Mito Simulation v.1.1

Main program files

config.py-

Contains all of the adjustable parameters (including input file paths for more complex parameters) needed to control the simulation, with the exception of optional selection sensitivity control (please adjust in **sim_core.py** if needed; see **selection model parameters** section for more details). **Required for running**

init_fasta.py-

Processes input files for initializing cells and mtDNA sequences. **Required for running**

mutation_tracks.py-

Processes input files for initializing mutation parameters. **Optional** (if no mutation parameters desired)

selection_tracks.py-

Processes input files for initializing selection parameters. **Optional** (if no selection parameters desired)

main.py-

Contains the general simulation workflow. **Required for running**

sim_core.py-

Contains main methods for running individual simulation processes (eg. death and replication cycles, selection mechanism). Also contains optional sensitivity control for additive selection. **Required for running**

snapshot.py-

Outputs snapshots of simulated cell population states (eg. list of current cells and their respective mitochondrial copies, sequences). **Required for running**

Plotting files

fixation_plot.py-

Outputs a plot of the % of fixed cells (averaged across replicates for a particular run, meaning % of cells that have homogenous mtDNA copies) by timestep. Can plot multiple runs for comparison. **Optional**

cell_fixation_plot.py-

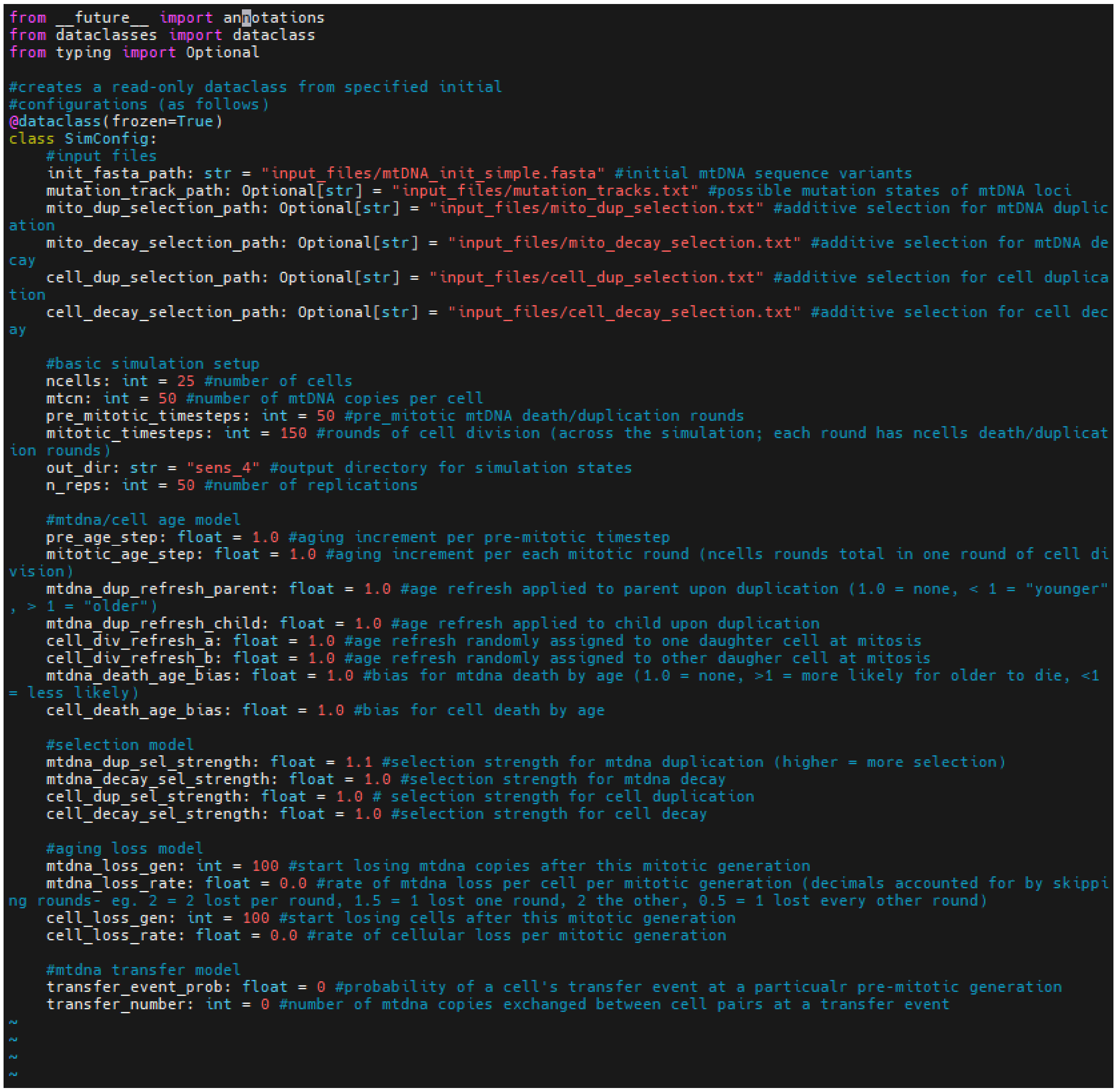
Outputs a plot of the % of cells that have fixed to the most predominant haplotype of fixation (for example, among cells that have fixed- mtDNA homogenized to some haplotype, take the most common haplotype of fixation, and check what % of cells are fixed to that). Measures homogenization on the cellular level. Can plot multiple runs for comparison. **Optional**

basefreq_position.py-

Outputs a plot of the frequency of a particular base pair (among A, T, G, C) at a specified position per run, across all mitochondria, averaged across cells and then replicates. Can plot multiple runs for comparison. **Optional**

General simulation instructions

To run the simulation, please gather all needed program files into one directory level. Before submitting a job (eg. using **sbatch main.sh** from the example directories), adjust parameters and input files for **config.py** (see screenshot below).

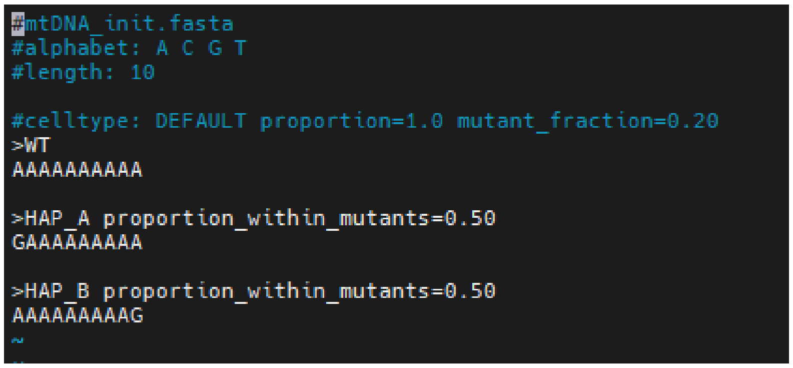


```
from __future__ import annotations
from dataclasses import dataclass
from typing import Optional

#creates a read-only dataclass from specified initial
#configurations (as follows)
@dataclass(frozen=True)
class SimConfig:
    #input files
    init_fasta_path: str = "input_files/mtDNA_init_simple.fasta" #initial mtDNA sequence variants
    mutation_track_path: Optional[str] = "input_files/mutation_tracks.txt" #possible mutation states of mtDNA loci
    mito_dup_selection_path: Optional[str] = "input_files/mito_dup_selection.txt" #additive selection for mtDNA duplic
ation
    mito_decay_selection_path: Optional[str] = "input_files/mito_decay_selection.txt" #additive selection for mtDNA de
cay
    cell_dup_selection_path: Optional[str] = "input_files/cell_dup_selection.txt" #additive selection for cell duplica
tion
    cell_decay_selection_path: Optional[str] = "input_files/cell_decay_selection.txt" #additive selection for cell dec
ay

    #basic simulation setup
    ncells: int = 25 #number of cells
    mtcn: int = 50 #number of mtDNA copies per cell
    pre_mitotic_timesteps: int = 50 #pre_mitotic mtDNA death/duplication rounds
    mitotic_timesteps: int = 150 #rounds of cell division (across the simulation; each round has ncells death/duplicat
ion rounds)
    out_dir: str = "sens_4" #output directory for simulation states
    n_reps: int = 50 #number of replications

    #mtdna/cell age model
    pre_age_step: float = 1.0 #aging increment per pre-mitotic timestep
    mitotic_age_step: float = 1.0 #aging increment per each mitotic round (ncells rounds total in one round of cell di
vision)
    mtdna_dup_refresh_parent: float = 1.0 #age refresh applied to parent upon duplication (1.0 = none, < 1 = "younger"
, > 1 = "older")
    mtdna_dup_refresh_child: float = 1.0 #age refresh applied to child upon duplication
    cell_div_refresh_a: float = 1.0 #age refresh randomly assigned to one daughter cell at mitosis
    cell_div_refresh_b: float = 1.0 #age refresh randomly assigned to other daugher cell at mitosis
    mtdna_death_age_bias: float = 1.0 #bias for mtdna death by age (1.0 = none, >1 = more likely for older to die, <1
= less likely)
    cell_death_age_bias: float = 1.0 #bias for cell death by age

    #selection model
    mtdna_dup_sel_strength: float = 1.1 #selection strength for mtdna duplication (higher = more selection)
    mtdna_decay_sel_strength: float = 1.0 #selection strength for mtdna decay
    cell_dup_sel_strength: float = 1.0 # selection strength for cell duplication
    cell_decay_sel_strength: float = 1.0 #selection strength for cell decay

    #aging loss model
    mtdna_loss_gen: int = 100 #start losing mtdna copies after this mitotic generation
    mtdna_loss_rate: float = 0.0 #rate of mtdna loss per cell per mitotic generation (decimals accounted for by skippi
ng rounds- eg. 2 = 2 lost per round, 1.5 = 1 lost one round, 2 the other, 0.5 = 1 lost every other round)
    cell_loss_gen: int = 100 #start losing cells after this mitotic generation
    cell_loss_rate: float = 0.0 #rate of cellular loss per mitotic generation

    #mtdna transfer model
    transfer_event_prob: float = 0 #probability of a cell's transfer event at a particualr pre-mitotic generation
    transfer_number: int = 0 #number of mtdna copies exchanged between cell pairs at a transfer event
~
~
~
~
```

1. Input files

You may specify your own paths to all the input files. In the default simulation setup provided (seen in the screenshot and in the main github/shared directory), they are put in the folder **input_files**.

a. **init_fasta_path** (path to file containing init sequences + cell-type proportions), **Required for running**



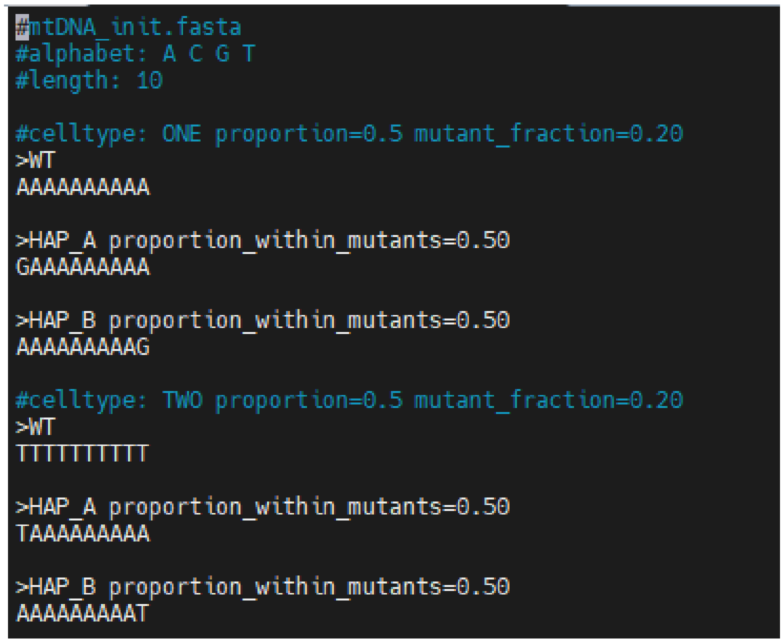
```
#mtDNA_init.fasta
#alphabet: A C G T
#length: 10

#celltype: DEFAULT proportion=1.0 mutant_fraction=0.20
>WT
AAAAAAAAAA

>HAP_A proportion_within_mutants=0.50
GAAAAAAAAA

>HAP_B proportion_within_mutants=0.50
AAAAAAAAAG
~
```

As seen here, first you would specify the **#alphabet** parameter (usually just A C G T) for which specific base pairs are allowed. **#length** here specifies how long a particular mtDNA sequence can be (for instance 10 here, and if you wanted the full mtDNA then something along the lines of 16000).

Then, we have the **#celltype**… line which denotes the name of the cell type (put after **celltype**, eg. **celltype: DEFAULT** or **celltype: one**), what proportion it constitutes within the cell population (put after **proportion**, eg. **proportion=1.0** or **proportion=0.8**), and what percent of the mtDNA copies within the cell are mutants (put after **mutant_fraction**, eg. **mutant_fraction=0.20** or **mutant_fraction=0.30**).

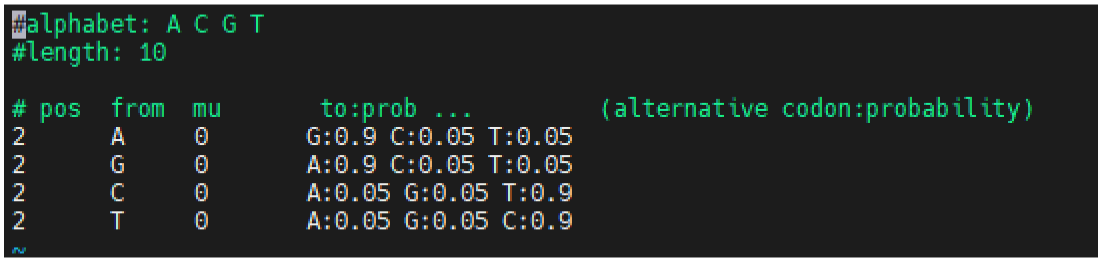
We then list all the possible haplotypes that can occur within the specific cell type. **>WT** denotes the wild-type, which will occur to 1 - **mutant_fraction** proportion. The individual haplotypes you list with **>name proportion_within_mutants** (eg. >HAP_A **proportion_within_mutants=0.50** meaning HAP_A occurs among 50% of mutants, so overall 10% if **mutant_fraction** is 0.20).

To list additional cell types, repeat the same format for another block (with proportions between cell types adding up to 1.0). For instance:



```
#mtDNA_init.fasta
#alphabet: A C G T
#length: 10

#celltype: ONE proportion=0.5 mutant_fraction=0.20
>WT
AAAAAAAAAA

>HAP_A proportion_within_mutants=0.50
GAAAAAAAAA

>HAP_B proportion_within_mutants=0.50
AAAAAAAAAG

#celltype: TWO proportion=0.5 mutant_fraction=0.20
>WT
TTTTTTTTTT

>HAP_A proportion_within_mutants=0.50
TAAAAAAAAA

>HAP_B proportion_within_mutants=0.50
AAAAAAAAAT
```

b. **mutation_track_path** (path to file containing mutation rules), **Optional**



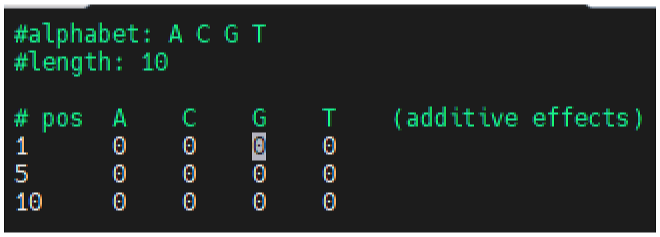
```
#alphabet: A C G T
#length: 10

# pos   from   mu       to:prob ...         (alternative codon:probability)
2       A      0       G:0.9 C:0.05 T:0.05
2       G      0       A:0.9 C:0.05 T:0.05
2       C      0       A:0.05 G:0.05 T:0.9
2       T      0       A:0.05 G:0.05 C:0.9
~
```

Like with the file specified in **init_fasta_path**, first we specify the alphabet of possible base pairs and the length of the input mtDNA sequences (exact sequences not shown here but found in **init_fasta_path**).

Each row here specifies a particular mutation rule. In column **#pos**, we specify which position we are referring to. **from** refers to the starting base pair. For example, if we have 2, A for **#pos**, **from**, the mutation rule refers to the ways base pair A at position 2 can mutate. **mu** specifies the overall mutation rate, and **to:prob** specifies the exact proportion of other base pairs from can mutate to. For instance, if we set **mu** to 0.01, that means at each replication, there is a 0.01 chance that **from** mutates to say G at 0.9, C at 0.05, T at 0.05 (given **to:prob** is G:0.9 C:0.05 T:0.05). To add another position/base pair's mutation rule, simply start another row.

c. **mito_[dup/decay]_selection_path** (paths to files containing mtDNA selection tracks for duplication and decay respectively), **Optional**

d. **cell_[dup/decay]_selection_path** (paths to files containing cell selection tracks for duplication and decay respectively), **Optional**


```
#alphabet: A C G T
#length: 10

# pos  A    C    G    T    (additive effects)
1      0    0    0    0
5      0    0    0    0
10     0    0    0    0
```

Selection rule files are all formatted similarly (eg. **mito_[dup/decay]_selection_path** files look similar except one specifies selection for duplication, and the other for decay at the mitochondrial level, and **cell_[dup/decay]_selection_path** the same but at the cellular level). Alphabet of base pairs and length are specified as in previous files.

Each row's **#pos** column specifies the specific position the selection rule corresponds to. For example if **#pos** is 1, then we are referring to base pairs at position 1. Then, under each base pair, we list the additive effect that base pair contributes to the property (duplication, decay) under selection. If under A we have 1 for instance, then it will contribute 1 to either the mitochondria or cell for duplication or decay. Positive values add to the chance that either duplication or decay occurs, and negative values subtract (making it less likely for that mitochondria or cell). More rows can be added under this format for additional positions.

Note: There are some important aspects of how selection is implemented that limit the range of values one may use here which are discussed under the **selection model parameters** section.

2. **Population and general run parameters**
    a. **ncells** (number of cells in simulation run)
    b. **mtcn** (number of mtDNA copies per cell)
    c. **pre_mitotic_timesteps** (number of pre-mitotic mtDNA death/duplication rounds)
    d. **mitotic_timesteps** (number of rounds of complete cell division rounds)
    e. **out_dir** (path to output directory for simulation states/snapshots)
    f. **n_reps** (number of replications for simulation run)

Here, our simulation will have **ncells** (integer) in the starting population, each with **mtcn** (integer) number of mtDNA copies per cell. It will run for **pre_mitotic_timesteps** (integer) before *each* mitotic timestep (eg. if pre_mitotic_timesteps = 50, then we will go through 50 moran rounds of mtDNA death and duplication within each cell). And then, in each **mitotic_timestep** (integer, specifies how many times this occurs in a given simulation), we go through **ncells** # of cell death and division (eg. one cell dies, another replicates all mtDNA, and divides into daughters replacing itself and the dead cell). In each timestep snapshots are output to **out_dir**, and this repeats for **n_reps** (integer) each with its own directory.

3. **mtDNA/cellular general age model parameters**
    a. **pre_age_step** (aging increment for each pre-mitotic timestep)

b. **mitotic_age_step** (aging increment for each mitotic round- note that there are **ncells** rounds for each mitotic timestep)
c. **mtdna_dup_refresh_parent** (age “refresh” applied to parent mitochondria upon duplication)
d. **mtdna_dup_refresh_child** (age “refresh” applied to child mitochondria upon duplication)
e. **cell_div_refresh_a** (age “refresh” applied to a daughter cell upon cell division, assigned randomly to one of the pair)
f. **cell_div_refresh_b** (counterpart to cell_div_refresh_a)
g. **mtdna_death_age_bias** (bias for mtdna death by age)
h. **cell_death_age_bias** (bias for cell death by age)

pre_age_step simply specifies how much older a cell or mitochondria grows in a particular pre-mitotic timestep, and **mitotic_age_step** the same for a specific round of the mitotic timestep (there are **ncells** or number of cells rounds per mitotic timestep, and in *each* round we increase the age by **mitotic_age_step**).

mtdna_dup_refresh_parent refers to how much we refresh the age by for the parent after a particular mitochondrial duplication, and **mtdna_dup_refresh_child** for the child. For example, if **mtdna_dup_refresh_parent** is 1 and **mtdna_dup_refresh_child** is 0, and the original age of the parent is 50, then after a round of duplication the parent remains 50 but the child duplicate is now 0.

cell_div_refresh_a and **cell_div_refresh_b** refer to how much each daughter cell’s age is refreshed by relative to the parent. They are assigned at random between pairs of daughter cells. If our parent cell for instance is 50, and then **cell_div_refresh_a** is 0.3 and **cell_div_refresh_b** is 1, then between our daughter cells, at random one will receive **cell_div_refresh_a** (making their new age 50x0.3 = 15), and the other **cell_div_refresh_b** (making their new age remain 50).

To implement bias for decaying older mtDNA and cells more quickly than younger ones (assuming greater damage), we use **mtdna_death_age_bias** and **cell_death_age_bias**. They behave the same way. If we set them to 1, then there is no age bias for decay. If we set them >1, then older mtDNA and/or cells will preferentially decay in a moran step. Conversely, while not the reason we implemented this factor, if we set them <1, older mtDNA and/or cells will be protected from decay.

More specifically, how bias for decay is implemented is as follows:

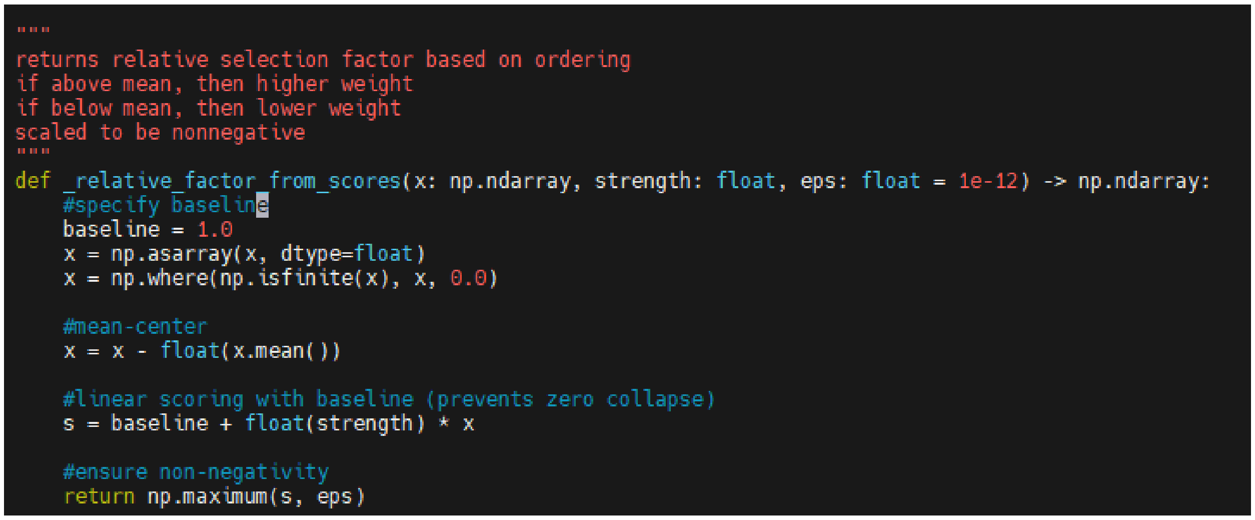
1) We start out with an array of equal “weights” for each mtDNA/cell, eg. [1, 1, …. 1]. This is the same for duplication.
2) For decay multiply each weight by (**age_bias**^{age}), eg. 1 x (1.1^{age}) for each to get our age-adjusted weight, which does not occur for duplication (as age bias does not apply there).

Note: See **selection model parameters** for a worked out example of how duplication/decay weights are computed in full including age bias and selection.

4. **Selection model parameters**
   a. **mtdna_dup_sel_strength** (selection strength for mtdna duplication)
   b. **mtdna_decay_sel_strength** (selection strength for mtdna decay)
   c. **cell_dup_sel_strength** (selection strength for cell duplication)
   d. **cell_decay_sel_strength** (selection strength for cell decay)

All of these simply refer to the factor that the additive selection effect for the process in question (eg. mtDNA duplication, decay, cellular duplication, decay) is multiplied by. As with additive effects, positive values denote selecting *for* the process, and negative values denote selecting *against*. However, this is not done on the *raw* additive selection effect (rather on the mean-adjusted variant). The following is an example of how this is computed for mitochondrial decay (including age bias which figures into the "decay weight"):

1) Let's say that we have an array of additive effects for mtDNA decay (summed up from factors specified in **mito_decay_selection_path**) in a particular cell = [0, 0, 0, 0.1, 0.1]. Let's also say our **mtdna_decay_sel_strength** is 1.1.
2) First, for each mtDNA per cell we start out with a uniform array of "decay weights" = [1, 1, 1, 1, 1]
3) We multiply each one by **age_bias**^{age} (yielding just **age_bias**^{age}) as we specified in the previous section to get our age-adjusted decay weight. Again, this step is skipped for duplication.
4) Then, to get our mean-centered additive effects, we find our mean (0.2/5) = 0.04 and adjust our array by the mean [0, 0, 0, 0.1, 0.1] - 0.04 = [-0.04, -0.04, -0.04, 0.06, 0.06].
5) With an optionally adjustable parameter specified in **sim_core.py** (and *not* in **config.py**); see the function **_relative_factor_from_scores** here:


```
"""
returns relative selection factor based on ordering
if above mean, then higher weight
if below mean, then lower weight
scaled to be nonnegative
"""
def _relative_factor_from_scores(x: np.ndarray, strength: float, eps: float = 1e-12) -> np.ndarray:
    #specify baseline
    baseline = 1.0
    x = np.asarray(x, dtype=float)
    x = np.where(np.isfinite(x), x, 0.0)

    #mean-center
    x = x - float(x.mean())

    #linear scoring with baseline (prevents zero collapse)
    s = baseline + float(strength) * x

    #ensure non-negativity
    return np.maximum(s, eps)
```

for the **baseline** parameter, we compute the selection factor we actually end up using which is **baseline + mtdna_decay_sel_strength x [-0.04, -0.04, -0.04, 0.06, 0.06]** = 1 + 1.1 x [-0.04, -0.04, -0.04, 0.06, 0.06] = [0.956, 0.956, 0.956, 1.066, 1.066].

6) Then, to ensure non-negativity, we take the largest of our computed selection factor (each value in the array from (5)) and 1e-12 which yields the selection factor we actually use. This means that negative values or values ~0 will end up being defaulted to 1e-12 though, **so it is best to select selection strengths, ranges of per-position additive effects, and baseline parameters such that baseline + selection strength x [summed additive effect] > 1e-12.** Additionally, the larger the **baseline** is relative to **selection strength x [summed additive effect]** the less sensitive selection will be. So these values have to be selected carefully.
7) We then multiply this by our age-adjusted decay weight to get the weight we use to compute the probability of mitochondrial decay in this round. This comes out to **$\text{age\_bias}^{\text{age}}$ x selection factor** overall. For duplication, it would just be the **selection factor** due to no age bias.
8) We calculate the probability of each mitochondria decaying by dividing their individual weights by the sum of all weights. This is used to pick the mitochondria that decays this round.

5. Age-related loss model

a. **mtdna_loss_gen** (mitotic generation after which mitochondrial copy loss starts to occur)
b. **mtdna_loss_rate** (rate of mitochondrial copy loss per cell per mitotic generation post-mtdna_loss_gen)
c. **cell_loss_gen** (mitotic generation after which cell loss starts to occur)
d. **cell_loss_rate** (rate of cellular loss per mitotic generation post-cell_loss_gen)

After the specified (**mtdna_loss_gen**, **cell_loss_gen**) mitotic generation, we uniformly reduce the per-cell mitochondrial copy number by **mtdna_loss_rate** (eg. losing 1 per mitotic generation, for *each* cell) and/or the cell population by **cell_loss_rate**.

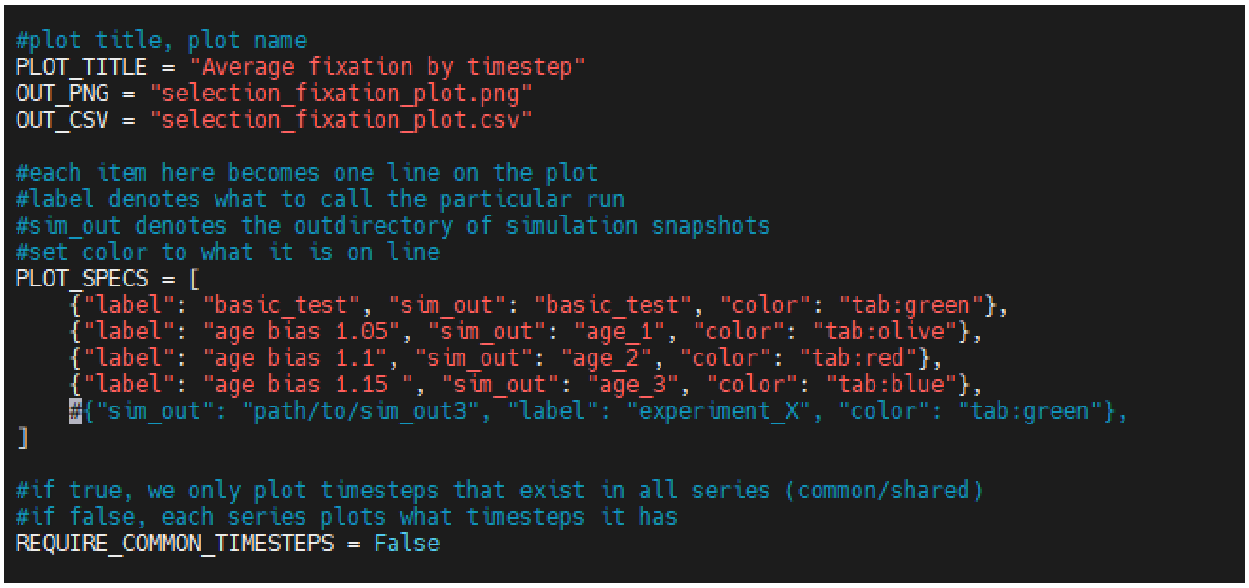
6. mtDNA transfer model

a. **transfer_event_prob** (probability of a particular cell engaging in a transfer event at a particular pre-mitotic generation)
b. **transfer_number** (number of mtDNA copies exchanged between cell pairs in a transfer event)

Each pre-mitotic generation, each cell has **transfer_event_prob** chance of engaging in a mitochondrial transfer event. What happens is that it will then pair up with a randomly selected other cell, and exchange at random **transfer_number** mtDNA copies between them. For example, if **transfer_event_prob** is 0.001 and **transfer_number** is 2, then each pre-mitotic generation all cells have a 0.001 chance of pairing up with a randomly selected other cell and exchanging 2 randomly selected mitochondria.

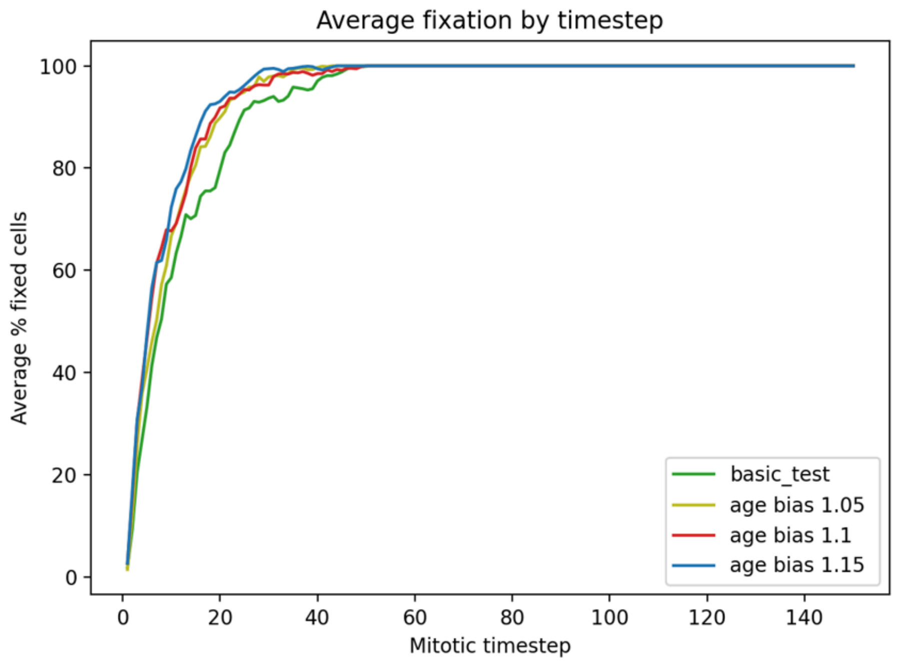
Plotting

fixation_plot.py-

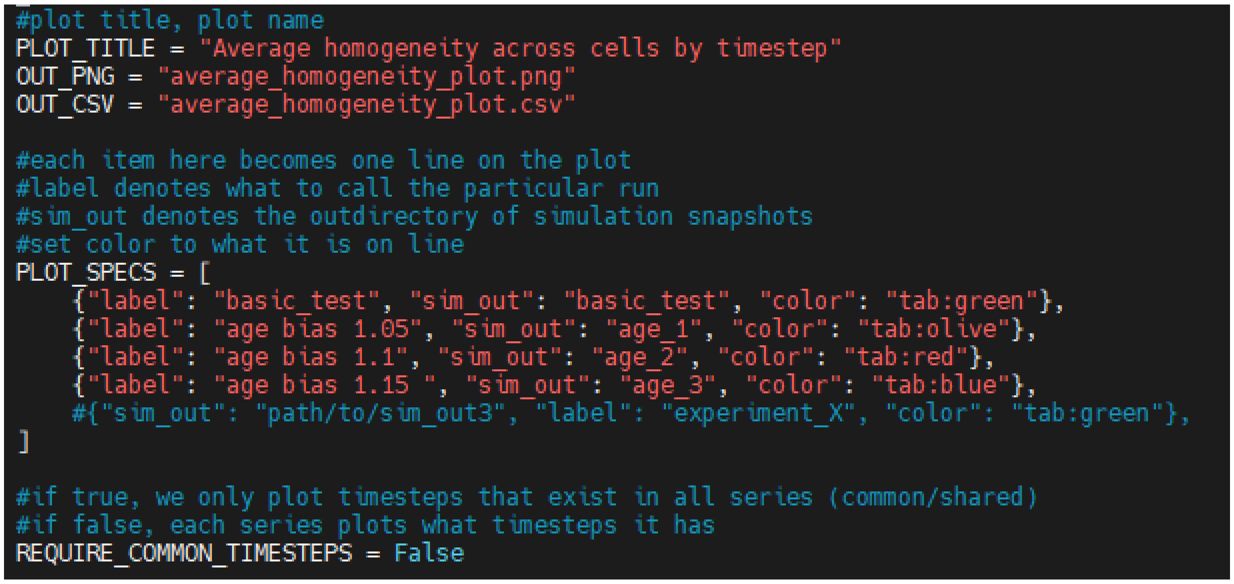


```python
#plot title, plot name
PLOT_TITLE = "Average fixation by timestep"
OUT_PNG = "selection_fixation_plot.png"
OUT_CSV = "selection_fixation_plot.csv"

#each item here becomes one line on the plot
#label denotes what to call the particular run
#sim_out denotes the outdirectory of simulation snapshots
#set color to what it is on line
PLOT_SPECS = [
    {"label": "basic_test", "sim_out": "basic_test", "color": "tab:green"},
    {"label": "age bias 1.05", "sim_out": "age_1", "color": "tab:olive"},
    {"label": "age bias 1.1", "sim_out": "age_2", "color": "tab:red"},
    {"label": "age bias 1.15 ", "sim_out": "age_3", "color": "tab:blue"},
    #{"sim_out": "path/to/sim_out3", "label": "experiment_X", "color": "tab:green"},
]

#if true, we only plot timesteps that exist in all series (common/shared)
#if false, each series plots what timesteps it has
REQUIRE_COMMON_TIMESTEPS = False
```

Parameters for plotting % of fixed cells by timestep are found at the top of **fixation_plot.py**. Title is specified in **PLOT_TITLE**, and the names of the output image in **OUT_PNG** and corresponding CSV in **OUT_CSV**. In the next section, you can specify the run label after **label** and snapshot directory to use (data for plotting that run) after **sim_out**. Color is specified after **color**. Finally, you can specify whether to only use timesteps shared between all series in **REQUIRE_COMMON_TIMESTEPS**. Example output (from the top inputs) is as follows:

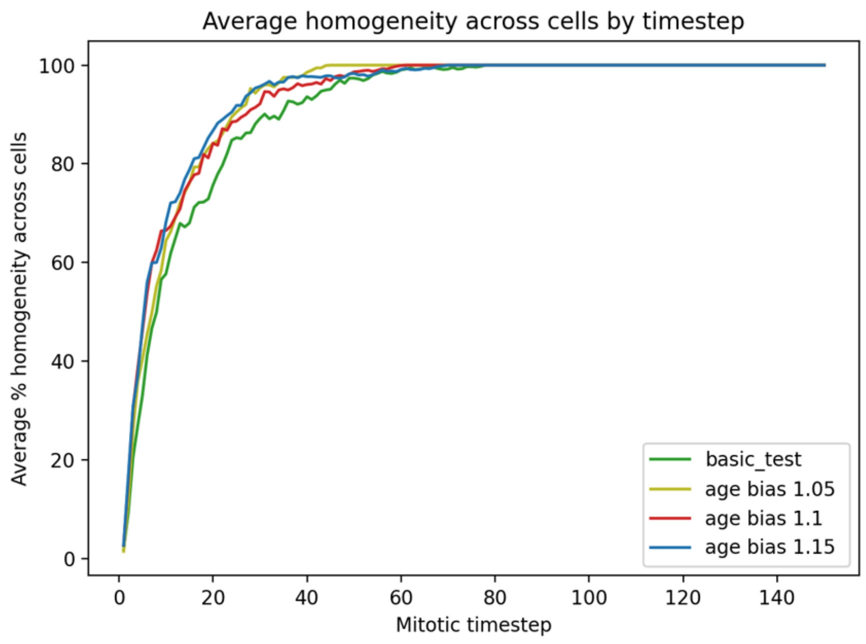


cell_fixation_plot.py-

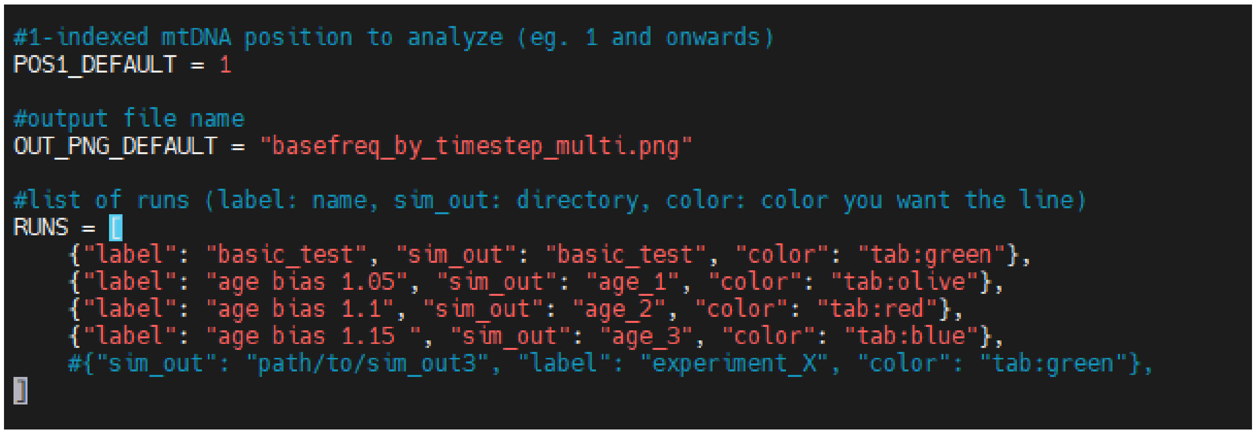


```
#plot title, plot name
PLOT_TITLE = "Average homogeneity across cells by timestep"
OUT_PNG = "average_homogeneity_plot.png"
OUT_CSV = "average_homogeneity_plot.csv"

#each item here becomes one line on the plot
#label denotes what to call the particular run
#sim_out denotes the outdirectory of simulation snapshots
#set color to what it is on line
PLOT_SPECS = [
    {"label": "basic_test", "sim_out": "basic_test", "color": "tab:green"},
    {"label": "age bias 1.05", "sim_out": "age_1", "color": "tab:olive"},
    {"label": "age bias 1.1", "sim_out": "age_2", "color": "tab:red"},
    {"label": "age bias 1.15 ", "sim_out": "age_3", "color": "tab:blue"},
    #{"sim_out": "path/to/sim_out3", "label": "experiment_X", "color": "tab:green"},
]

#if true, we only plot timesteps that exist in all series (common/shared)
#if false, each series plots what timesteps it has
REQUIRE_COMMON_TIMESTEPS = False
```

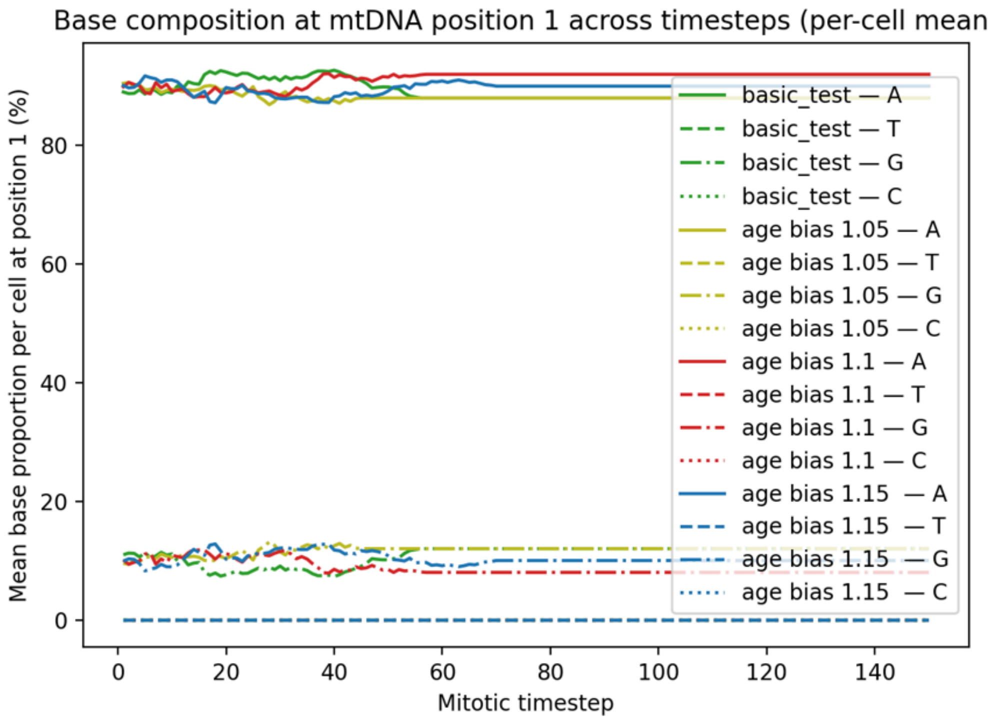
As can be seen here, input parameters are identical to **fixation_plot.py**. Example output is as follows:



basefreq_position.py-


```
#1-indexed mtDNA position to analyze (eg. 1 and onwards)
POS1_DEFAULT = 1

#output file name
OUT_PNG_DEFAULT = "basefreq_by_timestep_multi.png"

#list of runs (label: name, sim_out: directory, color: color you want the line)
RUNS = [
    {"label": "basic_test", "sim_out": "basic_test", "color": "tab:green"},
    {"label": "age bias 1.05", "sim_out": "age_1", "color": "tab:olive"},
    {"label": "age bias 1.1", "sim_out": "age_2", "color": "tab:red"},
    {"label": "age bias 1.15 ", "sim_out": "age_3", "color": "tab:blue"},
    #{"sim_out": "path/to/sim_out3", "label": "experiment_X", "color": "tab:green"},
]
```

While the **RUNS** section is identical in format to the **PLOT_SPECS** section in **fixation_plot.py** and **cell_fixation_plot.py**, we now specify which mtDNA position we analyze in **POS1_DEFAULT** (here using 1). **OUT_PNG_DEFAULT** specifies the name of the output png. This also outputs CSV files for each individual run (containing frequencies of base pairs at the position specified). Example output is as follows:



Base composition at mtDNA position 1 across timesteps (per-cell mean
Mean base proportion per cell at position 1 (%)
0
20
40
60
80
0
20
40
60
80
100
120
140
Mitotic timestep
basic_test — A
basic_test — T
basic_test — G
basic_test — C
age bias 1.05 — A
age bias 1.05 — T
age bias 1.05 — G
age bias 1.05 — C
age bias 1.1 — A
age bias 1.1 — T
age bias 1.1 — G
age bias 1.1 — C
age bias 1.15 — A
age bias 1.15 — T
age bias 1.15 — G
age bias 1.15 — C