**Report of daily progress of my dissertation**

**Day #1 (reported) – 01/04/25**

Today I spent 5 hours trying to adapt what I had of the neutral transmission model to the time averaged variation. I’ve called the former the “Snapshot” model, and the latter the “Time Averaged”. The first hour or so I was trying to figure how to average the generations recorded in the model, and I created a “time averaging factor” and differentiated between total generations or time steps and observed time steps, which are the ones we’re interested in ultimately. To sample which time step out of the factor chosen (in this case it was 10), I simply told the simulation to pick the last one, but I eventually changed it to sample all individuals in the window averaged. It’s not very relevant right now, because I gave it up momentarily to work on sorting the neutrality “snapshot” model.

I spent most of the afternoon/evening battling with the code to detect possible caveats in the current model. I removed and rewrote many chunks of code, until I finally hit the right note. Firstly, I reported 60% of detection of neutrality, but then I modified some bits and went down to 3-6%, which implied an error in the model or conversion of the model to the FIT parameters. I had to go back to Ben Marwick’s repository several times to adequately convert the simulated data into long format, filter variants with less than 3 time points and run the test regardless of NA results. I finally made it, and after some time wrestling with R, I finally got some more realistic results: 43% of neutrality detection.

Next step is to refine the code and modify the parameters to evaluate different outcomes of the test. So far, with an innovation rate of 0.01 after filtering, fit\_result has only 54 observations, which means 54 distinct variants that appear at least in 3 time steps. Out of the 100 variants that start off, and after the burn-in stage, we’re left with 54 observed variants. This is likely due to a low innovation rate which leads to a rapid turnover of variants (a phenomenon that we would not be able to record as precisely in the time averaged model). Adjusting the model parameters would likely result in different outcomes:

* With higher *N* or population size, the effect of drift (random fluctuations) would be reduced, meaning that we would expect more observed variants.
* With higher *µ* or innovation rate, we should expect more new variants emerging or reemerging, leading to more observed variants after equilibrium. And if we increase *N,* we should expect an even higher number of variants, as the effect of drift is reduced.

This is all under the assumption of a finite population, however, we should bear in mind that some cultural traits may be better represented within the frame of the infinite allele model (Wright-Fisher Infinite Allele), such as decorative motifs in pottery (Madsen, 2012). In essence, the number of observed variants is defining how diverse or homogeneous was our population in the past 200 generations/time steps.

To conclude, for tomorrow I must **review the current code and look for possible caveats or ways of improving it** (too tired now and might’ve incurred in some mistakes while writing); **modify the parameters of the model** and make predictions; **clean the pipeline to transfer it into the time averaged variation**; make **visualisations of the simulated data** to better understand what’s going on. And, if possible, start off with the time averaged model, which I’ve already started but will need to rewrite.

Another goal for tomorrow is to go forward with the reference writing and summarising for the introductory chapter of the dissertation. So far, I’ve read and summarised the elementary papers on the application of neutrality models, and I’ve been able to raise some questions and get things a little bit clearer. I should carry on with this matter, as it will save me a lot of work later while writing the thesis.

**Day #2 – 02/04/25**

I am currently editing the model’s parameters, and as I predicted, when increasing *N*, we get more variants observed after equilibrium, and the opposite if we indicate a lower number. In fact, with only 10 variants only 4 survive, and three of them record *NA* values after the test, so only one is detected as neutral. I’ll assign 50 number of initial observations, as I think it might relate better with a real-life scenario of, for example, pottery motifs (other traits may vary).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | µ | Runs | % | Time |
| 50 | 0.01 | 100 | ~44% | 13” |
| 50 | 0.01 | 1000 | ~44% | 2’ 50” |
| 50 | 0.01 | 10000 | ~44% | 16’ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | µ | Runs | % | Time |
| 50 | 0.02 | 100 | ~84% | 20” |
| 50 | 0.02 | 1000 | ~84% | 3’ 15” |
| 50 | 0.04 | 100 | ~159% (?) | 34” |
| 50 | 0.06 | 100 | ~224% | 40” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | µ | Runs | % | Time |
| 20 | 0.01 | 100 | ~41% | 10” |
| 40 | 0.01 | 100 | ~42% | 18” |
| 60 | 0.01 | 100 | ~45% | 15” |
| 80 | 0.01 | 100 | ~44% | 16” |

As we can see from the tables, innovation rate is biasing the test results. This is likely due to the faster rate of innovation leading to the introduction/invention of new variants to the population, surpassing the initial population of *N*. When computing the average, we divide the sum of neutral detections between the number of runs and *N*, meaning that we are comparing the observed population with the initial one, and this leads to **overrepresentation of variants when innovation is high**. Number of runs and population size does not seem to affect the test results, so perhaps is not worth paying the costs of more time for more runs.

Today, after modifying the parameters of the model and revising the code, I’ve found a possible caveat, which is what I refer to as the **overrepresentation of variants in the test result** due to a high rate of innovation. A task for tomorrow will be **solving the problem of overrepresentation when high µ**. For the rest of the tasks I had for today, the code is almost ready to apply the time averaging factor, tomorrow I’ll be able to go forward in that regard.

I’ve correctly modified one of the plots to show all the observed variants in terms of absolute frequencies, and not relative. Ben Marwick mentions that the FIT requires absolute frequencies, not relative, which is something I had to correct from previous models. So, the division is done between number of unique variants ever recorder and the initial population (N). For this reason, it is more reasonable to represent absolute frequencies, instead of relative frequencies each time step, although it could be more intuitive to plot the latter for actual representations of real data.

For tomorrow, the tasks to complete are as follows: **solve the overrepresentation of variants** in fit\_results, **implement time averaging into the new model**, **plot both models** with different parameters, and **hint at different paths to go** next.

**Day #3 – 03/04/2025**

Today I’ve read the article titled “Random drift and culture change”, published in 2004 in the *Proceedings of the Royal Society of London* by Alex Bentley, Matthew Hann and Stephen Shennan. The article essentially covers the question of how efficient the neutral models of genetic evolution and their adaptation to cultural evolution with real case studies are. It puts forward the relationship between the notion of power-law slope (α) and the finite allele drift model (2Nµ). The authors predict that the slope is directly proportional to the product of *N* and *µ*, meaning that if the population is large and new variants are introduced frequently, the distribution of variants is more spread. On the other hand, if the population is small and innovation is low the effect of drift is amplified and variant number is reduced, which leads to an uneven distribution.

|  |  |  |
| --- | --- | --- |
| N | µ | α |
| High | High | High |
| High | Low | High |
| Low | High | High (for short term) |
| Low | Low | Low |

Don’t know how this will be useful for this work, but it’s always worth looking at the references and writing down whatever comes to mind throughout.

Today I fixed the issue of the overrepresentation of variants. Apparently, what I was doing was wrong. I was computing the average of counted detected neutral and initial amount of variants/number of unique variants, but this doesn’t tell me anything about the test itself, it just computes how many of the unique or initial variants are left in one run. What I did instead was:

1. Sum all the neutral counts per run and store it in an object.
2. Count the **total variants tested and expected** in a single run (95% threshold).
3. Sum the **actual number of observed neutral** variants in a run.
4. Compute the **average**: Observed neutral variants / Expected neutral variants.
5. Store the result in an empty object, and index it for every run.
6. Compute the **mean across all runs**.

The first mean after 100 runs, *N* = 80, *burn-in stages* = 100, *time steps* = 100 and *µ* = 0.05, is **~86%**. Which means that, out of 100 runs, the test observed 86% of variants that were neutrally transmitted (in theory all are, although some were not detected and returned NA), under the assumption that to be statistically significant it should reach at least 95%. This is a positive result, as the ratio is not as low as it was before (~44%), is close to the expected value, but it still doesn’t quite reach it. Recording the time that different number of runs take was crucial, because now more runs imply more robustness of results. I should try next to run the loop 10000 times, but it’ll most likely take ~15-20 minutes. Perhaps it’s time to get my hands on vim…

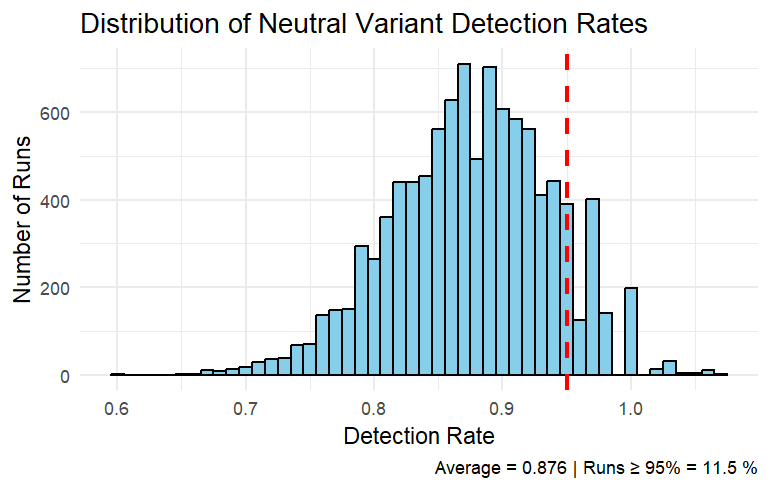
I ran the simulation again with the same parameters, except *µ* = 0.1, and the ratio is ~84%, slightly less, but still quite high. Now is the *time* for *time averaging*. I couldn’t complete the visualisation and time averaging task, but I took a great leap in the current work, so now I can get my hands on time averaging. For tomorrow, the objective will be to **run the test 10,000 times** and compare with less runs; **apply time averaging** to the neutral model, which shouldn’t be hard, and I’ve already wrote some of the code; and finally, **read another article** and take notes, as today I read one even though I didn’t intend in the beginning.

**Day #4 – 07/04/25**

Before going further, I’ve counted how many NA values the FIT returns given some parameters (N = 80, µ = 0.02, burn-in = 100, time steps = 100, significance level = 0.05, and number of runs = 100). The ratio of detection is ~88%, higher than in previous examples in which µ was higher (translated into more variants observed and tested by the FIT). What is worth mentioning is that ~28% of the variants analysed by the test are recorded as NA, all of them scoring three or four time steps maximum. Two things are going on:

1. Innovation rate (µ) could influence the test results insofar as with higher value, more variants are observed and analysed by the test, perhaps reducing its accuracy (mo’ variants mo’ problems?).
2. There seems to be a correlation between capacity of the test to detect neutrality/selection (hence high % NA) and small number of time steps in which the variant is present within the simulations.

With 10,000 runs and µ = 0.01 the test returns a mean of ~88% successfully detected neutral variants, and ~23% of NA. Knowing the mean value of neutral detections is important, but we also want to know the **proportion out of the total number of runs**. Within the same simulation (10,000 runs), there is a **11.52%** of cases in which the test has detected neutral variants over 95% of the times. When plotting the distribution of neutral variant detection rates across runs, most observations are aggregated around the mean (0.876), and some even surpass the 0.95 threshold of accuracy:



Nevertheless, some observations go beyond 100%, meaning that there is an error, as the rate must be bounded by the total number of runs. After reevaluating the pipeline, the expected\_neutral\_count should really be 100% of the times if the simulation is doing what is meant to, instead of 95% (a threshold I established based on statistical significance, but which is purely arbitrary). We know that sampling is random, and innovation is random as well, so all the frequencies should follow a neutrality/unbiased pattern. The null is true, and I need to account for any false positive (probability of rejecting the null when it is true or **Type I error**). For this reason, I created two new objects, FPR and TNR, which stand for False Positive Rate (α) and **True Neutral Rate** (1-α), the latter being our measurement of interest.

# Previous syntax to compute the number of neutral variants detected per run

# Observed vs total neutral detection

# count neutral cases per run

neutral\_counts\_per\_run\_snapshot**[**run**]** **<-** sum**(**fit\_results**$**sig **==** "neutral", na.rm **=** **TRUE)**

# count total variants tested and the expected proportion

total\_variants\_tested **<-** nrow**(**fit\_results**)**

expected\_neutral\_count **<-** round**(**0.95 **\*** total\_variants\_tested**)** # Expected proportion based on threshold

# observed neutral variants in one run

actual\_neutral\_count **<-** sum**(**fit\_results**$**sig **==** "neutral", na.rm **=** **TRUE)**

# match rate

neutral\_match\_rate **<-** actual\_neutral\_count **/** expected\_neutral\_count

# store results per run in the empty object

accuracy\_snapshot**[**run**]** **<-** neutral\_match\_rate

# New Syntax

# Store metrics:

total\_variants **<-** nrow**(**fit\_results**)**

FPR **<-** sum**(**fit\_results**$**sig **==** "selection"**)** **/** total\_variants

# False positives

TNR **<-** sum**(**fit\_results**$**sig **==** "neutral"**)** **/** total\_variants

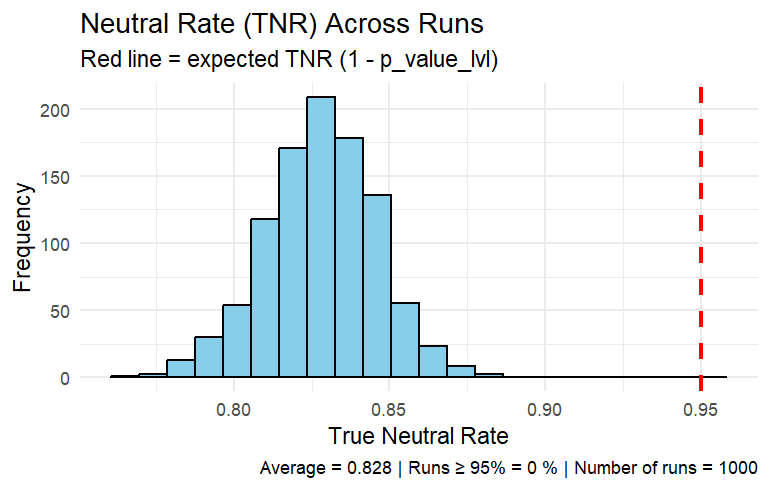
# True negatives

# or

TNR **<-** 1 **-** FPR

accuracy\_snapshot**[**run**]** **<-** TNR # Store true negatives across runs

Added a plot of the distribution of True Neutral Rate, after running the simulation with the following parameters: N = 100, µ = 0.01, burn-in = 100, time steps = 1,000, significance level = 0.05, number of runs = 1,000.



The distribution looks more logical than before, and we lack values beyond 100%. With 1,000 runs we don’t have a single run in which the test has detected more than 95%, however, we can propose that, on average, the Signal Selection Test has a statistical power of 82% to detect successfully neutral transmission with no time averaging, at equilibrium.

I still record a ~21% of NA values after running the test, most of them again related to <5 time steps; perhaps the **model needs more filtering for better results**. In regard to the relationship between **innovation rate and test accuracy**, we have no current evidence to support it, but it should be an issue to raise in the future.

Summary

I couldn’t apply time averaging, but several advances were made today: the pipeline looks clearer, and the rate of neutral detection is calculated over the actual number of total variants, instead of an artificial threshold of 95% (as it was at the start of the day). Furthermore, I ran the simulation 10,000 times, and through visualisation realised that some of the observations were higher than 1. This contradicts the assumption that the numerator (observed neutral counts) should never be bigger than the denominator (expected neutral count) to compute a rate. That explains the relatively high number of detections over the 0.95 threshold (11.52%). To solve this, I just had to compute the probability of accepting and rejecting the null when it is true (Type I error, we’ll incur in Type II error when discussing content bias transmission). I successfully fixed the problem, and now all observations are distributed between 0 and 1, presenting a normal distribution. The output is that now no observations exceed 0.95, but they aggregate around 0.82 (mean). What this means is that, if the model and the rest of the pipeline is correct, **the Signal Selection Test has on average 82% of probabilities of detecting neutrality**, even though some variants were not recorded in the process (Nas, although considered when computing the TNR).

A new implementation to the workflow has been made: **recording metadata**. I’ve created a Notepad++ file to record all the modifications of the main code, in case I ever want to reproduce past problems and new/different ways of fixing them. It’ll be conveniently pushed to my GitHub.

Tasks

For tomorrow, my tasks comprise: **applying time averaging** once and for all (lol) and **read a new article** for the literature review (perhaps I ought to summarise them in a different document, although I still write notes in my notebook).

**Day #5 – 08/04/25**

Finally, I’ve implemented time averaging into the pipeline. Essentially what I did is resumed in the following code:

# Time averaging step -----------------

averaged\_rows **<-** floor**(**timesteps **/** time\_window**)**

averaged\_matrix **<-** matrix**(NA**, nrow **=** averaged\_rows, ncol **=** N**)** # new matrix

**for** **(**j **in** 1**:**averaged\_rows**)** **{**

start **<-** **(**j **-** 1**)** **\*** time\_window **+** 1 # indicate start of window (1, 26, 51...)

end **<-** j **\*** time\_window # indicate end of window (25, 50, 75...)

averaged\_matrix**[**j, **]** **<-** apply**(**traitmatrix**[**start**:**end, **]**, 2, **function(**x**)** sample**(**x, 1**))**

**}**

What I’m doing here is creating the number of time steps observed when time averaging is at play and creating a new matrix to store the simulation output before applying the FIT. The loop indicates the start and end of each time window, sample one time step out of the number (in this case, it was 25), and store what each variant does within that generation in the new matrix.

The results this time are more promising, as predicted though. When time averaging is happening, the test selects on average ~94% neutral variants (knowing that 100% should be neutral). 19% of the runs show equal or more than 95% neutral variants detected, with only 6.4% of NA. The parameters of this simulation are as follows: N = 100, µ = 0.05, burn in = 1,000, time steps = 1,000, significance level = 0.05, runs = 100, and **time window** = 25. The time window determines ultimately **how many time steps we are counting when sampling our data**. With 25-time windows and 1,000-time steps, we get 40 generations (in archaeological terms, these would be our strata).

Here's a comparison of both models using the same parameters (snapshot and time averaged):

A graph of a number of different numbers

AI-generated content may be incorrect.

If we modify the parameters of time averaging, we should expect some changes. The small-scale fluctuations are smoothed by the time averaging effect. We gain accuracy to detect neutrality at the cost of losing small-scale information. This has some advantages and disadvantages:

Advantages:

* Knowing that the test performs well under time averaging is convenient insofar as the archaeological record is time averaged.
* The high proportion of neutral variants detected indicates good power.
* We’re able to make more solid conclusions about neutrality within our own assemblage.

Disadvantages:

* Fine-grained information is lost, such as short-lived variants, bursts of innovation and rapid turnovers (although it depends on the size of the time window).
* We lose individual-level trajectories, since they are averaged.
* We make the processes look smoother than they are.

We can tell from the plots that the distribution of TNR is compressed and less equally distributed, probably because the number of runs is lower than in previous simulations. When running 1,000 simulations the data is more widely distributed, so ideally, we would like to have as many runs as possible. Unfortunately, in this laptop 1,000 runs is a lot, and 10,000 I cannot even imagine how long. It might be time to test this in the cloud.

So today I read another article (not fully, but almost), and implemented time averaging into the neutrality model. The results drawn from both experiments are positive insofar as the test has proved a good rate of neutral detection. For tomorrow, I will **run the test with parameter modifications** and **compare results**, changing µ (referenced in one of the issues written down yesterday), time window, and time steps.

With increasing time steps, we eventually see more unique variants mainly because the model iterates the input for longer. In the last simulation (the one shown in the last figure) the output was 2131 unique variants (over 3-time steps, remember), most of the ones with NA being present at 3 – 5 time steps. This is telling us that perhaps **more variants are not a problem for the FIT, and performance is not detracted**. We will examine this question tomorrow.