R_Assignment_Simulations

Ian Dworkin

11/27/2018

R take home assignment 4

Due Monday December 3rd, 5:59PM

Question 1

Please do the DataCamp course (all chapters) on "Introduction to Bioconductor".

Question 2

Question

In class we developed the code to perform the deterministic simulation for a simple population genetics model with one locus with 2 alleles in a haploid. Use what you learned in class, and the following information to write a general simulator for diploids. Your code should be flexible enough (i.e. arguments in a function call for instance) to try different sets of parameters including initial allele frequencies, fitness of each genotype, number of generations. Your script should do the following at a minimum:

- Report whether the beneficial allele fixes by the end of your simulation.
- Plot the trajectory of the beneficial allele across generations.

Some things you will need:

mean population fitness in a given generation (t) is

$\overline{W} = p^2 W_{AA} + 2pq W_{Aa} + q^2 W_{aa}$

and

$p_{(t+1)} = p^2_{(t)} \frac{W_{AA}}{\overline{W}} + p_{(t)} q_{(t)} \frac{W_{Aa}}{\overline{W}}$

Where

- $p_{(t+1)}$ is the frequency of the A allele in the population in generation $t+1$.
- $p_{(t)}$ and $q_{(t)}$ are the allele frequencies of A and a in generation t.
- $p^2_{(t)}$ is the genotypic frequency for the AA diploid genotype in generation t.
- W_{AA}, W_{Aa} & W_{aa} are the fitnesses for each genotype AA, Aa and aa respectively.
- You can also use (if you want, but do not need to) the fact that $p + q = 1$ for allele frequencies, and $p^2 + 2pq + q^2 = 1$ for genotypic frequencies.
- While mean fitness of the population changes with allele frequencies (according to the equation above), individual genotypic fitnesses do not (i.e. no frequency dependent selection)

Answer

At its heart, we just need to re-write the haploid function based on the diploid model. So for the basic function across one generation would look like something like:

```r
p_t1 <- function(w_AA, w_Aa, w_aa, p_t0) {
        w_bar <- (w_AA * p_t0^2) +
            (2 * w_Aa* p_t0 * (1 - p_t0)) +
            (w_aa * (1- p_t0)^2)

        p_t1 <- ((w_AA* p_t0^2) + (w_Aa)*p_t0*(1-p_t0)) / w_bar
        return(p_t1)}

p_t1(w_AA = 1, w_Aa = 0.975, w_aa = 0.95, p_t0 = 0.5)
```

```
## [1] 0.5064103
```

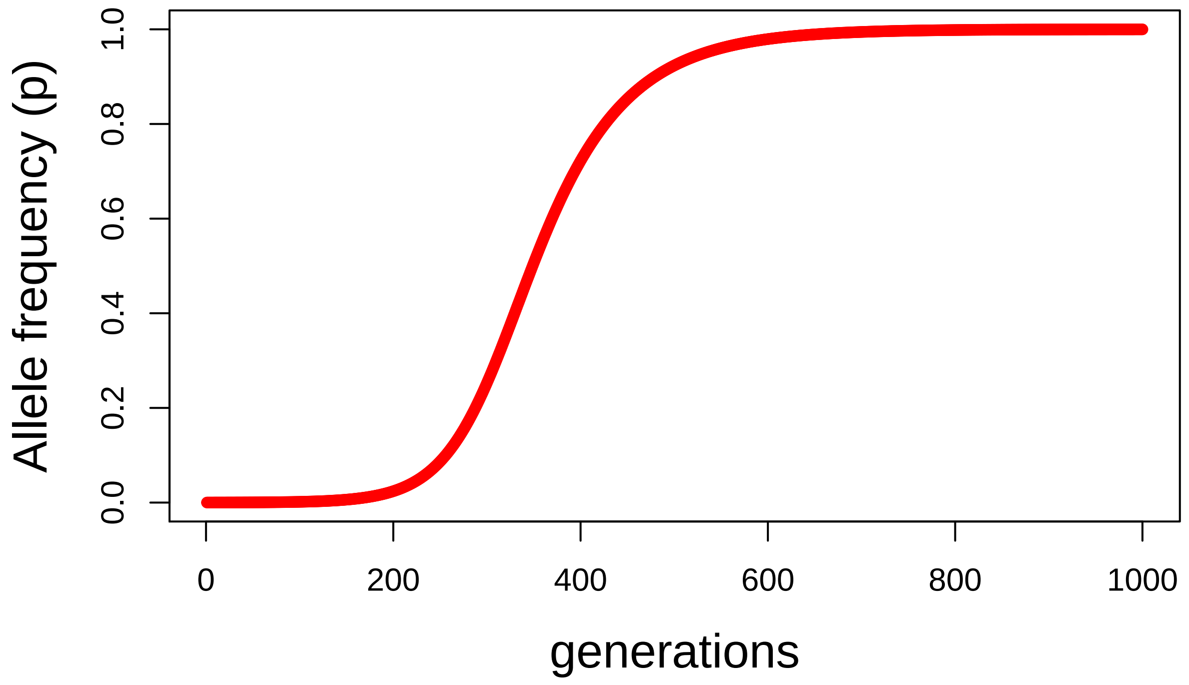
And then we can just modify the larger bit of code from the haploid function

```r
diploid.selection <- function(p0 = 0.01, w_AA = 1, w_Aa = 0.9, w_aa, n = 100) {

# Initialize vectors to store p
p <- rep(NA,n)

# starting conditions
p[1] <- p0 # starting allele frequencies

# now we need to loop from generation to generation
for ( i in 2:n) {
     w_bar <- (w_AA * p[i - 1]^2) +
            (2 * w_Aa* p[i - 1] * (1 - p[i - 1])) +
            (w_aa * (1- p[i - 1])^2)

     p[i] <- ((w_AA * p[i - 1]^2) + (w_Aa)*p[i - 1]*(1-p[i - 1])) / w_bar
     rm(w_bar)
    }

if (any(p > 0.9999)) {
    fixation <- min(which.max(p > 0.9999))
    cat("fixation for A1 occurs approximately at generation:", fixation )
    } else {
        maxAlleleFreq <- max(p)
        cat("fixation of A1 does not occur, max. allele frequency is: ", maxAlleleFreq)
    }

# Let's make the plot
par(mfrow=c(1,1))

# 1. change in allele frequency over time
plot(x = 1:n, y = p,
     xlab="generations",
     ylab="Allele frequency (p)", ylim = c(0,1),
     pch = 20, col = "red", cex.lab = 1.5)

}
```

Let's see what this does.

```
diploid.selection(p0 = 0.0001, w_AA = 1, w_Aa = 0.987, w_aa = 0.96, n = 1000)
```

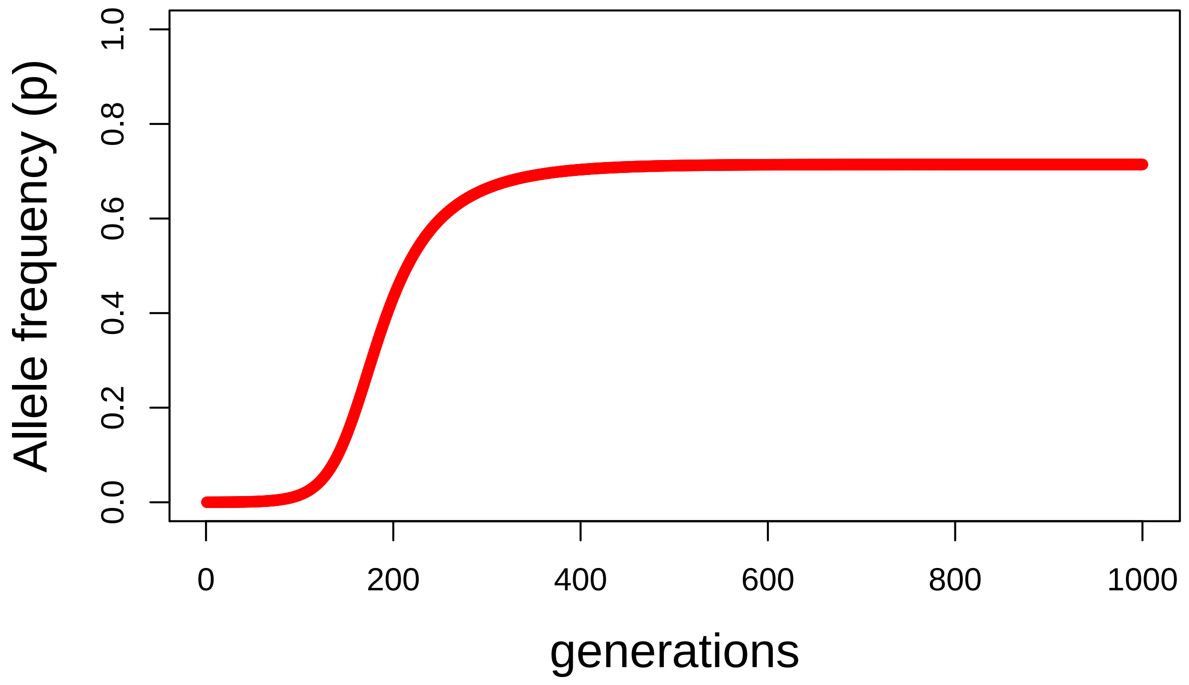
```
## fixation of A1 does not occur, max. allele frequency is:  0.9998903
```



or

```
diploid.selection(p0 = 0.0001, w_AA = 0.98, w_Aa = 1, w_aa = 0.95, n = 1000)
```

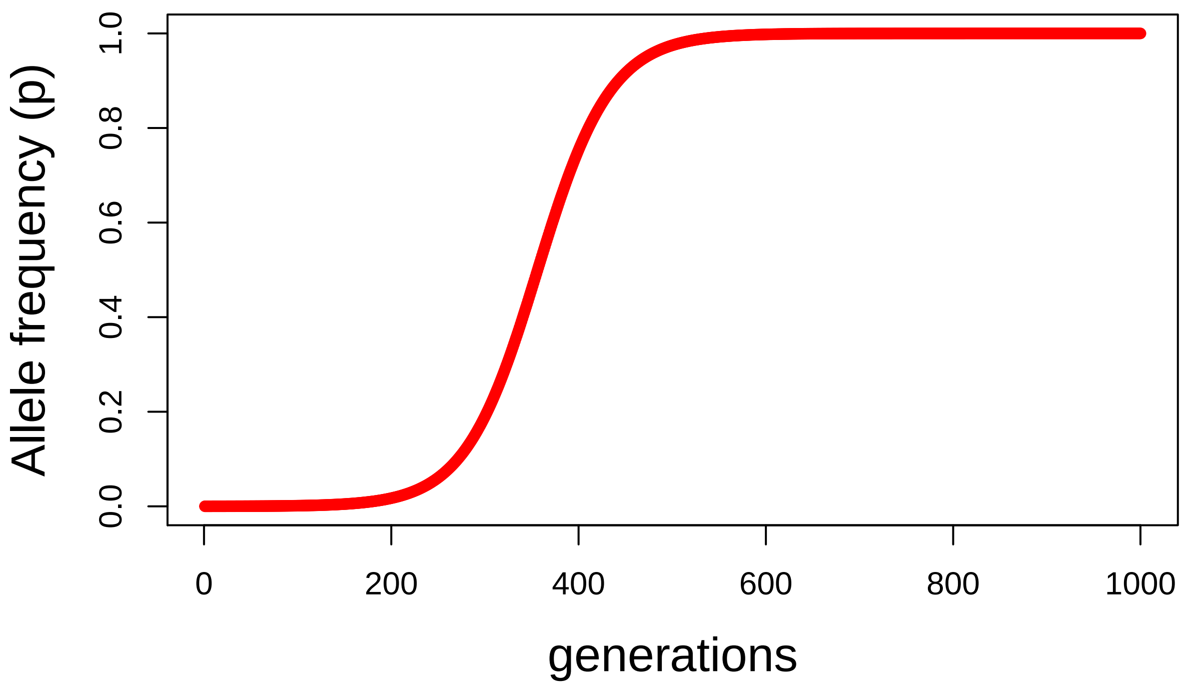
```
## fixation of A1 does not occur, max. allele frequency is:  0.714284
```



or

```
diploid.selection(p0 = 0.0001, w_AA = 1, w_Aa = 0.975, w_aa = 0.95, n = 1000)
```

```
## fixation for A1 occurs approximately at generation: 720
```



Question 3 - Genetic drift simulator

Question

In class we saw how with a simple stochastic simulation, and the use of the `sample()` function we could simulate how allele frequencies changed in small populations due to genetic drift alone (i.e. random sampling of gametes). However we only did this over the course of two generations. Please write a general purpose simulator that allows you to examine the effects of genetic drift allowing for variable arguments/parameters minimally including:

- Number of alleles in population (i.e. for diploids 2 times the number of individuals).
- Starting allele frequency
- Number of generations.
- plot allele frequency changes over time

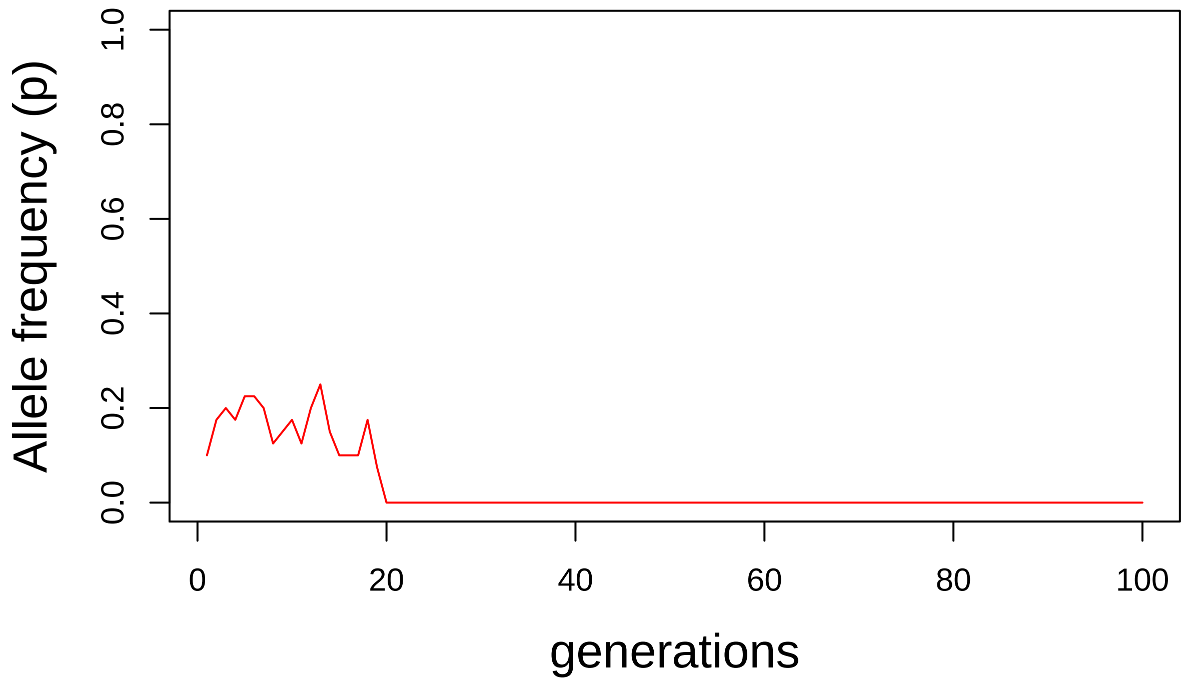
Answer

In some ways this is just like our previous simulator, but just using a line with the sample function to allow for changes in allele frequency

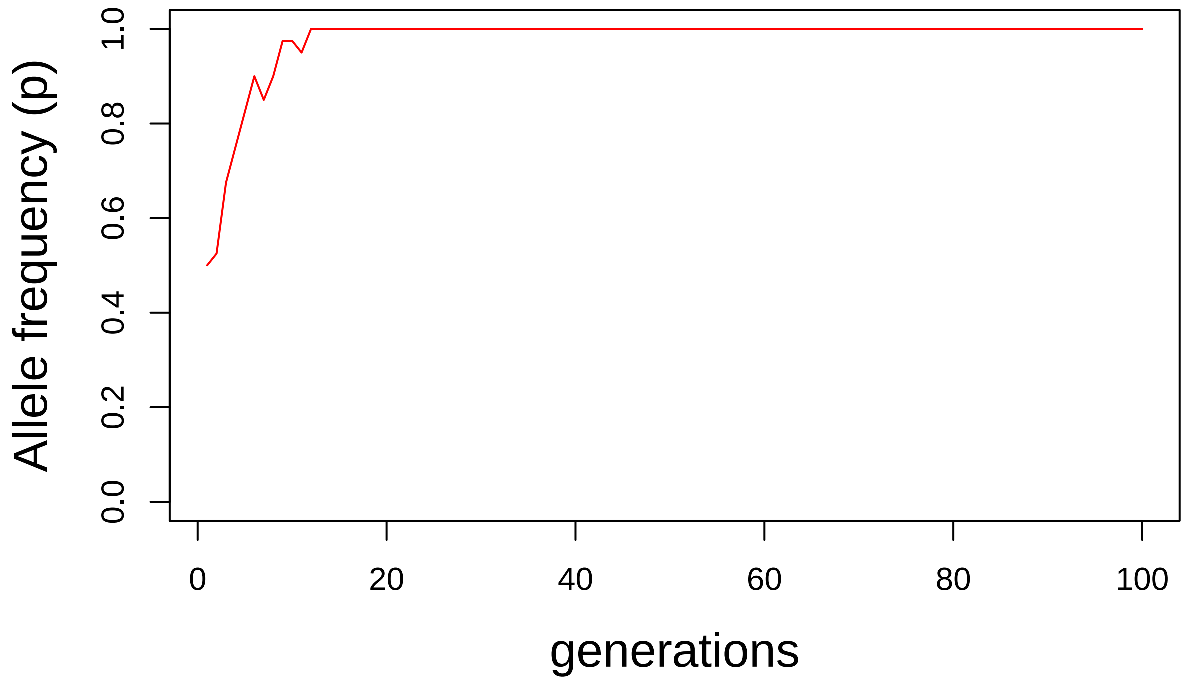
```
GeneticDrift <- function(alleles, p0, gen){

    p <- rep(NA, gen)
    p[1] <- p0

    for (i in 2:gen) {
        allele_counts <- sample(c("A", "a"),
                                        size = alleles, replace = T,
                                        prob = c(p[i-1], (1 - p[i-1])))

        p[i] <- sum(grepl("A", allele_counts))/alleles # allele frequency of "A"
    }

    plot(x = 1:gen, y = p, type = "l",
     xlab="generations",
     ylab="Allele frequency (p)", ylim = c(0,1),
     pch = 20, col = "red", cex.lab = 1.5)

}
```

And we can check a few times

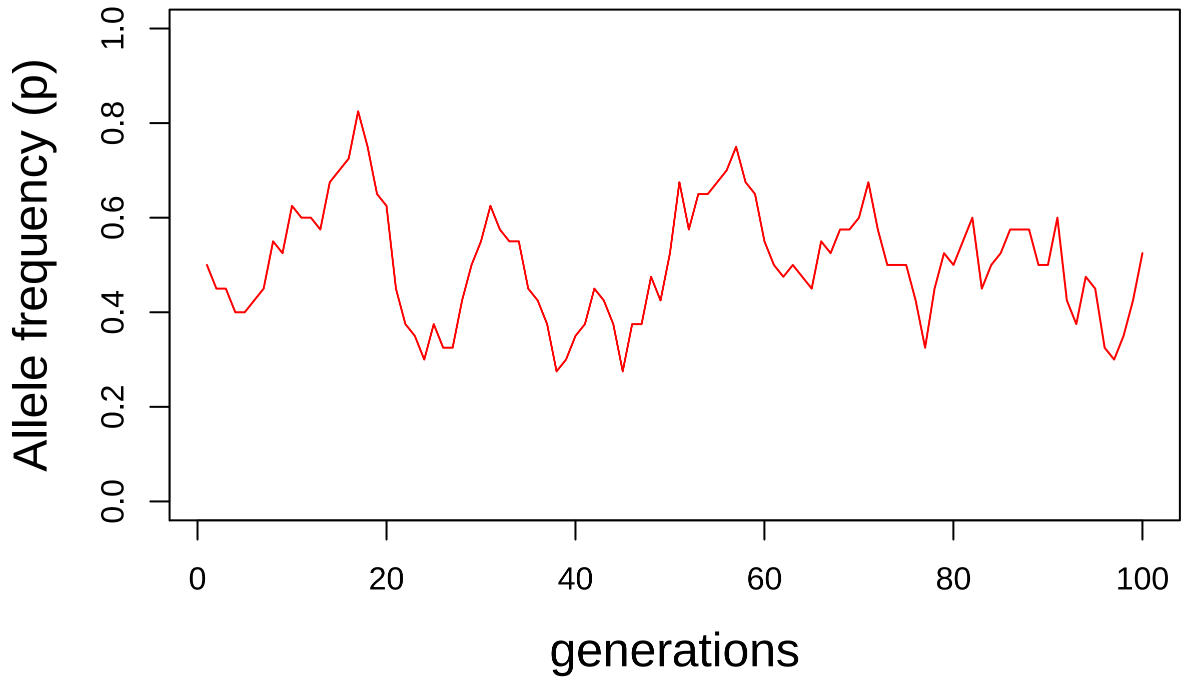
```
GeneticDrift(alleles = 40, p0 = 0.1, gen = 100)
```



```
GeneticDrift(alleles = 40, p0 = 0.5, gen = 100)
```



```r
GeneticDrift(alleles = 40, p0 = 0.5, gen = 100)
```



Question 4

Question

Repeating your stochastic simulation. Now that you have a working simulator of genetic drift, you want to use it to assess how likely it is for the allele to be lost ($p = 0$) after a certain number of generations of drift (we will use 100 generations, but your function should be flexible). Using your function (you can modify it if you need to), perform a simulation 1000 times each with starting allele frequencies of $p = f(A)$ of 0.5, 0.25 and 0.1, with 200 diploid individuals in the population each generation. Have your function record the proportion (out of 1000) of simulated runs that the A allele is lost from the population ($p = 0$).

Answer

First Let's modify the original function to just spit out the allele frequency in the final generartion of a single simulation. Not necessary, but it means we don't have to store very much. For such a small simulation not really necessary, but for larger ones it may be useful to not store too much.

```r
GeneticDrift_v2 <- function(alleles, p0, gen){

    p <- rep(NA, gen)
    p[1] <- p0

    for (i in 2:gen) {
        allele_counts <- sample(c("A", "a"),
                                size = alleles, replace = T,
                                prob = c(p[i-1], (1 - p[i-1])))

        p[i] <- sum(grepl("A", allele_counts))/alleles # allele frequency of "A"
    }
    return(p[gen]) # only keeping the last allele frequency in final generation.
}
```

Try this a few times

```r
GeneticDrift_v2(alleles = 40, p0 = 0.5, gen = 100)
```

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

Now we can go ahead and write function to collect everything we need.

```r
Extinction_prob <- function( sims = 1000, alleles = 400, p0 = 0.5, gen = 100) {
    end_alleles <- replicate(sims,
                             GeneticDrift_v2(alleles = alleles, p0 = p0, gen = gen))
    return(sum(end_alleles == 0)/sims) # how often does a run end in loss of "A" from pop
}
```

And now we can test (I only did the simulation 200 times to speed it up, but easy to change)

```r
Extinction_prob(sims = 200, alleles = 400, p0 = 0.5, gen = 100)
```

```
## [1] 0.015
```

```r
Extinction_prob(sims = 200, alleles = 400, p0 = 0.25, gen = 100)
```

```
## [1] 0.115
```

```r
Extinction_prob(sims = 200, alleles = 400, p0 = 0.1, gen = 100)
```

```
## [1] 0.405
```

And as expected, as the "A" allele gets rarer in the populatin it is lost more often due to drift.

Question 5

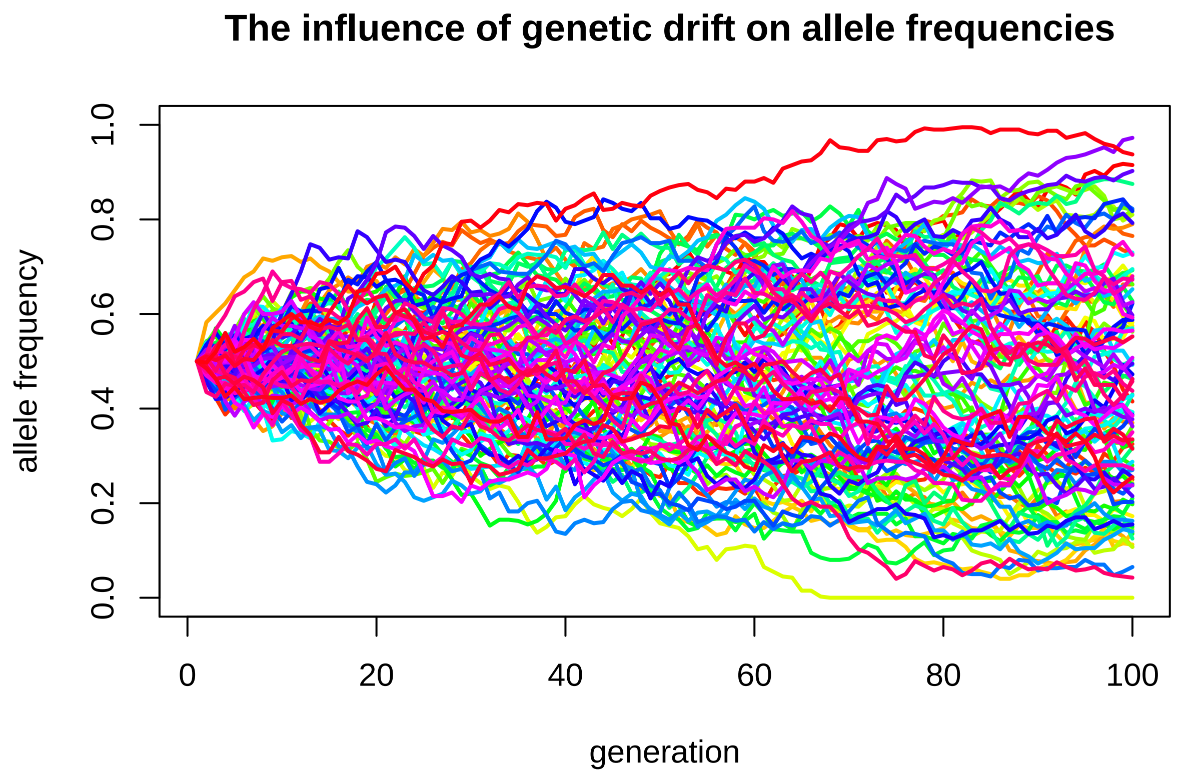
Question

Write some code that allows you to plot the allele trajectories for drift for 100 of the simulations starting at $p = 0.5$. hint: I showed you an example of how to draw multiple lines with a single function last week

Answer

Just a slight modification of the previous question. In this case we do want to store the allele dynamics for each simulation.

```r
GeneticDrift_v3 <- function(alleles, p0, gen){

    p <- rep(NA, gen)
    p[1] <- p0

    for (i in 2:gen) {
        allele_counts <- sample(c("A", "a"),
                                size = alleles, replace = T,
                                prob = c(p[i-1], (1 - p[i-1])))

        p[i] <- sum(grepl("A", allele_counts))/alleles # allele frequency of "A"
    }
   return(p)
}

sims <- 100 # number of simulations
gen <- 100 # number of generations

# This repeats the simulation sims times.
drift_frequencies <- replicate(sims,
    GeneticDrift_v3(alleles = 400, p0 = 0.5, gen = gen))

drift_frequencies <- t(drift_frequencies)

matplot(t(drift_frequencies),
    ylim = c(0, 1), type = "l", lty = 1, col = rainbow(sims), lwd =2,
    ylab = "allele frequency", xlab = "generation",
    main = " The influence of genetic drift on allele frequencies")
```



Question 6 - A simple power analysis

Question

One common use for stochastic simulations is to evaluate the statistical power of a particular method or model given some known parameters such as effect size (like the slope of the regression line), sample size, and residual standard error (the uncertainty in your estimator). While I kind of detest teaching anything with *p-values* (it is not a great way of evaluating statistical models) we will use them for this exercise.

We saw in class how to simulate data for the relationship between y and x and plot the regression fits (best fit lines) for the simulated data. You are going to use that same approach to determine how often you observe a "significant" p-value ($p < 0.05$) under some different scenarios.

In class we did something approximately like this (with a few changes)

```
x <- seq(from =1, to = 10, length.out = 20) # length.out is how many observations we will have
a <- 0.5 # intercept
b <- 0.1 # slope
y_deterministic <- a + b*x

y_simulated <- rnorm(length(x), mean = y_deterministic, sd = 2)

mod_sim <- lm(y_simulated ~ x)
p_val_slope <- summary(mod_sim)$coef[2,4] # extracts the p-value
p_val_slope
```

```
## [1] 0.3811058
```

- 6A First re-write this as a function where the intercept, slope, sample size (number of observations) and residual standard error (the stochastic variation) are all arguments in the function, so that they can be changed easily to different values. Check that it works. How can you confirm (given this is stochastic) that you can get the same results whether it is in the code I wrote above or the function you wrote?

Answer 6A

This only reports the p-value associated with the slope of the regression line. For this assignment I only asked for that, but more generally I suggest keeping track of the slope itself `coef(mod_sim)[2]` and examine variation in it relative to the known value you use for *b*.

```r
regression_sim <- function(sample_size, a, b, rse) {

x <- seq(from = 1, to = 10, length.out = sample_size) # length.out is how many observations we will hav
a <- a # intercept
b <- b # slope

y_deterministic <- a + b*x

y_simulated <- rnorm(length(x),
                     mean = y_deterministic,
                     sd = rse)

mod_sim <- lm(y_simulated ~ x)
p_val_slope <- summary(mod_sim)$coef[2,4] # extracts the p-value
p_val_slope
}
```

Check it (you can `set.seed()` to some value to confirm that you get the same result with and without the explicit function call.

```r
regression_sim(a = 0.5, b = 0.1, rse = 2, sample_size = 20)
```

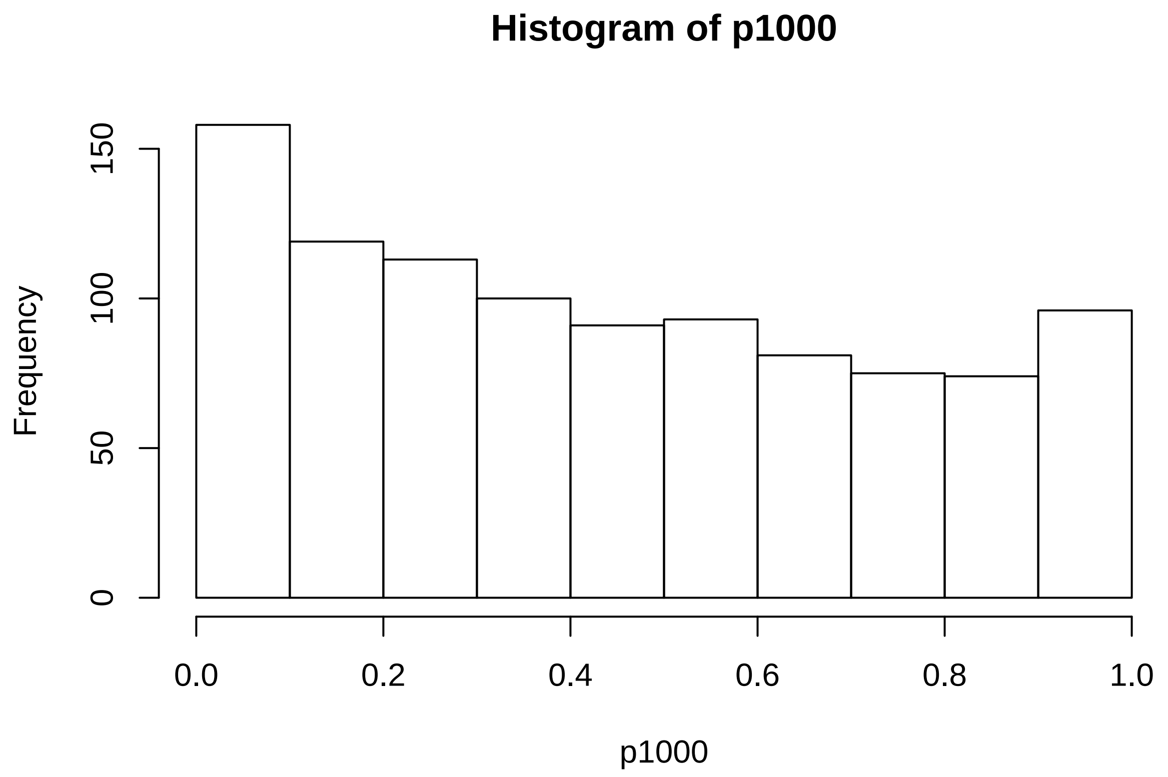
```
## [1] 0.4083753
```

More generally I would re-write this to allow more flexibility in what values I include for my predictor (x). Here it is always between 1 and 10, and just how many (evenly spaced) observations I should include. You could also use `x <- rnorm(sample_size, some_mean, some_var)` instead!

- 6B Now with your new function, run it 1000 times and generate a histogram of the p-values from these simulations? Check what proportion of times the p-value was less than 0.05.

Answer 6B

```r
p1000 <- replicate(1000,
                   regression_sim(a = 0.5, b = 0.1, rse = 2, sample_size = 20))

hist(p1000)
```



```
sum(p1000 <= 0.05)/length(p1000)
```

```
## [1] 0.1
```

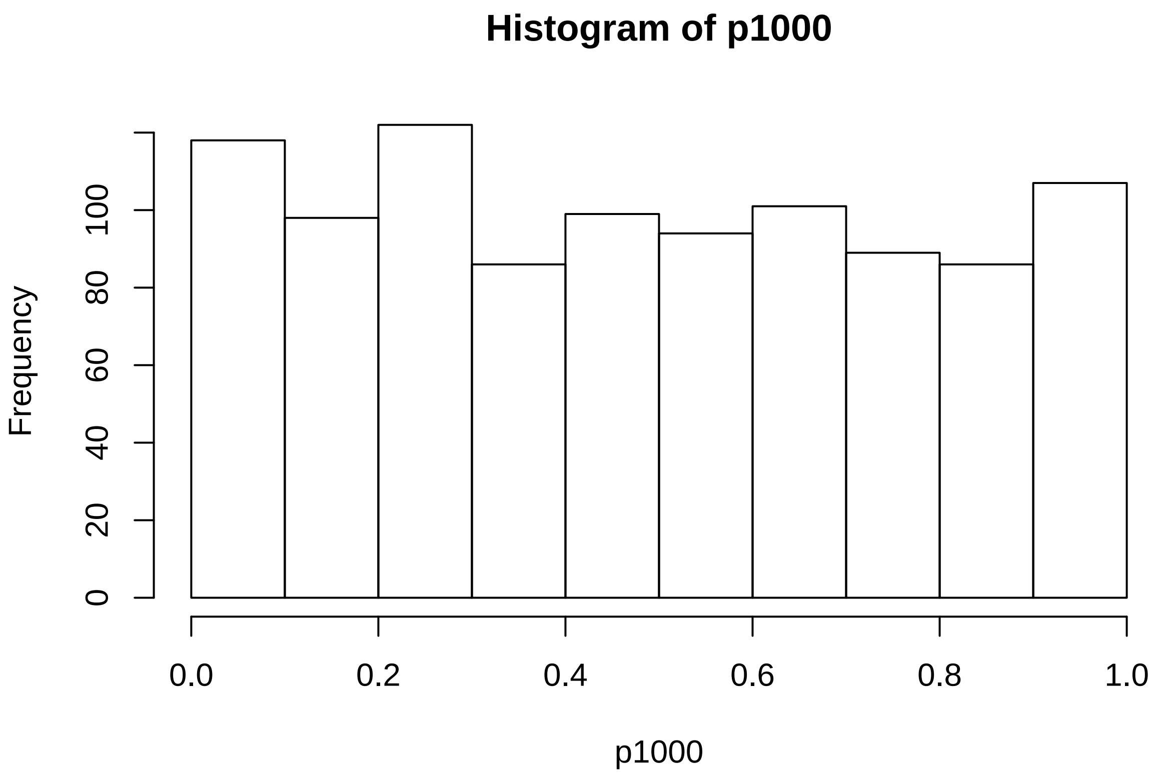
It was "significant" with a $p < 0.05$ about 9% of the time. Consider that we decided that the slope is non-zero (0.1) so why is it not significant each and every time? Relatively low statistical power because of modest sample size, small magnitude of the effect (slope) and high level of sampling variation (rse) mean that most of the time you can not detect an effect!

Of course statistical power based on just a p-value is not a useful way to really go, and this was more to illustrate a computational point, not a statistical one!

- 6C Redo this, but change the slope to 0. Examine this histogram and determine the proportion of times that the p-value is less than 0.05. Explain this result.

Answer 6C

```
p1000 <- replicate(1000,
                   regression_sim(a = 0.5, b = 0, rse = 2, sample_size = 20))

hist(p1000)
```



```r
sum(p1000 <= 0.05)/length(p1000)
```

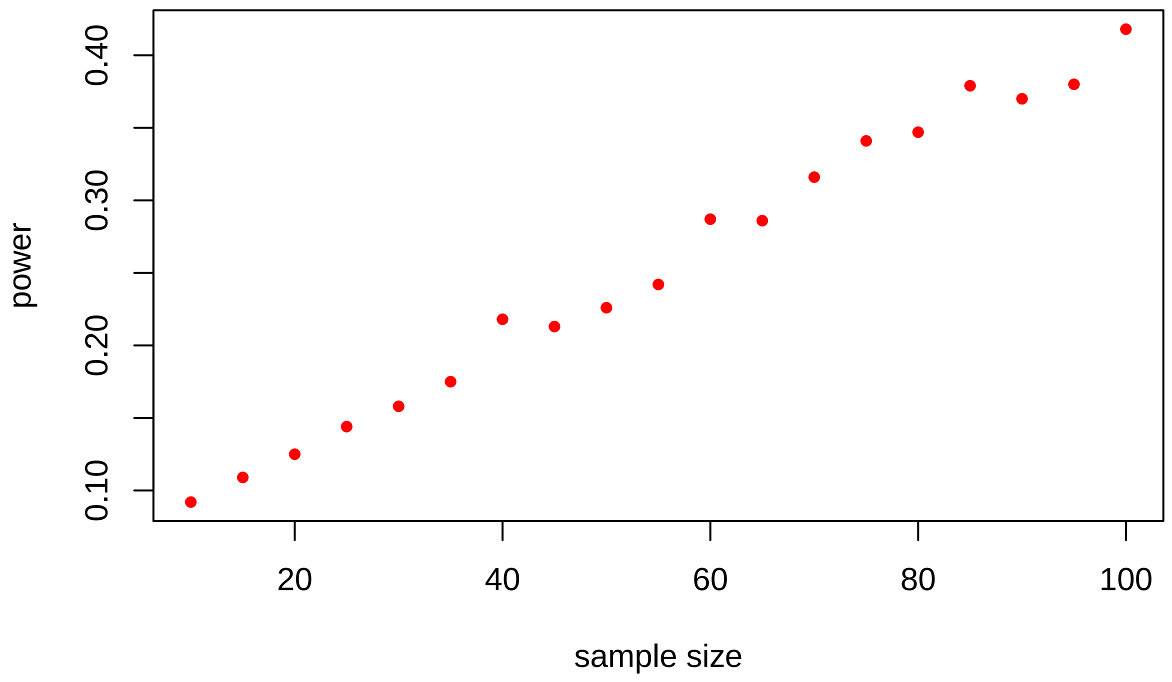
```
## [1] 0.061
```

With a "true" (expected) value of the slope at 0, the only "significant" with a p less than 0.05 should occur 5% of the time (by definition). So this is just confirming random sampling for this simple design!

- 6C Finally, using either a for loop or an *Rish* method (i.e. one of the apply family of functions) try a *grid* of sample sizes from 10 to 100 by 5 (i.e. 10, 15, 20. . . ., 95, 100) with the slope being fixed at 0.1, intercept = 0.5 and residual standard error as 1.5. At each different sample size, run the simulation 100 times and report the frequency at which the p value is less than 0.05. What pattern do you observe. Plotting the proportion of p-values less than 0.05 on the Y axis VS sample size (X axis) may be very helpful

Answer 6C

```r
sample_sizes <- seq(from = 10, to = 100, by = 5)

pow <- rep (NA, length(sample_sizes))
for (i in 1:(length(sample_sizes))) {
   p <- replicate(1000,
             regression_sim(a = 0.5, b = 0.1, rse = 1.5, sample_size = sample_sizes[i]))
  pow[i] <- sum(p < 0.05)/length(p)}

plot(pow ~ sample_sizes,
     ylab = "power", xlab = "sample size",
     pch = 20, col = "red")
```



While your power to detect a “statistically significant” effect of slope (i.e. that your estimate of the slope exceeds your uncertainty in the estimate in terms of a t-statistic) increases with sample size, even with 100 individuals you can only detect this significant effect 40% of the time! Mostly because our slope in reality is small relative to the variability.

Try making the slope bigger or rse smaller, and see what happens!