This document is a manual for using the StemCellSim program. Here we provide detailed explanations for simulating data and inferring model parameters using ABC from either real or simulated data. We also provide implementation details of our algorithms at the end of the document.

**StemCellSim class**

StemCellSim is a class implemented in c++. The StemCellSim class consists of two files: StemCellSim.h and StemCellSim.cpp. StemCellSim.h contains the class definition and StemCellSim.cpp contains the implementation of its member functions. To carry out simulation, the user must write a main.cpp file and use the functionality of the StemCellSim class by declaring a StemCellSim object and calling its member functions. Example main files for simulating data and inferring model parameters from real and simulated data are provided in our gitlab readme (https://gitlab.com/hormozlab/stemcellsim).

**Elementary member functions**

There are three basic member functions of StemCellSim which are used by other member functions to carry out ABC. The first of these is

bool StemCellSim::runClonalExpansion(double s, int N, int g, int L, double x)

The function parameters of runClonalExpansion() s, N, g, and L correspond to the parameters of the WF model (defined in the Supplement), and x is the parameter that corresponds to the level of feedback in the WF model (also see Supplement). runClonalExpansion() simulates the number of mutant cells as a function of time by doing a series of g binomial draws, where the parameters of each binomial draw depend on the current number of cancer cells. The number of mutant cells at each generation are stored internally in the StemCellSim class in the array cloneSize\_simulated. runClonalExpansion() also stores the value of the function parameter g internally as g\_simulated to hold the length of the cloneSize\_simulated array.

The following function is used to generate a tree, and then an LTT curve from the tree, from the clonal expansion:

double StemCellSim::generateLTT(double mutationRate)

generateLTT() is always run immediately after runClonalExpansion(), and uses the number of mutant cells at each generation stored in the array cloneSize\_simulated to construct a tree that has been averaged (for a high-level description of the algorithm, see our pseudocode for the averaging algorithm in Methods. For implementation details, see the end of the document) with branch lengths in mutations. The mutation rate per generations is determined by the function parameter mutationRate. After the tree is constructed, its LTT curve is then stored in a vector called LTT\_sim. More precisely, the elements of LTT\_sim hold the coalescence times of the tree in number of mutations from the present to the time at which the coalescence event occurs. This is equivalent to knowing the LTT curve of the tree, since vector element i = 1 corresponds to the coalescence time of k lineages, i = 2 to the coalescence time of k – 1 lineages, etc., up until the last coalescence event which corresponds to the time of common ancestry of the sample. i = 0 always holds the value 0, a technical detail that simplifies the algorithm used in the function computeEpsSimAndData() (High-level description of this function is in the section called ABC on simulated data, and implementation details are at the end of the document). generateLTT() also returns the total length of the tree in number of mutations from the present to the birth of the patient. This is used when simulated tree data. The tree data must have both the coalescence times of the tree, along with the total length in number of mutations because it is necessary to know the total length to estimate the mutation rate when carrying out ABC on the data tree. In particular, the return value is used in the function simulateDataAndStoreObservation() (more details on this function are in the section called Simulating data). Implementation details of the algorithm used in generateLTT() are at the end of this document.

After generating the LTT curve of a tree and storing it as coalescence times in the vector LTT\_sim, we can find the area between the LTT curve of a data tree with the LTT curve from a tree generated by ABC. This is accomplished using the function:

double StemCellSim::computeEpsSimAndData()

computeEpsSimAndData() does not have any function parameters. This is because the LTT curves from ABC and the LTT curve from the data tree are stored internally in StemCellSim as coalescence times in LTT\_sim and LTT\_data respectively. computeEpsSimAndData() uses these coalescence times to compute the area between the LTT curves. The return value of computeEpsSimAndData() is the area. Implementation details of the algorithm are at the end of the document.

**Simulating data**

To simulate data, we call the member function

void StemCellSim::simulateDataAndStoreObservation(double ageOfIndividual, int numOfLineages, double mutationsAccrued, double model\_x)

where the parameters denote:

ageOfIndividual = age of patient

numOfLineages = number of mutant cells will sample from clonal expansion

mutationsAccrued = age of patient in mutations (This is equivalent to knowing the mutation rate in mutations per year, since the mutation rate is mutationsAccrued/ageOfIndividual)

model\_x = specifies the amount of feedback (x = 0 corresponds to the usual WF model with no feedback)

simulateDataAndStoreObservation() draws parameter values from prior distributions and stores them internally, and then simulates clonal expansions by calling runClonalExpanion(). runClonalExpansion() is called repeatedly with those parameter values until it generates an expansion that does not go extinct. At this point, the population size as a function of time from the simulated clonal expansion is saved internally in the StemCellSim object in the array cloneSize\_simulated. simulateDataAndStoreObservation() then calls generateLTT() using the mutation rate (mutations per generation) as input. generateLTT() uses the population size as a function of time stored in cloneSize\_simulated from the clonal expansion to simulate the genealogical history, the mutational process, and to perform the averaging algorithm. Information about the tree topology is ignored, and the resulting LTT curve is stored internally as coalescence times in LTT\_sim. Note that the prior distributions that the parameters are drawn from in simulateDataAndStoreObservation() (See code in StemCellSim.cpp) can be easily changed to produce trees with fixed parameter values, or with different priors. This was done in Methods to test the robustness of our model assumptions.

After calling simulateDataAndStoreObservation(), we can print the simulated data tree (more precisely the simulated LTT curve or coalescence times. Trees and their corresponding LTT curves are equivalent since tree topology contains no information about the parameter values that produced the tree) along with the parameters that produced it to a text file. We can accomplish this by calling the printData() function, which takes a file name as input and stores the tree in a text file with that file name. The output file has the following format:

The output is a text file in the following format:

    g = 21

    L = 34

    g/L = 0.617647

    N = 17415

    n = 70

    s = 0.233745

    age = 34

    k = 22

    trajectory: 1,2,3,3,8,12,11,10,16,12,13,18,24,25,28,29,32,39,50,72,70,

    LTT: 20.782,20.782,37.5372,42.1257,43.011,43.011,56.5434,57.0947,57.0947,

    61.9103,67.7438,79.4597,79.4597,79.4597,102.075,108.873,193.097,

    199.235,299.869,300.742,354.301,

    mutationsAccrued: 706.301

The line that begins with “LTT: ” contains the coalescence times of the tree in mutations, delimited by “,”, and the remaining lines contain information about the parameter values that gave rise to the tree. Text files in this format can be read into the StemCellSim object to carry out ABC.

**ABC on simulated data**

ABC is run on simulated data by first reading the data into the StemCellSim object using

void StemCellSim::readParamFile(std::string paramFile),

where the parameter paramFile is the name of the text file storing the tree data in the format shown in the previous section, and then by putting the following function in a for loop:

void StemCellSim::samplePosterior(double eps, double model\_x,

const std::tuple<std::string, std::string,

std::string, std::string, std::string,

std::string, std::string, std::string> & outputFiles)

The samplePosterior() function simulates clonal expansions using runClonalExpansion() under different model parameter values drawn from prior distributions, and then trees from those clonal expansions using generateLTT() until a simulated tree that is epsilon close to the data tree is found. The amount of feedback for generating clonal expansions is specified with the function parameter model\_x, and the epsilon threshold is specified with the function parameter eps. In general, the model\_x parameter for samplePosterior() always takes on the values of zero, since we are generally only interested in varying the ground truth to show how well we can infer parameter values from other models. The metric distance between the data tree and the simulated data tree is computed by calling the function

double StemCellSim::computeEpsSimAndData()

computeEpsSimAndData() returns the area between the data tree and the simulated tree which is used to check if the epsilon threshold is met. computeEpsSimAndData() does not need parameter values as input because the LTT curve of the data tree and the simulated tree are both stored internally in the StemCellSim object.

When a tree from ABC is found that is sufficiently close to the data tree, the model parameter values s, n, N, g, L corresponding to the tree from ABC (along with the trajectories of the corresponding clonal expansion, the LTT curve, and the epsilon distance) are printed to the files specified in the function parameter outputFiles, where outputFiles is a tuple and its components correspond to s, n, g, L, the trajectory of the clonal expansion, the LTT curve of the tree, and the epsilon distance between the simulated tree and the data tree respectively.

Similar to how the prior distributions for simulating data trees are easily changed by the user, the prior distributions for ABC are also easily changed in samplePosterior() in StemCellSim.cpp.

**ABC on real data**

ABC on real data is identical to carrying out ABC on simulated data trees, except that instead of using the readParamFile() function to read data, the function

void StemCellSim::readDataAndStoreLTT(std::string newickFile, double ageOfIndividual)

is used instead. The function samplePosterior() is then called in an identical manner to the previous section on ABC inference from simulated data trees. The function parameter newickFile in readDataAndStoreLTT() specifies the name of the file where the data tree is stored, and the function parameter ageOfIndividual specifies the age of the patient that corresponds to the data tree. For real data, the file containing the data tree must also have a different format to that of simulated data trees. In particular, it must contain a tree in Newick format that looks as follows:

((((MPP-39:391,HSC-10:364):9,((HSC-32:373,HSC-37:316):2,HSC-35:365):12):33,((HSC-51:353,HSC-47:345):4,HSC-46:300):34):117,(((((HSC-45:338,HSC-48:355):9,(((MPP-48:291,HSC-5:304):10,HSC-50:323):37,HSC-30:361):2):20,(((MPP-22:243,HSC-49:281):46,HSC-34:279):66,MPP-34:313):40):103,(MPP-1:480,HSC-39:488):36):11,(HSC-6:423,HSC-33:454):57):2):219;

Take note of number 219 before the ‘;’ at the very end. 219 is the number of mutations from birth of the patient to the most recent common ancestor of the sampled cells. It is very important to specify this in the tree since ABC uses the number of mutations accrued from birth of the patient to the present (the total length of the tree in mutations) to estimate the mutation rate (mutations per generation) when calling samplePosterior() by dividing the total number of mutations accrued by the L drawn from the prior in the samplePosterior() function. The function parameter ageOfIndividual must also be specified to determine the default prior on L when calling samplePosterior(), where we have assumed 1 generation per year, so that L = ageOfIndividual + round(Gaussian noise), where the Gaussian noise has mean 0 with an std of roughly 15% of the ageOfIndividual to account for uncertainty, and where round() is used because L must be an integer.

After reading in the tree in Newick format, the readDataAndStoreLTT() function performs the averaging algorithm on the tree and converts it to an LTT curve, where by LTT curve we mean the coalescence times of the tree. The LTT curve is kept internally, and samplePosterior() can then be called to infer the parameter values. The averaging algorithm in readDataAndStoreLTT() is the same one used in generateLTT() when constructing the LTT curve of a tree from a simulated clonal expansion, but the implementation is different.

**Implementation details of the averaging algorithm**

This section provides details for the implementation of our averaging algorithm described in Methods.

Our averaging algorithm is used in the generateLTT() function and in the readDataAndStoreLTT() function, and both are implemented differently. Recall that generateLTT() is used when simulating data trees, and when carrying out ABC. readDataAndStoreLTT() on the other hand is used when reading a real data tree from a text file in Newick format.

The algorithm implemented in generateLTT() begins with a sample of k cells from the final generation. Those cells choose parents from the previous generation at random, one at a time. Mutations are assigned to the edge associated with each pick so that the mutational process is implemented simultaneously with the genealogical process. For each pick, a coalescence event may or may not occur. If at any point a coalescence event occur, the length in number of mutations of the left and right subtree of the point of coalescence (the parent chosen by both cells) are immediately averaged, and the length of the subtrees are both scaled so that they equal the average. The coalescence times of the subtrees are kept in vectors, and thus we are actually scaling the coalescence times of the subtrees, rather than the branches. This means that we always ignore information about topology. When a coalescence event occurs, we therefore scale the coalescence times of the left and right subtrees so that their lengths equal the average, and then combine the coalescence times of both vectors into one vector. We therefore always begin with k vectors, and after the final coalescence event recover one vector which corresponds to the coalescence times of the averaged tree of the random sample.

The algorithm implemented in readDataAndStoreLTT() works by first reading the Newick format tree into a string. A vector of vectors called array is then declared, and an empty vector is pushed into the back. The vectors within the vector represent subtrees, and contain the coalescence times of the subtrees. The algorithm then loops over every element of the string containing the Newick tree. At each iteration, it checks the current string element. If the current string element is “:”, the algorithm pushes the numbers that come after it to the back of last vector in array. If the current string element is “,”, the algorithm pushes an empty vector to the back of the array. If the current string is “)”, the algorithm does the following: it first takes the last and penultimate vector in array, which are the coalescence times of subtrees, and averages the lengths of the subtrees so that they equal the average length between the two. This is done by scaling the coalescence times of the subtrees accordingly. After this, the last and penultimate vectors are combined to form one vector to form a newly averaged subtree. Lastly, it checks to see if the following string element containing the Newick tree is “:” followed by a number. If so, the algorithm pushes that number to the back of the last vector in array, which corresponds to the newly averaged subtree. After these 3 conditions are checked, the loop proceeds to the next iteration.

These algorithms are statistically equivalent to the pseudocode in the methods only if more than two coalescence events never occur simultaneously. For example, suppose 3 lineages coalesce simultaneously. Then averaging lineage 1 and 2, and then averaging with lineage 3 may produce different coalescence times than averaging lineage 2 and 3 first, and then averaging with lineage 1. In this case, the averaging algorithm would depend on which node it starts on. More than 2 coalescence events rarely occur, and if they did, either averaged tree would produce a good estimate of the coalescence times. We therefore believe the contribution to the statistics is negligible and can be safely ignored. Furthermore, the implementation of the algorithm on tree data is equivalent to the implementation of the algorithm on trees generated by ABC. Therefore, any contribution to the statistics from multiple coalescence events occurring simultaneously would be expressed as uncertainty in the error bars.

**Implementation details of the algorithm for computing the area between two LTT curves**

Whenever simulated or real tree data is read in and stored internally in StemCellSim, the LTT curve of the data tree is stored in the LTT\_data vector as coalescence times. Whenever a tree is generated by ABC, the LTT curve of the simulated tree is stored in the vector LTT\_sim as coalescence times. The coalescence times are in other words a representation of the LTT curves of a tree, since the vector elements from i = 1, … , k – 1 correspond to the time in number of mutations at which k, k – 1, … , 2 lineages coalesce respectively. Recall that to simplify the algorithm, the 0th element of LTT\_data and LTT\_sim both hold the number zero.

To compute the area between the LTT curves, we apply an algorithm that uses the coalescence times stored in LTT\_sim and LTT\_data. The algorithm first initializes two int variables called counterSim and counterData, where counterSim corresponds to LTT\_sim and counterData corresponds to LTT\_data, to equal 1. The algorithm then incrementally computes the area by implementing the following pseudocode:

Let k be the number of sampled lineages

Let A = 0

do{

1. Let x1 be the min of the two coalescence times LTT\_sim[counterSim] and LTT\_data[counterData], and let y1 be the number of lineages that exist immediately before x1.
2. Let x0 be the max of the two coalescence times LTT\_sim[counterSim - 1] and LTT\_data[counterData - 1], and let y0 be the number of lineages that exist immediately after x0.
3. A += (y1 – y0)\*(x1 – x0)
4. If counterSim minimized the two coalescence times in 1), then increment counterSim. Else, increment counterData.

while(A has not accrued the full are between the two LTT curves)

There is one important difference in the implementation to the pseudocode. In the implementation, we instead do A += [ (y1 – y0)\*(x1 – x0) ]/LTT\_data[LTT\_data.size() - 1]. In other words, we divide the area by the number of mutations from the present until the most recent common ancestor of the sample. Intuitively this is like finding the percent difference between the trees. The rest of the pseudocode only differs in notation and technical detail. The full implementation can be seen in StemCellSim.cpp.