**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…

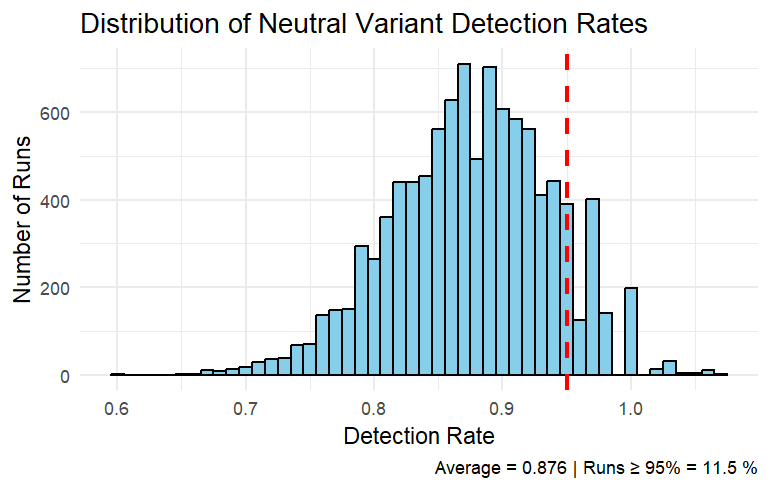
I ran the simulation again with the same parameters, except *µ* = 0.1, and the ratio is ~84%, slightly less, but still quite high. Now is the *time* for *time averaging*. I couldn’t complete the visualisation and time averaging task, but I took a great leap in the current work, so now I can get my hands 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

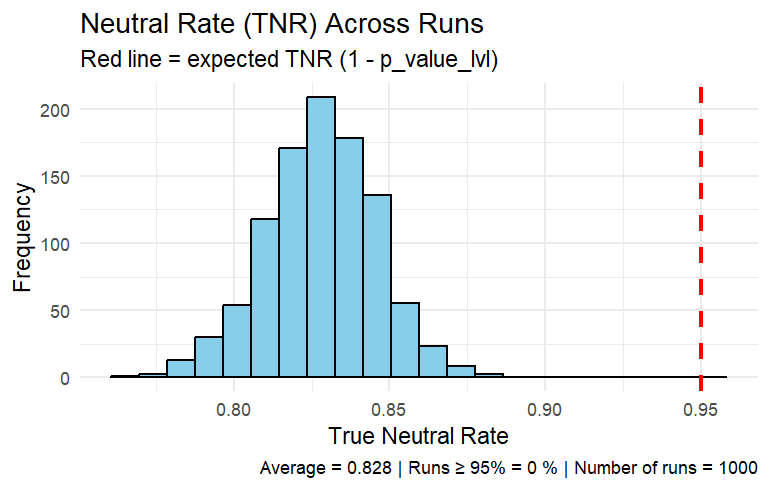
# True negatives

# 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...)

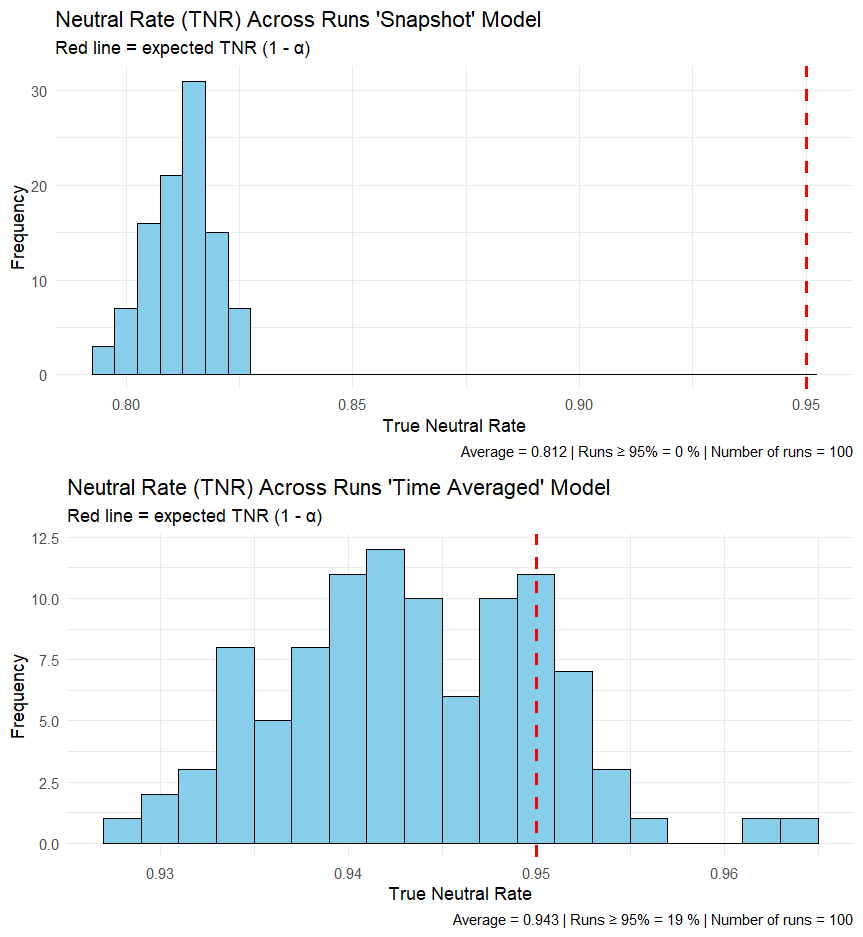
averaged\_matrix**[**j, **]** **<-** apply**(**traitmatrix**[**start**:**end, **]**, 2, **function(**x**)** sample**(**x, 1**))**

**}**

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):



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 input was 2131 unique variants (equal or more than 3 time counts, 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.

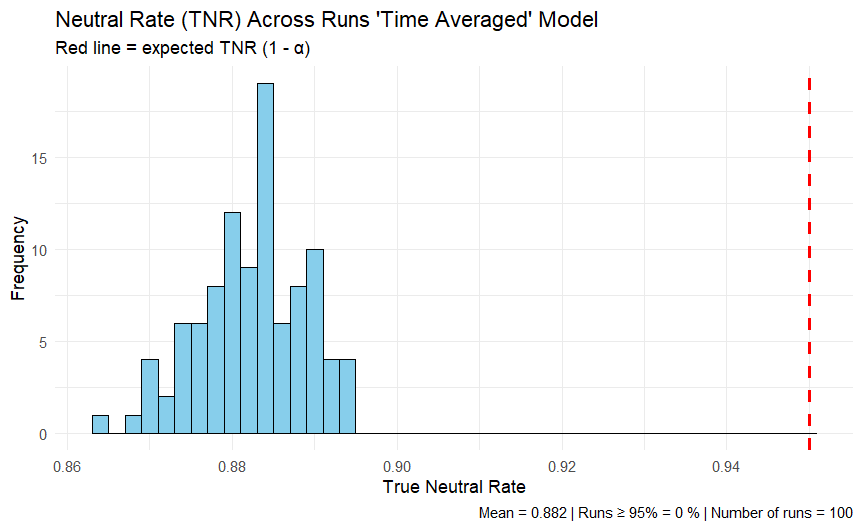
**Day #6 – 11/04/25**

I am writing this amidst the working session to address something that I should´ve addressed before: reference organisation. So far what I’ve done within this session is go through Acerbi et al. (2022) handbook of IBM for cultural evolution, specifically, chapter 4 focused on conformity bias. I won’t include this mode of transmission in my dissertation, but knowledge doesn’t hurt. In fact, I should read all that’s been written about it for the methods justification. I must state the *whys* of my decision to choose only unbiased transmission and content biased transmission.

For this task I am reading all the literature on cultural transmission applied to archaeology, and in the process, I’ve realised that I am once again reading without a purpose. That’s why I’ve decided to keep track of: what I am currently reading, what I read and how can it help me for literature review, and which references to go next. So far, I was reading everything saved in the “References” folder in chronological order, however, I’ve skipped the order several times, so there’s no point in committing anymore. Instead, I should read intelligently, highlighting what is important and what is not.

In the end, I created a **spreadsheet** with all the references I have so far, and with annotations and information that should be useful when writing the dissertation or looking back for references or ideas I had written in my notebook or floating in my head.

Back to the code. I ran again the pipeline, this time with higher innovation rate value. What the plot shows me is that with higher innovation rate, and therefore, higher number of variants for the FIT to analyse, the results are clearly worse, although not drastically worse. Instead of 2131 variants, it’s 2630. The mean of neutral detections equals 0.882 for 100 runs, with 14.6% of NAs and 0 runs with a success rate of over 95%.



This is pointing us that **with more variants analysed throughout the time series, the FIT performs slightly worse**, however, I still must discover why.

**Day #7 – 17/04/25**

Today I spent most time reading Lipo et al. (1997), an article that essentially covers the topic of sampling frequencies in archaeological assemblages following the criteria of real population dynamics (spatiotemporal distribution and its relationship with trait interaction).

**Day #8 – 18/04/25**

After reading some more of Lipo et al., I’ve decided to skip it and read more relevant articles regarding content bias, which relates to the next step of the dissertation: formulating the content biased transmission experiment and its justification.

One important thing I want to refer here is the structure of the draft, and the timeline to write it. The first thing to write should be the methods section, in which I ought to describe the methodology employed and the justification that lies behind. Let’s break it down:

Methods

* Describe SST: parameters, input, output and implementation details.
  + - The logic behind the outcome of the test.
* Experiments: parameters, logic, details (equilibrium, time averaging), justification.
  + - Why time averaging, and what differences are expected from both scenarios.
    - What insights do changing the parameters yield in relation to the theory of cultural transmission.
    - Why is equilibrium important?

And here’s the preferred references to achieve it as I wrote in the initial proposal. Most of them I’ve already read or skimmed, but a more comprehensive reading is required.

|  |  |
| --- | --- |
| Read | Why |
| Crema et al. (2024) | Experiment design. |
| Henrich (2001) | Quantitative definition of content bias and how it operates over time. |
| O'Dwyer & Kandler (2017) | How to distinguish neutrality from bias, |
| Kandler & Crema (2019) | Methodological overview of selection detection. Use to defend SST usage. |
| Feder et al. (2014) | Original SST logic. |
| Newberry et al. (2017) | SST applied to cultural (language) data. Cross-domain use. |

**Day #9 – 24/04/25**

Time averaging follows the idea that with longer time spans there’s more cumulative behaviour, more **items are deposited** hence a **bigger assemblage**. Number of individuals is relevant, but the key point is time. For the current model:

**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**))**

**}**

I am sampling one variant per individual for each window, when I should be sampling the whole accumulation of the time\_window. In an assemblage we don’t simply excavate the individuals’ behaviour subsampled across 20 time steps, but what we do is sample the accumulation of those behaviours over 20 time steps. Innovations, although new, emerge from known information which is accumulated. Therefore, sampling randomly from each time window makes us lose track of that cumulative nature.

**Day #10 – 03/05/25**

For today I will be re-reading Feder et al. (2014) and work on the code using Enrico’s suggestions from the last meeting.

* To start off writing the methods section I shall refer to Feder et al. (2014) and Newberry et al. (2017) to summarise the FIT and Signal Selection Test.
* After describing the basics of the method, i.e. the statistical foundation, objective, data input and output, I should mention again (because it should’ve been mentioned already in the Introduction) the objectives of the study and the methodology employed (**simulation based**). To achieve this, I should refer to Crema et al. (2024).
* And finally, the most important part, to describe each experiment and the conditions. This would require a brief explanation of the basics of neutral and biased transmission, a quick formalisation of the models (snapshot and time averaged models), and the **expected outcome**. In our case, the interesting bit is the **power analysis**, i.e. what I called “True Neutral Rate” (TNR) and “False Positive Rate” (FPR). Here I should refer to Neiman et al. (1995), Shennan & Wilkinson (2001), Kandler & Shennan (2014), Kandler & Crema (2019), Premo (2014), Crema et al. (2016), Acerbi & Bentley (2014).

After going through Feder et al. (2014) again I’ve picked up some interesting things which could be useful for the discussion and methods section. Let’s break down the latter, as it is the one I’m busy working on right now.

The main method used in this study is the Frequency Increase Test, which is an algorithm developed by Feder et al. (2014) and presented in that same paper. The most accurate description is as follows: *it is a statistical test that rejects neutrality if the distribution of normalised allele frequency increments exhibits a mean that deviates significantly from zero*. So, applied to cultural evolution, the test rejects neutrality if the distribution of normalised cultural variants increments exhibits a mean that deviates significantly from zero. It is important noting that the authors state that FIT is limited to “(…) time series of allele frequencies (…), assuming independence from all other loci” (p. 510).

The description of the technique is quite straightforward. In the draft I should reference Ben Marwick’s application of the technique to an archaeological dataset (Merzbach LBK), because it is the coding logic I’ve followed in my own application.

I picked up two new subjects that could be explored in the discussion section: **sampling noise** and **selection coefficient**. The former I haven’t read the section which covers it in the article, but the latter I have, and there’s some interesting things that can be said. Firstly, in the paper the authors apply a power analysis between three different techniques applied to time series genetic data (with limited conditions). The conclusion is that the **FIT is the one which performs best**, with some implications:

1. **FIT gains power with stronger *s*** (coefficient of selection), but in long time series they start to lose power when *s* becomes stronger. This can be explained because the test essentially detects if an increase in allele frequency is statistically significant (i.e., different from zero with α = 0.05), so:
   * A graph of a number of frequency

     AI-generated content may be incorrect.If time series are short, the dynamic of increase due to strong *s* will be detected by the test.
   * If time series are long, and *s* is strong, alleles selected will reach fixation and sampling from that point bears no information about frequency dynamics (generations 800-1000), i.e. samples become **uninformative**. This is something similar to what is expressed by (Lyman, 2008) as “sampling until redundancy”.
2. There is an **optimal sample interval** in which the test performs best as it captures the information relevant to the frequency fluctuations within the time series. So, if our objective is to maximise the power of the test, we should tell the algorithm to apply the test in that **“maximised window**” *spower(T)*. And another relevant matter is that with more *L* or number of sampled points, FIT power increases, so we should maximise the number of sampled points within the maximised window.

Figure 3 from Feder et al. (2014), illustrating the theoretical frequency fluctuations based on different coefficients of selection. Dashed lines represent sampling points.

Moving on to the next task for today, two subtasks emerge: reporting non-NA cases as the “Neutral Detection Rate” (NDR) reporting the proportion of NAs, and correctly applying time averaging as it is conceived in the archaeological realm.

The first task stems from a misinterpretation of the results from my part. Enrico carefully pointed out that what I called the “True Neutral Rate”, usually 83% recorded over 100 runs, effectively is not 83%, it’s somewhere between 83-100% because there are 79 which should be classified but they are not. If we don’t account for 79 NAs there is a 100% accuracy, but this is biased by the user-specified rule of more than 3 time steps (it’s not really user-specified, but a methodological rule, the test cannot run successfully with less than 3 time steps). Therefore, **we gain accuracy, but we lose representation of the population**. For this reason, I should report “Neutral Detection Rate” (NDR) as well as NAs and “False Positive Rate” (FPR), instead of a True Neutral Rate, because it is not a true neutral rate because we ignore the instances in which the test couldn’t complete the analysis, answering with NA.

The time averaging pipeline went from this:

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**))**

**}**

unique\_variants **<-** sort**(**unique**(**as.vector**(**averaged\_matrix**)))** # store variants

averaged\_rows **<-** floor**(**timesteps **/** time\_window**)** # round down to nearest whole number

# list of time-averaged samples (each is a vector of N \* time\_window variants)

averaged\_samples **<-** vector**(**"list", averaged\_rows**)**

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

start **<-** **(**j **-** 1**)** **\*** time\_window **+** 1

end **<-** j **\*** time\_window

# Flatten all traits in the time window into one vector

averaged\_samples**[[**j**]]** **<-** as.vector**(**traitmatrix**[**start**:**end, **])**

**}**

unique\_variants **<-** sort**(**unique**(**unlist**(**averaged\_samples**)))** # store unique variants across all bins

And with subsequent modifications to the pipeline:

After running with the following parameters: N = 100, µ = 0.01, burnin = 1000, time steps = 1000, significance level = 0.05, runs = 100, time window = 20. The results are as follows:

# Trait matrix to frequency matrix

freq\_mat **<-** t**(**apply**(**traitmatrix, 1, **function(**row**)** **{**

tab **<-** table**(**factor**(**row, levels **=** unique\_variants**))**

as.numeric**(**tab**)/**N # Convert to frequencies

**}))**

colnames**(**freq\_mat**)** **<-** unique\_variants # give names

# Prepare FIT input

freq\_long **<-** as.data.frame**(**freq\_mat**)** %>%

mutate**(**time **=** 1**:**timesteps**)** %>%

pivot\_longer**(-**time, names\_to**=**"variant", values\_to**=**"freq"**)** %>% # long format

filter**(**freq **>** 0**)** %>% # remove zeros

mutate**(**variant **=** as.integer**(**variant**))**

# Trait matrix to frequency matrix (row = bins, col = variants)

freq\_mat **<-** t**(**sapply**(**averaged\_samples, **function(**traits**)** **{**

tab **<-** table**(**factor**(**traits, levels **=** unique\_variants**))**

as.numeric**(**tab**)** **/** length**(**traits**)** # Proportions relative to N \* time\_window

**}))**

colnames**(**freq\_mat**)** **<-** unique\_variants

# Prepare FIT input

freq\_long **<-** as.data.frame**(**freq\_mat**)** %>%

mutate**(**time **=** 1**:**nrow**(**.**))** %>%

pivot\_longer**(-**time, names\_to**=**"variant", values\_to**=**"freq"**)** %>% # long format

filter**(**freq **>** 0**)** %>% # remove zeros

mutate**(**variant **=** as.integer**(**variant**))**

A graph with blue squares

AI-generated content may be incorrect.

Clearly, something’s going on. But it will be discussed tomorrow. The explanation of the code modifications will be done tomorrow too.

**Day #11 – 05/05/25**

Let’s explain the changes carried out to the time averaging section of the pipeline:

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**))**

**}**

unique\_variants **<-** sort**(**unique**(**as.vector**(**averaged\_matrix**)))** # store variants

averaged\_rows **<-** floor**(**timesteps **/** time\_window**)** # round down to nearest whole number

# list of time-averaged samples (each is a vector of N \* time\_window variants)

averaged\_samples **<-** vector**(**"list", averaged\_rows**)**

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

start **<-** **(**j **-** 1**)** **\*** time\_window **+** 1

end **<-** j **\*** time\_window

# Flatten all traits in the time window into one vector

averaged\_samples**[[**j**]]** **<-** as.vector**(**traitmatrix**[**start**:**end, **])**

**}**

unique\_variants **<-** sort**(**unique**(**unlist**(**averaged\_samples**)))** # store unique variants across all bins

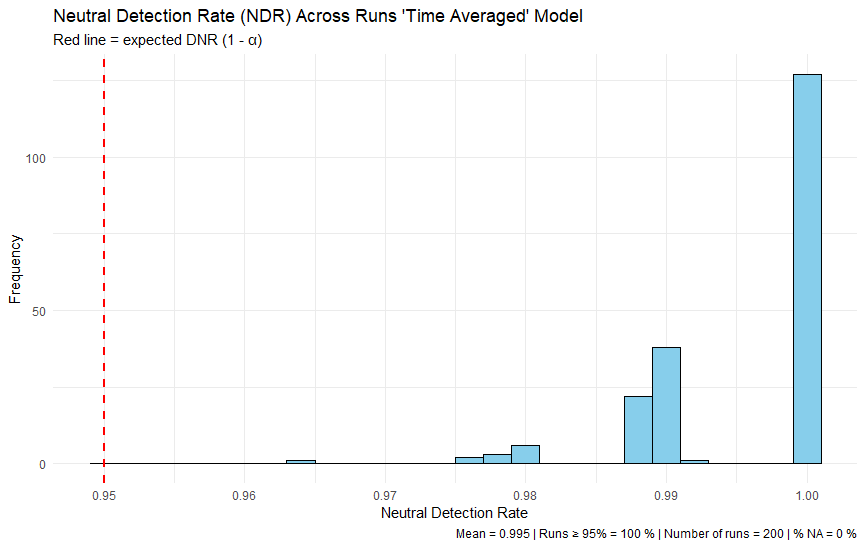
What I had previously done is sampling randomly one variant for each time window for each individual, so per time window I get N observations, we are subsampling the actual population.

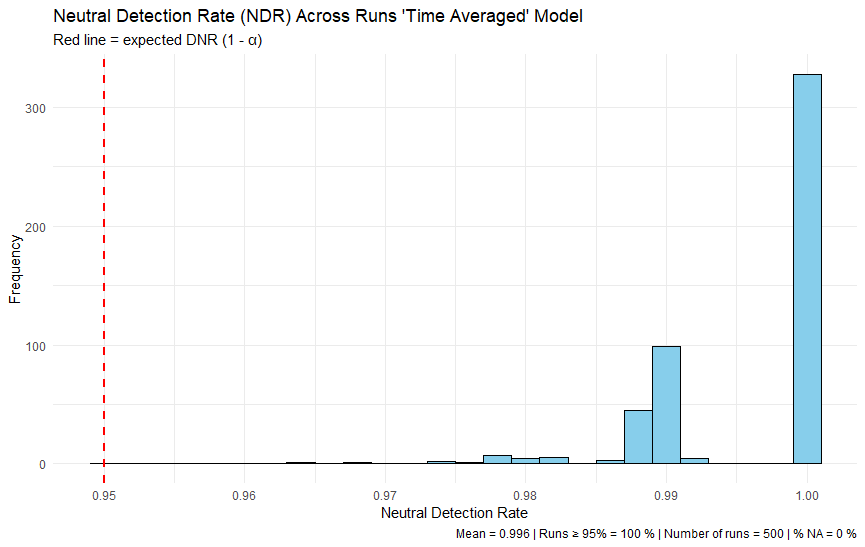
The logic of time averaging is that if our population is defined by N number of individuals per generation (in this case, 100), and the hypothetical stratum excavated spans for 20 generations (time window size), we potentially have **2000 discarded artefacts** per time window to sample in an assemblage, not 100, if behaviour is stable. Therefore, with longer time window size we’ll have more discarded artefacts.

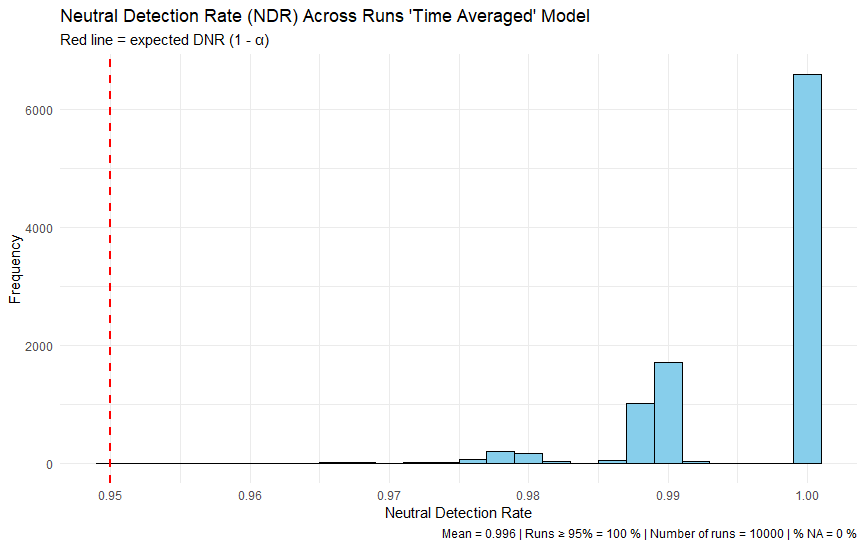
The number we give to time window size is indicative of how many generations span the stratum or layer that we excavate. It is important to highlight that **time windows are not homogeneous in archaeological deposits**, i.e. all strata are not the same. But for the sake of the experiment, we’ll keep the parameter constant.

In the pipeline we are essentially flattening all traits in the time window into a vector and storing it into a list in which each element is a bin (number of time averaged units).

Now that it is clearer, I tried running some simulations with the current pipeline and the results are as follows. Parameters are N = 100, µ = 0.02, burn-in = 1000, time steps = 1000, p value = 0.05 and number of runs is as indicated in the legend of the figures, 200, 500 and 10,000.

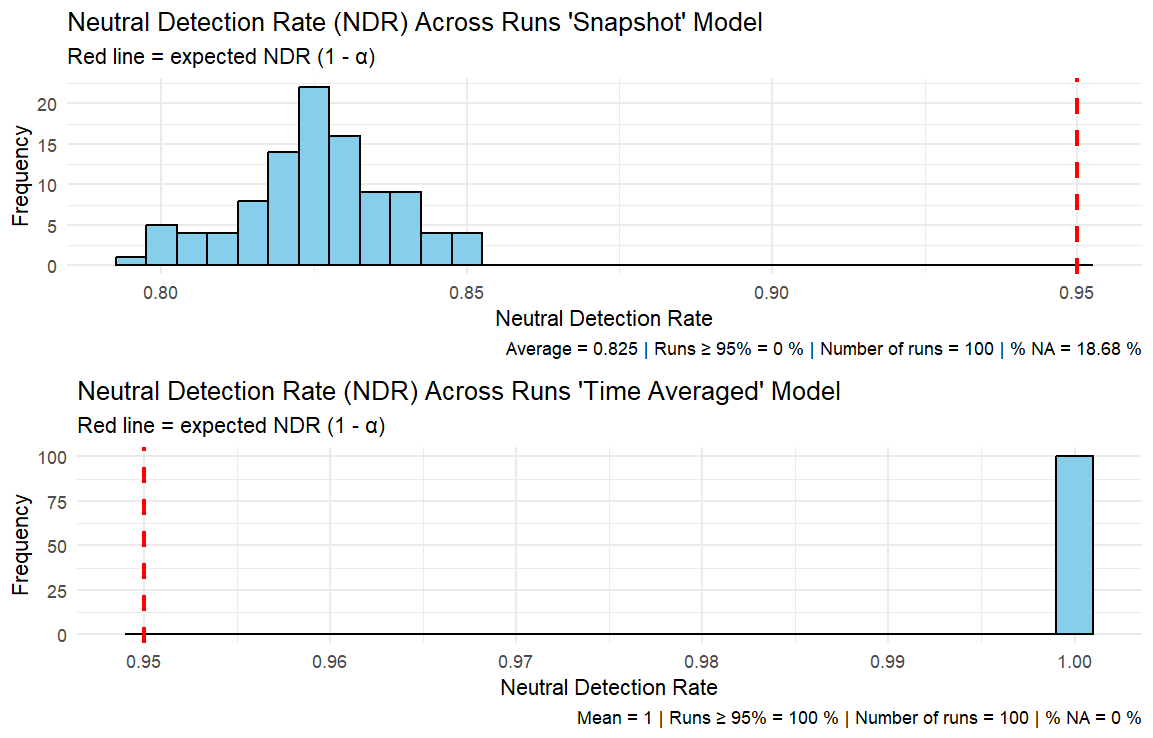






The most surprising thing is not only that the accuracy is quite high (99.6%), but rather that NAs have virtually disappeared. All runs surpass the 95% threshold, which is good, but let’s examine exactly what’s going on.

If there are no NAs it means that the test has detected, on average, 99.6% of the times the stochasticity we simulated (Neutral Detection Rate), but somehow it has detected 0.04% of selection (False Positive Rate). Does this mean that the test has successfully ran on all eligible variants across all bins? In one instance, in the previous version of the time averaged pipeline I got 14.6% NAs, which was less than the snapshot model (~21%). But now it’s 0. The following figure illustrates two simulations with the following parameters: N = 100, µ = 0.2, burn-in = 1000, time steps = 1000, significance level = 0.05, time window size = 20, and number of runs = 100.



There were **no model-fitting failures**, so all variants had at least 3 time points. This is because by aggregating more observations (before we had one variant per individual per bin, now we have one variant per individual \* time window per bin) we reduce the likelihood that variants appear too briefly. This is what Enrico pointed out in the meeting: the 83% that we were recording before was ~100% if we accounted on all eligible variants. But now that all observations are eligible, we get close to the true number.

Summary

Today I conveniently modified and ran the pipeline with varying parameters, as shown in the figures above. In the time averaged version, I think I’ve got rid of the model-fitting problem which was producing a significant proportion of NAs (~15%). Now the model is provisioning more observations, and across all runs**, the test has at its disposal all eligible observations**, therefore not producing any NAs. As a result, the average neutral detection rate is close to 100%, ~99.6%. The instances in which selection is detected probably show a very low coefficient of selection, something worth exploring.

For tomorrow:

1. Close reading of Newberry et al. (2017) for the methods section.
2. Start writing the methods as it is organised in the document.
3. Compare distributions with similar parameters and write down observations worth making.
4. vApply coefficient of selection and population size estimates using tsinfer() and explore possibilities.

**Day #12 – 07/05/25**

Today I started the PPT presentation due on Monday. I want to finish it before the weekend so I can spend full time on the second experiment. I created a new R file with the code that I’m using to generate the figures for the presentation. I am basically giving an overview of the technique (FIT), inputs and outputs, the objective of the dissertation and how to achieve it (power analysis, true positive and true negative for both experiments), the experiment design and justification, some preliminary results and a brief discussion/conclusion.

There’s not much that I can say right now, apart from the fact that the test performs well in the neutral snapshot model (average of ~84%), but even better when time averaging is applied (~99.6%). This seems logical for the following reasons:

* It aggregates generations, so small fluctuations in the short term are smoothed and **sampling variance is reduced**. We get rid of the “noise” of short-term processes.
* The simulation creates more eligible variants per bin, so **short-lived variants persist** and are analysable.
* With more frequencies analysed by the test, there is enough data to estimate the slope with which the FIT performs.

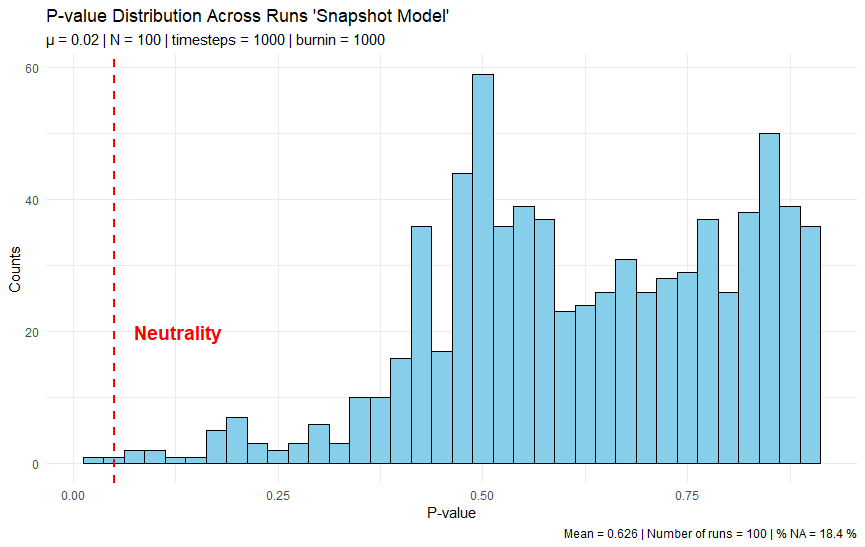
We can start considering a better theoretical performance in real life scenarios in which time averaging is at play (archaeological assemblages). We gain accuracy but we lose precision, because effectively we are missing out the real fluctuations. Therefore, it is **scientifically sufficient**, which is the goal.

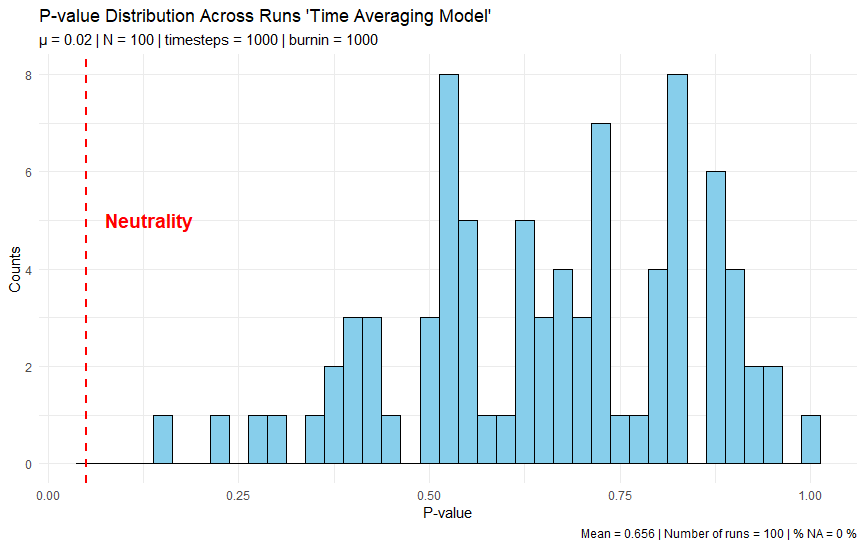
For tomorrow I’ll carry on with the tasks I left undone for today (proposed in day #11), plus go forward in the presentation. First thing will be tasks 1 and 2, followed by presentation, and if there’s time, task 4 (3 I did that same day, hence the figure attached above, although recreating the time averaging figure with a smaller bin size would be more convenient to illustrate the results with more detail).

**Day #13 – 08/05/25**

After reading Newberry et al. (2017) I decided to add a new plot to visualise p-value distribution across runs. This is done to directly observe how many observations fall into the acceptance region, which is what we’re interested in. It’s a basic visualisation, but still useful. The plot illustrates that none of the versions of the neutral model show relevant aggregations around or even below the significance level. There is one thing worth noting though, the snapshot model does present few significant observations and a more regular distribution (**right skewed**, as shown in the figure below). On the other hand, the time averaged model shows a more uniform and discrete distribution.

We should bear in mind that the snapshot model is effectively showing us the p-value of those observations which were not recorded as NA. We still report the proportion of NAs (18.4%) just to inform us that these are not included in the plot. Snapshot model mean p-value is **0.626**, and the TA version mean p-value is **0.656**, which aligns with what is expected. Some variants even record p-values close to 1.0 (8 variants over 0.90).





The distribution of the time averaging model version shows less counts per p-value recorded because we have less time points (averaged out) and **there are more variants which were recorded for less than 3 time points that we filtered out**. Yes, we have more unique variants, but over 3 time points much less than the snapshot version. One thing worth trying next day is running the time averaged version without the 3-time-points rule; perhaps the issue behind the rule is neutralised with smoother time conditions as simulated by time averaging. Also tracking variants that are filtered out in both variations and comparing could be interesting.

And both codes:

p\_value\_distribution\_snapshot **<-** ggplot**(**data **=** fit\_results, aes**(**x **=** fit\_p**))** **+**

geom\_histogram**(**binwidth **=** 0.025, fill **=** "skyblue", color **=** "black"**)** **+**

geom\_vline**(**xintercept **=** 0.05, linetype **=** "dashed", color **=** "red", linewidth **=** 1**)** **+**

annotate**(**"text", x **=** 0.075, y **=** 5, label **=** "Neutrality",

color **=** "red", size **=** 3.5, fontface **=** "bold", hjust **=** 0**)** **+**

labs**(**

title **=** "P-value Distribution Across Runs 'Snapshot Model'",

subtitle **=** paste**(**"µ =", mu, "| N =", N, "| timesteps =", timesteps, "| burnin =", burnin**)**,

x **=** "P-value",

y **=** "Counts",

caption **=** paste**(**

"Mean =", round**(**mean**(**fit\_results**$**fit\_p, na.rm **=** **TRUE)**, 3**)**, "|",

"Number of runs =", n\_runs, "|",

"% NA =", round**(**proportionNA, 2**)**, "%"

**)**

**)** **+**

coord\_cartesian**(**clip **=** "off"**)** **+** # allows text to overflow if needed

theme\_minimal**()**

p\_value\_distribution\_snapshot

p\_value\_distribution\_ta **<-** ggplot**(**data **=** fit\_results, aes**(**x **=** fit\_p**))** **+**

geom\_histogram**(**binwidth **=** 0.025, fill **=** "skyblue", color **=** "black"**)** **+**

geom\_vline**(**xintercept **=** 0.05, linetype **=** "dashed", color **=** "red", linewidth **=** 1**)** **+**

annotate**(**"text", x **=** 0.075, y **=** 5, label **=** "Neutrality",

color **=** "red", size **=** 3.5, fontface **=** "bold", hjust **=** 0**)** **+**

labs**(**

title **=** "P-value Distribution Across Runs 'Time Averaging Model'",

subtitle **=** paste**(**"µ =", mu, "| N =", N, "| timesteps =", timesteps, "| burnin =", burnin**)**,

x **=** "P-value",

y **=** "Counts",

caption **=** paste**(**

"Mean =", round**(**mean**(**fit\_results**$**fit\_p, na.rm **=** **TRUE)**, 3**)**, "|",

"Number of runs =", n\_runs, "|",

"% NA =", round**(**proportionNA, 2**)**, "%"

**)**

**)** **+**

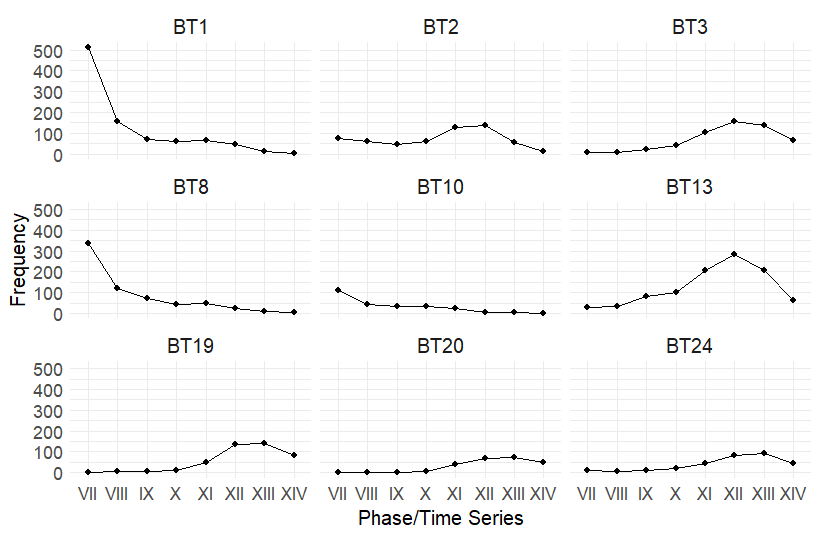
coord\_cartesian**(**clip **=** "off"**)** **+** # allows text to overflow if needed

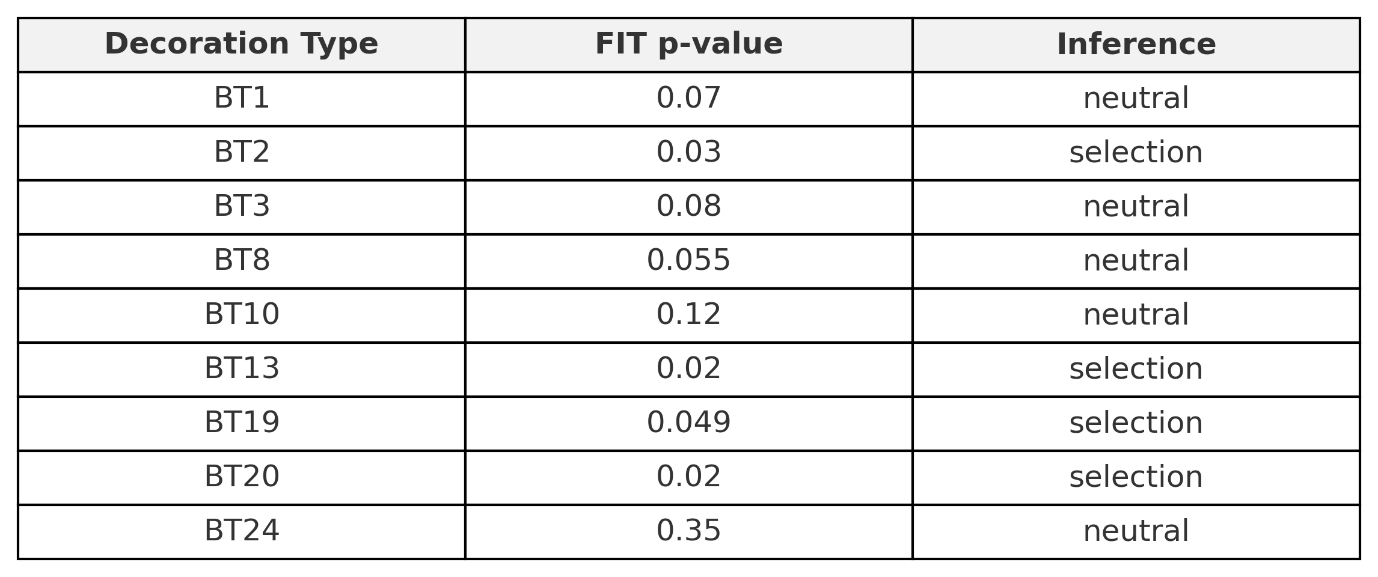
theme\_minimal**()**

p\_value\_distribution\_ta

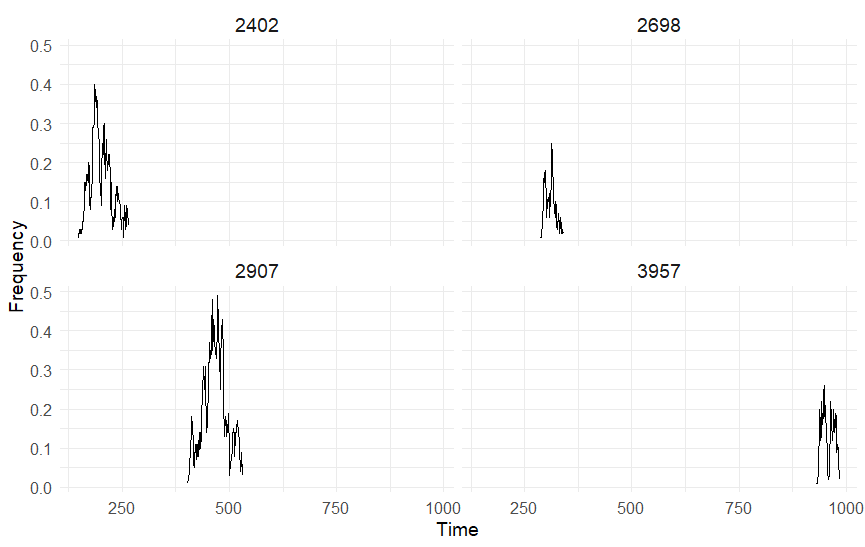
**Day #14 – 11/05/25**

I spent the whole session of today doing the presentation. I had to prepare more figures because they are the forte of the dissertation. It’s quite difficult to illustrate the kind of thing I’m doing because it’s abstract and purely theoretical; I could spend some time thinking ways to describe the idea in a creative way, but for the time that I am asked to speak (5 minutes), it’s not really worth.

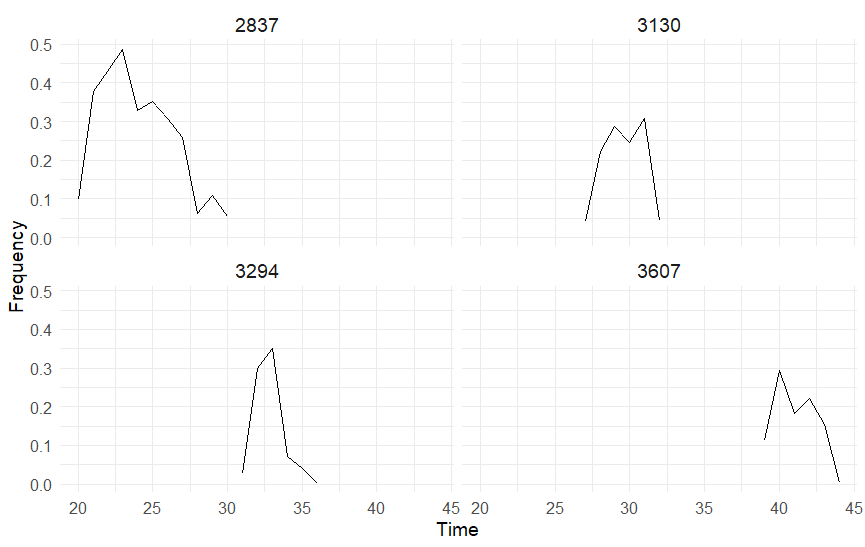




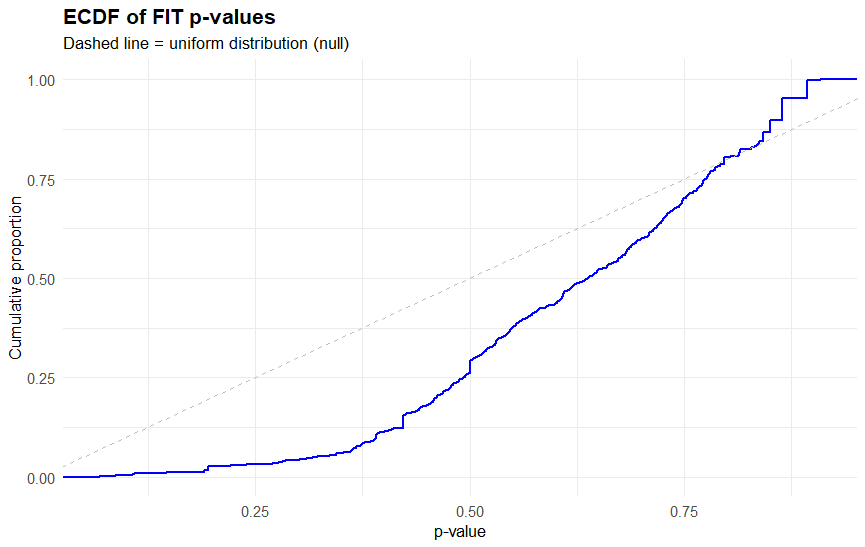
This figure and table are shown to illustrate how the FIT works: the input and output of the test. The data used comes from “evoarchdata”, the R package containing datasets from several assemblage collected to which an evolutionary approach was carried out. In particular, this one is the widely used LBK Merzbach dataset.



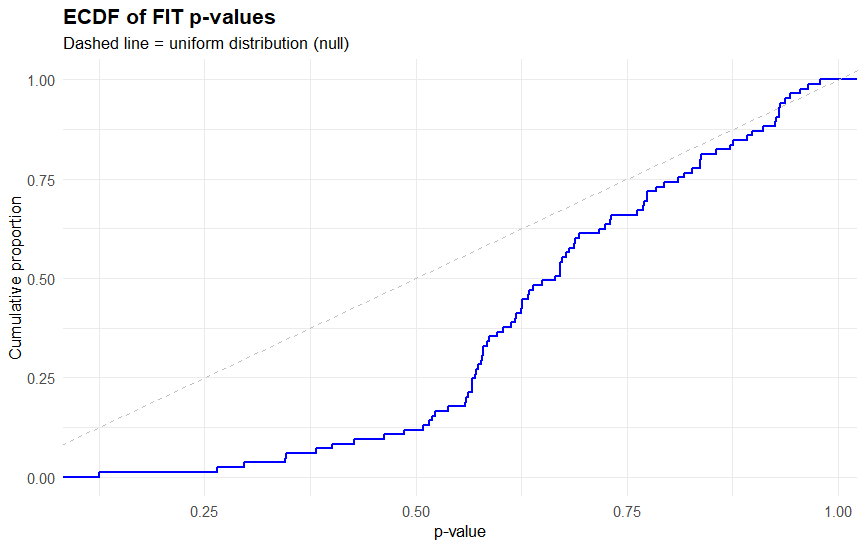
This figure represents four variants taken randomly from the simulation of the neutral snapshot model.



And this figure represents four variants taken randomly from the simulation of the neutral time averaging model. The differences are clear: the small-scale fluctuations are averaged out by aggregating time steps into wider windows. Each curve is smoothed out by the effects of time averaging, meaning that our approach, although not perfect, it’s close to the phenomenon described.



Snapshot version of p-value cumulative proportion.



Time averaged version of p-value cumulative proportion (estimated cumulative d

**Day #14 – 12/05/25**

Tuesday, I spent the whole session preparing the presentation for the next day. I finished the Power Point and wrote the script so that I wouldn’t exceed 5 minutes. As I was writing it, some ideas came to mind.

1. In the snapshot version of the neutral transmission experiment the possible correlation between low time points and NAs, if statistically significant, might act as a **handicap** which can’t be controlled in real life archaeological scenarios. The snapshot model is just a tarting point to test the accuracy of the test; its utility does not transcend that purpose.
2. The meeting with Enrico also helped me organise the parameter exploration in a more systematic way. He suggested **running multiple simulations** with a **fixed combination of parameters**. And then store the results and plot them in both ways: **boxplots** to illustrate different parameters in a discontinuous way; **scatterplot** to show the gradient of parameter value (e.g., 0.1 to 0.9). A good way to initially try this is carrying it out with **extreme values** first and recording NAs under different parameters.
3. In the time averaged version, I can mention the debate over **what is truly N**, number of artifacts, number of individuals who can craft those artifacts… For this question might be worth giving another look at Neiman (1995) who wrote a brief discussion on what assumption of N is best.

**Day #15 – 15/05/25**

Today I have added to the code an attempt to make the pipeline a function() to speed up the process of running the simulations with different parameters.

The pipeline is identical, except that I added the function and parameters and inserted the vectors to record NDR and a list to store p-values.

neutral\_snapshot **<-** **function(**N, mu, burnin, timesteps, p\_value\_lvl, n\_runs**)** **{**

neutral\_counts\_per\_run\_snapshot **<-** numeric**(**n\_runs**)** # empty vector for counting neutral variants

accuracy\_snapshot **<-** numeric**(**n\_runs**)** # empty vector for accuracy tracking each run

fit\_p\_count **<-** vector**(**"list", n\_runs**)** # store p-values

# Store results

mean\_accuracy **<-** mean**(**accuracy\_snapshot, na.rm **=** **TRUE)** # mean accuracy across runs

# Proportion runs have 95% detection

over95 **<-** sum**(**accuracy\_snapshot **>=** 0.95**)**

high\_accuracy\_runs **<-** over95**/**n\_runs**\***100

# Proportion NA across runs

sumNA **<-** sum**(**fit\_results**$**sig **==** "NA"**)**

proportionNA **<-** sumNA**/**length**(**fit\_results**$**sig**)\***100

# p-value count

fit\_p\_count**[[**run**]]** **<-** fit\_results**$**fit\_p

Additionally, at the end of the function, I stored the output and parameters to plot them:

# Store output

return**(**list**(**

accuracy\_snapshot **=** accuracy\_snapshot,

mean\_accuracy **=** mean\_accuracy,

high\_accuracy\_runs **=** high\_accuracy\_runs,

sumNA **=** sumNA,

proportionNA **=** proportionNA,

fit\_p\_count **=** fit\_p\_count,

N **=** N,

mu **=** mu,

burnin **=** burnin,

timesteps **=** timesteps,

p\_value\_lvl **=** p\_value\_lvl,

n\_runs **=** n\_runs

**))**

**}**

And to call the function:

# Run simulation with parameters

n\_snap\_sim **<-** neutral\_snapshot**(**N **=** 100, mu **=** 0.02, burnin **=** 1000,

timesteps **=** 1000, p\_value\_lvl **=** 0.05, n\_runs **=** 100**)**

And made two new functions for both plots, NDR and p-value distribution:

# NDR

plot\_neutral\_snapshot **<-** **function(**n\_snap\_sim, binwidth **=** 0.005**)** **{**

ggplot**(**data **=** data.frame**(**NDR **=** n\_snap\_sim**$**accuracy\_snapshot**)**, aes**(**x **=** NDR**))** **+**

geom\_histogram**(**binwidth **=** binwidth, fill **=** "skyblue", color **=** "black"**)** **+**

geom\_vline**(**xintercept **=** 0.95, linetype **=** "dashed", color **=** "red", linewidth **=** 1**)** **+**

labs**(**title **=** "Neutral Detection Rate (NDR) Across Runs 'Snapshot' Model",

subtitle **=** "Red line = expected NDR (1 - α)",

x **=** "Neutral Detection Rate",

y **=** "Frequency",

caption **=** paste**(**"Average =", round**(**mean**(**n\_snap\_sim**$**accuracy\_snapshot**)**, 3**)**, "|",

"Runs ≥ 95% =", round**(**n\_snap\_sim**$**high\_accuracy\_runs, 1**)**, "%", "|",

"Number of runs =", n\_snap\_sim**$**n\_runs, "|",

"% NA =", round**(**n\_snap\_sim**$**proportionNA, 2**)**, "%"**))** **+**

theme\_minimal**()**

**}**

plot\_neutral\_snapshot**(**n\_snap\_sim**)**

# P-value distribution

p\_value\_distr\_snap **<-** **function(**n\_snap\_sim, binwidth **=** 0.025**)** **{**

p\_vals **<-** unlist**(**n\_snap\_sim**$**fit\_p\_count**)**

ggplot**(**data **=** data.frame**(**p\_value **=** p\_vals**)**, aes**(**x **=** p\_value**))** **+**

geom\_histogram**(**binwidth **=** binwidth, fill **=** "skyblue", color **=** "black", na.rm **=** **TRUE)** **+**

geom\_vline**(**xintercept **=** n\_snap\_sim**$**p\_value\_lvl, linetype **=** "dashed", color **=** "red", linewidth **=** 1**)** **+**

labs**(**title **=** "DIstribution of FIT P-values Across Runs",

subtitle **=** paste0**(**"Red line = α threshold (", n\_snap\_sim**$**p\_value\_lvl, ")"**)**,

x **=** "p-value",

y **=** "Frequency",

caption **=** paste**(**"Total p-values =", length**(**p\_vals**)**, "|",

"Runs =", n\_snap\_sim**$**n\_runs, "|",

"% NA =", round**(**mean**(**is.na**(**p\_vals**))** **\*** 100, 2**)**,"%"

**)**

**)** **+**

theme\_minimal**()**

**}**

p\_value\_distr\_snap**(**n\_snap\_sim**)**

**Day #16 – 17/05/25**

I spent some time evaluating possible caveats of the pipeline, and discovered several:

1. I am not using the vector neutral\_counts\_per\_run\_snapshot <- numeric(n\_runs) in the pipeline, so I got rid of it. I already compute the neutral count at the end, so it’s not essential.
2. I added fit\_p\_count <- vector(“list”, n\_runs) to store the p-values across runs, so that I can compute summary statistics in the results section. I will refer to the p-values to compute the sum and proportion of NA.
3. There is a chance that in one run the number of total variants that survived is 0, turning the NDR formula into a dead-end. For this reason, I added:

**if(**total\_variants **==** 0**)** **{** # if no variants survive NA is returned

FPR **<-** **NA**; NDR **<-** **NA**

**}**

1. Also, by storing all the p-values across runs, we slightly change the syntax of the output to this:

all\_pvals **<-** unlist**(**fit\_p\_count**)** # list of p-values

# Proportion of NA from the simulation

sumNA **<-** sum**(**is.na**(**all\_pvals**))**

sumNA

proportionNA **<-** sumNA **/** length**(**all\_pvals**)** **\*** 100

proportionNA

# P-values across runs

mean\_p\_value **<-** mean**(**all\_pvals, na.rm **=** T**)**

mean\_p\_value

1. I added a conditional, if the number of summed NAs equals 0, the proportion of NA is 0. This is done to avoid NaN in a table where the results of the simulations are stored:

**if(**sumNA **==** 0**)** **{**

proportionNA **<-** 0

**}**

In every pipeline I’ve added two new features: a table to store the output of the experiment (a), and a table storing several parameter-sets (b) (swap “snap” for “ta” to change models):

a)

# Store output across runs in a table

results\_table\_neutral\_snapshot **<-** tibble**(**

N **=** n\_snap\_sim**$**N,

mu **=** n\_snap\_sim**$**mu,

burnin **=** n\_snap\_sim**$**burnin,

timesteps **=** n\_snap\_sim**$**timesteps,

p\_value\_lvl **=** n\_snap\_sim**$**p\_value\_lvl,

n\_runs **=** n\_snap\_sim**$**n\_runs,

mean\_accuracy **=** n\_snap\_sim**$**mean\_accuracy,

high\_accuracy\_runs **=** n\_snap\_sim**$**high\_accuracy\_runs,

proportionNA **=** n\_snap\_sim**$**proportionNA,

mean\_p\_value **=** mean**(**n\_snap\_sim**$**all\_pvals, na.rm **=** **TRUE)**

**)**

results\_table\_neutral\_snapshot

b)

# Run many parameter-sets and stack the results

params\_neutral\_snapshot **<-** list**(**

list**(**N**=**100, mu**=**0.02, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**200, mu**=**0.02, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**

**)**

all\_results\_neutral\_snapshot **<-** map\_dfr**(**params\_neutral\_snapshot, **~** **{**

sim **<-** do.call**(**neutral\_snapshot, args **=** .x**)**

tibble**(**N **=** .x**$**N,

mu **=** .x**$**mu,

burnin **=** .x**$**burnin,

timesteps **=** .x**$**timesteps,

p\_value\_lvl **=** .x**$**p\_value\_lvl,

n\_runs **=** .x**$**n\_runs,

mean\_accuracy **=** sim**$**mean\_accuracy,

high\_accuracy\_runs **=** sim**$**high\_accuracy\_runs,

proportionNA **=** sim**$**proportionNA,

mean\_p\_value **=** mean**(**sim**$**all\_pvals, na.rm **=** **TRUE)**

**)**

**})**

print**(**all\_results\_neutral\_snapshot**)**

The purpose of creating two functions storing both snapshot and time averaging pipelines is to be able to run each model independently. In the previous version the parameters were shared, so every time I wanted to plot each version, I had to run the pipeline. Now I’ve got two functions to run the simulations and plot both NDR and p-value distribution, each calling one different version. The results were stored in fit\_results, but **now they are stored within each function as a list**, and I can call and plot the outcomes and parameters independently.

I successfully implemented the function task, so now I have two functions calling: each version of the neutral transmission experiment, and two plots (NDR and p-value distribution) for each one. The following step is applying content bias following the same procedure. I’ve kept the older syntax of the pipeline (where the output is stores outside of the loop, and there is no function wrapping everything), but I might delete it or keep it in a file. At some point I should format the code and get rid of some messy aspects, but this I will do later on. For now, I will focus on the second experiment.

**Day #17 & #18 – 18-19/05/25**

Today the objective was to apply content bias, but first, I had to revisit the theoretical notion to emulate it in the simulations as similar as possible. It’s not easy, especially when reaching the output storing in the pipeline. Effectively, the NDR was calculated under the assumption that all variants were selectively neutral, i.e., their frequencies changed relative to the frequency of the rest of the variants within the population:

A black background with white spots

AI-generated content may be incorrect.

So, all variants were expected to be neutral. But when selection is at play **all variants can’t be selected** at a single time point. Throughout the time series all variants could have undergone selection, but they are not expected to, therefore computing the SST relative to the total number of variants tested does not express what we are looking for. We’re also interested in the **coefficient of selection**, which is the estimation of the strength of selection. According to Acerbi et al. (2022) it equals the relative difference between the biased probability of that variant and the rest in the same generation (expressed as vi as follows):

A black background with a black square

AI-generated content may be incorrect.

It is the measurement of the quality of that variant that increases its probability of being socially learned. In a neutral transmission case, the global S must be 0, at least in our simulations because all variants represent the same probability. The tsinfer() function has a parameter to measure the relative changes in frequency similar to coefficient of selection, which is expressed by **s.0**. Selected variants must show 0 s.0, but their corresponding S. So, our expectations are, if variants are neutral, they should be equal or less than 0:

* If equal to 0, it means that their frequency will remain constant (but this is unlikely if several variants are at play).
* If less than 0 it means that they lose frequency because another variant is being learned, hence selectively negative.

There is, although, one misleading evidence shown in the application of Ben Marwick which I don’t understand: there are neutrally detected variants in the LBK Merzbach dataset that are detected as **neutral** but show a positive S (such as BT13 and BT2). Furthermore, if we plot the frequency across time series, we see that **BT13 notoriously increases its relative frequency** in phase 12 to 0.3 but is still detected as neutral.

However, in the Signal Selection Test, S represents a global estimate; it is a **maximum likelihood estimation** of the bias towards each variant under the Wright-Fisher neutral model assumption with selection. It’s not an absolute measurement, but instead, it is a likelihood estimation. This means that **sampling noise** can lead some variants which are not statistically different from 0 (neutral) to still present a tiny proportion of S. This is the case for the variants pointed out, although the frequency increase remains a weird matter… The rest of neutral variants have a negative S (BT1 even reaches -1, which equals extinction).

| **type** | **s** | **f0** | **LL** | **p\_value** | **inference\_tag** |
| --- | --- | --- | --- | --- | --- |
| BT20 | 0.69865040 | 0.0119200970 | -11,568.20895 | 0.0089461917 | selection |
| BT19 | 0.24232994 | 0.0002989482 | -1,406.15933 | 0.0011454026 | selection |
| BT24 | 0.12437378 | 0.9928740526 | -134.34427 | 0.0004282439 | selection |
| BT3 | 0.21293353 | 0.8263663325 | -429.96496 | 0.0061753689 | selection |
| BT13 | 0.06702107 | 0.9862020469 | 36.31373 | 0.3221977173 | neutral |
| BT2 | 0.08610147 | 0.0099631665 | 32.03084 | 0.9464648980 | neutral |
| BT10 | -0.26641737 | 0.0012675424 | -1,724.20636 | 0.1411941851 | neutral |
| BT8 | -0.32301459 | 0.0017857961 | -580.34247 | 0.0039190078 | selection |
| BT1 | -1.00405576 | 0.9984530922 | Inf | 0.0501846274 | neutral |

A graph of different colored lines

AI-generated content may be incorrect.

This is a good clarification, because now we know that **the output of our second experiment must account for positive S**. But as not all positive S variants are detected as selection due to sampling error, we should draw a line that acts as a **statistical threshold**. However, we are adding a bias, so the next thing to do is figure out an unbiased way of applying that threshold.

**Day #19 & #20 – 20-21/05/25**

I think I am addressing the question in the wrong way. Let’s go back to the basic theory:

|  |  |  |
| --- | --- | --- |
|  | ***H0 =* True** | ***H0 =* False** |
| Reject *H0* | Type I error  (α) | “True” Negative  (1 – β) |
| Accept *H0* | “True” Positive (1 – α) | Type II error  (β) |

When computing NDR the result is not actually reflecting 1 – α because of the high number of variants that were not detected by the test. When running the simulation >100 iterations there are some instances in which the test detects selection, probably due to sampling error. However, in 100 iterations usually no variants are selected, therefore, the NDR is 100%. It is ~82% because of the NA proportion (which should add up to 100).

To revisit the main question now: what denominator represents best what I am aiming for, statistical power.

Before going into detail, I should report that I have finally ran the different simulation settings, with the following outputs:

Neutral Snapshot

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | µ | Burn-in | | Time steps | α | Runs | NDR | % NA | Mean p-value |
| 100 | 0,01 | 1000 | 1000 | | 0,05 | 10 | 0,822 | 17,62 | 0,618 |
| 100 | 0,025 | 1000 | 1000 | | 0,05 | 10 | 0,82 | 17,92 | 0,626 |
| 100 | 0,05 | 1000 | 1000 | | 0,05 | 10 | 0,813 | 18,56 | 0,632 |
| 100 | 0,075 | 1000 | 1000 | | 0,05 | 10 | 0,805 | 19,45 | 0,639 |
| 100 | 0,1 | 1000 | 1000 | | 0,05 | 10 | 0,791 | 20,82 | 0,644 |
| 100 | 0,125 | 1000 | 1000 | | 0,05 | 10 | 0,784 | 21,52 | 0,65 |
| 100 | 0,15 | 1000 | 1000 | | 0,05 | 10 | 0,771 | 22,82 | 0,655 |
| 100 | 0,175 | 1000 | 1000 | | 0,05 | 10 | 0,757 | 24,24 | 0,66 |
| 100 | 0,2 | 1000 | 1000 | | 0,05 | 10 | 0,745 | 25,45 | 0,665 |

Neutral Time Averaging

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | µ | Burn-in | Time steps | α | Runs | Window size | NDR | % NA | Mean p-value |
| 100 | 0,01 | 1000 | 1000 | 0,05 | 100 | 20 | 0,995 | 0 | 0,632 |
| 100 | 0,025 | 1000 | 1000 | 0,05 | 100 | 20 | 0,996 | 0 | 0,646 |
| 100 | 0,05 | 1000 | 1000 | 0,05 | 100 | 20 | 0,996 | 0 | 0,661 |
| 100 | 0,075 | 1000 | 1000 | 0,05 | 100 | 20 | 0,997 | 0 | 0,674 |
| 100 | 0,1 | 1000 | 1000 | 0,05 | 100 | 20 | 0,998 | 0 | 0,676 |
| 100 | 0,125 | 1000 | 1000 | 0,05 | 100 | 20 | 0,999 | 0 | 0,675 |
| 100 | 0,15 | 1000 | 1000 | 0,05 | 100 | 20 | 1 | 0 | 0,678 |
| 100 | 0,175 | 1000 | 1000 | 0,05 | 100 | 20 | 0,999 | 0 | 0,68 |
| 100 | 0,2 | 1000 | 1000 | 0,05 | 100 | 20 | 0,998 | 0 | 0,677 |

Firstly, the only parameter modification was innovation rate. The expectation for the snapshot version is that with higher innovation rate more variants are created (increased probability each time step), and the test performs “worse”, at least when it is forced to record frequency changes at each time step. In the column NDR we can see how innovation rate is inversely proportional to NDR, which is something worth plotting and discussing.

This is only a first step in this whole process, and I have probably rushed into parameter exploration before settling content bias, but I wanted to try with both models just to briefly confirm my suspicions. In fact, Wednesday I spend quite a lot of time battling with the troughton server to run the parameter setting list:

params\_neutral\_snapshot **<-** list**(**

list**(**N**=**100, mu**=**0.01, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.025, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.05, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.075, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.1, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.125, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.15, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.175, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**,

list**(**N**=**100, mu**=**0.2, burnin**=**1000, timesteps**=**1000, p\_value\_lvl**=**0.05, n\_runs**=**10**)**

**)**

all\_results\_neutral\_snapshot **<-** map\_dfr**(**params\_neutral\_snapshot, **~** **{**

sim **<-** do.call**(**neutral\_snapshot, args **=** .x**)**

tibble**(**N **=** .x**$**N,

mu **=** .x**$**mu,

burnin **=** .x**$**burnin,

timesteps **=** .x**$**timesteps,

p\_value\_lvl **=** .x**$**p\_value\_lvl,

n\_runs **=** .x**$**n\_runs,

mean\_accuracy **=** round**(**sim**$**mean\_accuracy, 3**)**,

proportionNA **=** round**(**sim**$**proportionNA, 2**)**,

mean\_p\_value **=** round**(**mean**(**sim**$**all\_pvals, na.rm **=** **TRUE)**, 3**)**

**)**

**})**

I basically went through the notes I took from the last meeting with Enrico and tried to run the code. I am becoming more comfortable with the GNU/LINUX syntax and commands, but is still quite hard, I just need to spend some time memorising the keyboard bindings. I created a new session and two .R scripts with both models (neutral snapshot and time averaged), successfully pasting the codes. However, I encountered an error when loading the packages because *signatselect* is not available in CRAN, it is stored in a GitHub repository, so I haven’t managed to install it properly.

Anyway, this shouldn’t bother me at the moment. I should be focusing on figuring out the right approach to implement content bias into the pipeline and store the output. Therefore, tomorrow my main objective will be addressing this exact problem. So far, what I have envisioned is establishing a low threshold (perhaps 0.05) of coefficient of selection as the minimum required for the true negative rate (SSR). We basically presuppose that if a variant shows positive S is selected, but in some rare instances neutral variants can present it only at low values. That’s why setting a low threshold is of vital importance, but still arbitrarily induced…