]), ncol = no.replicates, byrow = FALSE)
xs.10k = matrix(unlist(data10k[1,]), ncol = no.replicates, byrow = FALSE)
```

```r
fs.10k = matrix(unlist(data10k[2,]), ncol = no.replicates, byrow = FALSE)

mean10 = apply(xs.10, 1, mean)
mean10k = apply(xs.10k, 1, mean)
stdev10 = apply(xs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
stdev10k = apply(xs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

f.mean10 = apply(fs.10, 1, mean)
f.mean10k = apply(fs.10k, 1, mean)
f.stdev10 = apply(fs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
f.stdev10k = apply(fs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

par(mfrow = c(2, 1), oma = c(5, 2, 2, 0) + 0.0, mai = c(.0, 1, .0, 1))
plot(
  mean10,
  type = "l",
  #  ylim = c(-0, 1),
  ylim = c(-0.05, .5),
  lwd = 2,
  col = 1,
  xaxt = "n",
  xlab = "",
  ylab = "x"
)
lines(mean10k,
      type = "l",
      col = 2,
      lwd = 2)
# lines(
#   1:no.generations,
#   sin(2 * pi * lambda * 1:no.generations) / 2 + 1 / 2,
#   lty = 2,
#   lwd = 2
# )
skip = 250
arrows((2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(1, no.generations, skip)] - stdev10[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(2, no.generations, skip)] + stdev10[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 1,
       lwd = 3
)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       mean10k[seq(1, no.generations, skip)] - stdev10k[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       mean10k[seq(2, no.generations, skip)] + stdev10k[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 2,
       lwd = 3
)

# Plot those f-stats
plot(
  f.mean10,
  col = 1,
  type = "l",
  lwd = 2,
  ylab = "Summed change \n in fitness",
  xlab = "Generation",
  ylim = c(0, 0.55)
)

lines(f.mean10k, col = 2, lwd = 2)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       f.mean10[seq(1, no.generations, skip)] - f.stdev10[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       f.mean10[seq(2, no.generations, skip)] + f.stdev10[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 1,
       lwd = 3
)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       f.mean10k[seq(1, no.generations, skip)] - f.stdev10k[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       f.mean10k[seq(2, no.generations, skip)] + f.stdev10k[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 2,
       lwd = 3
)

# plot(
#   cumsum(f.mean10),
#   col = 1,
#   type = "l",
#   lwd = 2,
#   ylab = "Summed change \n in fitness",
#   # xlab = "Generation"
#   # ylim = c(0, 0.55)
# )
# lines(
#   cumsum(f.mean10k),
#   col = 2,
#   type = "l",
#   lwd = 2,
#   xlab = ""
#   # ylim = c(0, 0.55)
# )
mtext("Generation", side = 1, line = 2.5)
legend(
  "bottomright",
  c("N = 10", "N = 1000"),
  inset = 0.02,
  cex = 1.5,
  col = 1:2,
  lty = c(1, 1),
  lwd = 2,
  bg = "white"
)


# plot(apply(xs.10, 1, sd), type = "l", ylim=c(-2.5,2.5))
# lines(apply(xs.10k, 1, sd), type = "l", col = "red")
```

../code/fisherb.R

```r
#!/usr/bin/env Rscript
setwd("~/compbio/src/assignments/pga1/code/")
library(RColorBrewer)
library(scales) #imports alpha
library(stats)
palette(brewer.pal(n = 8, name = "Set1"))
lw.s = 3
```

```r

# Params
x.init = 0.5
no.generations = 5000
pop.size = 0
no.replicates = 1
lambda = 0.001
linew = 3

# Data
x = vector(mode = "numeric", no.generations)
sum.df = vector(mode = "numeric", no.generations)
x[1] = x.init

# Run!
run.sim = function(poppy.size) {
  target = 0
  for (t in 2:no.generations) {
    target = sin(2 * pi * lambda * t) / 2 + 1 / 2
    dx = runif(1, min = -0.1, max = 0.1)
    if (x[t - 1] + dx < 0) {
      if (x[t - 1] == 0) {
        x[t] = x[t - 1]
        sum.df[t] = sum.df[t - 1]
        next
      }
      dx = -x[t - 1]
    }

    df = -(abs(x[t - 1] + dx - target) - abs(x[t - 1] - target))
    p.fix = (1 - exp(-2 * df)) / (1 - exp(-2 * poppy.size * df))

    if (runif(1) < p.fix) {
      x[t] = x[t - 1] + dx
      sum.df[t] = sum.df[t - 1] + df
    } else {
      x[t] = x[t - 1]
      sum.df[t] = sum.df[t - 1]
    }
  }
  return(list(x, sum.df))
}

data10 = replicate(no.replicates, run.sim(10))
data10k = replicate(no.replicates, run.sim(1000))
xs.10  = matrix(unlist(data10[1,]), ncol = no.replicates, byrow = FALSE)
fs.10  = matrix(unlist(data10[2,]), ncol = no.replicates, byrow = FALSE)
xs.10k = matrix(unlist(data10k[1,]), ncol = no.replicates, byrow = FALSE)
fs.10k = matrix(unlist(data10k[2,]), ncol = no.replicates, byrow = FALSE)

mean10 = apply(xs.10, 1, mean)
mean10k = apply(xs.10k, 1, mean)
stdev10 = apply(xs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
stdev10k = apply(xs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

f.mean10 = apply(fs.10, 1, mean)
f.mean10k = apply(fs.10k, 1, mean)
f.stdev10 = apply(fs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
f.stdev10k = apply(fs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

par(
  mfrow = c(2, 1),
  oma = c(5, 2, 2, 0) + 0.0,
  mai = c(.0, 1, .0, 1)
)
plot(
  mean10,
  type = "l",
  #  ylim = c(-0, 1),
  ylim = c(-0.05, 1.1),
  col = 1,
  xaxt = "n",
  xlab = "",
  ylab = "x",
  lwd = linew
)
lines(mean10k,
      type = "l",
      col = 2,
      lwd = linew)
lines(
  1:no.generations,
  sin(2 * pi * lambda * 1:no.generations) / 2 + 1 / 2,
  lty = 2,
  lwd = linew
)
skip = 250
arrows((2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(1, no.generations, skip)] - stdev10[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(2, no.generations, skip)] + stdev10[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 1,
       lwd = linew
)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       mean10k[seq(1, no.generations, skip)] - stdev10k[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       mean10k[seq(2, no.generations, skip)] + stdev10k[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 2,
       lwd = linew
)

# Plot those f-stats
plot(
  f.mean10,
  col = 1,
  type = "l",
  ylab = "Summed change \n in fitness",
  xlab = "Generation",
  ylim=c(0,11),
  lwd = linew
)
mtext("Generation", side = 1, line = 2.5)
lines(f.mean10k, col = 2, lwd = 2)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       f.mean10[seq(1, no.generations, skip)] - f.stdev10[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       f.mean10[seq(2, no.generations, skip)] + f.stdev10[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 1,
       lwd = linew
)
```

```r
arrows((2:no.generations)[seq(2, no.generations, skip)],
       f.mean10k[seq(1, no.generations, skip)] - f.stdev10k[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       f.mean10k[seq(2, no.generations, skip)] + f.stdev10k[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 2,
       lwd = linew
)

legend(
  "topleft",
  c("N = 10", "N = 1000"),
  inset = 0.02,
  cex = 1.5,
  col = c(1, 2),
  lty = c(1, 1),
  bg = "white",
  lwd = linew
)


# plot(apply(xs.10, 1, sd), type = "l", ylim=c(-2.5,2.5))
# lines(apply(xs.10k, 1, sd), type = "l", col = "red")
```

../code/fisherc.R

```r
#!/usr/bin/env Rscript
setwd("~/compbio/src/assignments/pga1/code/")
library(RColorBrewer)
library(scales) #imports alpha
library(stats)
palette(brewer.pal(n = 8, name = "Set1"))
lw.s = 3

# Params
x.init = 0.5
no.generations = 10000
pop.size = 1000
no.replicates = 1
mi = 0.0
ma = 0.001
lambda = seq(mi, ma, by = (ma - mi) / no.generations)

linew = 3

# Data
x = vector(mode = "numeric", no.generations)
sum.df = vector(mode = "numeric", no.generations)
x[1] = x.init

# Run!
run.sim = function(poppy.size) {
  target = 0
  for (t in 2:no.generations) {
    target = sin(2 * pi * lambda[t] * t) / 2 + 1 / 2
    dx = runif(1, min = -0.1, max = 0.1)
    if (x[t - 1] + dx < 0) {
      if (x[t - 1] == 0) {
        x[t] = x[t - 1]
        sum.df[t] = sum.df[t - 1]
        next
      }
      dx = -x[t - 1]
    }

    df = -(abs(x[t - 1] + dx - target) - abs(x[t - 1] - target))
    p.fix = (1 - exp(-2 * df)) / (1 - exp(-2 * poppy.size * df))

    if (runif(1) < p.fix) {
      x[t] = x[t - 1] + dx
      sum.df[t] = sum.df[t - 1] + df
    } else {
      x[t] = x[t - 1]
      sum.df[t] = sum.df[t - 1]
    }
  }
  return(list(x, sum.df))
}

data10 = replicate(no.replicates, run.sim(10))
data10k = replicate(no.replicates, run.sim(1000))
xs.10  = matrix(unlist(data10[1,]), ncol = no.replicates, byrow = FALSE)
fs.10  = matrix(unlist(data10[2,]), ncol = no.replicates, byrow = FALSE)
xs.10k = matrix(unlist(data10k[1,]), ncol = no.replicates, byrow = FALSE)
fs.10k = matrix(unlist(data10k[2,]), ncol = no.replicates, byrow = FALSE)

mean10 = apply(xs.10, 1, mean)
mean10k = apply(xs.10k, 1, mean)
stdev10 = apply(xs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
stdev10k = apply(xs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

f.mean10 = apply(fs.10, 1, mean)
f.mean10k = apply(fs.10k, 1, mean)
f.stdev10 = apply(fs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
f.stdev10k = apply(fs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

par(
  mfrow = c(2, 1),
  oma = c(5, 2, 2, 0) + 0.0,
  mai = c(.0, 1, .0, 1)
)
plot(
  1:no.generations,
  sin(2 * pi * lambda[1:no.generations] * 1:no.generations) / 2 + 1 / 2,
  lty = 2,
  lwd = linew - 0,
  type = "l",
  #  ylim = c(-0, 1),
  ylim = c(-0.05, 1.1),
  xaxt = "n",
  xlab = "",
  ylab = "x"
)
lines(mean10k,
      type = "l",
      col = 2,
      lwd = linew)
lines(
  mean10,
  col = 1,
  lwd = linew
)
skip = 250
arrows((2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(1, no.generations, skip)] - stdev10[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(2, no.generations, skip)] + stdev10[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
        code = 3,
        col = 1,
        lwd = linew
)
arrows((2:no.generations)[seq(2, no.generations, skip)],
        mean10k[seq(1, no.generations, skip)] − stdev10k[seq(2, no.generations, skip)],
        (2:no.generations)[seq(2, no.generations, skip)],
        mean10k[seq(2, no.generations, skip)] + stdev10k[seq(2, no.generations, skip)],
        length = 0.05,
        angle = 90,
        code = 3,
        col = 2,
        lwd = linew
)

# Plot those f−stats
plot(
  f.mean10,
  col = 1,
  type = "l",
  ylab = "Summed change \n in fitness",
  xlab = "Generation",
  ylim=c(0,27.5),
  lwd = linew
)
mtext("Generation", side = 1, line = 2.5)
lines(f.mean10k, col = 2, lwd = 2)
arrows((2:no.generations)[seq(2, no.generations, skip)],
        f.mean10[seq(1, no.generations, skip)] − f.stdev10[seq(2, no.generations, skip)],
        (2:no.generations)[seq(2, no.generations, skip)],
        f.mean10[seq(2, no.generations, skip)] + f.stdev10[seq(2, no.generations, skip)],
        length = 0.05,
        angle = 90,
        code = 3,
        col = 1,
        lwd = linew
)
arrows((2:no.generations)[seq(2, no.generations, skip)],
        f.mean10k[seq(1, no.generations, skip)] − f.stdev10k[seq(2, no.generations, skip)],
        (2:no.generations)[seq(2, no.generations, skip)],
        f.mean10k[seq(2, no.generations, skip)] + f.stdev10k[seq(2, no.generations, skip)],
        length = 0.05,
        angle = 90,
        code = 3,
        col = 2,
        lwd = linew
)

legend(
  "topleft",
  c("N = 10", "N = 1000"),
  inset = 0.02,
  cex = 1.5,
  col = c(1, 2),
  lty = c(1, 1),
  bg = "white",
  lwd = linew
)
```

../code/fisherd.R

```
#!/usr/bin/env Rscript
setwd("~/compbio/src/assignments/pga1/code/")
library(RColorBrewer)
library(scales) #imports alpha
library(stats)
palette(brewer.pal(n = 8, name = "Set1"))
lw.s = 3
```

```r

# Params
x.init = 0.5
no.generations = 10000
pop.size = 1000
no.replicates = 10
mi = 0.0
ma = 0.001
lambda = seq(mi, ma, by = (ma - mi) / no.generations)
linew = 3

# Data
x = vector(mode = "numeric", no.generations)
sum.df = vector(mode = "numeric", no.generations)
x[1] = x.init

# Run!
run.sim = function(poppy.size) {
  target = 0
  for (t in 2:no.generations) {
    target = sin(2 * pi * lambda[t] * t) / 2 + 1 / 2
    dx = runif(1, min = -0.1, max = 0.1)
    if (x[t - 1] + dx < 0) {
      if (x[t - 1] == 0) {
        x[t] = x[t - 1]
        sum.df[t] = sum.df[t - 1]
        next
      }
      dx = -x[t - 1]
    }

    df = -(abs(x[t - 1] + dx - target) - abs(x[t - 1] - target))
    p.fix = (1 - exp(-2 * df)) / (1 - exp(-2 * poppy.size * df))

    if (runif(1) < p.fix) {
      x[t] = x[t - 1] + dx
      sum.df[t] = sum.df[t - 1] + df
    } else {
      x[t] = x[t - 1]
      sum.df[t] = sum.df[t - 1]
    }
  }
  return(list(x, sum.df))
}

data10 = replicate(no.replicates, run.sim(10))
data10k = replicate(no.replicates, run.sim(1000))
xs.10  = matrix(unlist(data10[1,]), ncol = no.replicates, byrow = FALSE)
fs.10  = matrix(unlist(data10[2,]), ncol = no.replicates, byrow = FALSE)
xs.10k = matrix(unlist(data10k[1,]), ncol = no.replicates, byrow = FALSE)
fs.10k = matrix(unlist(data10k[2,]), ncol = no.replicates, byrow = FALSE)

mean10 = apply(xs.10, 1, mean)
mean10k = apply(xs.10k, 1, mean)
stdev10 = apply(xs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
stdev10k = apply(xs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

f.mean10 = apply(fs.10, 1, mean)
f.mean10k = apply(fs.10k, 1, mean)
f.stdev10 = apply(fs.10, 1, function(x)
  sd(x) / sqrt(length(mean10)))
f.stdev10k = apply(fs.10k, 1, function(x)
  sd(x) / sqrt(length(mean10k)))

par(
  mfrow = c(2, 1),
  oma = c(5, 2, 2, 0) + 0.0,
  mai = c(.0, 1, .0, 1)
)
plot(
  1:no.generations,
  sin(2 * pi * lambda[1:no.generations] * 1:no.generations) / 2 + 1 / 2,
  lty = 2,
  lwd = linew - 0,
  type = "l",
  #  ylim = c(-0, 1),
  ylim = c(-0.05, 1.1),
  xaxt = "n",
  xlab = "",
  ylab = "x"
)
lines(mean10k,
      type = "l",
      col = 2,
      lwd = linew)
lines(
  mean10,
  col = 1,
  lwd = linew
)
skip = 250
arrows((2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(1, no.generations, skip)] - stdev10[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       mean10[seq(2, no.generations, skip)] + stdev10[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 1,
       lwd = linew
)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       mean10k[seq(1, no.generations, skip)] - stdev10k[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       mean10k[seq(2, no.generations, skip)] + stdev10k[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 2,
       lwd = linew
)

# Plot those f-stats
plot(
  f.mean10,
  col = 1,
  type = "l",
  ylab = "Summed change \n in fitness",
  xlab = "Generation",
  ylim=c(0,27.5),
  lwd = linew
)
mtext("Generation", side = 1, line = 2.5)
lines(f.mean10k, col = 2, lwd = 2)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       f.mean10[seq(1, no.generations, skip)] - f.stdev10[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       f.mean10[seq(2, no.generations, skip)] + f.stdev10[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 1,
       lwd = linew
)
arrows((2:no.generations)[seq(2, no.generations, skip)],
       f.mean10k[seq(1, no.generations, skip)] - f.stdev10k[seq(2, no.generations, skip)],
       (2:no.generations)[seq(2, no.generations, skip)],
       f.mean10k[seq(2, no.generations, skip)] + f.stdev10k[seq(2, no.generations, skip)],
       length = 0.05,
       angle = 90,
       code = 3,
       col = 2,
       lwd = linew
)

legend(
  "topleft",
  c("N = 10", "N = 1000"),
  inset = 0.02,
  cex = 1.5,
  col = c(1, 2),
  lty = c(1, 1),
  bg = "white",
  lwd = linew
)
```

../code/fisherd_ext.R