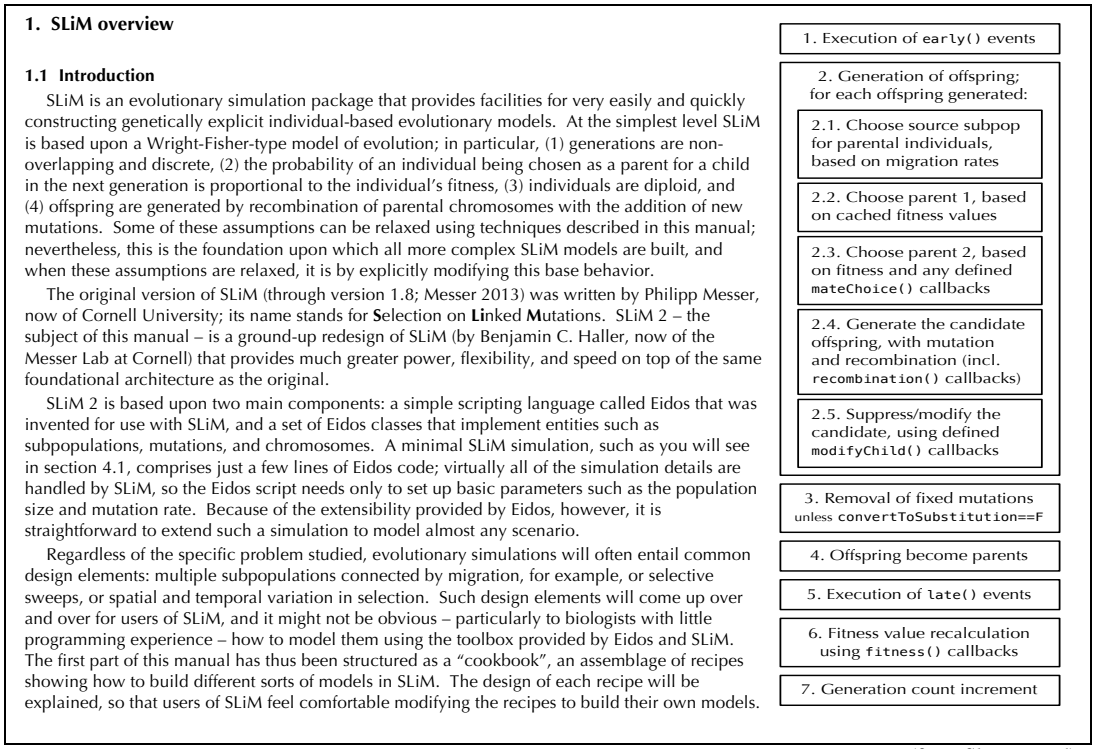
Population Genomics to Adaptation

In this practical session we will simulate a number of different evolutionary scenarios and we will see the different patterns by studying a number of summary statistics: The variability (Theta=4Neμ), The Site Frequency Spectrum (SFS) and the estimation of the proportion of adaptive substitutions (alpha).

1. We will simulate a number of different scenarios using ***Slim*** (Messer, Genetics 2013, Haller and Messer, MBE 2017). Slim is a forward simulator that allows to simulate many selective positions at the same time in complex demographic patterns. Slim2 has a graphical interface for MacOSX but here we will use the **command line program** for the practical session.



1. SLiM overview

1.1 Introduction

SLiM is an evolutionary simulation package that provides facilities for very easily and quickly constructing genetically explicit individual-based evolutionary models. At the simplest level SLiM is based upon a Wright-Fisher-type model of evolution; in particular, (1) generations are non-overlapping and discrete, (2) the probability of an individual being chosen as a parent for a child in the next generation is proportional to the individual's fitness, (3) individuals are diploid, and (4) offspring are generated by recombination of parental chromosomes with the addition of new mutations. Some of these assumptions can be relaxed using techniques described in this manual; nevertheless, this is the foundation upon which all more complex SLiM models are built, and when these assumptions are relaxed, it is by explicitly modifying this base behavior.

The original version of SLiM (through version 1.8; Messer 2013) was written by Philipp Messer, now of Cornell University; its name stands for **S**election on **Li**nked **M**utations. SLiM 2 – the subject of this manual – is a ground-up redesign of SLiM (by Benjamin C. Haller, now of the Messer Lab at Cornell) that provides much greater power, flexibility, and speed on top of the same foundational architecture as the original.

SLiM 2 is based upon two main components: a simple scripting language called Eidos that was invented for use with SLiM, and a set of Eidos classes that implement entities such as subpopulations, mutations, and chromosomes. A minimal SLiM simulation, such as you will see in section 4.1, comprises just a few lines of Eidos code; virtually all of the simulation details are handled by SLiM, so the Eidos script needs only to set up basic parameters such as the population size and mutation rate. Because of the extensibility provided by Eidos, however, it is straightforward to extend such a simulation to model almost any scenario.

Regardless of the specific problem studied, evolutionary simulations will often entail common design elements: multiple subpopulations connected by migration, for example, or selective sweeps, or spatial and temporal variation in selection. Such design elements will come up over and over for users of SLiM, and it might not be obvious – particularly to biologists with little programming experience – how to model them using the toolbox provided by Eidos and SLiM. The first part of this manual has thus been structured as a "cookbook", an assemblage of recipes showing how to build different sorts of models in SLiM. The design of each recipe will be explained, so that users of SLiM feel comfortable modifying the recipes to build their own models.

(from *Slim* manual)

2. In order to run Slim, we need to construct the scripts with the models to study. We will use a **template** that we will modify in relation to the different scenarios, with the help of the *Slim* manual. This template contains orders to run a simulation and extract a sample in ***ms*** format.

   a. We will simulate a 30Kb DNA fragment of a coding region of an initial population of 10K individuals. Then we sample 25 genomes from population of interest and 5 genomes from the outgroup:

```
// set up a simple neutral scenario simulation
initialize() {

  // set the overall uniform mutation rate
  initializeMutationRate(1e-5);
  // uniform recombination along the chromosome
  initializeRecombinationRate(1e-4);

  // m1 mutation type: (neutral)
  initializeMutationType("m1", 0.5, "f", 0.0);
  // m2 mutation type: (deleterious)
  initializeMutationType("m2", 1.0, "f", -1.0);
  // m3 mutation type: (beneficial)
  initializeMutationType("m3", 0.5, "f", 0.005);

  // g1 genomic element type: (synonymous) uses m1 for all mutations
  initializeGenomicElementType("g1", m1, 1.0);
  // g2 genomic element type: (nonsynonymous) uses all mutations
  initializeGenomicElementType("g2", c(m1,m2,m3), c(2,8,0));
```

```
  // Define positions to genomic elements at a chromosome of length 30 kb
  count = 0;
  for (i in 0:30000) {
       count2 = count;
       count = count2 + 1;
       if (count == 1) {
            initializeGenomicElement(g2, i, i);
       } else if (count == 2) {
            initializeGenomicElement(g2, i, i);
       } else if (count == 3) {
            initializeGenomicElement(g1, i, i);
            count = 0;
       }
  }
}

// create a population of 1000 individuals
1{
  sim.addSubpop("p1", 1000);
}

// Split de p1 in the generation 10000.
// Now we have the target pop and the outgroup pop.
5000 { sim.addSubpopSplit("p2", 1000, p1); }

// If required, sudden change in population size
9500 { p2.setSubpopulationSize(1000); }

//if required, force a strong selective sweep
//9900 {
//     target = sample(p2.genomes, 100); //the mutation is in 100 individuals
//     target.addNewDrawnMutation(m3, 15000); //the mutation occurs at position 15000
//}

// Run to the final
10000 late() {
  // Select 5 samples of the outgroup and 25 samples of the target population and output to
MS format
  // obtain random samples of genomes from the three subpopulations
  g_1 = sample(p2.genomes,25,replace=F);
  g_2 = sample(p1.genomes,5,replace=F);
  //Concatenate the 2 population samples
  g_12=c(g_1,g_2);
       //Get the unique mutations in the sample, sorted by position
       m = sortBy(unique(g_12.mutations),"position");
       // print the number of segregating sites
       cat("//" + "\n");
       cat("segsites: "+ size(m) + "\n");
       //print the positions
       positions = format("%.6f", m.position / sim.chromosome.lastPosition);
       cat("positions: "+ paste(positions," ") + "\n");
       //print the sampled genomes
       for (genome in g_12){
            hasMuts = (match(m,genome.mutations) >= 0);
            cat(paste(asInteger(hasMuts),"") + "\n");
       }
}
```

3. The scenarios are the following (15 scenarios, we will divide the task in 15 groups separately):

   a. A two species model **with constant population size** and **no selection** on functional positions. **No recombination** between positions. (SNM-R0)

```
// uniform recombination along the chromosome
initializeRecombinationRate(0.0);
```

   b. A two species model with constant population size and no selection on functional positions. **High recombination** between positions. (SNM-Rh)

```
// uniform recombination along the chromosome
initializeRecombinationRate(1e-4);
```

   c. A two species model with constant population size plus recent **strong positive selection on a single** position in the target population. No recombination between positions. (SS-R0)

```
// uniform recombination along the chromosome
initializeRecombinationRate(0.0);

//if required, force a strong selective sweep
9900 {
     target = sample(p2.genomes, 100); //the mutation is in 100 individuals
     target.addNewDrawnMutation(m3, 15000); //the mutation occurs at position 15000
}
```

d. A two species model with constant population size plus recent strong positive selection on a single position in the target population. **High recombination** between positions. (SS-Rh)

```
// uniform recombination along the chromosome
initializeRecombinationRate(1e-4);

//if required, force a strong selective sweep
9900 {
     target = sample(p2.genomes, 100); //the mutation is in 100 individuals
     target.addNewDrawnMutation(m3, 15000); //the mutation occurs at position 15000
}
```

e. A two species model with constant population size plus **beneficial selection (low proportion in relation to neutral)** on functional positions in the target population. High recombination between positions. (PSL)

```
// g2 genomic element type: (nonsynonymous) uses all mutations
initializeGenomicElementType("g2", c(m1,m2,m3), c(2,8,0.1));
```

f. A two species model with constant population size plus **beneficial selection (middle proportion in relation to neutral)** on functional positions in the target population. High recombination between positions. (PSM)

```
// g2 genomic element type: (nonsynonymous) uses all mutations
initializeGenomicElementType("g2", c(m1,m2,m3), c(2,8,0.3));
```

A two species model with constant population size plus **beneficial selection (high proportion in relation to neutral)** on functional positions in the target population. High recombination between positions. (PSH)

```
// g2 genomic element type: (nonsynonymous) uses all mutations
initializeGenomicElementType("g2", c(m1,m2,m3), c(2,8,1));
```

g. A two species model, **demographic expansion (5x)** in the target population and no selection on functional positions in the target population. High recombination between positions. (SNM-EXP)

```
// set the overall uniform mutation rate
initializeMutationRate(5e-6);

// If required, sudden change in population size
9500 { p2.setSubpopulationSize(5000); }
```

A two species model, **demographic reduction (0.2x)** in the target population and **no selection** on functional position in the target population s. High recombination between positions. (SNM-RED)

```
// If required, sudden change in population size
9500 { p2.setSubpopulationSize(200); }
```

h. A two species model with constant population size and **deleterious mutations** on functional positions in the target population. **Mutations highly deleterious but codominant**. High recombination between positions. (DELF)

```
// m2 mutation type: (deleterious)
  initializeMutationType("m2", 0.5, "f", -0.005);
```

i. A two species model with constant population size and **deleterious mutations** on functional positions in the target population. **Gamma distribution and codominant**. High recombination between positions. (DELG)

```
// m2 mutation type: (deleterious)
initializeMutationType("m2", 0.5, "g", -0.005, 1); // mean -0.01, shape 0.01 rate 1.0
```

j. A two species model, **demographic expansion** in the target population and **deleterious mutations** on functional positions in the target population. High recombination between positions. (DELF-EXP)

```
// set the overall uniform mutation rate
initializeMutationRate(5e-6);

// m2 mutation type: (deleterious)
initializeMutationType("m2", 0.5, "f", -0.005);

// If required, sudden change in population size
9500 { p2.setSubpopulationSize(5000); }
```

k. A two species model, demographic expansion in the target population, deleterious **and beneficial selection (low proportion in relation to neutral)** on functional positions in the target population. High recombination between positions. (PSL-DELF)

```
// m2 mutation type: (deleterious)
initializeMutationType("m2", 0.5, "f", -0.005);

// g2 genomic element type: (nonsynonymous) uses all mutations
initializeGenomicElementType("g2", c(m1,m2,m3), c(2,8,0.));
```

l. A two species model, demographic expansion in the target population, deleterious **and beneficial selection (middle proportion in relation to neutral)** on functional positions in the target population. High recombination between positions. (PSM-DELF)

```
// m2 mutation type: (deleterious)
initializeMutationType("m2", 0.5, "f", -0.005);

// g2 genomic element type: (nonsynonymous) uses all mutations
initializeGenomicElementType("g2", c(m1,m2,m3), c(1,9,0.3));
```

m. A two species model, demographic expansion in the target population, deleterious **and beneficial selection (high proportion in relation to neutral)** on functional positions in the target population. High recombination between positions. (PSH-DELF)

```
// m2 mutation type: (deleterious)
initializeMutationType("m2", 0.5, "f", -0.005);

// g2 genomic element type: (nonsynonymous) uses all mutations
initializeGenomicElementType("g2", c(m1,m2,m3), c(2,8,1));
```

4. (each group separately). Create a variable with the name of the script slim file:

   For example:

```
n="slim_template_SNMR0.slim"
```

5. (each group separately). Run Slim:

```
slim -t -m $n > $n.out
```

NOTE: All the next steps can be performed automatically by just including the filename of your simulation scenario, that is: "sh ./run_alpha_steps_after_slim_smulation.sh $n.out", although it is convenientthat you first read and understand all the next steps.

6. (each group separately). Once we run the scripts in *Slim* and the output is obtained, we need to obtain the summary statistics that explain the patterns of variability:
   a. First cut the header of the output and keep only the *ms* simulation data:

```
sed '1,/Starting run/d'< $n.out > $n.out.ms
```

   b. We will use *mstatspop* program to calculate the variability across the genome (theta Watterson) and the Site Frequency Spectrum (SFS) for neutral (synonymous) and functional positions (nonsynonymous).
      i. You need to introduce **a file including the name of the chromosome** and **a *mask file* to separate neutral from functional positions**. Therefore, the program must

be run twice: for neutral and for functional positions. The command line to run *mstatspop* (https://github.com/CRAGENOMICA/mstatspop) using the *ms* format as input is the following:

```
1. mstatspop -f ms -i $n.out.ms -o 0 -N 2 25 5 -G 1 -l 30000 -r 1
   -n name_scaffold.txt -m mask_neutral.txt >
   $n.out.ms.neutral_statistics.txt

2. mstatspop -f ms -i file_name_ms -o 0 -N 2 25 5 -G 1 -l 30000 -r
   1 -n name_scaffold.txt -m mask_functional.txt >
   $n.out.ms.functional_statistics.txt
```

ii. The (variability levels and Site Frequency Spectrum is available inside the output file:

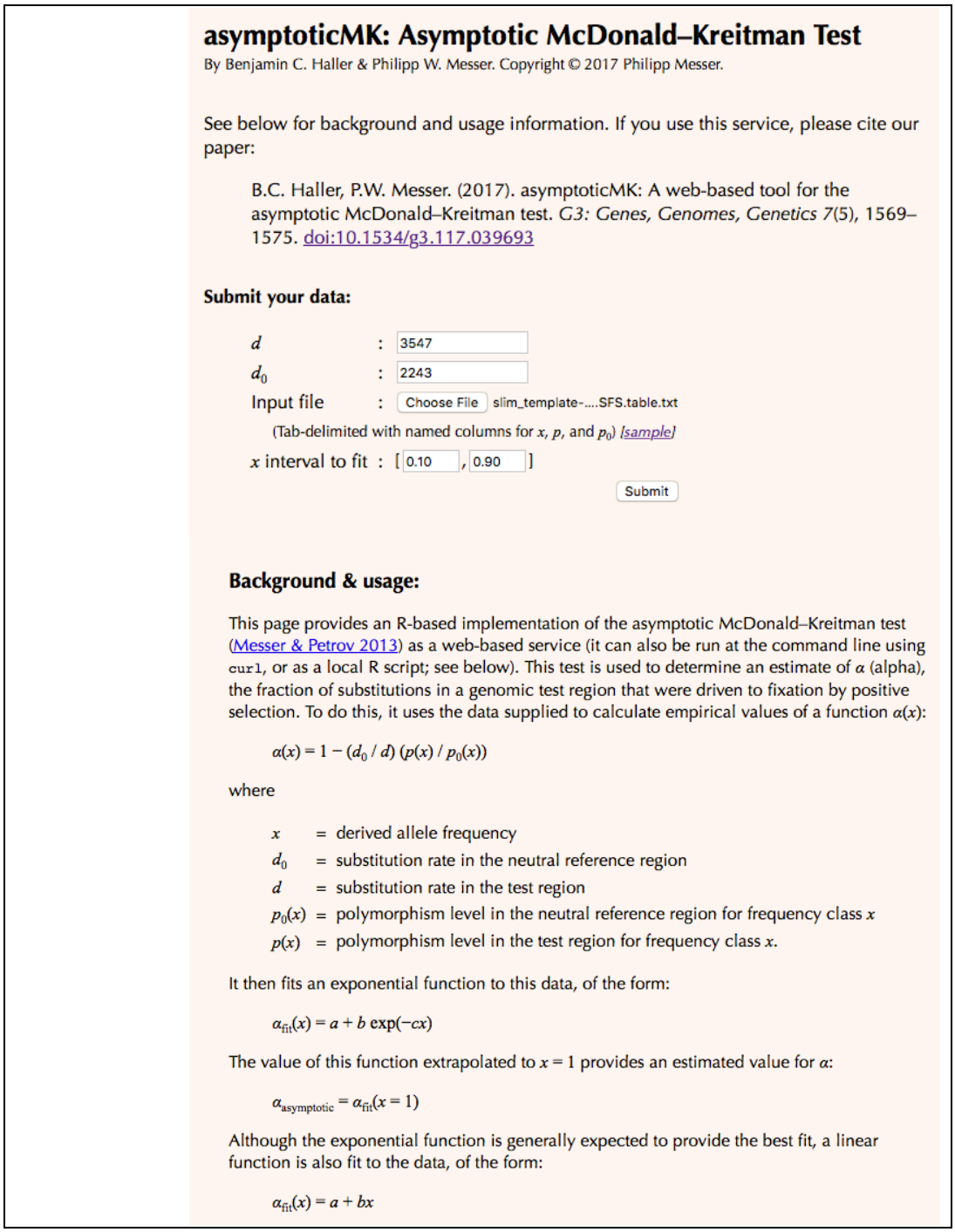
1. Example:

(…)
Estimates of variability for each population (an and bn for the variant positions):
S[0]: 23 **Theta(Wat)[0]: 4.442403** Theta(Taj)[0]: 3.378182
Theta(Fu&Li)[0]: 3.000000 Theta(Fay&Wu)[0]: 1.389495
Theta(Zeng)[0]: 2.383838 Theta(Achaz,Wat)[0]: 4.787693
Theta(Achaz,Taj)[0]: 3.385900 **Divergence[0]: 91.660000**
Estimates of NUCLEOTIDE variability for each population (if missing, corrected by the averaged effective positions):
S[0]: 23 **Theta/nt(Wat)[0]: 0.000444** Theta/nt(Taj)[0]: 0.000338
Theta/nt(Fu&Li)[0]: 0.000300 Theta/nt(Fay&Wu)[0]: 0.000139
Theta/nt(Zeng)[0]: 0.000238 Theta/nt(Achaz,Wat)[0]: 0.000479
Theta/nt(Achaz,Taj)[0]: 0.000339 **Divergence[0]: 0.009166**
(…)
Frequency of variants for each population:
fr[0,1]: 3 fr[0,2]: 5 fr[0,3]: 2 fr[0,4]: 2 fr[0,5]: 2 fr[0,6]: 2 fr[0,7]: 0 fr[0,8]: 0
fr[0,9]: 1 fr[0,10]: 1 fr[0,11]: 0 fr[0,12]: 0 fr[0,13]: 0 fr[0,14]: 0 fr[0,15]: 0
fr[0,16]: 1 fr[0,17]: 0 fr[0,18]: 0 fr[0,19]: 0 fr[0,20]: 0 fr[0,21]: 0 fr[0,22]: 0
fr[0,23]: 0 fr[0,24]: 0
(…)

iii. Then extract the levels of variability (Theta), the divergence and the site frequency spectrum (SFS) for **neutral** and for **functional** positions. Variability and divergence can be obtained by hand. For extracting SFS and make a table with functional and neutral, you can do the following. **A necessary file with the frequencies** (*freq_col.txt*) is already generated:

```
echo "freq\tsfs_funct\tsfs_neutral" > ${n}.SFS.table.txt #include a
     header.
grep 'fr\[0,' ${n}.out.ms.functional_statistics.txt | tr '\t' '\n'|cut
     -d ' ' -f2> ${n}.out.ms.functional_statistics.txt.SFS.txt
     #create a column with functional SFS.
grep 'fr\[0,' ${n}.out.ms.neutral_statistics.txt | tr '\t' '\n' | cut
     -d ' ' -f2 > ${n}.out.ms.neutral_statistics.txt.SFS.txt
     #create a column with neutral SFS.
paste freq_col.txt ${n}.out.ms.functional_statistics.txt.SFS.txt
     ${n}.out.ms.neutral_statistics.txt.SFS.txt > ${n}.cols.txt
     #join the frequencies, the SFS for functional and for neutral
     columns.
perl -ane 'print if $F[2]' ${n}.cols.txt > ${n}.cols_.txt #eliminate
     rows where syn is zero
cat ${n}.cols_.txt >> ${n}.SFS.table.txt #join header with SFS columns.
```

c. We will use the **asymptotic MK approach** to estimate the proportion of adaptive substitutions. (http://benhaller.com/messerlab/asymptoticMK.html, from Haller and Meeser G3 2017).



asymptoticMK: Asymptotic McDonald–Kreitman Test

By Benjamin C. Haller & Philipp W. Messer. Copyright © 2017 Philipp Messer.

See below for background and usage information. If you use this service, please cite our paper:

B.C. Haller, P.W. Messer. (2017). asymptoticMK: A web-based tool for the asymptotic McDonald–Kreitman test. *G3: Genes, Genomes, Genetics* 7(5), 1569–1575. doi:10.1534/g3.117.039693

Submit your data:

d : 3547

d_0 : 2243

Input file : Choose File slim_template-....SFS.table.txt

(Tab-delimited with named columns for x, p, and p_0) *[sample]*

x interval to fit : [0.10 , 0.90]

Submit

Background & usage:

This page provides an R-based implementation of the asymptotic McDonald–Kreitman test (Messer & Petrov 2013) as a web-based service (it can also be run at the command line using `curl`, or as a local R script; see below). This test is used to determine an estimate of α (alpha), the fraction of substitutions in a genomic test region that were driven to fixation by positive selection. To do this, it uses the data supplied to calculate empirical values of a function $\alpha(x)$:

$$\alpha(x) = 1 - (d_0 / d)\,(p(x) / p_0(x))$$

where

- x = derived allele frequency
- d_0 = substitution rate in the neutral reference region
- d = substitution rate in the test region
- $p_0(x)$ = polymorphism level in the neutral reference region for frequency class x
- $p(x)$ = polymorphism level in the test region for frequency class x.

It then fits an exponential function to this data, of the form:

$$\alpha_{fit}(x) = a + b\exp(-cx)$$

The value of this function extrapolated to $x = 1$ provides an estimated value for α:

$$\alpha_{asymptotic} = \alpha_{fit}(x = 1)$$

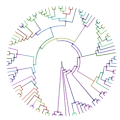
Although the exponential function is generally expected to provide the best fit, a linear function is also fit to the data, of the form:

$$\alpha_{fit}(x) = a + bx$$

(From asymptotic-MK web application)

i. It is necessary to introduce the functional and neutral divergence (from mstatspop file) and the file with a table containing the three columns (the file previously constructed): the frequency of the derived allele, the number of functional variants for each frequency and the number of neutral variants for each frequency.
ii. This algorithm estimates the value of α for each frequency separately and make a plot. Finally, estimate asymptotically the value of α from the whole data, to eliminate the effect of deleterious mutations.
iii. If you still have time, repeat simulations (more replicates) and keep all values!

7. Finally each group, evaluate their results:



a. Observe the level of variability in relation to the value included in the simulation (Theta=4Neμ).
b. The comparison of the Site frequency Spectrum between neutral and functional positions. (make 2 plots: neutral and functional).
c. The estimation of the proportion of adaptive substitutions in relation to the value included in the simulation (using asymptotic MKT approach). Take the raw value, the asymptotic value and the plot.

8. **Comparison of the results from all studied conditions**. Make a table with the results given all the simulated conditions, Interpretation.

9. **alpha using estimates of variability**: It is also possible to estimate alpha using estimates of variability (per nucleotide). This is convenient in cases where the data is not large, or when there is a presence of abundant missing data. These estimates provide values of alpha considering low, medium and high frequency variants and thus they are a proxy when the SFS is not really available. Take a glance at the R script *run_plots_Theta_alpha.R* to see how is calculated.