PGA Assignment

404018

April 2021

1 Wright Fisher simulation

1.1 Build a Haploid simulator

I built the simulator using a repeated binomial draw starting at $p = 1$ and ran the simulation 1000 times. I used a new consecutive seed for each run to make my results reproducible. Once n fixed or was lost I continued the simulation up until $t = 500$ with $n = 0$ or $n = 100$ for loss and fixing respectively.

I found that the number of times it fixed was 10, the mean time to fixation was $t = 199.3$, and the mean time to loss was $t = 9.84$.

1.1.1 Plots for haploid Wright Fisher

Next I plotted the M_i, T_i for situations where there was fixation and loss, and a density plot of $C(n, t)$. I plotted the density plot as a heatmap and overlaid the individual paths so this could be clearly viewed.

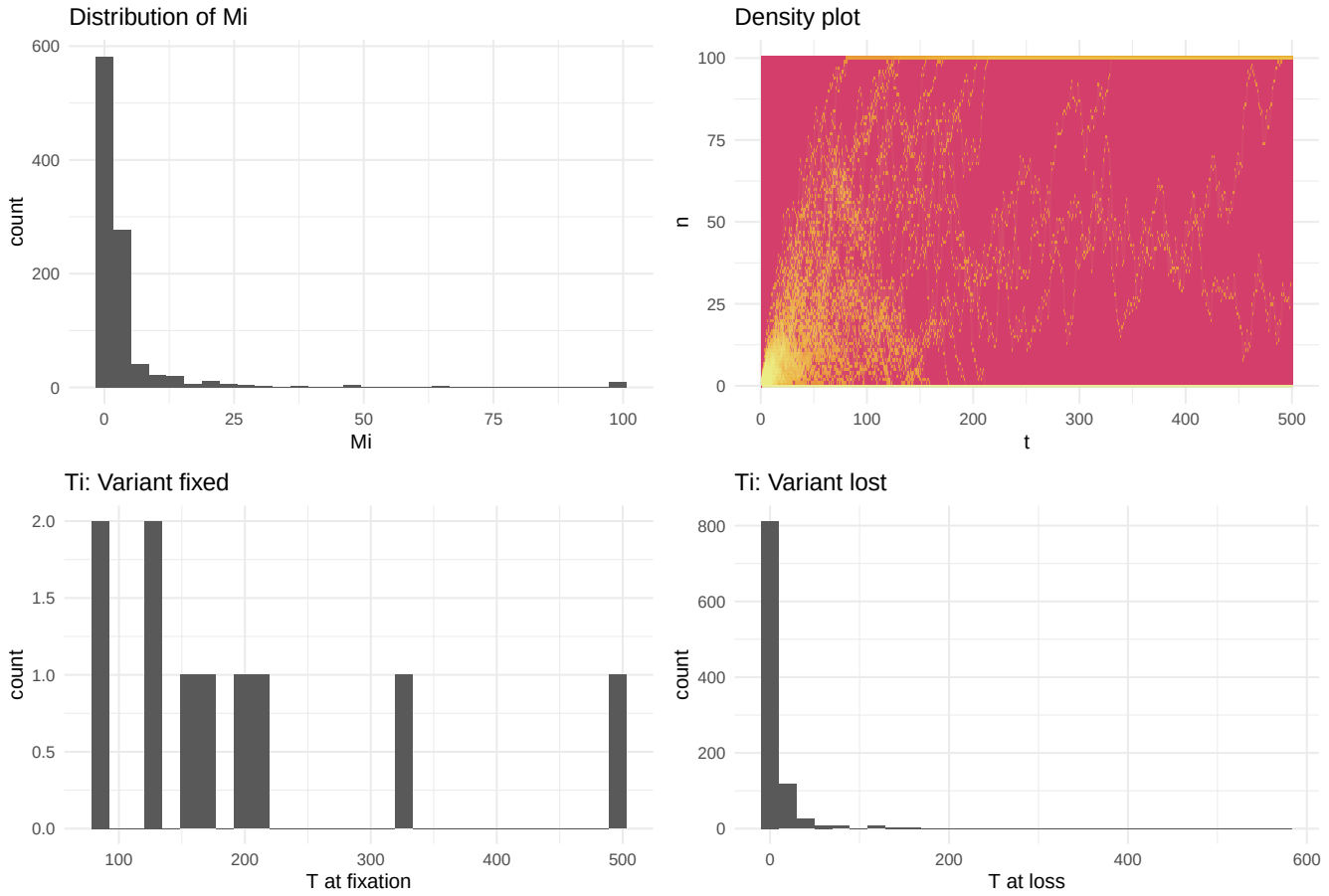

Figure 1: Plots of M_i, density plot of $C(n, t)$ and T_i for loss and fixation. For the density plot the paths of the simulations are overlaid so that the exact paths .

Here you can see (fig 1, top left) the plot of M_i for the model. You can see here that in the majority of simulations the mutation does not fix, and the maximum n reached is low, clustering around 1 and 2. This is unsurprising, as the expectation of p‘ is p, and as $p = 0.01$ initially we would expect p to either be lost immediately or drift around

low values before being lost. We can see this behaviour in the bottom left corner of the density plot (fig 1, top right). It is also to be expected that there are very few cases where $50 \leq M_i \leq 99$. This is again unsurprising as few simulations had $M_i \geq 50$ and when $p > 0.5$ the expectation would be that these simulations would fix as they are more likely to reach $n = 100$ than fall back to 0. This behaviour can be seen in the density plot; once a simulation has risen above $n = 50$ the majority continue up to fixation. This explains the small peak at 100, as these mutations we would expect to rise to 100 and fix, leaving a small peak.

Similarly, for the plot of T_i for loss (fig 1, top bottom right) most simulations are rapidly lost causing most of the density to appear at low values of t for the plot of T_i for loss (again seen by the high density at the bottom left of the density plot). There is however a long tail. This can be accounted for by variants that rise to values of $n \approx 50$, as from this point as $E(p') = p$ once higher values of n are attained then there would be an expected longer period of drift. We can see this in the lines in the lower right quadrant of the density plot.

By contrast, for the plot of T_i for simulations where the variant fixed (fig 1, bottom left) the values start at around $t = 100$. This is seen in the density plot as lines slowly rising up to the top of the plot. We would expect there to be a large number of steps to move from $p = 0.01$ to $p = 1$, so this distribution is to be expected. We can also see that a total of 10 simulations fixed - this is to be expected as $p_{fix} = p$ for the Wright Fisher model without selection, and here 10 out of 1000 simulations have fixed, so in this case exactly 0.01 of the simulations fixed.

In addition the expected variance under Wright Fisher is $\frac{pq}{N}$ and this can be seen in the density plots; there is a low variance (how quickly the path moves) for small values of p and q but when $p \approx q$ the there is significantly more variability in n from generation to generation.

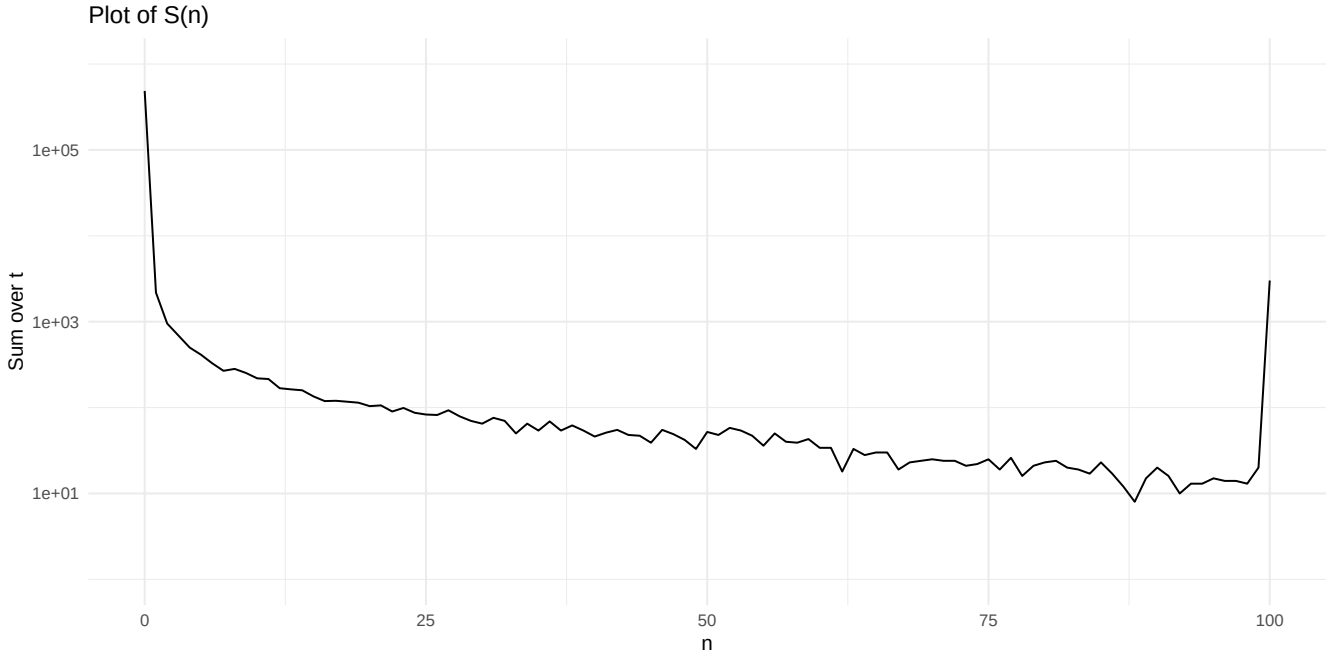

Figure 2: Plot of distribution of $S(n)$, with y axis log scaled.

The plot of $S(n)$ is shown in fig 2. This plot represents the sum across all values of t for each n. It could be thought of by taking a snapshot in time of a population. For a population of 100 individuals ($N = 100$) and 1000 loci (the number of simulations) this plot would represent the number of loci at each variant allele frequency (for every n on the x-axis). To then sum over all t we can consider taking repeated snapshots over time (500 snapshots in total for $t = 500$) and then summing up these curves. For example, in a population of haploid cells such as bacteria, if you sequenced their genomes once every generation (assuming that every locus of interest is independently inherited, that there is no selection at any locus, that the population is in equilibrium, that every individual is replaced at each generation, and that every variant allele starts at $p = 1/N$) and at each snapshot plotted the number variants at each value of n, adding up the curves at every generation we would get a plot that appears like this. This sort of plot is useful for investigating the dynamics of drift in a population as it summarises all of the other plots above.

1.2 Haploid Wright Fisher with selection

Next I considered the behaviour of the simulations when selection was applied. I used the same model as in the Wright Fisher without selection but instead of using $\frac{p}{N}$ for the expectation of the binomial I used $\frac{p}{p+(1-s)(1-p)}$. I then made the plots as above for all different levels of selection (fig 3).

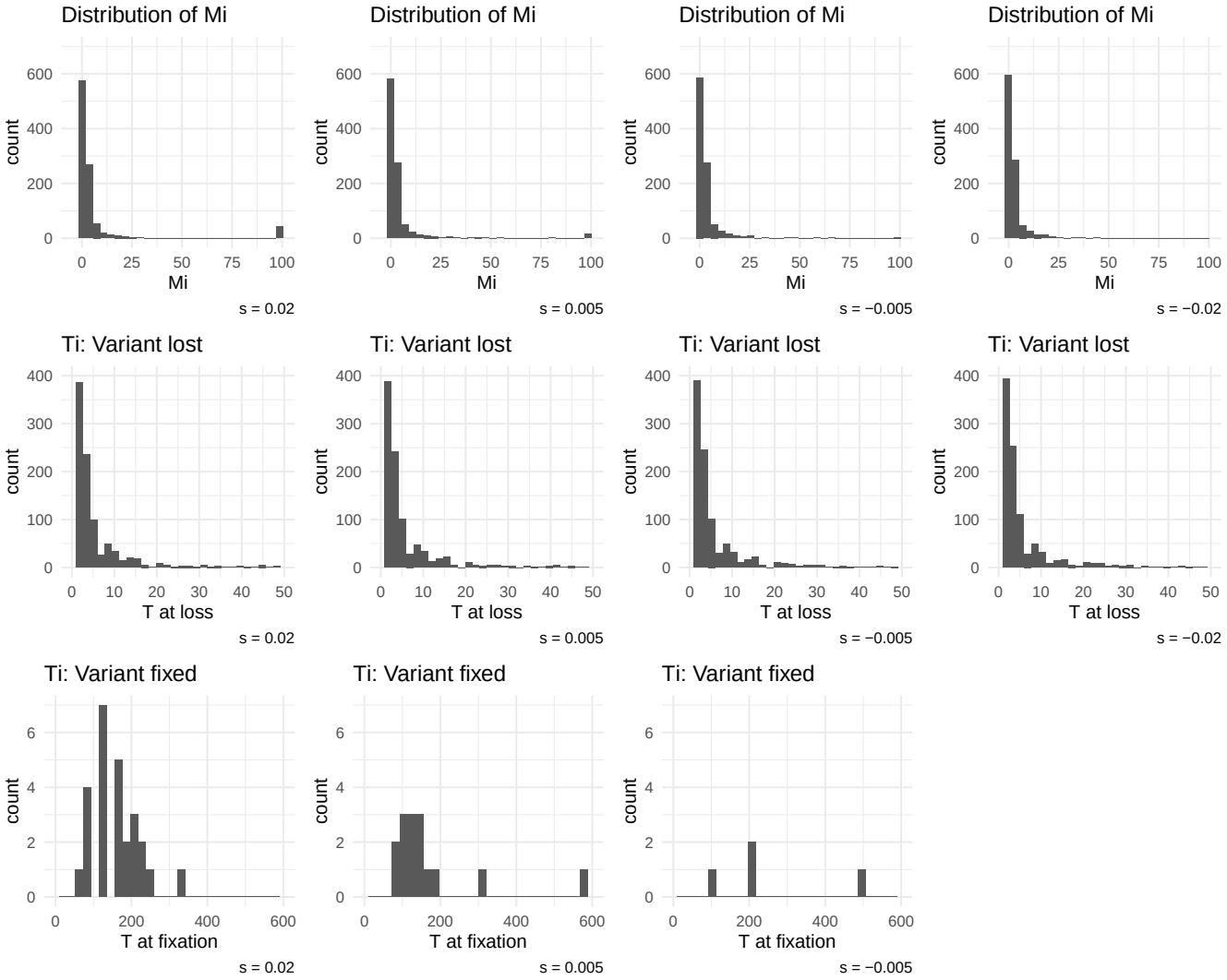

Figure 3: Plots of M_i, T at loss and fixation for different levels of selection. I have not included the plot of T at fixation for $s = -0.02$ as none of the simulations fixed at this level of selection.

Here you can see that for increasing positive selection (left side plots) that more simulations fixed, meaning that the peak at the left of the plots of T_i for loss (second row) there is a lower peak at the left for increasing positive selection, and a higher peak for increasing negative selection. However the plots are very similar as the mutations which do not fix behave quite similarly. There is also a shift to the right for the plots of M_i for increasing positive selection. This is all unsurprising, as the positive selection biases the binomial draw in the Wright Fisher model to increase the chance of fixation. The plots of T_i for gain show that increasing positive selection causes an increased number of variants to fix and that fixation happens more rapidly. This is again as expected as the biasing of the binomial draw means that n is more likley to increase.

As we know that $\pi_1 = \frac{1-e^{-s}}{1-e^{-Ns}}$ we can calculate the values for different levels of s: when $s = -0.02$, $\pi_1 = 0.00076$, when $s = -0.005$ $\pi_1 = 0.00584$, when $s = 0.005$ $\pi_1 = 0.0157$ and when $s = 0.02$, $\pi_1 = 0.0399$. This is mirrored in the data: 26 simulations fixed when $s = 0.02$ while none fixed when $s = -0.02$.

Selection is less effective below $s = \frac{1}{N}$ so for the cases where $s = \pm 0.005$ the simulation behaved more like a drift regime - this is clear from density plots (4). However for $s = \pm 0.02$ selection predominates: the strong negative selection caused no variants to fix whilst strong positive selection caused many more variants to fix. However even in the case of $s = 0.02$ most advantageoous new mutations are lost. If $s > \frac{1}{N}$ and $s << 1$ then $\pi_1 \approx s$: this is the case for $s = 0.02$. From these plots we can see that in the regime with selection using the equations for the expectations for Wright Fisher with selection the plots appear as expected.

Note that I haven't included the plots of $S(n)$ here; these are included the question 2 (fig 7). They show a progressive shift to the right as positive selection is increased: this would be expected from the behaviour with increasing positive selection as above.

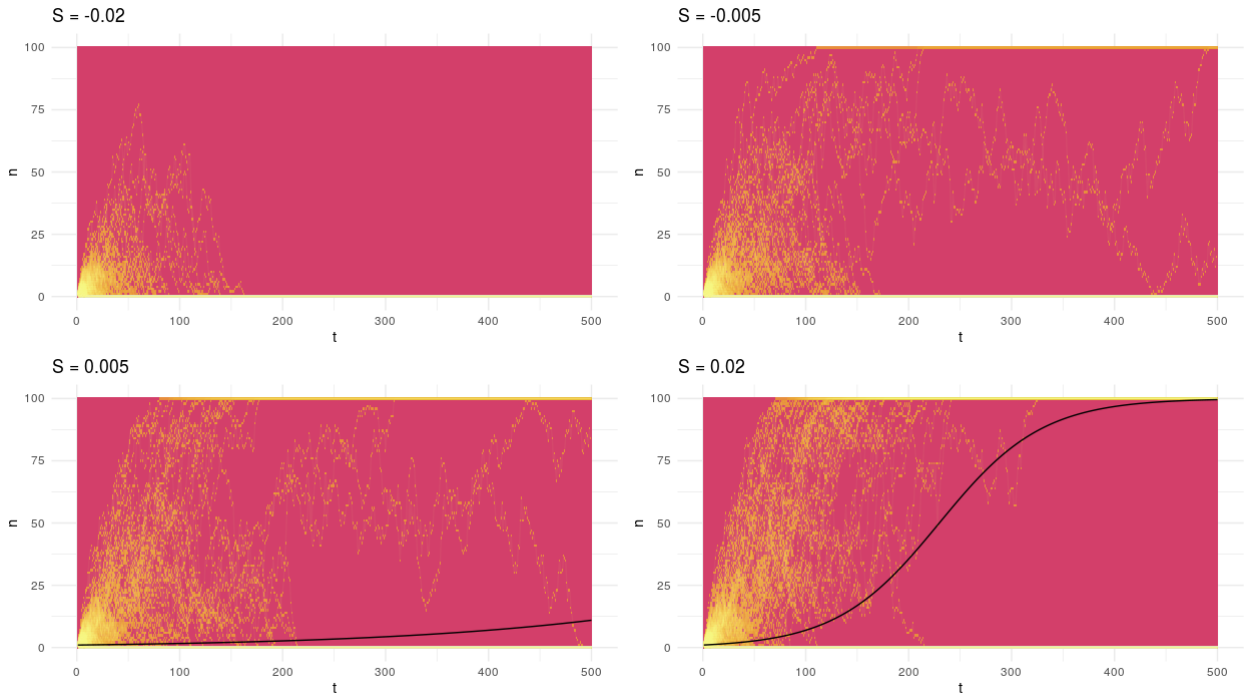

Figure 4: Heatmaps of selection with deterministic lines overlaid. Note that this line is not overlaid for negative selection as it immediately falls to 0.

1.3 Diploid Wright Fisher

Next I repeated the simulation for Wright Fisher with a diploid model, with N = 50. I used the expectation of the binomial draw to be $\frac{p^2+p(1-p)(1-hs)}{p^2+2p(1-p)(1-hs)+(1-p)^2(1-s)}$ with $2N$ binomial draws.

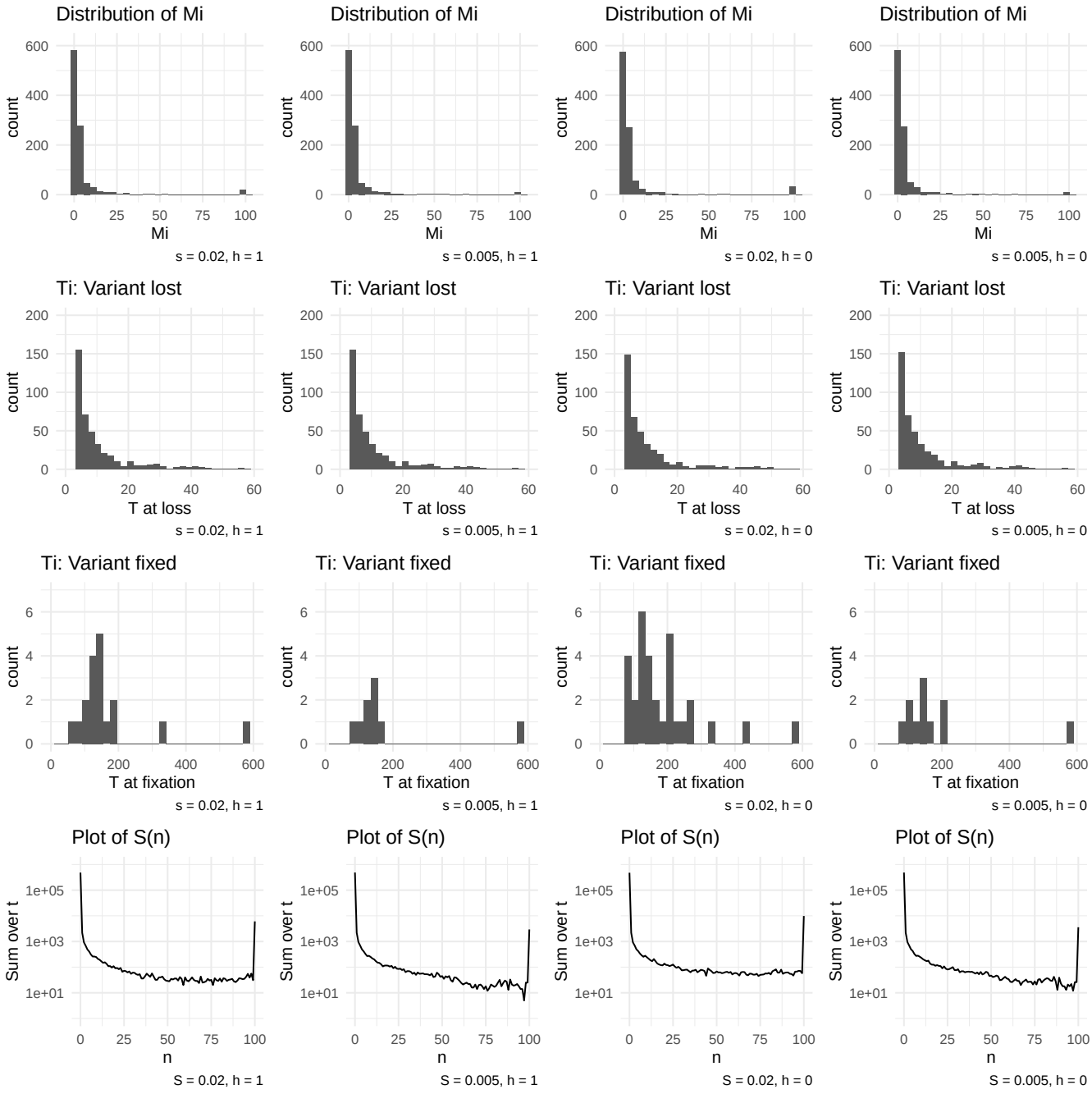

Figure 5: Plots of M_i, T_i at loss and fixation and $S(n)$ for different s and h for the diploid Wright-Fisher simulation

For these plots (fig 5) we can see that as above more variants fixed when selection was stronger. However we can also see that h plays an important role. When $h = 0$ (variant effect dominant) the variant was more likely to fix than when $h = 1$, however it also took longer to get there, with the plots of T_i for fixation shifting to the right for $h = 1$. This is to be expected from the interaction of the two variants: when n is small then the variant allele has more of an effect under selection in the case of a dominant variant as selection is exerted on the heterozygotes as well as the homozygotes. In the effect of the recessive effect the selection is only exerted on homozygotes, so the variant must reach a threshold ($p > \frac{1}{\sqrt{N}}$, in this case $p = 0.1$) where there are enough homozygotes with both copies of the variant before the selection effect occurs. This means that below this threshold drift predominates, but for $s = 0.02$ above this threshold selection predominates.

However close to fixation the reverse occurs: in the case where the variant is dominant then above a certain threshold ($p > 1-\frac{1}{\sqrt{N}}$) the majority of individuals with the wild-type variant are heterozygous and so still experience the positive selective effect (there is effectively a reservoir of wild-type deleterious effect recessive alleles). This means

that drift predominates and so it takes longer for the variant to fix in these cases. By contrast in the case where the variant is recessive at p close to 1 the wild-type is dominant so any individuals with a wild-type allele are selected against, meaning that the although recessive variants are less likely to be selected for they then fix quicker once they are above the threshold.

This behaviour is clear from the density plots (fig 6): for the recessive variant effect once it rises above a certain level fixation happens quickly but for the dominant variant effect it takes much longer for the paths to fix.

We can also see this from the plots of $S(n)$: for the plots where $h = 0$ there is an increase in density on the right had side of the curve (more pronounced for $s = 0.02$ than $s = 0.005$) in the plots relative to the plots where $h = 1$, where drift predominates meaning the more time is spent in that region before fixation occurs.

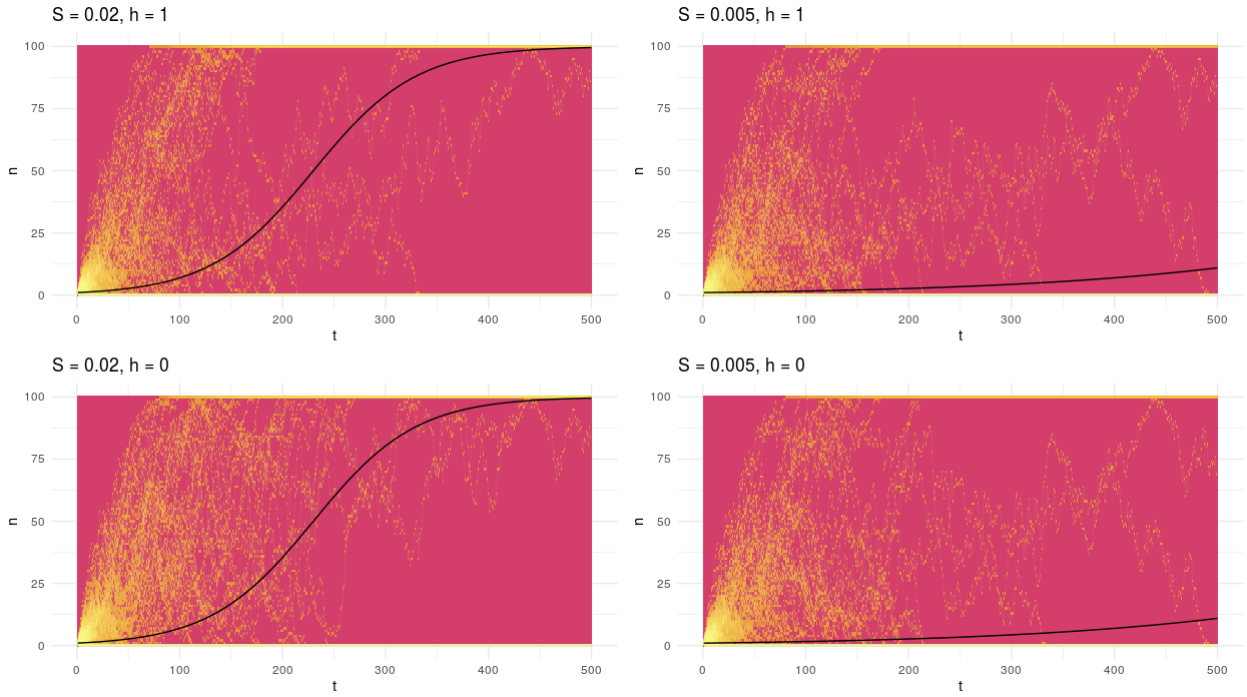

Figure 6: Heatmaps of selection for diploid Wright Fisher model with haploid deterministic lines overlaid

2 Estimating selection from the curve of the sum

These plots (fig 7) show us the expected amount of time spent at any given variant allele frequency in the Wright Fisher model with selection. They show that for negative selection of the variant more time is spent at lower variant allele frequencies whilst the curve shifts to the right with increasing positive selection, with progressively more time spent at higher variant allele frequences and $VAF = 1$ at more positive s.

We can calculate $\pi(p)$ (p_{fix}) from the line by comparing the ratio of the numbers of simulations which fix with the ratio of those that do not. However we can also use the other points in the line to get a more accurate estimate of $\pi(p)$ from which we can calculate s. To do this I took a weighted mean of $S(n)$: $\sum_{n=1}^{100} \frac{S_n n}{100} / \sum_{n=1}^{100} S_n$. This gives the $\pi(p)$ which we can use then to calculate s. To compare the estimates against the true calculated $\pi(p)$ (using the equation $\pi_1 = \frac{1-e^{-s}}{1-e^{-Ns}}$) I plotted them against each other (fig 8): we can see that we get an accurate estimate of $\pi(p)$ and therefore s using this method of estimation. Using this method and the repeated snapshots of a genome over time, for example as suggested in the assignment by comparing amino acid substituions it would be possible to accurately estimate s for a set of loci in an organism.

However as above there are a number of assumptions that have to be made for this approach to be used, including that they are independently inherited, and that every generation is fully replaced at every time point. There aren't enough chromosomes for all genes to be independently inherited (no organism has 1000 chromosomes) so there will always be some linkage of haplotypes. In addition, very few organisms would have a breeding cycle perfectly in sync such that every generation is completely and perfectly replaced by the next one.

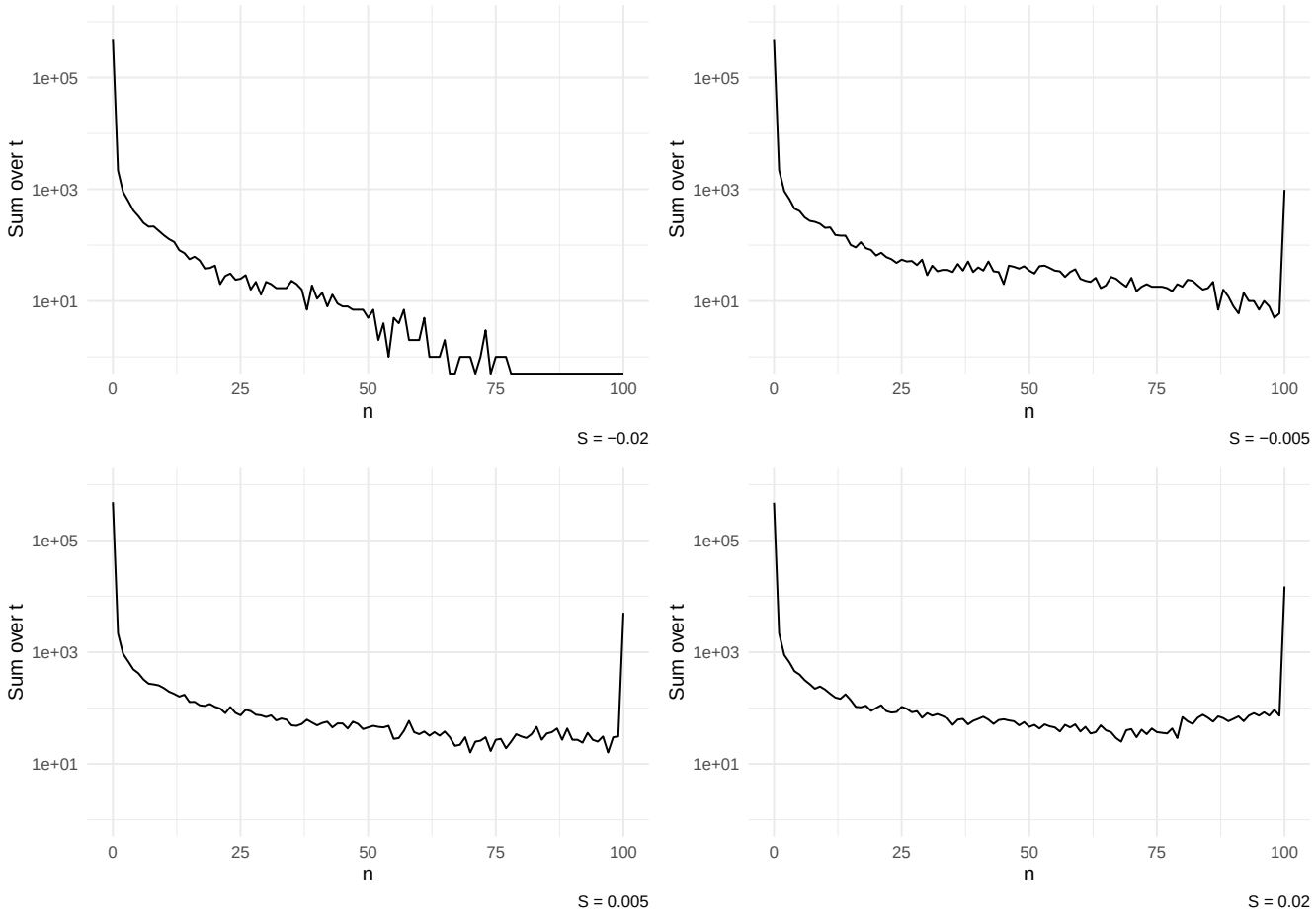

Figure 7: Plots of S(n)

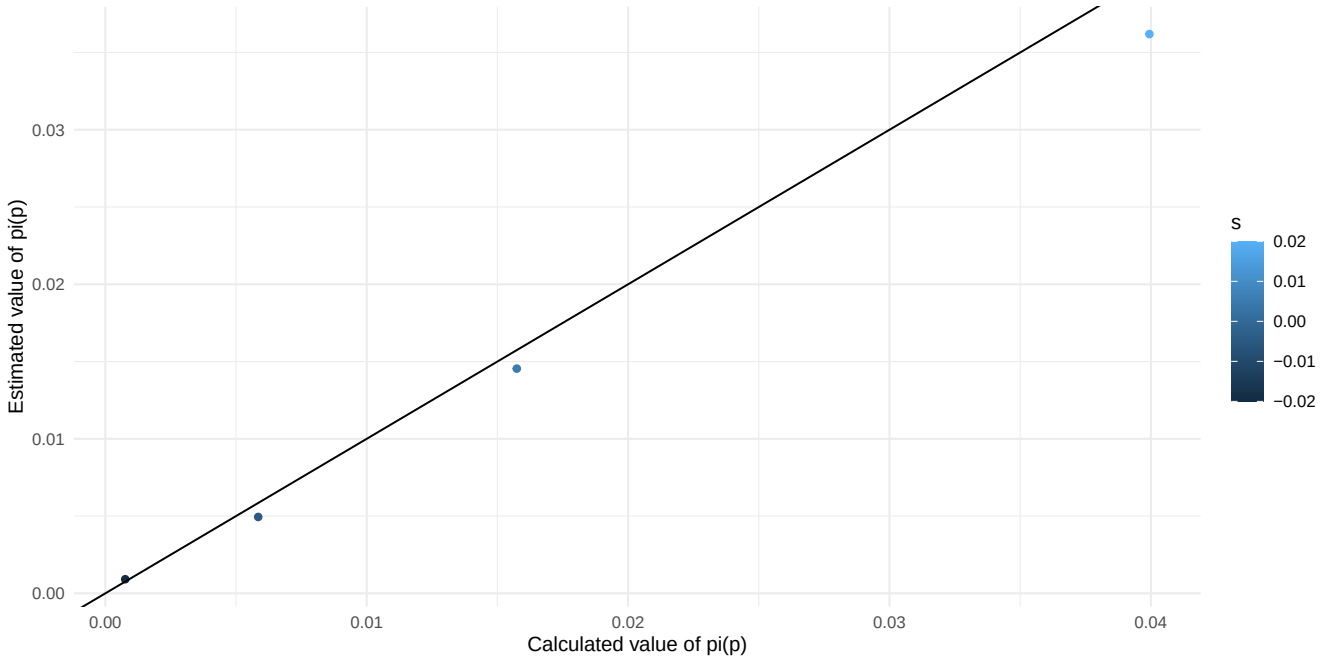

Figure 8: Estimating $\pi(p)$ using the method explained above. Calculated values of $\pi(p)$ using the equation are on the x-axis whilst the estimates using the method explained above are plotted on the y-axis. The black line is where the estimated and calculated values would be equal.

3 Calculating allele frequencies given known population frequencies

Please see all workings in the appendix in the code under HWE. I found that the allele frequency of S was 0.0908, the expected genotype fractions were 0.827 for AA, 0.165 for AS and 0.00825 for SS. $\chi^2 = 167.7$.

I found that the probability of survival when $N = 40000$ infants born under HWE for AA was 0.767, for AS was 0.83 and for SS was 0.203. I therefore caculate s to be 0.735 for A with respect to S, and h to be -0.110957. This shows that there is in fact an over dominance behaviour occuring for S and A, where heterozygotes are selected for, but that A is strongly selected for over A. I found these values to be invariant of N: this is unsuprising as when calculating W_{SS} and W_{AS} you normalise by W_{SS} which cancels out the effect of N, and $s = 1 - W_{SS}$, $h = \frac{1-W_{AS}}{s}$:

$$W_{SS} = \frac{HbSS}{q^2 \times N\frac{HbAA}{p^2 \times N}} \tag{1}$$

Where $HbSS$ is the number of individuals homozygous for S, $HbAA$ is the number of individuals homozygous for A, q is the allele frequency for S and p is the allele frequency for A.

3.1 Reasons for errors in calculation

There are a number of reasons why an error may occur when making these estimates. As this assumes Hardy-Weinberg equilibrium this must meet the assumptions HWE: the population must be in equilibrium (equally mixed and equally likely to breed with another individual), that there is no selection in who is able to mate with whom, a very large pool of people, no new mutations in this locus and no movement of people. In fact patients with sickle-cell anaemia are more likely to have relationships with other people with SCA or sickle-cell trait for cultural and social reasons. These estimates could be corrected by determining the number of children born to each group and then the probability of surviving until they are able to have children. Sickle-cell anaemia has no effect on the fetus as fetal haemoglobin is coded for at a different locus so considering survival from birth to having children should be sufficient to detect selection pressures.

4 Simulating fluctuating selection

I simulated fluctuating selection. I initially set the f_i^a for each gene (the positive selection for state 1 vs state 0) using an exponential distribution with mean 0.01. These remained fixed throughout the simulation. Next I calculated the probability of fixation with the equation:

$$\pi_1 = \frac{1 - e^{-2f_i^a}}{1 - e^{-2Nf_i^a}} \tag{2}$$

I then chose genes at random and sampled from this distribution so that the probability of fixing was π_p and the probability of a mutation being lost was $1 - \pi_p$. When a gene was in state 1 the fixation probability (ie the probability of reverting to state 0) was calculated using the equation:

$$\pi_1 = \frac{1 - e^{2f_i^a}}{1 - e^{2Nf_i^a}} \tag{3}$$

I continued simulating for 10^6 iterations of this process.

I then repeated this for changes in selection. To do this I sampled from a poisson distribution with λ equal to the set rate of change, and used the poisson random variable to determine whether or not the fitnesses were reversed. I plotted the number of genes in state 1 over time for these different scenarios:

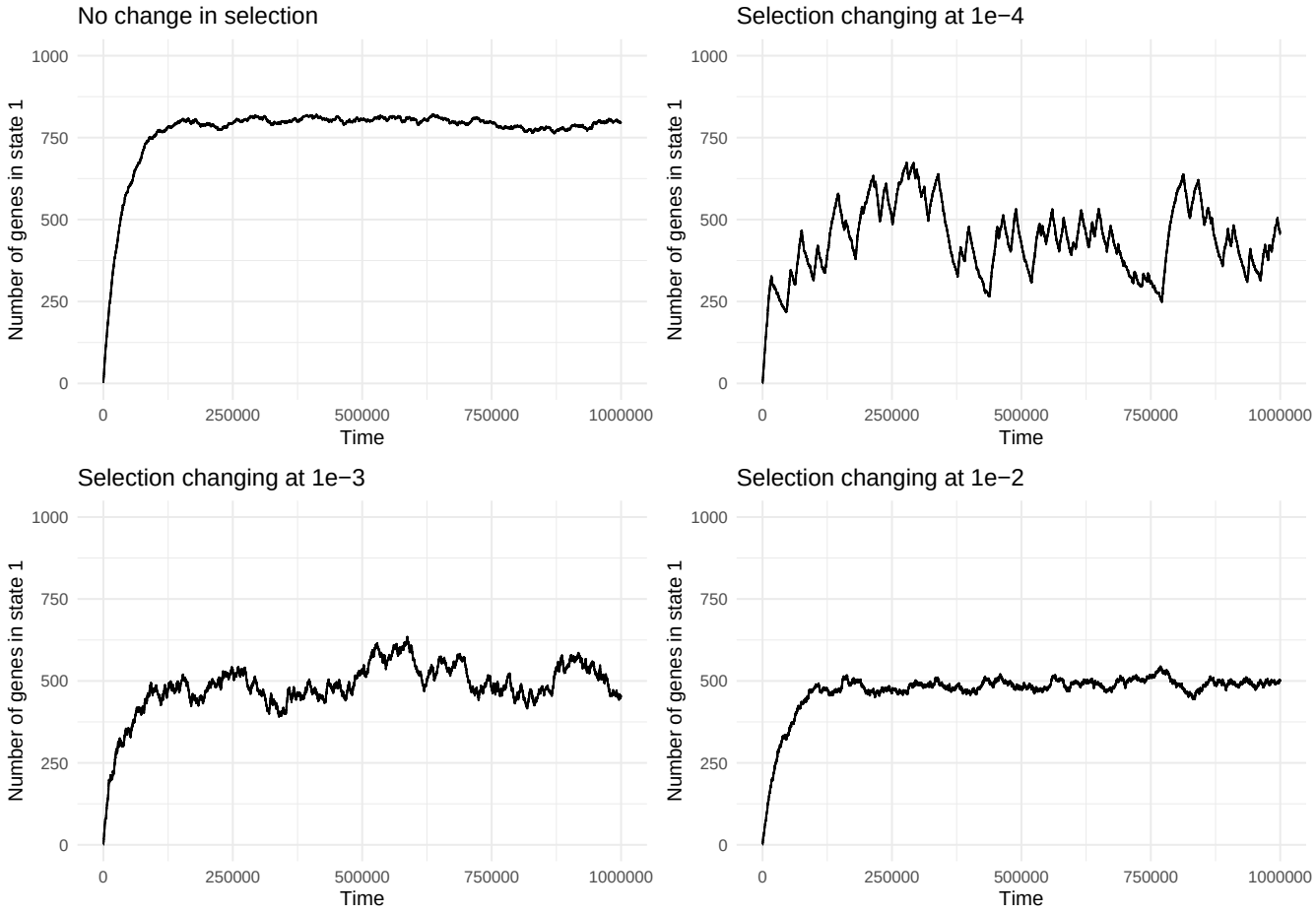

Figure 9: Plots of number of genes in state 1 over time with and without changing selection

Here you can see that without changes in fitnesses the number of genes in state 1 equilibrates around 800. However with changes in selection caused the equilibrium to shift to 500, with a higher rate of change of selection causing less fluctuation around the equilibrium state.

I considered the fitnesses of the genes that fixed in state 1. I considered predicted probability of being in state 1 vs state 0 as given by a logistic regression and plotted this as a curve (fig 10 left). I also recorded the mean amount of time in the state which was selected for after equilibration (which I considered to have occurred at 2e5 iterations), which I plotted (fig 10 right).

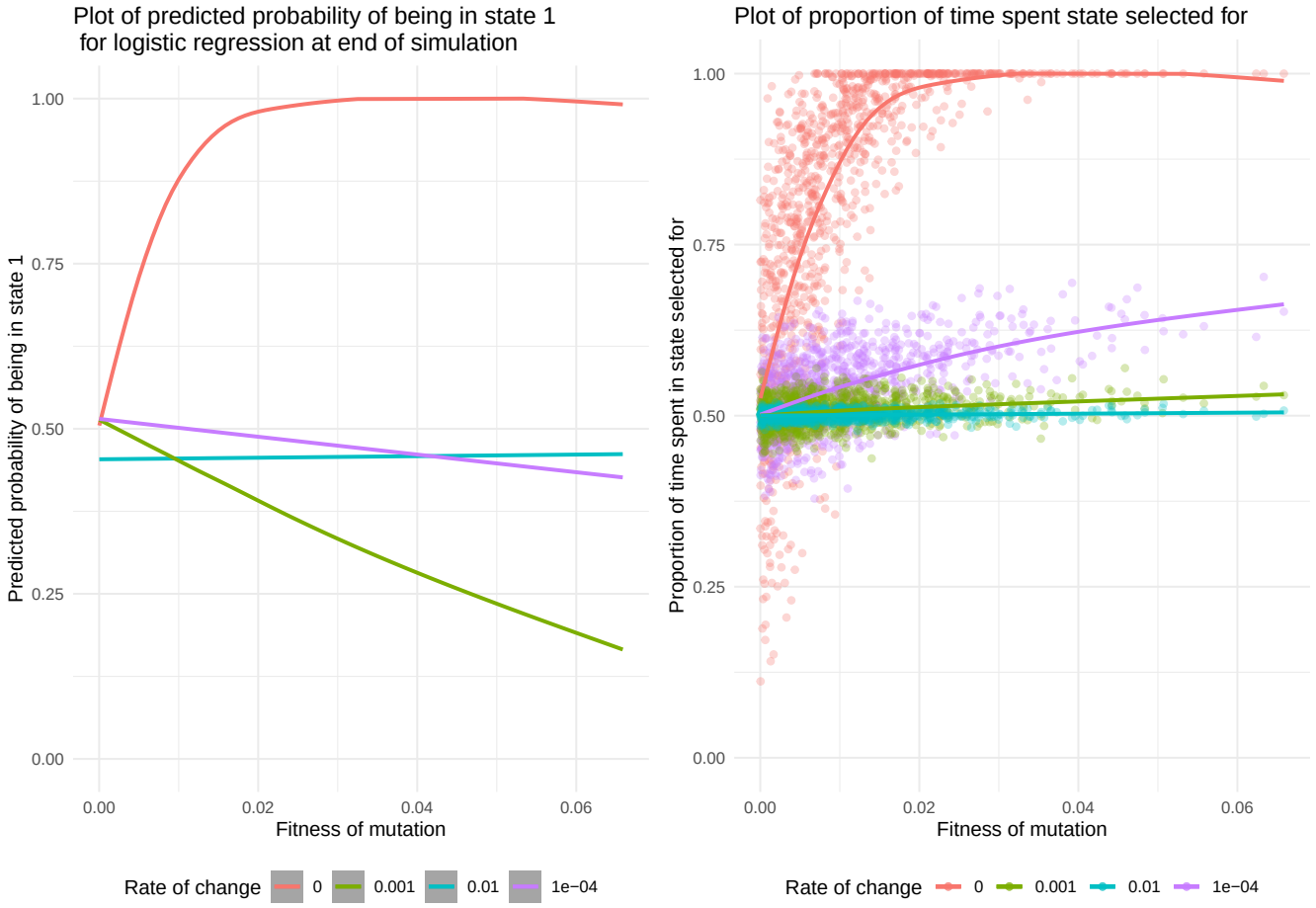

Figure 10: Plots of number of genes in seleceted for state over time with and without changing selection

Here you can see that in the case where there is no change in selection increasing fitness was positively correlated with being in state 1, with the fittest mutations spending the most time in state 1. The logistic regression performed on the states and fitnesses at the end of the simulation mirrored this: when there was no change in selection the fitness of mutation was positively correlated with likelihood of being in state 1. However when changes in selection were introduced there was a reduced correlation between fitness and time spent in the state where the selection was positve. Of note I corrected for whether the state that the gene was in was the one that was selected for (when calculated the mean time spent in state 1 I reversed the states when there was a reversal of selection pressure). As the rate of change of selection increased the correlation between fitness and time spent in the state which was selected for reduced. However it is worth noting that for the case of selection changing on average every 10^{-3} unit time the logistic regression showed a negative correlation between time fitness and time spent in state 1. This is likely a stochastic effect and we can see from the averaged values that this effect disappears when averaged over time. These plots show that using a logistic regression to check whether fitness is correlated to time spent in a certain state is a good proxy for measurement over time as the logistic regression approximates the proportion of time spent in state 1.

In addition for the scenario where there is no change in selection pressure we can see that there is a change in behaviour occuring at around $f = 0.01$ which is the point at which $f > \frac{1}{N}$. Above this, and more noticeably above $f = 0.02$ selection predominates, with genes where $f > 0.02$ spending the vast majority of the time in state 1, but below this drift behaviour dominates as $f < \frac{1}{N}$ in this case.

The reason for there being a large proportion of beneficial mutations in drosophila may stem from the fact that conditions change rapidly over time for drosophila, which have a lifespan of 70 days. This means that conditions can change rapidly with a single season per generation. Mutatations which are beneficial in the summer may therefore be deleterious in the winter. It means that mutations which are beneficial may not fix in the population, as the selection pressures fluctuate so rapidly. If beneficial mutations fix in the population then they would not be seen as beneficial as they would no longer be mutations but become part of the reference sequence. This constantly fluctuating selection would mean that beneficial mutations do not fix and therefore this variation would still appear in the drosophila genotype, meaning that many mutations seen would appear as beneficial. By contrast in humans the vast majority of mutations seen are deleterious as the majority of beneficial mutations have fixed in the population.

Table 1: Table with internal nodes and leaves included

internal_node	daughter_node1	dist1	daughter_node2	dist2	Internal_nodes_below	Leaves_below
9	2	2.7	3	2.7	0	2
10	1	1.8	9	4.5	1	3
11	4	1.1	5	1.1	0	2
12	11	1.9	6	3.0	1	3
13	10	1.5	12	3.0	4	6
14	7	4.7	8	4.7	0	2
15	13	2.2	14	3.5	6	8

Table 2: Table of distances from leaf to root

leaves	distances
1	5.5
2	10.9
3	10.9
4	8.2
5	8.2
6	8.2
7	8.2
8	8.2

5 Felsenstein's Pruning Algorithm

I built a function to read in the table and compute the number of leaves and internal nodes below each internal node (see table 1). I also computed the distances to the root for each leaf (see table 2). We can see from this table that there are different distances from the leaves to the root showing that this table is not consistent with an evolutionary tree where branch lengths represent times and all the leaves are in the present time, with a constant mutation rate. However if there were local differences in mutation rate then this may be possible.

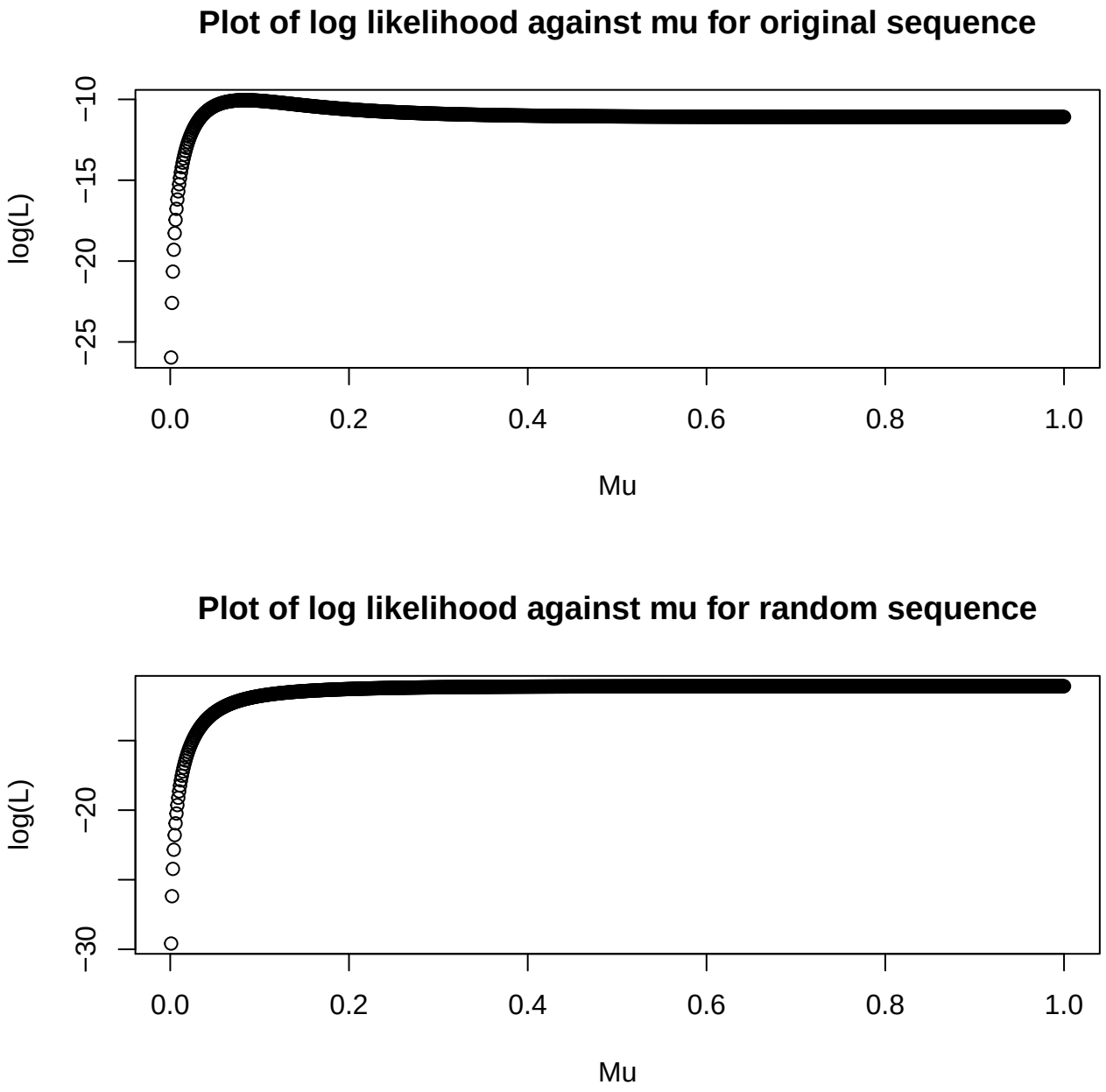

Figure 11: Plots of mu against log(L) for the original sequence and a random sequence GCATATGA

Here we can see the different log likelihoods for this tree accross different μ (fig 11). The maximum likelihood estimate of μ for the original sequence was 0.084 and the MLE for the random sequence was 1.

6 Appendix with code

```
library(ggplot2)

## Question 1

#Build a wright fisher simulator
simulate_wf <- function(N, seed = 1, s = 0) {
  #Function to make reproducible Wright-Fisher simulator
  set.seed(seed)

  #Set up some storage
  c_nt <- data.frame(n = NA, t = 1:500)
  m_max <- 1
  n <- 1
  t <- 1

  #Run simulator until fixation or loss
  while (n %in% seq(1, N-1)) {

    #Store previous n
    if (t < 500) {c_nt$n[t] <- n}

    #Set probability (considering selection)
    p <- n/N
    binom_prob <- p*(1/(1*p + (1 - s)*(1 - p)))

    #Do binomial sim
    n_prime <- rbinom(N, 1, binom_prob)
    n <- sum(n_prime)

    #Update m_max
    if (n > m_max) {m_max <- n}

    #Update t
    t <- t + 1

  }

  #Ensure rest of ns in c_nt are the same as last value
  if (t < 500) {c_nt$n[t:500] <- n}

  return(c(n, t, m_max, c_nt$n))
}

haploid_wf <- sapply(1:1000, simulate_wf, N = 100, s = 0)

num_fixed <- which(haploid_wf[1,] == 100)

#Plot distribution of T where variant fixes and is lost -
#Plot distribution of m_max

#Consider scaling by proportions
plot_fixed_not <- function(df, fixedT = T, sub = '') {
  fixed <- which(df[1,] >= 100)

  if (length(fixed) > 0) {
    if (fixedT == T) {ggplot() + geom_histogram(aes(x = df[2,fixed]), bins = 30) + xlab('T at fixation')
                      labs(title = 'Ti: Variant fixed', caption = sub) + theme_minimal()
    } else {
        ggplot() + geom_histogram(aes(x = df[2,-fixed]), bins = 30) + xlab('T at loss') +
```

```
                    labs(title = 'Ti: Variant lost', caption = sub) + theme_minimal()
      }
  } else {
    ggplot() + geom_histogram(aes(x = df[2,]), bins = 30) + xlab('T at loss') +
                    labs(title = 'Ti: Variant lost', caption = sub) + theme_minimal()
  }

}

make_heatmatrix <- function(input) {
  #Function to convert each column to matrix

  df <- input[4:dim(input)[1],]

  #Set up matrix
  c_nt_map <- matrix(0, nrow = 101, ncol = 500)

  #Loop over it to set squares
  for (index in 1:1000) {
    for (i in 1:500) {
      c_nt_map[df[i, index] + 1, i] <- c_nt_map[df[i, index] + 1, i] + 1

    }
  }

  c_nt_map
}

calc_fixation <- function(f_ai,  N = 100) {
  #Function to calculate fixation probability

  numerator = 1 - exp(-2 * f_ai)
  denominator = 1 - exp(-2 * f_ai * N)

  numerator/denominator
}

#solve_pip <- function(input) {
#  #Function to solve for pi_p
#
#  diff <- 1
#  start <- 0.02
#
#  while(diff > 0.0001) {
#    range <- calc_fixation(seq())
#
#  }
#
#}

plot_sums <- function(input, title = '', plotting = T) {
  #Function to plot the distribution of the sums

  #Make matrix
  mat <- make_heatmatrix(input)

  #Get sums - could colour by which ones fixed? - or do a cumulative density plot?
  #plot(apply(mat[2:101,], 2, sum), type = 'l', log = 'y', xlab = 't', ylab = 'count', col = 'green')
  #lines(y = mat[1,], x = 1:length(mat[1,]), col = 'red')

  sum_t <- apply(mat, 1, sum)
```

```
  p <- ggplot() + geom_line(aes(x = 0:100, y = sum_t)) + theme_minimal() +
    xlab('n') + ylab('Sum over t') + scale_y_log10(limits = c(1, 1e6))

  if (plotting == F) {sum(sum_t*seq(0, 1, 0.01))/sum(sum_t)


    } else {p}

}

plot_paths_gg <- function(input, title = '') {
  #Function to plot paths of the drift

  #Take only rows with paths in (remove top few)
  input_n <- input[4:dim(input)[1],]

  #Set rownames
  t <- rownames(input_n) <- 1:500

  #Get heatmatrix
  heat_matrix <- make_heatmatrix(input)
  heatmap_df <- data.frame(count = as.vector(unlist(heat_matrix)),
                           n = rep(0:100, times = 500),
                           t = rep(1:500, each = 101))
  my_breaks = c(1, 3, 10, 50, 100, 1000)

  #Make plot
  p <- ggplot() +
    geom_tile(data = heatmap_df, aes(y = n, x = t, fill=count + 0.01)) +
    theme_minimal() + scale_fill_gradientn(colours = colorspace::heat_hcl(7), name = "count", trans = "lo
                                                      breaks = my_breaks, labels = my_breaks) +
    labs(title = title)

  #Add lines
  for (i in 1:length(input_n[1,])) {
    temp <- data.frame(on_x = t, on_y = input_n[,i])
    p <- p + geom_line(data = temp, aes(x = on_x, y = on_y), alpha = 0.05, col = '#FFFF99', show.legend =
  }

  p
}

a <- ggplot() + geom_histogram(aes(x = haploid_wf[3,]), bins = 30) + xlab('Mi') + theme_minimal() +
                                labs(title = 'Distribution of Mi')
b <- plot_paths_gg(haploid_wf, 'Density plot') + theme(legend.position = 'none')
c <- plot_fixed_not(haploid_wf, T)
d <- plot_fixed_not(haploid_wf, F)
gridExtra::grid.arrange(a, b, c, d, nrow = 2)

plot_sums(haploid_wf, 'Plot S(n)') + labs(title = 'Plot of S(n)')


#### Now repeat with selection
haploid_wf_neg0.02 <- sapply(1:1000, simulate_wf, N = 100, s = -0.02)
haploid_wf_neg0.005 <- sapply(1:1000, simulate_wf, N = 100, s = -0.005)
haploid_wf_0.02 <- sapply(1:1000, simulate_wf, N = 100, s = 0.02)
haploid_wf_0.005 <- sapply(1:1000, simulate_wf, N = 100, s = 0.005)


#Make sigmoid deterministic curve
deterministic_curve <- function(s, t) {
  #Function to make deterministic curve
```

```
  100/ (1 + 99 * exp(-s * t))

}

mi3 <-  ggplot() + geom_histogram(aes(x = haploid_wf_neg0.005[3,]), bins = 30) + xlab('Mi') + theme_minim
                                labs(title = 'Distribution of Mi', caption = 's = -0.005') + xlim(c(-5, 1
mi4 <-  ggplot() + geom_histogram(aes(x = haploid_wf_neg0.02[3,]), bins = 30) + xlab('Mi') + theme_minima
                                labs(title = 'Distribution of Mi', caption = 's = -0.02') + xlim(c(-5, 10
mi2 <-  ggplot() + geom_histogram(aes(x = haploid_wf_0.005[3,]), bins = 30) + xlab('Mi') + theme_minimal(
                                labs(title = 'Distribution of Mi', caption = 's = 0.005') + xlim(c(-5, 10
mi1 <-  ggplot() + geom_histogram(aes(x = haploid_wf_0.02[3,]), bins = 30) + xlab('Mi') + theme_minimal()
                                labs(title = 'Distribution of Mi', caption = 's = 0.02') + xlim(c(-5, 101
notfixed1 <- plot_fixed_not(haploid_wf_0.02, F, 's = 0.02') + xlim(c(0, 50)) + ylim(c(0, 400))
fixed1 <- plot_fixed_not(haploid_wf_0.02, T, 's = 0.02') + xlim(c(0, 600)) + ylim(c(0, 7))
notfixed2 <- plot_fixed_not(haploid_wf_0.005, F, 's = 0.005') + xlim(c(0, 50)) + ylim(c(0, 400))
fixed2 <- plot_fixed_not(haploid_wf_0.005, T, 's = 0.005') + xlim(c(0, 600)) + ylim(c(0, 7))
notfixed3 <- plot_fixed_not(haploid_wf_neg0.005, F, 's = -0.005') + xlim(c(0, 50)) + ylim(c(0, 400))
fixed3 <- plot_fixed_not(haploid_wf_neg0.005, T, 's = -0.005') + xlim(c(0, 600)) + ylim(c(0, 7))
notfixed4 <- plot_fixed_not(haploid_wf_neg0.02, F, 's = -0.02') + xlim(c(0, 50)) + ylim(c(0, 400))


gridExtra::grid.arrange(mi1, mi2, mi3, mi4,
                        notfixed1, notfixed2, notfixed3, notfixed4,
                        fixed1, fixed2, fixed3, nrow = 3, ncol = 4)

a <- plot_paths_gg(haploid_wf_neg0.02, 'S = -0.02') + theme(legend.position = 'none')
b <- plot_paths_gg(haploid_wf_neg0.005, 'S = -0.005') + theme(legend.position = 'none')
d <- plot_paths_gg(haploid_wf_0.02, 'S = 0.02') + geom_line(aes(x = 1:500, y = deterministic_curve(0.02,
c <- plot_paths_gg(haploid_wf_0.005, 'S = 0.005') + geom_line(aes(x = 1:500, y = deterministic_curve(0.00
gridExtra::grid.arrange(a, b, c, d, nrow = 2)

#### Now do diploid simulation
#Build a wright fisher simulator
simulate_diploid_wf <- function(N, seed = 1, s = 0, h = 0) {
  #Function to make reproducible Wright-Fisher simulator
  set.seed(seed)

  #Set up some storage
  c_nt <- data.frame(n = NA, t = 1:500)
  m_max <- 1
  n <- 1
  t <- 1

  #Run simulator until fixation or loss
  while (n %in% seq(1, (2*N)-1)) {

    #Store previous n
    if (t < 500) {c_nt$n[t] <- n}

    #Set probability (considering selection)
    p <- n/(2*N)

    numerator <- p^2 + p*(1 - p)*(1 - h*s)
    denominator <- p^2 + 2*p*(1 - p)*(1 - h*s) + (1 - p)^2 * (1 - s)
    binom_prob <- numerator/ denominator

    #Do binomial sim
    n_prime <- rbinom(2*N, 1, binom_prob)
    n <- sum(n_prime)

    #Update m_max
```

```
    if (n > m_max) {m_max <- n}

    #Update t
    t <- t + 1

  }

  #Ensure rest of ns in c_nt are the same as last value
  if (t < 500) {c_nt$n[t:500] <- n}

  return(c(n, t, m_max, c_nt$n))
}
diploid_wf_0.02_1 <- sapply(1:1000, simulate_diploid_wf, N = 50, s = 0.02, h = 1)
diploid_wf_0.005_1 <- sapply(1:1000, simulate_diploid_wf, N = 50, s = 0.005, h = 1)
diploid_wf_0.02_0 <- sapply(1:1000, simulate_diploid_wf, N = 50, s = 0.02, h = 0)
diploid_wf_0.005_0 <- sapply(1:1000, simulate_diploid_wf, N = 50, s = 0.005, h = 0)

mi1 <-  ggplot() + geom_histogram(aes(x = diploid_wf_0.02_1[3,]), bins = 30) + xlab('Mi') + theme_minimal
                                labs(title = 'Distribution of Mi', caption = 's = 0.02, h = 1') + xlim(c(
mi2 <-  ggplot() + geom_histogram(aes(x = diploid_wf_0.005_1[3,]), bins = 30) + xlab('Mi') + theme_minima
                                labs(title = 'Distribution of Mi', caption = 's = 0.005, h = 1') + xlim(c
mi3 <-  ggplot() + geom_histogram(aes(x = diploid_wf_0.02_0[3,]), bins = 30) + xlab('Mi') + theme_minimal
                                labs(title = 'Distribution of Mi', caption = 's = 0.02, h = 0') + xlim(c(
mi4 <-  ggplot() + geom_histogram(aes(x = diploid_wf_0.005_0[3,]), bins = 30) + xlab('Mi') + theme_minima
                                labs(title = 'Distribution of Mi', caption = 's = 0.005, h = 0') + xlim(c
notfixed1 <- plot_fixed_not(diploid_wf_0.02_1, F, 's = 0.02, h = 1') + xlim(c(0, 60)) + ylim(c(0, 200))
fixed1 <- plot_fixed_not(diploid_wf_0.02_1, T, 's = 0.02, h = 1') + xlim(c(0, 600)) + ylim(c(0, 7))
notfixed2 <- plot_fixed_not(diploid_wf_0.005_1, F, 's = 0.005, h = 1') + xlim(c(0, 60)) + ylim(c(0, 200))
fixed2 <- plot_fixed_not(diploid_wf_0.005_1, T, 's = 0.005, h = 1') + xlim(c(0, 600)) + ylim(c(0, 7))
notfixed3 <- plot_fixed_not(diploid_wf_0.02_0, F, 's = 0.02, h = 0') + xlim(c(0, 60)) + ylim(c(0, 200))
fixed3 <- plot_fixed_not(diploid_wf_0.02_0, T, 's = 0.02, h = 0') + xlim(c(0, 600)) + ylim(c(0, 7))
notfixed4 <- plot_fixed_not(diploid_wf_0.005_0, F, 's = 0.005, h = 0') + xlim(c(0, 60)) + ylim(c(0, 200))
fixed4 <- plot_fixed_not(diploid_wf_0.005_0, T, 's = 0.005, h = 0') + xlim(c(0, 600)) + ylim(c(0, 7))

sums1 <- plot_sums(diploid_wf_0.02_1) + labs(title = 'Plot of S(n)', caption = 'S = 0.02, h = 1')
sums2 <- plot_sums(diploid_wf_0.005_1) + labs(title = 'Plot of S(n)', caption =  'S = 0.005, h = 1')
sums3 <- plot_sums(diploid_wf_0.02_0) + labs(title = 'Plot of S(n)', caption =  'S = 0.02, h = 0')
sums4 <- plot_sums(diploid_wf_0.005_0) + labs(title = 'Plot of S(n)', caption = 'S = 0.005, h = 0')

gridExtra::grid.arrange(mi1, mi2, mi3, mi4,
                        notfixed1, notfixed2, notfixed3, notfixed4,
                        fixed1, fixed2, fixed3, fixed4,
                        sums1, sums2, sums3, sums4, nrow = 4, ncol = 4)

a <- plot_paths_gg(diploid_wf_0.02_1, 'S = 0.02, h = 1') + geom_line(aes(x = 1:500, y = deterministic_cur
b <- plot_paths_gg(diploid_wf_0.005_1, 'S = 0.005, h = 1') + geom_line(aes(x = 1:500, y = deterministic_c
c <- plot_paths_gg(diploid_wf_0.02_0, 'S = 0.02, h = 0') + geom_line(aes(x = 1:500, y = deterministic_cur
d <- plot_paths_gg(diploid_wf_0.005_0, 'S = 0.005, h = 0') + geom_line(aes(x = 1:500, y = deterministic_c
gridExtra::grid.arrange(a, b, c, d, nrow = 2)

## Make the colsum things
a <- plot_sums(haploid_wf_neg0.02) + labs(caption = 'S = -0.02')
b <- plot_sums(haploid_wf_neg0.005) + labs(caption = 'S = -0.005')
d <- plot_sums(haploid_wf_0.02) + labs(caption = 'S = 0.02')
c <- plot_sums(haploid_wf_0.005) + labs(caption = 'S = 0.005')
gridExtra::grid.arrange(a, b, c, d, nrow = 2)

pfix1 <- plot_sums(haploid_wf_neg0.02, plotting = F)
pfix2 <- plot_sums(haploid_wf_neg0.005, plotting = F)
pfix3 <- plot_sums(haploid_wf_0.02, plotting = F)
pfix4 <- plot_sums(haploid_wf_0.005, plotting = F)
```

```
ss <- c(-0.02, -0.005, 0.02, 0.005)
estimated_pfix <- data.frame(estimate <- c(pfix1, pfix2, pfix3, pfix4), true = calc_fixation(ss), s = ss)

#Could just use a black line with slope = 1?
ggplot(estimated_pfix) + geom_point(aes(x = true, y = estimate, col = s)) + geom_abline(intercept = 0, sl
  xlab('Calculated value of pi(p)') + ylab('Estimated value of pi(p)')

#HWE
#Set up some paramters
total_adults = 30923
HbAA = 25374
HbAS = 5482
HbSS = 67

#Calculate allele frequencies
freq_A = (HbAA*2 + HbAS)/(total_adults*2)
freq_S = (HbSS*2 + HbAS)/(total_adults*2)

#Calculate expected values if HWE = true
exp_AA = (freq_A^2)*total_adults
exp_AS = (2*freq_A*freq_S)*total_adults
exp_SS = (freq_S^2)*total_adults

#Make contingency table
chi_table <- as.table(rbind(c(HbAA, HbAS, HbSS), c(exp_AA, exp_AS, exp_SS)))

dimnames(chi_table) <- list(real = c("Real", "Expected"),
                            genotype = c("HbAA","HbAS", "HbSS"))

chi.value = ((chi_table[1, 1] - chi_table[2, 1])^2)/chi_table[2, 1] +
            ((chi_table[1, 2] - chi_table[2, 2])^2)/chi_table[2, 2] +
            ((chi_table[1, 3] - chi_table[2, 3])^2)/chi_table[2, 3]

#3b
N = 40000

#Get birth numbers under HWE
born_AA = (freq_A^2)*N
born_AS = (2*freq_A*freq_S)*N
born_SS = (freq_S^2)*N

#Get genotype fitnesses
W_AA = (HbAA/born_AA)
W_SS = (HbSS/born_SS)/W_AA
W_AS = (HbAS/born_AS)/W_AA

#Calculate S and H
s = 1 - W_SS
h = (1 - W_AS)/s

#### Question 4
calc_fixation <- function(f_ai,  N = 100) {
  #Function to calculate fixation probability

  numerator = 1 - exp(-2 * f_ai)
  denominator = 1 - exp(-2 * f_ai * N)

  numerator/denominator
}


#Do gibbs sampling update and then track number of genes in state 1
```

```r
simulate_population <- function(cycles, N = 100, genes = 1000, seed = 1, rate_change = 0) {
  #Function to do simulation

  #Set up a vector with the genes
  genotype <- rep(0, times = 1000)
  n1 <- rep(NA, times = cycles)

  #Make it reproducible
  set.seed(seed)

  #Set selection coefficients
  f_ais <- rexp(genes, 1/0.01)

  #Get fixation probabilities
  pi_p <- calc_fixation(f_ais)
  pi_p_df <- data.frame(lost = 1 - pi_p, fixed = pi_p)

  #Get reverse fixation probabilities
  pi_p_neg <- calc_fixation(-f_ais)
  pi_p_neg_df <- data.frame(lost = 1 - pi_p_neg, fixed = pi_p_neg)

  #track when there is a switch
  t <- c()

  #Track proportion of time in state 1 for each gene
  prop_time <- genotype
  prop_time_cor <- genotype

  #This needs to be a totally random walk
  for (i in 1:cycles) {

    #Randomly choose a blob to update
    gene <- sample(1:genes, 1)


    #Allow random reversing of direction
    if (rpois(1, rate_change) > 0) {
      pi_p_neg_df_temp <- pi_p_neg_df
      pi_p_neg_df <- pi_p_df
      pi_p_df <- pi_p_neg_df_temp

      #Track the switches
      t <- c(t, i)
    }

    #Decide do p_pi depending on if its 1 or 0
    if (genotype[gene] == 0) {
      genotype[gene] <- sample(c(0,1), 1, prob = pi_p_df[gene,])

    } else {
      genotype[gene] <- sample(c(1,0), 1, prob = pi_p_neg_df[gene,])

    }

    #Track statistics
    n1[i] <- sum(genotype)

    #Update proportional times
    #Could somehow track if they are fixed in beneficial state? like reverse the states if in negative st
    if (i > 2e5) {

      #Track whether a switch has occurred:
```

```
      if (length(t) %% 2 == 1) {
        prop_time_cor <- (prop_time_cor*(i - 1) + (1 - genotype))/i

      } else {
        prop_time_cor <- (prop_time_cor*(i - 1) + genotype)/i
      }

      prop_time <- (prop_time*(i - 1) + genotype)/i
    }

  }


  return(list(n1, genotype, f_ais, t, prop_time/0.8, prop_time_cor/0.8))

}

n1 <- simulate_population(1e6)
n2 <- simulate_population(1e6, rate_change = 1e-4)
n3 <- simulate_population(1e6, rate_change = 1e-3)
n4 <- simulate_population(1e6, rate_change = 1e-2)

a <- ggplot() + geom_line(aes(y = n1[[1]], x = 1:1e6)) + xlab('Time') + ylab('Number of genes in state 1'
  theme_minimal() + labs(title = 'No change in selection') + ylim(c(0, 1000))


b <- ggplot() + geom_line(aes(y = n2[[1]], x = 1:1e6)) + xlab('Time') + ylab('Number of genes in state 1'
  theme_minimal() + labs(title = 'Selection changing at 1e-4') + ylim(c(0, 1000))

c <- ggplot() + geom_line(aes(y = n3[[1]], x = 1:1e6)) + xlab('Time') + ylab('Number of genes in state 1'
  theme_minimal() + labs(title = 'Selection changing at 1e-3') + ylim(c(0, 1000))


d <- ggplot() + geom_line(aes(y = n4[[1]], x = 1:1e6)) + xlab('Time') + ylab('Number of genes in state 1'
  theme_minimal() + labs(title = 'Selection changing at 1e-2') + ylim(c(0, 1000))

gridExtra::grid.arrange(a, b, c, d, nrow = 2)

#Consider this to be logit model
pop_outcomes0 <- data.frame(genotype = n1[[2]], f_ais = n1[[3]])
fit0 <- glm(genotype ~ f_ais, pop_outcomes0, family = "binomial")

pop_outcomes1 <- data.frame(genotype = n2[[2]], f_ais = n2[[3]])
fit1 <- glm(genotype ~ f_ais, pop_outcomes1, family = "binomial")

pop_outcomes2 <- data.frame(genotype = n3[[2]], f_ais = n3[[3]])
fit2 <- glm(genotype ~ f_ais, pop_outcomes2, family = "binomial")

pop_outcomes3 <- data.frame(genotype = n4[[2]], f_ais = n4[[3]])
fit3 <- glm(genotype ~ f_ais, pop_outcomes3, family = "binomial")

#Predict outcomes using the model
pop_outcomes0$predicted = predict(fit0, n1$f_ais, type = 'response')
pop_outcomes1$predicted = predict(fit1, n2$f_ais, type = 'response')
pop_outcomes2$predicted = predict(fit2, n3$f_ais, type = 'response')
pop_outcomes3$predicted = predict(fit3, n4$f_ais, type = 'response')


#Plot them
a <- ggplot() + #geom_point(data = pop_outcomes0, aes(x = f_ais, y = genotype, col = as.factor(0))) +
            geom_smooth(data = pop_outcomes0, aes(x = f_ais, y = predicted, col = as.factor(0))) + theme_
  #geom_point(data = pop_outcomes1, aes(x = f_ais, y = genotype, col = as.factor(1e-2))) +
```

```
  geom_smooth(data = pop_outcomes1, aes(x = f_ais, y = predicted, col = as.factor(1e-2))) +
  #geom_point(data = pop_outcomes2, aes(x = f_ais, y = genotype, col = as.factor(1e-3))) +
  geom_smooth(data = pop_outcomes2, aes(x = f_ais, y = predicted, col = as.factor(1e-3))) +
  #geom_point(data = pop_outcomes3, aes(x = f_ais, y = genotype, col = as.factor(1e-4))) +
  geom_smooth(data = pop_outcomes3, aes(x = f_ais, y = predicted, col = as.factor(1e-4))) +
  labs(col = 'Rate of change', title = 'Plot of predicted probability of being in state 1 \n for logistic
  theme(legend.position = 'bottom') + ylim(c(0, 1))

b <- ggplot() + geom_point(aes(x = n1[[3]], y = n1[[6]], col = as.factor(0)), alpha = 0.3) +
                geom_point(aes(x = n2[[3]], y = n2[[6]], col = as.factor(1e-4)), alpha = 0.3) +
                geom_point(aes(x = n3[[3]], y = n3[[6]], col = as.factor(1e-3)), alpha = 0.3) +
                geom_point(aes(x = n4[[3]], y = n4[[6]], col = as.factor(1e-2)), alpha = 0.3) +
                geom_smooth(aes(x = n1[[3]], y = n1[[6]], col = as.factor(0)), se = F) +
                geom_smooth(aes(x = n2[[3]], y = n2[[6]], col = as.factor(1e-4)), se = F) +
                geom_smooth(aes(x = n3[[3]], y = n3[[6]], col = as.factor(1e-3)), se = F) +
                geom_smooth(aes(x = n4[[3]], y = n4[[6]], col = as.factor(1e-2)), se = F) + theme_minimal
                labs(col = 'Rate of change', title = 'Plot of proportion of time spent state selected for
                ylab('Proportion of time spent in state selected for') + theme(legend.position = 'bottom'

gridExtra::grid.arrange(a, b, nrow = 1)
#Plot the proportion of time spent in each state over time

tree_tab <- data.frame(internal_node = c(9:15),
                       daughter_node1 = c(2, 1, 4, 11, 10, 7, 13),
                       dist1 = c(2.7, 1.8, 1.1, 1.9, 1.5, 4.7, 2.2),
                       daughter_node2 = c(3, 9, 5, 6, 12, 8, 14),
                       dist2 = c(2.7, 4.5, 1.1, 3.0, 3.0, 4.7, 3.5))

#Question 1
make_node_tab <- function(input, row) {
  #Function to get daughter nodes and leaves for each internal node
  #Make some sort of recursive function?

  #Make table of nodes below leaves
  node_tab <- input[row,]

  if (input$daughter_node1[row] %in% input$internal_node) {

    #if daughter 1 is an internal node recursively do function, append it to tab
    recursive_nodes <- make_node_tab(input, which(input$internal_node == input$daughter_node1[row]))
    node_tab <- rbind(node_tab, recursive_nodes)

  }

  if (input$daughter_node2[row] %in% input$internal_node) {

    #if daughter 2 is an internal node recursively do function, append it to tab
    recursive_nodes <- make_node_tab(input, which(input$internal_node == input$daughter_node2[row]))
    node_tab <- rbind(node_tab, recursive_nodes)

  }

  return(node_tab)

}

compute_tab_leafnodes <- function(input, row) {
  #Function to compute numbers of leaves and internal nodes

  node_tab <- make_node_tab(input, row)

  #Work out which daughter nodes are internal
```

```
  daughter_internal1 <- length(which(node_tab$daughter_node1 %in% node_tab$internal_node))
  daughter_internal2 <- length(which(node_tab$daughter_node2 %in% node_tab$internal_node))

  #Now calc number of nodes and leaves
  internal_nodes <- dim(node_tab)[1] - 1
  leaves <- dim(node_tab)[1]*2 - (daughter_internal1 + daughter_internal2)

  c(internal_nodes, leaves)
}

tree_tab_stats <- cbind(tree_tab, t(sapply(1:dim(tree_tab)[1], compute_tab_leafnodes, input = tree_tab)))
colnames(tree_tab_stats)[6:7] <- c('Internal_nodes_below', 'Leaves_below')
kable(tree_tab_stats, caption = 'Table with internal nodes and leaves included')

#### Question 2 ---
calc_leaf_dist <- function(leaf, input) {
  #Function to calculate distance from leaf to root
  dist <- 0

  #Find loc and dist of leaf
  if (leaf %in% input$daughter_node1) {
    #If leaf is in first daughter column

    leaf_row <- which(c(input$daughter_node1) == leaf)
    dist <- dist + input$dist1[leaf_row]

    #Do recursion
    if (input$internal_node[leaf_row] %in% c(input$daughter_node1, input$daughter_node2)) {
      dist <- dist + calc_leaf_dist(input$internal_node[leaf_row], input)
    }

  } else if (leaf %in% input$daughter_node2) {
    #Repeat for second column

    leaf_row <- which(c(input$daughter_node2) == leaf)
    dist <- dist + input$dist2[leaf_row]

    #Do recursion
    if (input$internal_node[leaf_row] %in% c(input$daughter_node1, input$daughter_node2)) {
      dist <- dist + calc_leaf_dist(input$internal_node[leaf_row], input)
    }
  }



  return(dist)

}

calc_root_dist <- function(input) {
  #Function to calculate distance to root node from leavees

  #Work out which are leaves
  notleaves1 <- which(input$daughter_node1 %in% input$internal_node)
  notleaves2 <- which(input$daughter_node2 %in% input$internal_node)
  leaves <- sort(c(input$daughter_node1[-notleaves1], input$daughter_node2[-notleaves2]))

  #Now return the distances
  distances <- sapply(leaves, calc_leaf_dist, input)
  data.frame(leaves, distances)

}
```

```r
kable(calc_root_dist(tree_tab), caption = 'Table of distances from leaf to root')

##### Make the pruning algo ----
mu <- 0.01

make_transmat <- function(mu, t) {
  #Function to make transition matrix
  #Do you need to normalise over the rows?

  #Get aii and aij
  a_ij <- 0.25*(1 - exp(-4*mu*t))
  a_ii <- 0.25*(1 + 3*exp(-4*mu*t))

  #Make the matrix
  tmat <- matrix(a_ij, nrow = 4, ncol = 4)
  for (i in 1:4) {tmat[i,i] <- a_ii}

  tmat
}

get_pair_L <- function(internal_node, input, L_values, mu) {
  #Function to get likelihood of a pair of daughter nodes

  #Get daughter nodes
  internal_node_row <- which(input$internal_node == internal_node)
  daughter1 <- input$daughter_node1[internal_node_row]
  daughter1_probs <- L_values[daughter1, ]
  daughter2 <- input$daughter_node2[internal_node_row]
  daughter2_probs <- L_values[daughter2, ]

  #Get distances
  dist1 <- input$dist1[internal_node_row]
  dist2 <- input$dist2[internal_node_row]

  #Make transition matrices given above
  tmat1 <- make_transmat(mu, dist1)
  tmat2 <- make_transmat(mu, dist2)

  #Get L for the root node
  prob1 <- daughter1_probs %*% tmat1
  prob2 <- daughter2_probs %*% tmat2

  #Get the dot product of the probabilities
  prob1 * prob2

}



make_base_prob <- function(base, bases) {
  #Function to make base into vector

  base <- bases[base]

  switch(base,
         'A' = {c(1, 0, 0, 0)},
         'C' = {c(0, 1, 0, 0)},
         'T' = {c(0, 0, 1, 0)},
         'G' = {c(0, 0, 0, 1)})

}
```

```r
run_felsenstein <- function(input, bases, mu) {
  #Function to put it all together

  #Work out which are leaves
  notleaves1 <- which(input$daughter_node1 %in% input$internal_node)
  notleaves2 <- which(input$daughter_node2 %in% input$internal_node)
  leaves <- sort(c(input$daughter_node1[-notleaves1], input$daughter_node2[-notleaves2]))

  #Make mat to hold probabilities
  L_values <- matrix(NA, nrow = 15, ncol = 4)

  #Fill in known probs
  L_values[1:8,] <- t(sapply(1:8, make_base_prob, bases = bases))

  #Work out which leaves share internal nodes
  leaf_nodes <- data.frame(row = NULL, node = NULL)

  #Make vector to hold all nodes and leaves
  all_nodes <- 1:15

  while (sort(unique(is.na(L_values[,1])), decreasing = T)[1]) {
    #While not all of them have been filled in

    #Find out which haven't been filled in
    unfilled <- which(is.na(L_values[,1]) == T)

    for (i in unfilled) {
      #Work through them to get which ones can be filled in (both daughter nodes filled in)
      internal_node_row <- which(input$internal_node == i)

      #IE does the current unfilled node have both duaghter nodes that are filled
      if (all(input[internal_node_row, c('daughter_node1', 'daughter_node2')] %in% all_nodes[-unfilled])) {

        L_values[i,] <- get_pair_L(i, input, L_values, mu)

      }

    }

    if (length(unfilled) == 1) {
      #If finally at root node

      L <- L_values[unfilled] * rep(0.25, 4)
    }

  }

  #Now return the summed probabilities
  log(sum(L))

}

par(mfrow = c(2,1))

bases <- c('A','G','T','G','G','G','C','G')
L_mu1 <- sapply(seq(1e-3, 1, 1e-3), run_felsenstein, input = tree_tab, bases = bases)
plot(y = L_mu1, x = seq(1e-3, 1, 1e-3), xlab = 'Mu', ylab = 'log(L)',
     main = 'Plot of log likelihood against mu for original sequence')

MLE1 <- seq(1e-3, 1, 1e-3)[which(L_mu1 == max(L_mu1))[1]]

base1 <- c('G','C','A', 'T','A','T','G','A')
```

```r
L_mu2 <- sapply(seq(1e-3, 1, 1e-3), run_felsenstein, input = tree_tab, bases = base1)
plot(y = L_mu2, x = seq(1e-3, 1, 1e-3), xlab = 'Mu', ylab = 'log(L)',
     main = 'Plot of log likelihood against mu for random sequence')

MLE2 <- seq(1e-3, 1, 1e-3)[which(L_mu2 == max(L_mu2))[1]]
```