`simDoSe`: simulate Dominance and Selection

Daniel J. Balick[1,2]

[1]*Department of Biomedical Informatics, Harvard Medical School, Boston, MA, 02115, USA*

[2]*Division of Genetics, Brigham and Women's Hospital, Harvard Medical School, Boston, MA, 02115, USA*

Last updated: November 10, 2021

Please address questions and bug reports to:

`dbalick@hms.harvard.edu` or `dbalick@gmail.com`

Brief overview of `simDoSe`

`simDoSe` is a fast Python Wright-Fisher simulator of independent sites (i.e, in the infinite recombination limit) evolving under diploid selection with or without recurrent mutations through a realistic non-equilibrium human demography. Each simulation outputs both raw (i.e., true population) and sample (i.e., sequenced cohort) site frequency spectra, as well as a collection of relevant summary statistics for each. Arbitrary selection and dominance coefficients can be specified, along with mutation rate, whether or not recurrent mutations are allowed, target size, choice of demography (and ancestry when the demography holds multiple). Users can also specify a number of simulated genes, either with uniform target size or pulled from an imported target size distribution, to be sampled from the larger target to model a distribution of per-gene site frequency spectra. Additionally, multiple sets of genes of differing sizes can be produced simultaneously.

Table of Contents

`simDoSe` User manual

This document is intended to be a brief user manual for `simDoSe`, designed to be a fast and flexible simulator to produce the site frequency spectrum (SFS) resulting from the independent evolution of alleles with various diploid selection coefficients through models of realistic human demographic history. `simDoSe` is specifically designed to properly handle high mutation rates in the recurrent mutation limit to more accurately simulate large modern population samples where this is expected to be relevant. Additionally, the simulator allows users to generate smaller simulated "genes" from a single larger simulation, and can be used to treat genes as biallelic loci (e.g., to model LOF mutations with similar consequence within a single gene). The general `simDoSe` output can be used, for example, to create expected SFSs for comparison to empirical data with a likelihood-based inference of the distribution of selective effects under the Poisson Random Field model [1]. Several literature-derived demographies are encoded (see References [2–4]), as well as simple toy models for linear and exponential population growth. Many output options are available, several of which specify ways to produce multiple SFSs (i.e., simulated 'genes') from a single simulation. In this manual, the population genetic kernels will be explained, command line inputs and dependencies will be detailed, and instructions for running will be explained along with some brief examples for various types of simulations and output.

1 Population genetic details

If you're interested in using a population genetic simulator, you should know both how it works and what assumptions are made. This simulator is divided into several relevant 'kernels", which are Python-scripted functions that control various population genetic forces. The relevant population genetic forces at play here are selection, drift, and mutation. Note that recombination is omitted; this simulator assumes that all sites evolve independently, which implies they are simulated in the infinite recombination limit of sexual evolution. This dramatically increases the speed of simulations and attainable population sizes (e.g., 2N=many millions), but limits the applicability of the output. If you are interested in simulating the often complex effects of linkage, read no further and use one of the many elegant simulators written for this purpose. If you either want to simulate in the infinite recombination limit or want to simulate multiple loci (e.g., short exons) that are each in perfect linkage, keep reading. For those still here, we will start with the mutation kernel.

1.1 Mutation kernel

First, mutation rate μ should be specified on the per-site basis, but a single site can represent a locus with many bases in perfect linkage. In other words, you can choose mutation rates appropriate for single bases at $\mu = 10^{-8}$ per site (this is the default), or simulate larger biallelic loci at a higher mutation rate of $\mu = 10^{-6}$ per site (corresponding to roughly 100 bases). There is no explicit assumption of infinite target size in `simDoSe`. Next, the target size L, or number of sites, should be specified. Mutations are generated pseudo-randomly with `numpy.random.poisson()` at the rate μL per generation. If $\mu L > L$ for any reason (e.g., μ is set very high or L is very small), only L mutations will be generated. New mutations are added to the population at a rate $2N\mu L$, where N is the number of diploid individuals in the previous generation (corresponding to $2N$ haploid individuals). These each enter the population at frequency $1/2N$.

Recurrent mutations can be enabled via the command line if you are simulating very larger mutation rates or are modeling a very large sample of the population (such that $2N\mu L$ is non-negligible compared to one). This switches to a different mutation kernel. Several recurrent mutation kernels are enabled, but it is highly recommended to use the most recent called `v3`. If enabled, the recurrent kernel pseudo-randomly adds mutations with `numpy.random.poisson` to each of L sites, independent of whether they are monomorphic or polymorphic. Unlike the non-recurrent kernel, polymorphic sites each gain `poisson(`$2N\mu$`)` individuals such that the frequency of each is increased by μ on average.

Note that this simulator assumes mutation rates are uniform across all sites. Since there is no linkage, a distribution of mutation rates can be modeled by taking a linear combination of outputted SFSs with different mutation rates to form a distribution of interest.

1.2 Selection kernel

Since we are modeling diploid selection coefficients, we must first define the dominance coefficient h and selection coefficients s in terms of the selection acting on homozygotes s_{hom} and heterozygotes s_{het} genotypes.

$$\begin{aligned} s_{hom} &= s \\ s_{het} &= hs \end{aligned} \tag{1}$$

Henceforth, we will refer only to h and s. The expected frequency of an allele at time $t+1$ due to selection is given by the following dependence on the frequency at time t.

$$E[p_{t+1}(h,s)] = \frac{p_t^2(1+s) + 2p_t(1-p_t)(1+hs)}{p_t^2(1+s) + 2p_t(1-p_t)(1+hs) + (1-p_t)^2} \tag{2}$$

In words, selection acting on a derived allele with frequency p acts to change the frequency of heterozygotes by $2p(1-p)$ by $(1+hs)$ and change the frequency of homozygotes p^2 by $(1+s)$. This is measured relative to the selection on all three haplotypes, homozygous derived, heterozygous, and homozygous wild type, as seen in the denominator. If selection is negative (purifying selection), the frequency will be reduced, while for positive (beneficial) selection it will be increased. This is the *deterministic* change in frequency due to selection, but will be promoted to a stochastic change due to the action of drift, as detailed below. The fact that we track changes in frequency (i.e., in aggregate), rather than the birth or death rate of each individual in the population, is what makes this a Wright-Fisher simulation and is responsible for a dramatic increase in speed.

Note that selection and dominance coefficients are uniform across all sites. Since there is no linkage, a distribution of selection effects can be created by taking a linear combination of SFS output for various values of s and h.

1.3 Drift and demography

Genetic drift, the introduction of stochastic changes in the frequency due to finite population size, has a simple implementation in the simulator. The expected change in frequency due to selection $E[p(h,s)]$ is detailed above. The realized change in frequency for each simulated derived allele is binomially sampled (using `numpy.random.binomial`) around this expectation.

$$p_{drift} = \text{Binom}[E[p(h,s)], 2N] \tag{3}$$

Note that, for diploids, drift depends on the number of haplotypes $2N$, rather than the number of diploid individuals N. Again, the fact that changes are based on the frequency of an allele ,rather than a stochastic birth-death process, is what makes this a fast Wright-Fisher simulation.

Demography, as implemented in `simDoSe`, is simply the change in population size as a function of time $N(t)$. The drift applied in generation $t+1$ depends on the new population size N_{t+1} for an allele with frequency p_t at time t.

2 Command line options

`simDoSe` has many command line options used to specify simulation parameters and the style of output. The simulation parameters are specified by the following commands. Aliases for commands are written immediately after separated with a comma. Commands with input parameters are followed by `<type>` specifying the type of the input value. Commands that take no additional input are specified as flags (with `[FLAG]`).

`-U, --mutation-rate <float>`

Specify mutation rate per site. Default is set to $\mu = 10^{-8}$ corresponding roughly to the genomic average of per-site mutation rates in the human genome.

`-L, --length <int>`

Specify the number of independent sites to simulate (i.e. target size). Overall mutation rate is given by $U = \mu L$, where μ is the mutation rate per site and L is the number of sites.

`-R, --recurrent [FLAG]`

Enable recurrent mutations in the mutation kernel. Multiple mutations can occur at the same site if mutation rate is high. Use this flag if $2N\mu$ in the final generation is larger than roughly 0.01, either due to large μ or large $2N$. Simulations performed with the recurrent mutation kernel will be placed in a separate folder labeled with the suffix `_recurrent`. By default recurrent mutations are deactivated.

`-S, --selection <float>`

Specify selection coefficient s (equivalent to selection on homozygotes) for all sites. Default is set to -0.1. Make sure you specify the sign of selection, where negative values are interpreted as purifying selection and positive values correspond to positive selection.

`-H, --dominance <float>`

Specify dominance coefficient h (where heterozygote selection corresponds to hs) for all sites. Additive selection corresponds to $h = 0.5$, recessive selection corresponds to $h = 0$, and dominant selection corresponds to $h = 1.0$. Underdominant selection can be specified with $s < 0$ and $h > 1.0$. Overdominant selection (i.e., heterozygote advantage) can be specified with $s < 0$ and $h < 0$ such that heterozygote selection hs is positive. Default is set to $h = 0.5$ (additive selection).

`--underdom, --underdominant [FLAG]`

Flag. Manually sets homozygote selection to zero. For fully underdominant selection, both homozygotes are under the same strength of selection (neutral). This can be approximated by taking $s = 10^{-n}$ and $h = -c10^{n}$ such that heterozygotes experience selection $-c$ and both homozygotes are essentially neutral. Since this is annoying to keep track of, you can use the underdominant flag and any h and s. Homozygotes will experience selection hs and homozygotes will be neutral. This flag can also be used for overdominant positive selection corresponding to a positively selected heterozygote and neutral homozygotes.

`--burnin, -T <int>` Specify a multiple of $2N$ to define the length of the burn-in before starting demography. The burn-in period evolves the population at constant size to fully equilibrate prior to initiating the demography. All population genetic simulators require a burn-in period to remove transient effects of the initial conditions. This generally takes $t = 8N$ generations to fully equilibrate. Default is set to 5.0 corresponding to 10N generations. If you are interested in the results of the specified demography, it is highly recommended that you specify `-T 4.0` or more generations (just leave it at 5.0 to be safe). Any value below 4.0 may result in transient effects confounding your results.

`-N, --initpopsize <int>`

Specify initial haploid population size $2N_0$ (double the number of diploid individuals N_0). Burn-in occurs at this population size and this is used as the initial population size for the specified demography. This should not be used for literature-derived demographies (e.g., `tennessen, supertennessen`), since they include the initial size of the population in the demography itself, which means that using alternative sizes will result in a downsampling on the first generation. Default is set to $2N_0 = 28948$, which corresponds to the ancestral human population size inferred to be 14474 individuals prior to the Out of Africa bottleneck that resulted in a population split. If `initpopsize` is set to any other value, you will get a cheeky warning that you are deviating from the human ancestral size (to remind you not to change this for literature-derived demographies), but the simulation will still run.

`-D, --demography <string>`

Choose a demography from a list of pre-scripted choices. Each demography uses a distinct kernel to dictate the relationship between the current population size and the population size in the following generation. If you would like to use a demography that is not included in the following list, you can either script it yourself or email `dbalick@hms.harvard.org` or `dbalick@gmail.com` to request help incorporating it. The default demography is `equilibrium` (see `-D equilibrium` below). Currently available demographies:

- `-D equilibrium`

  Equilibrium demography. After burn-in, population size remains constant at the initial population size. This sets the strength of drift by binomially sampling $2N$ individuals for the following generation.

- `-D linear`

  Linear population growth. After burn-in, population increases linearly with time as $2N_0 + gt$, where $2N_0$ is the initial population size, g is the growth rate, and t is the generation after the demography begins (post-burn-in). You should specify `--growthrate g` for this demography (see `--growthrate` below).

- `-D exponential`

  Exponential population growth. The population grows exponentially with time as $2N_0e^{gt}$, where N_0, where $2N_0$ is the initial population size, g is the growth rate, and t is the generation after the demography begins (post-burn-in). You should specify `growthrate g` for this demography (see `--growthrate` below).

- `-D tennessen`

  "Tennessen" demography. This demography was inferred by Tennessen et al. in [2]. This demography has two subpopulations corresponding to inferred African and European demographic history between the modern day and prior to the Out of Africa event. Specification of these ancestries should be included in the command line (see `--ancestry` below). Schematically, the African population size stays relatively constant and then grows exponentially. The European population size starts at the same size as the African population size, experiences one bottleneck followed by a second bottleneck, and then grow exponentially (much weaker than the African exponential growth), followed by a second, more extreme epoch of exponential growth (more rapidly but at a later generation than the African exponential population growth). Do not specify `--initipopsize` for this demography.

- `-D supertennessen`

  Tennessen et al. demography with faster exponential growth affectionately referred to as the "Super-Tennessen" demography. This European demography was originally inferred in Weghorn, Balick, et al [3] using a tenfold larger sample size than in Tennessen et al. [2] to better estimate the rate of growth in the most recent exponential epoch. The demography is identical to `tennessen` until the final exponential epoch, for which users can specify the growth rate (see `--growthrate`). To emulate the value inferred in Weghorn, Balick, et al., use the value `--growthrate 0.03`. Using the options `-D supertennessen --growthrate 0.0195` will reproduce `-D tennessen`. Specifying a larger value will result in more extreme recent growth, and a lower value weaker recent growth. If `--growthrate` is not set, the default is set to 0.0195 (`tennessen` growth rate), so be sure to specify a different value. Do not specify `--initipopsize` for this demography.

- `-D gaussian`

  Tennessen et al demography with faster than exponential growth in the most recent epoch. This demography allows for non-exponential growth for the most recent epoch of European demographic history. The population size during the most recent epoch grows as $2Ne^{g(t-t_0)^2}$ (like a Gaussian with the sign reversed), where $2N$ is the population size at the end of the previous epoch, g is the user-specified growth rate, and $t - t_0$ is the number of generations since the beginning of this final epoch.

This demography can be used as an alternative to exponential growth to fit to empirical observations (e.g., the number of singletons) from a European population sample. The growth rate should be specified using `--growthrate g`, where the default is again set to 0.0195 for reasons related to other demographies. Do not specify `--initipopsize` for this demography.

- `-D browning`

  Browning and Browning demography. This is another literature derived European demography, but from a different source of data. This demography was inferred from identical-by-descent (IBD) data by Browning and Browning [4]. There are substantial differences from the aforementioned demographic inferences, but key the features of a bottleneck followed by rapid exponential growth remain consistent. Like the Tennessen et al. demography, this model is fully specified and takes no input parameters. Do not specify `--initipopsize` for this demography.

`-G, --growthrate <int>`

As menntioned above, this parameter is used in a variety of demography kernels to specify the rate of (linear, exponential, non-exponential) growth. Default is set to 0.0195 (for which `supertennessen` is the same as `tennessen`). This parameter should always be specified, except in the cases of `equilibrium`, `tennessen`, and `browning` demographies.

`-A, --ancestry <string>`

This option is used to distinguish between the African and European demographies in the Tennessen et al model (`-D tennessen`). For African ancestry, use the command `--ancestry african`. For European ancestry, use `--ancestry european`. Default is set to `european` due to dependencies for some non-Tennessen demographies. Do not use this option with any demography other than `tennnessen`.

`--samplesize <int>`

Sample size specification. At the end of the simulation, `simDoSe` prints files for the SFS and then does a single generation binomial downsample to simulate the act of randomly choosing individuals from the full population to sequence and prints files for the sample SFS. For example, a full population of 20000 might have 2000 singletons (i.e., private mutations) at frequency 5×10^{-5} , but a sample of 10 individuals from this population could have only 2 singletons at frequency 0.1. For comparison to data (e.g., 1000 sequenced individuals), specify a number corresponding to twice the number of individuals in the sample (e.g., 2000), which corresponds to the number of haploid individuals in the sample. Default is set to 100. Do not set sample to zero, just choose not to print sample files using `--printsampleSFS 0`. If this option is not specified sample files will not be generated.

The following options are related to file name specification.

`-f, --fileprefix <string>`

Specifies the prefix you wish to use for all output files generated. For example, if you are simulating underdominance, you may wish to use the option `-f underdom` to produce files named e.g., `underdom_SFS.out`. Default is `stdout`, so if this option is not used, all files will be named e.g., `stdout_SFS.out`.

`--run <str>`

Specify sub-label for replicate runs. This option is useful for running many replicates with otherwise identical parameters to avoid overwriting existing files while retaining data for multiple runs. It is easiest to use this option as a numerical label from e.g., a bash loop over integers. Default is `XX`, which does not produce this sub-label.

`--geneSFS_startnumber <int>`

Start number for simulated gene labels. Similar to `run` labels, if genes are simulated using `--Ngenes` and `--Lgenes`, the corresponding files produced for each gene are labeled with `_gene_int`, where `int` is a number from zero to $(N_{genes} - 1)$. This option allows you to choose what number to start these gene labels with. This is particularly useful if printing simulated genes from multiple runs to the same folder. Default is set to 0.

`--linearU [FLAG]`

Filename specification for mutation rate. Note this is mutation rate per site μ, not total mutation rate $U = \mu L$. If using more than one mutation rate per half decade for multiple runs in the same folder (e.g., $\mu = 0.1, 0.2, 0.3, 0.01, 0.2, 0.3$), use this option to write filenames with linear μ rather than $\log 10\mu$. Default is `False`, which uses a rounded version of $\log 10\mu$ to avoid very large filenames for non-decade numbers. If this flag is not used and mutation rates with $\log 10\mu$ are the same within the first decimal place, the file will be overwritten.

`--linearL [FLAG]`

Filename specification for number of sites. If using more than one length L per half decade for multiple runs in the same folder (e.g., $L = 10, 13, 20, 30, 100$), use this option to write filenames with linear L rather than $\log 10L$. Default is `False`, which uses a rounded version of $\log 10L$ to avoid very large filenames for non-decade numbers. If this flag is not used and lengths with $\log 10L$ are the same within the first decimal place, the file will be overwritten.

`--linearS [FLAG]`

Filename specification for selection. If using more than one selection coefficient s per half decade for multiple runs in the same folder (e.g., $s = -0.3, -0.2, -0.1, 0.03, 0.02, -0.1$), use this option to write filenames with linear S rather than $\log 10s$. Default is `False`, which uses a rounded version of $\log 10s$ to avoid very large filenames for non-decade numbers. If this flag is not used and lengths with $\log 10s$ are the same within the first decimal place, the file will be overwritten.

The following options specify which files to print. Note that many of these options are intentionally binary, with `0=False` or `1=True`, to ensure that files are or are not printed after potentially long simulations.

`--printrawSFS <binary 0,1>`

Print the "raw" SFS for the whole population (prior to downsampling). This is a binary option, with `0=False` or `1=True`. Default is set to 1; the raw SFS is printed to file, along with various summary statistics. If you are only interested in the SFS after downsampling to the specified sample size, use the option `--printrawSFS 0`. To ensure you are printing the raw SFS regardless of defaults, use the option `--printrawSFS 1` (especially important for very long runs where you may not immediately know if this file was printed).

`--printsampleSFS <binary 0,1>`

Print the sample SFS for the whole population (after downsampling). This is a binary option, with `0=False` or `1=True`. Default is set to 1; the sample SFS is printed to file, along with various summary statistics. If you do not want to down-sample or are only interested in the SFS before downsampling to the specified sample size, use the option `--printsampleSFS 0`. To ensure you are printing the sample SFS regardless of defaults, use the option `--printsampleSFS 1` (especially important for very long runs where you may not immediately know if this file was printed).

`--printgeneSFS <binary 0,1>`

Print SFS file for each simulated gene. This is a binary option, with `0=False` or `1=True`. If you are creating short genes from a larger simulation (using the options `--Ngenes` and `--Lgenes` detailed below), this option specifies if you want to print SFS files *for each gene*. Be aware that this can produce a large number of files, potentially take up a (relatively) large amount of space, and may take a while for all of the i/o depending on how many genes you have created. Use the option `--printgenes 1` to create these per-gene SFS files, or use the option `--printgenes 0` to ensure these files are not printed. Default is set to 0; if genes are simulated, the SFS will not be printed (e.g., only the summary statistics files will be printed).

`--printgenestats <binary 0,1>`

Print summary statistics per simulated gene to a single file. This is a binary option, with `0=False` or `1=True`. If you are creating short genes from a larger simulation (using the options `--Ngenes` and `--Lgenes` detailed below), this option specifies if you want to create a file with several summary statistics per gene (in addition to summary stats for the collection of genes). Use the option `--printpergenestats 1` to create this per-gene summary statistics file, or use the option `--printpergenestats 0` to ensure this file is not printed. Default is set to 1; if genes are simulated, the per-gene summary statistics file will be printed.

`--printbiallelicSFS <binary 0,1>`

Create biallelic SFS files for simulated genes. This is a binary option, with `0=False` or `1=True`. If you are simulating genes that are considered biallelic (i.e., all mutations have the same consequence), this option will pool over sites to create a single frequency and then create an SFS by pooling these frequencies over all simulated genes. This may be useful, for example, when modeling loss of function mutations. Since `simDoSe` does not account for compound heterozygotes, this option should only be used if selection is additive (i.e., $h = 0.5$). Use the option `--printbiallelicSFS 1` to create a pooled SFS over biallelic genes, or use the option `--printbiallelicSFS 0` to ensure this is not done. If this option is activated, the per-gene summary file will be labeled `biallelic` and the summary statistics file will contain additional moments of the biallelic SFS. Default is set to 0; do not make a biallelic assumption or produce an SFS pooled over genes.

`--notsparse [FLAG]`

Print all bins instead of sparse SFS. Since site frequency spectra can be rather large vectors, by default `simDoSe` stores all SFS files as sparse vectors, where bins with zero counts, which are the majority in large samples, are not printed to file. This option instead prints all bins from 1 to $2n - 1$ haploid individuals, where $2n$ is the sample size. Using a sparse SFS saves a large amount of disk space when storing simulated SFS output. If you are working with smaller samples, this is not necessary and can add additional work in downstream analysis, but, if you are running many simulations, this may be a good idea regardless of the sample size. Default is set to `False`; sparse SFS will be printed.

`--fracrare <float>`
Choose the frequency cutoff for fraction of alleles that are rare. The summary statistics file includes a statistic called 'frac rare", which computes the percentage of segregating sites at or below a given frequency (see Output file details for more information). By convention, this is usually 0.01 (1%). For some applications this arbitrary cutoff may be less informative than other cutoffs. This function allows you to specify the frequency cutoff for frac rare. Default is set to 0.01.

The following options are related to the creation of smaller "genes" from a larger number of simulated sites.

`--Ngenes <int>`
Specify the number of genes to create. Since simulated sites are independent (i.e., are perfectly separates by recombination), many smaller target size "genes" can be created from a single simulation by breaking up the full target L into n parts. This option specifies the number of genes n to create, provided $L/n \geq 1$. You must specify the length of each gene using `--Lgenes` or a length distribution to sample using `--lengthlist` to use this option. If the length or length distribution is specified, `simDoSe` will produce summary stats (if `--printgenesummary 1` is given) and/or per-gene SFS files (if `--printgeneSFS` is given) and/or a biallelic SFS (if `--printbiallelicSFS 1` is given). Default is at 0; simulated genes are not produced.

`--Lgenes <int>`
Specify the length of genes to create. Since simulated sites are independent (i.e., are perfectly separates by recombination), many smaller target size "genes" can be created from a single simulation by breaking up the full target L into n parts. This option specifies the length of genes n to create, provided $L/n \geq 1$. You must specify the number genes using `--Ngenes` to use this option. If both are specified, `simDoSe` will produce summary stats (if `--printgenesummary 1` is given) and/or per-gene SFS files (if `--printgeneSFS` is given) and/or a biallelic SFS (if `--printbiallelicSFS 1` is given). Default is at 0; simulated genes are

not produced.

`--lengthlist <string>`

Import length distribution for simulated genes. If you are interested in simulating genes matching the length distribution of a gene set, this option will import a file from a specified path (with `--lengthlist path/filename`) and create a list of genes randomly sampled from the lengths in this list. Note that this list must be in multiples of the specified mutation rate μ, so some pre-computation is needed to use this option. The length list file must have a column named "length" (all lowercase) in order to be read in properly. Default is set to `XX`; no length list will be used.

`--russiandoll [FLAG]`

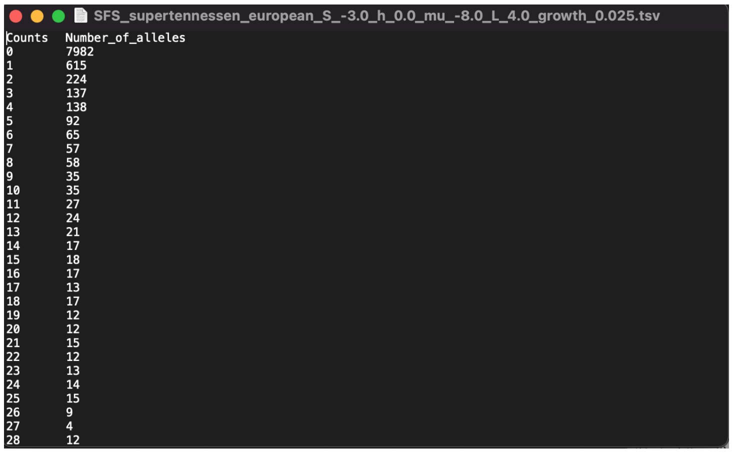
Create a series of gene sets with descending length. This allows you to create simulated genes with many lengths at the same time. This option takes the full simulation of L sites and cuts it into n genes of length $L/10^k$ for $k = 1, 2, 3, ...$ until $L/10^k < 10$. If L is divisible by 10, this produces sets of genes with length $l = 10, l = 100, ..., l = L/10$ simultaneously (like a Russian doll with ever smaller genes inside each gene set). The length of each decade is inherited from the total number of sites L, so to produce e.g., ten genes each of length 27 and 270, simulate a total of 2700 sites and use the option `--Ngenes 10`. The `--Ngenes` option must be used to use the `--russiandoll` option. Default is set to `False`; Russian doll gene sets will not be created.

3 Output file details

`simDoSe` creates output folders and a number of files depending on user specification. Three kinds of files can be created: SFS files, summary statistics files, and per-gene summary files. By default SFS and summary statistics files are created for all simulations for both the raw and sample SFS. If simulating genes from a longer simulation using `--Ngenes` and `--Lgenes`, an individual SFS will be created for each gene (if you used `--printgeneSFS 1`), per-gene summary files will be created by default (turn off with `--printgenestats 0`), and a summary statistics file will be created. The per-gene summary file includes a few summary statistics (e.g., sample size, mutation burden, heterozygosity) computed from the per-gene SFS (e.g., heterozygosity stored as `pi`, mutation burden stored as `xbar`), while the summary statistics files computes statistics by pooling the

Additional files can be created for gene simulations using the options `--printbiallelicSFS 1` and `--printbiallelicsummary 1`. The `--printbiallelicSFS` option creates a biallelic SFS by pooling frequencies within each gene under the assumption that it behaves as a biallelic locus, and then creating an SFS from the set of genes with each allele count representing a biallelic gene. The `--printbiallelicSFS 1` option creates a summary statistics file for the biallelic SFS, whether or not –printbiallelicSFS 1 is used.

The first class of files is the SFS file. The raw and sample SFS () files will contain two columns: "Counts" and "Number_of_alleles". "Counts" tells you the the number of haploid individuals a polymorphic allele is in from 1 to $2N-1$. This can be converted to the frequency of an allele by dividing by the final population size $2N_{final}$. Additionally, there is a zero counts bin, telling you the number of monomorphic ancestral alleles. The number of monomorphic derived alleles (i.e., fixations) can be computed by calculating the remaining alleles in a simulation of L sites, but is also explicitly stated in the summary statistics file. "Number of alleles" tells you the number of sites that appear at this frequency in the final simulated generation. By default, this file is sparse–all counts with zero alleles were removed before printing the file. To instead print the full SFS, use the option `--notsparse`, but be aware that this can take up substantially more disk space and runtime, especially when doing a large number runs or printing an SFS for each simulated gene. Since this is dependent on the final population size, this is only recommended for simulations of small populations. An example of an SFS file is shown in Figure 1.


SFS_supertennessen_european_S_-3.0_h_0.0_mu_-8.0_L_4.0_growth_0.025.tsv

```
Counts   Number_of_alleles
0        7982
1        615
2        224
3        137
4        138
5        92
6        65
7        57
8        58
9        35
10       35
11       27
12       24
13       21
14       17
15       18
16       17
17       13
18       17
19       12
20       12
21       15
22       12
23       13
24       14
25       15
26       9
27       4
28       12
```

Figure 1: Screenshot of an example SFS file produced by `simDoSe`.

The second type of file created is the summary statistics file. These files are stored in separate folders with the prefix `_summary_stats`. Summary statistics files contains information about the SFS without producing the full SFS. Additionally, they print the run parameters (e.g., mutation rate, selection and dominance coefficients, demography). They also display the initial and final population size, so if you need to know the final population size to simulate genes without downsampling, look here. In addition to these parameters the following summary statistics are displayed.

- Mutation burden, labeled `xbar`. Here, the "bar" indicates a weighted average over the site frequency spectrum ϕ. This is the first non-central moment of the SFS, and is defined by the following equation.

$$\overline{x} \equiv \sum_{x_i=1/2N}^{1-1/2N} x_i \; \phi(x_i; h, s, N(t)) \tag{4}$$

  This can be interpreted as the average number of mutations per haploid individual in the population/sample. This is proportional to the mutation load if all selection is additive.

- Homozygosity, labeled `x2bar`. This is the second non-central moment of the SFS, and is defined as follows.

$$\overline{x^2} \equiv \sum_{x_i=1/2N}^{1-1/2N} x_i^2 \; \phi(x_i; h, s, N(t)) \tag{5}$$

  This can be interpreted as the average number of homozygotes per individual in the population/sample. This is proportional to the mutation load if all selection is recessive.

- Heterozygosity, labeled `pi`. This is the following moment of the SFS.

$$\pi = \overline{2x(1-x)} \equiv \sum_{x_i=1/2N}^{1-1/2N} 2x_i(1-x_i) \; \phi(x_i; h, s, N(t)) \tag{6}$$

  This is the average number of heterozygote individuals in the population, and indicates the presence of selection.

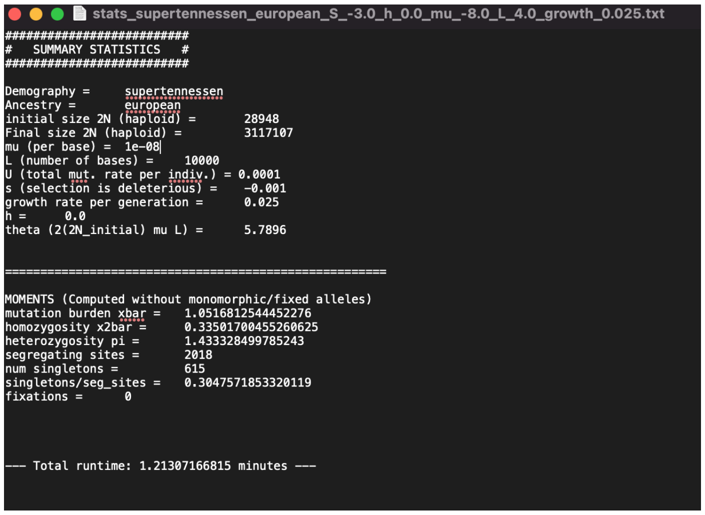
- Number of segregating sites. This is the integral of the SFS over all polymorphic sites.

$$SS \equiv \sum_{x_i=1/2N}^{1-1/2N} \phi(x_i; h, s, N(t)) \tag{7}$$

- Number of singletons. This is the number of alleles that exist in only one individual in the sample, or, equivalently, the number of alleles at frequency $x_i = 1/2N$.

- Number of fixations. Since the SFS only contains information about segregating sites, the number of fixations (i.e., monomorphic derived alleles) is not contained in the SFS file (though the number of monomorphic ancestral alleles is). This is the number of fixations since the beginning of the demography. Fixations during the burn-in phase are not included.

Finally, the total run time is printed at the end of the summary statistics file. An example of an SFS file is shown in Figure 2.

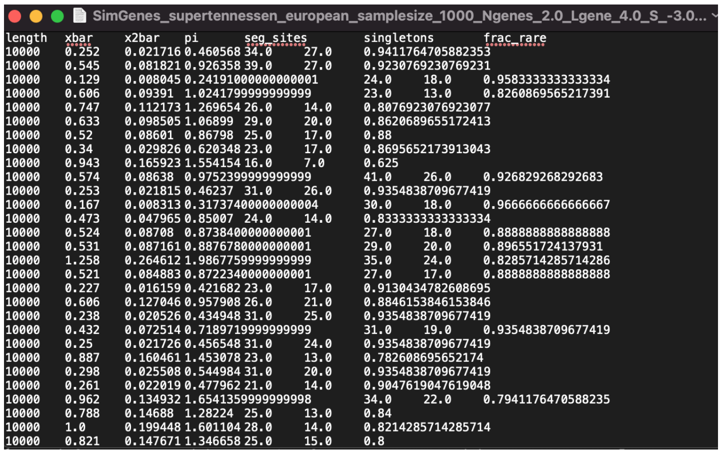


```
stats_supertennessen_european_S_-3.0_h_0.0_mu_-8.0_L_4.0_growth_0.025.txt
##########################
#   SUMMARY STATISTICS   #
##########################

Demography =     supertennessen
Ancestry =       european
initial size 2N (haploid) =       28948
Final size 2N (haploid) =         3117107
mu (per base) =  1e-08
L (number of bases) =    10000
U (total mut. rate per indiv.) = 0.0001
s (selection is deleterious) =    -0.001
growth rate per generation =      0.025
h =      0.0
theta (2(2N_initial) mu L) =      5.7896


=====================================================

MOMENTS (Computed without monomorphic/fixed alleles)
mutation burden xbar =    1.0516812544452276
homozygosity x2bar =      0.33501700455260625
heterozygosity pi =       1.433328499785243
segregating sites =       2018
num singletons =          615
singletons/seg_sites =    0.3047571853320119
fixations =      0




--- Total runtime: 1.21307166815 minutes ---
```

Figure 2: Screenshot of an example summary statistics file produced by `simDoSe`.

When simulating individual genes, additional SFS and summary stats files are produced beyond those for the full and sampled populations. The per-gene SFS files are identical in structure to the sample SFS files. Summary statistics files are not produced per gene (this is contained in the per-gene summary file), but a summary statistics file is produced for the whole set of simulated genes. If you are not printing results for biallelic assumptions (i.e. using `--printbiallelicSFS 0`), this summary statistics file just contains run parameters. If you are instead using `--printbiallelicSFS 1`, this summary file will contain a list of moments under the title `BIALLELIC MOMENTS`. These are statistics of the pooled SFS under the assumption that each simulated gene behaves as a single biallelic locus.

The per-gene summary statistics file contains a list of moments of the individual per-gene SFS, where each row corresponds to a simulated gene. The following moments are listed: gene length, mutation burden (`xbar`), homozygosity (`x2bar`), heterozygosity (`pi`), number of segregating sites, number of singletons, and the fraction of rare alleles. Additionally, if the option `--printbiallelicSFS 1` is passed, the per-gene summary file will be labeled with the suffix `_biallelic`, and will contain an additional column called `total_counts`. This column counts the total number of individuals with any mutation a given gene, and a number is computed for each simulated gene. An example of an SFS file is shown in Figure 3.



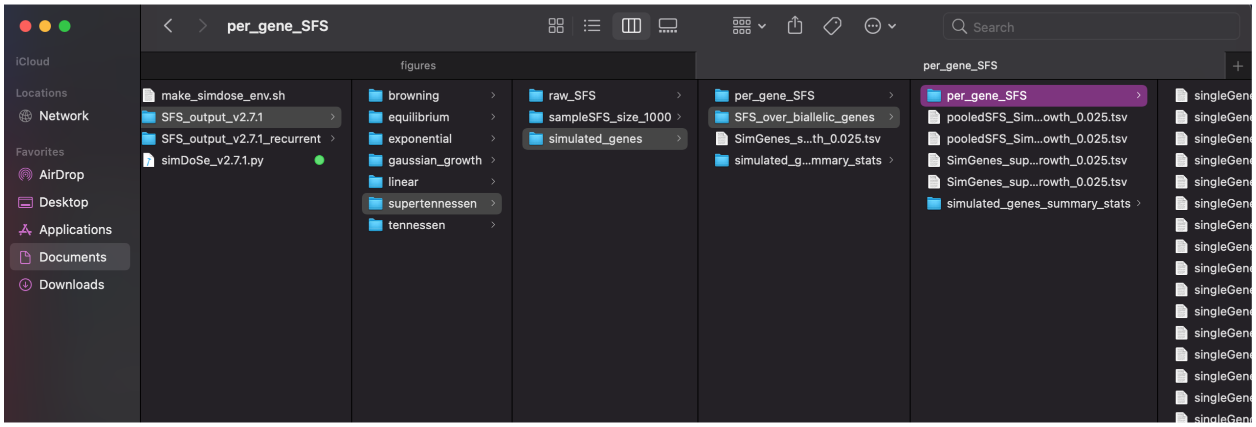
```
SimGenes_supertennessen_european_samplesize_1000_Ngenes_2.0_Lgene_4.0_S_-3.0...
length  xbar    x2bar   pi      seg_sites       singletons      frac_rare
10000   0.252   0.021716 0.460568 34.0  27.0    0.9411764705882353
10000   0.545   0.081821 0.926358 39.0  27.0    0.9230769230769231
10000   0.129   0.008045 0.24191000000000001    24.0    18.0    0.9583333333333334
10000   0.606   0.09391 1.0241799999999999      23.0    13.0    0.8260869565217391
10000   0.747   0.112173 1.269654 26.0  14.0    0.8076923076923077
10000   0.633   0.098505 1.06899 29.0   20.0    0.8620689655172413
10000   0.52    0.08601 0.86798 25.0    17.0    0.88
10000   0.34    0.029826 0.620348 23.0  17.0    0.8695652173913043
10000   0.943   0.165923 1.554154 16.0  7.0     0.625
10000   0.574   0.08638 0.9752399999999999      41.0    26.0    0.926829268292683
10000   0.253   0.021815 0.46237 31.0   26.0    0.9354838709677419
10000   0.167   0.008313 0.31737400000000004    30.0    18.0    0.9666666666666667
10000   0.473   0.047965 0.85007 24.0   14.0    0.8333333333333334
10000   0.524   0.08708 0.8738400000000001      27.0    18.0    0.8888888888888888
10000   0.531   0.087161 0.8876780000000001     29.0    20.0    0.896551724137931
10000   1.258   0.264612 1.9867759999999999     35.0    24.0    0.8285714285714286
10000   0.521   0.084883 0.8722340000000001     27.0    17.0    0.8888888888888888
10000   0.227   0.016159 0.421682 23.0  17.0    0.9130434782608695
10000   0.606   0.127046 0.957908 26.0  21.0    0.8846153846153846
10000   0.238   0.020526 0.434948 31.0  25.0    0.9354838709677419
10000   0.432   0.072514 0.7189719999999999     31.0    19.0    0.9354838709677419
10000   0.25    0.021726 0.456548 31.0  24.0    0.9354838709677419
10000   0.887   0.160461 1.453078 23.0  13.0    0.782608695652174
10000   0.298   0.025508 0.544984 31.0  20.0    0.9354838709677419
10000   0.261   0.022019 0.477962 21.0  14.0    0.9047619047619048
10000   0.962   0.134932 1.6541359999999998     34.0    22.0    0.7941176470588235
10000   0.788   0.14688 1.28224 25.0    13.0    0.84
10000   1.0     0.199448 1.601104 28.0  14.0    0.8214285714285714
10000   0.821   0.147671 1.346658 25.0  15.0    0.8
```

Figure 3: Screenshot of an example per-gene summary file produced by `simDoSe`.

`simDoSe` produces a number of folders with (hopefully) intuitive names. The basic output folder is named `SFS_output_version`, where `version` is replaced with the version of `simDoSe` used to generate the files contained within. To avoid mixing up recurrent and non-recurrent simulations, if you are using the `--recurrent` flag, this folder will have the suffix `_recurrent`. Inside this folder will be folders named with the demography specified for each simulation. For example, there may be a `equilibrium` folder, along with `tennessen` and `supertennessen` folders. Inside the demography folder, there are generally at least two folders within (unless `--printrawSFS 0` or `--printrsampleSFS 0` are specified). There is a folder containing simulation output for the full population, prior to downsampling, labeled `raw_SFS`. There is an analogous folder called `sampleSFS_size_xx`, where `xx` is replaced by the integer specified with `--samplesize`. A third folder labeled `simulated_genes` may exist if you used the options `--Ngenes` and `--Lgenes`, which contains all output for simulated genes.

Within the `raw_SFS` and `sampleSFS_...` folders, one simulated SFS will appear for each combination of parameters simulated. For each simulation, these SFS files will be produced for both the full population and population sample, unless `--printrawSFS 0` or `--printrsampleSFS 0` are specified. Additionally, these folders will contain a folder labeled `summary_stats` or `sample_summary_stats` for the full population or population sample, respectively. These folders contain one summary statistics file for each set of parameters simulated. The `simulated_genes` folder contains a per-gene summary file detailed above, and a folder labeled `simulated_genes_summary_stats` containing the parameters simulated to produce this file. Additionally, if

`--printgeneSFS 1` was specified, there will be a folder in the `simulated_genes` folder containing one SFS per simulated gene and numbered from 0 to ($N_{genes} - 1$). If the option `--printbiallelicSFS 1` was used, the `per_gene_SFS` and `simulated_genes_summary_stats` folders will instead be inside a folder labeled `SFS_over_biallelic_genes`, and an additional file with the prefix `pooledSFS_` will be created in this folder. This file contains the SFS created by assuming each simulated gene is a single biallelic locus and pooling over all simulated genes. The general folder structure produced by `simDoSe` is shown in Figure 4.



make_simdose_env.sh
SFS_output_v2.7.1
SFS_output_v2.7.1_recurrent
simDoSe_v2.7.1.py

browning
equilibrium
exponential
gaussian_growth
linear
supertennessen
tennessen

raw_SFS
sampleSFS_size_1000
simulated_genes

per_gene_SFS
SFS_over_biallelic_genes
SimGenes_s...th_0.025.tsv
simulated_g...mmary_stats

per_gene_SFS
pooledSFS_Sim...owth_0.025.tsv
pooledSFS_Sim...owth_0.025.tsv
SimGenes_sup...rowth_0.025.tsv
SimGenes_sup...rowth_0.025.tsv
simulated_genes_summary_stats

singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen
singleGen

Figure 4: Screenshot of the general `simDoSe` folder structure.

4 Dependencies

`simDoSe` is scripted in Python 2.7. This may present an annoyance to some who are used to Python 3, but this can be easily remedied (see below). The simulator uses the following packages that must be installed prior to use: `numpy`, `scipy`, `pandas`. Additionally, the following packages, which should be available with any standard Python installation, are used: `sys`, `os`, `optparse`, `itertools`, and `time`. For informational purposes, `simDoSe` relies heavily on `numpy.random` and `sciply.stats`.

If you happen not to have any of the above installed, particularly if you are primarily using Python 3, the easiest option is to use Anaconda. Anaconda for individual use is freely available at `https://www.anaconda.com/products/individual`, though the institution or company versions (if you don't already have one) may not be free. Once you have downloaded Anaconda, you can simply run the file `make_simdose_env.sh` with the following command in the terminal.

```
bash ./make_simdose_env.sh
```

Once you have finished creating the Anaconda environment, you can activate the environment with the following command.

```
conda activate simdose
```

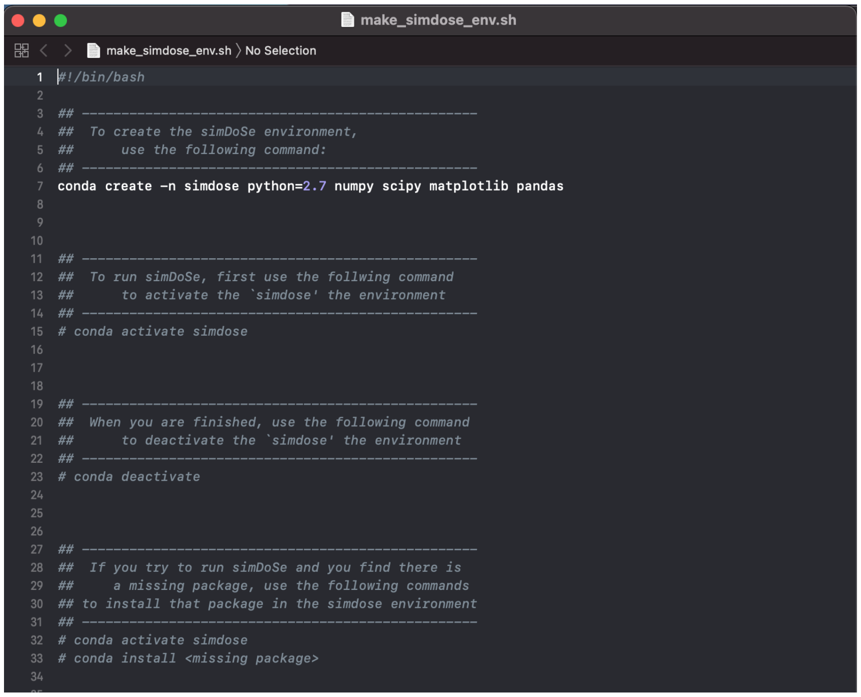
Now you are ready run any `simDoSe` simulations you can imagine! If you want to deactivate the environment to get back to your normal install of Python, use the following command.

```
conda deactivate
```

If you attempted to run `simDoSe` and were told that a package is missing (this is unlikely if Anaconda was installed correctly), first activate the simdose environment with `conda activate simdose`, then install the missing package with the following command.

```
conda install <package name>
```

Replace `<package name>` with the name of the package you were told is missing (e.g., `conda install optsparse` to install a missing `optsparse` package). For your convenience, all of these commands are commented out with explanations in the `make_simdose_env.sh` file, as shown in Figure 5.



```
#!/bin/bash

## --------------------------------------------------
##  To create the simDoSe environment,
##      use the following command:
## --------------------------------------------------
conda create -n simdose python=2.7 numpy scipy matplotlib pandas



## --------------------------------------------------
##  To run simDoSe, first use the follwing command
##      to activate the `simdose' the environment
## --------------------------------------------------
# conda activate simdose



## --------------------------------------------------
##  When you are finished, use the following command
##      to deactivate the `simdose' the environment
## --------------------------------------------------
# conda deactivate



## --------------------------------------------------
##  If you try to run simDoSe and you find there is
##     a missing package, use the following commands
## to install that package in the simdose environment
## --------------------------------------------------
# conda activate simdose
# conda install <missing package>

```

Figure 5: Contents of the `make_simdose_env.sh` file with all commands needed to set up the `simDoSe` environment.

5 Instructions for running `simDoSe`

Once the `simDoSe` environment is activated with `conda activate simdose`, running `simDoSe` is just a matter of specifying options in the command line. For example, let's say you want to simulate a population in equilibrium with 10^4 individuals under recessive selection ($h = 0$) with selection coefficient $s = 10^{-3}$ with mutation rate of 10^{-7} for $L = 10^4$ simulated sites. You want to downsample the population to match the size of a dataset with $2n = 1000$ individuals. If you want to run this simulation with `simDoSe_v2.7.1`, the following command will do just that.

```
python simDoSe_v2.7.1.py -D equilibrium --initpopsize 10000 -S -0.001 -H 0 -U 0.0000001
        -L 10000 --samplesize 10000
```

As you can see, the commands get quite long, so it's often easiest (and harder to forget parameters) if you write a short bash script to run the simulations you would like. Now let's say you want the same parameters, but want to also simulate 100 genes of length $L_{gene} = 1000$. In that case, just add the options `--Ngenes 100` and `--Lgenes 1000`. The full command is as follows.

```
python simDoSe_v2.7.1.py -D equilibrium --initpopsize 10000 -S -0.001 -H 0 -U 0.0000001
        -L 10000 --samplesize 10000 --Ngenes 100 --Lgenes 1000
```

This will only print a per-gene summary file for the simulated genes, but perhaps you want to save the SFS for each simulated gene to file. In that case, add the command `--printgeneSFS 1` (this is a binary option), which results in the following (very long) command.

```
python simDoSe_v2.7.1.py -D equilibrium --initpopsize 10000 -S -0.001 -H 0 -U 0.0000001
        -L 10000 --samplesize 10000 --Ngenes 100 --Lgenes 1000 --printgeneSFS 1
```

If you wanted to also simulate these genes under the assumption that they behave in a biallelic manner, you would add the command `--printbiallelicSFS 1`.

Generally, the names and aliases of each option were chosen to be as intuitive as possible, but if you forget any or get confused, you can refer to the Command Line Options section above. If you happen to run into any bugs while simulating, please email `dbalick@hms.harvard.edu` or `dbalick@gmail.com`. All options have defaults, discussed above, so `simDoSe` can technically run with no command line options. However, to run simulations of interest and produce the output of interest, it is best to specify as many options as possible, unless you are familiar with all of the default settings. Additionally, it is strongly recommended that you specify the binary output file options (e.g., `--printgeneSFS 0/1`) if you are running very long or computationally intensive simulation (e.g., on a computing cluster) to avoid runs that end without the files you intended to produce.

One peculiarity is that currently all simulated genes are assembled from the segregating sites after downsampling to simulated a cohort of individuals randomly sampled from the full population and sequenced. If, for any reason, you would like to simulate genes from the full population rather than from the sample (e.g., you would like to do downstream analysis prior to downsampling) you can approximate this by setting the sample size to the final population size. To do this, run a short simulation with the demography of interest (i.e., use `-D <some demography>` and potentially `--initpopsize` and/or `--growthrate`) using very quick parameters, specifically a small target size (e.g., `-L 100`), and open the summary statistics file in either the `raw_SFS` or `sampleSFS_...` folders. The final population size will be listed next to `Final size 2N (haploid) =`. Then run your intended simulation by specifying `--samplesize` with the final population size you found. If you happen to find any other peculiarities or options you would like added, please email `dbalick@hms.harvard.edu` or `dbalick@gmail.com`.

Finally, a brief discussion of parallelization. First, since all sites evolve independently, parallelization is quite straight forward: run many simulations with smaller target size and pool the resulting site frequency spectra to create a simulated spectra for a larger mutational target. For example, simulations of the `tennessen` demography, as with all `simDoSe` demographies, depend linearly on the simulated target size. To create a deep "reference" SFS (total target size $L = 10^9$) for use as a theoretical expectation in a likelihood-based inference, you can simulate 1000 times with a target size of $L = 10^6$ using the following loop in bash.

```
#!/bin/bash
for ((RUN=1;RUN<=1000;RUN++));
do
python simDoSe_v2.7.1.py -D tennessen -S -0.001 -H 0 -L 1000000 --samplesize 3000 --run ${RUN}
        --printrawSFS 1 --printsampleSFS 1 > simdose_output_run${RUN}.out &
done;
```

Note that the `--run` command is used here so that all of the 1000 simulation results will land in the same folder, labeled by the run number, without overwriting. The `--printrawSFS 1` and `--printsampleSFS 1` are used to ensure that both the pre and post-downsampled SFS are produced, and `--samplesize 3000` specifies that you will be comparing this data to a cohort with $2n = 3000$ haploid individuals (1500 diploid individuals). As always, `>` will write the output to a file, and labeling the filename with `${RUN}` will ensure that these output files are not overwritten.

6 Additional examples

To further illustrate command line specification of simulation and output parameters, here are a few, slightly more complicated, examples.

- Run the `tennessen` demography for African demographic history with negative selection $s = -0.1$ for additive selection $h = 0.5$ for $L = 10^6$ sites and a sample size of $2n = 10^4$.

  ```
  python simDoSe_v2.7.1.py -D tennessen --ancestry african -S -0.1 -H 0.5 -L 1000000
         --samplesize 10000
  ```

- Run the `exponential` demography with the same parameters and a growth rate of 10^{-4}

  ```
  python simDoSe_v2.7.1.py -D exponential --growthrate 0.0001 -S -0.1 -H 0.5 -L 1000000
         --samplesize 10000
  ```

- Run the `linear` demography with the same parameters and growth rate, but with initial population size $2N = 1000$ and with positive selection of the same strength.

  ```
  python simDoSe_v2.7.1.py -D linear --growthrate 0.0001 -S 0.1 -H 0.5 -L 1000000
         --initpopsize 1000 --samplesize 10000
  ```

- Run the `supertennessen` demography with a growth rate of 0.03 (corresponding to the inferred rate from [3]) for underdominant selection with heterozygous selection coefficient $hs = -0.01$.

```
python simDoSe_v2.7.1.py -D supertennessen --growthrate 0.03 -S -1 -H 0.01 --underdom
-L 1000000
       --initpopsize 1000 --samplesize 10000
```

Or alternatively:

```
python simDoSe_v2.7.1.py -D supertennessen --growthrate 0.03 -S -0.01 -H 1 --underdom
-L 1000000
       --initpopsize 1000 --samplesize 10000
```

Any combination of h and s that matches your intended value of hs will work.

- Run the same demography and parameters, but with heterozygote advantage (i.e., overdominant selection).

```
python simDoSe_v2.7.1.py -D supertennessen --growthrate 0.03 -S 0.01 -H 1 --underdom -L
1000000
       --initpopsize 1000 --samplesize 10000
```

Note that this just amounts to a sign flip in hs, since `--underdom` fixes the strength of selection of homozygotes at $s = 0$.

- Approximate underdominance without using the `--underdom` flag, and run the same parameters in an equilibrium demography.

```
python simDoSe_v2.7.1.py -D equilibrium -S -0.000001 -H 10000 -L 1000000
       --initpopsize 1000 --samplesize 10000
```

Again, any combination of h and s that matches your intended value of hs will work, but $s \ll 1/2N$ should be very weak and h should be very large.

- Run the `tennessen european` demography with $s = -0.001$ under recessive selection $h = 0$ and output the SFS for the full population, the downsampled population, simulate 100 genes with the `--russiandoll` flag for all decades smaller than L, and output the SFS for each gene.

```
python simDoSe_v2.7.1.py -D tennessen -A european -S -0.001 -H 0 -L 1000000
       --samplesize 10000 --russiandoll --Ngenes 100 --printrawSFS 1 --printsampleSFS 1
       --printgeneSFS 1
```

Acknowledgements

DJB would like to thank Shamil Sunyaev for helpful discussions about various functions. Additionally, Daniel M. Jordan was extremely helpful for identifying Python 2.7 particularities, helping to test all manner of the `simDoSe` functionality, and for advice in preparing this manual.

References

1. Bustamante, C., J., W., S., S. & Hartl, D. Directional selection and the site-frequency spectrum. *Genetics* **159,** 1779–1788 (2001).

2. Tennessen, J. A. *et al.* Evolution and Functional Impact of Rare Coding Variation from Deep Sequencing of Human Exomes. *Science* **337,** 64–69. doi:10.1126/science.1219240 (2012).

3. Weghorn, D., Balick, D. J., Cassa, C., Kosmicki, J. A., Daly, M. J., Beier, D. R. & Sunyaev, S. R. Applicability of the Mutation-Selection Balance Model to Population Genetics of Heterozygous Protein-Truncating Variants in Humans. *Mol. Biol. Evol.* **36,** 1701–1710. doi:10.1093/molbev/msz092 (2019).

4. Browning, S. R. & Browning, B. L. Accurate Non-parametric Estimation of Recent Effective Population Size from Segments of Identity by Descent. *American Journal of Human Genetics* **97,** 404–18. doi:10.1016/j.ajhg.2015.07.012 (2015).