CellLine, a stochastic cell lineage simulator: Manual

Andre S. Ribeiro, Daniel A. Charlebois, Jason Lloyd-Price

July 23, 2007

1 Introduction

This document explains how to work with the *CellLine* modules and the functionality provided.

*CellLine*consists of 3 modules and requires an initial reactions file. The reactions file defines the initial concentrations of all the substances and the reactions that can occur inside a single "cell".

Given an initial cell, defined in a file containing the list of possible reactions and initial reactants concentrations, one can use "CellLineGen" to generate a "cell line" from the initial cell. If desired, the third module "NCellsGen" can be used to generate a set of N dynamically independent cell copies of the initial cell or some cell of the lineage.

CellLineGen takes the initial reactions file, calls DynSim.exe to run its dynamics for a user defined time interval, and creates two new cells afterwards. The user also defines the number of cell generations desired.

The two new cells will have the same set of reactions as the "mother" cell. Each new cell's initial concentrations are equal to the concentrations of the same elements in the mother cell, at the moment of division.

Importantly, elements of the waitlist still not released in the system will also be passed on to the two daughter cells. This is critical since, for example, transcription and translation involve time delays, and the promoters could be in the waitlist at the moment of division.

Once their reactions files are generated, the two daughter cells' dynamics are then simulated by DynSim.exe. The cells' life time and sampling interval are the same as the mother cell. When the dynamics simulation is finished, DynSim.exe outputs the two cells' time series and divides each of them in two more daughter cells and so on.

CellLineGen provides each cell with a unique "random seed" and appends it to each cell's reactions files.

NCellsGen takes the user defined reactions file and makes N copies of it. The cell copies will have the same set of reactions but can be as distinct as desired on every other aspect such as concentrations, rate constants, and delays.

For example, the same reactions in different cells can have different rate constants, for one or more reactions. Initial chemical populations and time delays can also differ.

Each reactions file has a different random seed, thus the dynamics of each cell copy will be unique. The seed is appended to each cell's reactions file, so that experiments can be repeated when desired. The results will always be the same as long as no other random variables are defined (such as random rate constants).

After making N cell copies and creating an independent reactions file for each cell, NCellsGen calls DynSim.exe, which simulates the dynamics for a user defined time interval and outputs the resulting times series of each cell in independent files.

The dynamics resulting from the reactions files generated by NCellsGen or CellLineGen is simulated by DynSim.exe, which is an extension of a previously existing simulator, *SGNSim* [4].

This simulator's dynamics is driven by the "delayed SSA" [5], which consists of the original SSA [2, 1], but allows multiple time delayed reactions to be modelled. *CellLine* uses the Mersenne Twister pseudo-random number generator [3]. This generator has a period of $2^{19937} - 1$.

We now describe each of the program components.

2 Cell line generator module

CellLineGen generates a cell lineage from an initial reactions file provided by the user where the “mother” cell’s system of chemical reactions is defined. It then runs its dynamics for a user-defined amount of time and outputs the system state at a specified sampling interval.

When it completes this first step, the program takes a “snap-shot” of the mother cell’s state at the time of division (the concentrations of all chemical elements including those in the waitlist) and it creates two new daughter cells. The daughter cells inherit the set of reactions and the concentrations of the chemical species of the mother cell at the moment of division.

The reactions file of each cell in the lineage is named “Gener_*x*_cell_*y*.g”, where x is the generation number and y is the cell number within that generation. The mother cell’s reactions file (a copy of the reactions file provided by the user) is named “Gener_01_cell_01.g”.

After simulating the dynamics of any cell of the lineage, the program outputs a results file. This file, named “Gener_*x*_cell_*y*_out.xls”, contains the time series of each of the chemical species involved. Additionally, the program also outputs a “snapshot” file, called “Gener_*x*_cell_*y*_.snap.g” with each cell’s final state.

CellLineGen.exe can be called by invoking the following command:

`cellline` <*filename*> *[number of cell generations] [simulation time] [sampling interval]*

Filename is the reactions filename provided by the user. *Number of cell generations* is an integer specifying the number of generations spawned from the original cell. *Simulation time* is a positive real variable specifying the run time of each cell dynamics. *Sampling interval* is a positive real variable specifying the time series sampling interval.

Number of cell generations, *Simulation time*, and *Sampling interval* are all default to 1, when not provided.

CellLineGen appends each cell’s random seed to the cell’s reactions file and therefore all cells will differ in their dynamics.

3 NCellsGen generator module

Once the reactions file is constructed (either by the user or resulting from a cell lineage simulation), NCellsGen can be invoked to produce a number of independent cell copies that will be simulated for a specific length of time. This is done by executing the following command:

`NCellsGen` <*filename*> *[number of cells] [simulation time] [sampling interval]*

Filename is the reactions filename. *Number of cells* is an integer variable that specifies the number of independent cell copies of the original cell to simulate (from 1 to N). *Simulation time* is a real positive variable specifying the simulation time of the dynamics of each cell (from 0 to infinite). *Sampling interval* is a real positive variable specifying the sampling interval of the cell’s dynamics.

Number of cells, *Simulation time*, and *Sampling interval* are all default to 1, when not provided.

After the command line is correctly invoked, NCellsGen generates a single reactions file for each cell and names them “cell_*[cell index]*.g”. After running each cell’s dynamics, an output file of each cell is created that contains the time series of all species and is named “out_*[cell index]*.xls”.

Since each cell has a different random seed, their dynamics will differ. The seed is appended to each cell’s reactions file, so that experiments can be repeated when desired. Yet, one may which to vary some parameter from cell to cell, like a rate constant, an element’s initial concentration, or a time delay value. How to do this is explained in the following sections describing how to setup a reactions file.

Fourier spectrums of time series can only be obtained for single cells (i.e. one cell at a time).

For example, to calculate the fourier spectrum of the time series of cell x, open the reactions file from which cell x is created and introduce a new line, before the main lua block, with the following:

`fourier_file` *fourier_filename*;

Then run NCellsGen, using the reactions file as input and specifying that only a single cell is to be simulated with the following command:

```
Ncells <cell filename> 1 [simulation time] [sampling interval]
```

The program will output a results file "out_01.xls" with the time series of the system, *simulation time* seconds long, sampled every *sampling interval* seconds, and a second file, named *fourier_filename*, with the fourier spectrum of the time series of "out_01.xls". Both files are tab delimited text files.

4 Delayed Stochastic Simulation Algorithm

CellLine dynamics module, DynSim, is driven by the delayed SSA [5]. Unlike the original SSA [2], the delayed SSA stores delayed output events on a waitlist, handling multiple delayed output events for each input event. The waitlist contains the list of elements to be released some time delay after the reaction that produced them occurred (this delay is also stored for each element on the waitlist). The algorithm proceeds as follows [5]:

1) Set $t \leftarrow 0$, $t_{stop} \leftarrow$ stop time, read initial number of molecules and reactions, create empty waiting list L for delayed generating events.

2) Do an SSA step for reacting events to get the next reacting event R_1 and corresponding occurrence time $t + t_1$.

3) Compare $t + t_1$ with t_{min}, the least time in L. If $t + t_1 < t_{min}$ or L is empty, set $t \leftarrow t + t_1$. Update number of molecules by performing R_1, adding any delayed products to L with a time of $t + t_1 + \tau$, where τ is drawn from an appropriate distribution.

4) If L is not empty and if $t + t_1 \geq t_{min}$, set $t \leftarrow t_{min}$. Update number of molecules and L by releasing the first element in L.

5) If $t < t_{stop}$, go to step 2, else stop.

5 Input Reactions file

The initial reactions file must always be provided to the simulator. It consists of a list of the initial quantities of chemical species and the set of reactions (including rate constants and time delays) that each cell consists of. In the following sections we describe the syntax of the reactions file.

If the initial concentration of some substance is not defined, the simulator assumes it is zero at t = 0 s.

In general, the initial cell can contain any set of reactions. For example, it could consist of a toggle switch model, a repressilator, or the P53-Mdm2 feedback loop chemical system.

To input a reactions system into the simulator the reactions have to be listed in a file. The reaction system is specified by a series of *identifier-data* pairs. The *identifier* determines how to interpret the *data*. The next sections of this document defines the set of identifiers and the format of their data.

We provide several reactions files. These should be kept intact and any changes should be made in a new file.

5.1 Format

Information specified in a file follows the following formats:

1. *identifier* *data*`;`
2. *identifier* `{ (`*data*`;)* }`
3. *identifier* `!{` *data* `}!`

The first format is used to give simple information such as the initial random seed and the filename of the file to which the fourier spectrum should be output.

The second format is used to input a block of similar data and is equivalent to repeating the first form many times. Lists of reactions and initial populations of molecules can then be added quickly. The following example demonstrates how a set of initial population declarations (see section 5.5) for a Toggle Switch system between genes "A" and "B" can be condensed into a single block:

Format 1	Format 2

`population A = 0;`
`population B = 0;`
`population ProA = 1;`
`population ProB = 1;`
	`population {`
	`     A = 0;`
	`     B = 0;`
	`     ProA = 1;`
	`     ProB = 1;`
	`}`

The third and final format is designed to allow blocks of data which may contain semicolons or braces inside them to be input. This is primarily useful for writing a chunk of Lua code (see section 6).

5.2 Comments

Comments can be placed in an input file to allow inline documentation of the reaction system. Comments follow the C++ comment format, where `//` can be used to comment a line, and `/*` ⋯ `*/` can be used to comment a block of text.

5.3 Species Names

Species names follow the C identifier rules. They must begin with a letter or an underscore, followed by a string of letters, numbers or underscores. That is, they must satisfy the regular expression `[a-zA-Z_][a-zA-Z0-9_]*`.

5.4 Multimers

Often, molecules will bind to one another to form complexes. Names of complexes can be formed by concatenating the names of the elementary species with '`.`'. The complexes are treated internally as a completely separate species. That is, `A`, `B` and `A.B` are completely separate species and will be reported as such, rather than have `A.B` contribute to the count of the number of `A`s and `B`s.

5.5 Initial Populations

The initial populations of species are given by the `population` identifier. Populations are usually specified as '*speciesname* = *population*'. Possible uses of the `population` identifier are illustrated below:

Data	Description
`A = 5`	Set the initial concentration of A to 5.
`A += 3`	Add 3 to the previously declared initial population of A.
`A -= 2`	Subtract 2 from the previously declared initial population of A. Initial populations cannot be negative.
`A`	Equivalent to `A += 0`.

If a species name is encountered that has not yet been seen before in the input, its initial concentration is initialized to 0. This applies to species in reactions and the waiting list as well as the `+=` and `-=` forms of `population`.

By default, all species are output in the readout file. This can produce a lot of data that may not be necessary. Readout of a species can be explicitly enabled or disabled by placing a `!` or `#` before the species name, respectively.

Readout of newly-discovered species, from any identifier that adds molecular species (`population`, `reaction` and `queue`) can be enabled or disabled by using the `molecule_readout` identifier with data `show` or `hide`.

When using the NCellsGen module, the initial populations can vary among cells by adding a call to Lua's math.random here. For example, to initialize some substance X between 10 and 30, from a uniform distribution, write:

```
population X = math.random (10,30);
```

5.6 Reactions

Reactions are input with the `reaction` identifier in an intuitive and human-readable format:

substrate-list `--[`*rate-constant*`]-->` *product-list*

For example, the reaction $A + B \rightarrow C$ with a rate constant of 2 can be input as:

```
A + B --[2]--> C
```

Zero-order reactions (no substrates) and decay reactions (no products) can be input by simply omitting the *substrate-list* or the *product-list*. For example:

```
--[2]--> A
A --[2]-->
```

If more than one of a substrate is consumed in the reaction, such as the reaction $2Y \rightarrow Z$, then the number should be prepended to the substrate name, such as:

```
2Y --[2]--> Z
```

Additionally, the reaction rate is Lua-enabled (see section 6), allowing any formula that can be interpreted by Lua to be inserted, such as:

```
A + B --[math.sqrt(5) + 2 + math.random(4)]--> C
```

This could be used to adjust the reactions' rate constants to variations in the system properties (e.g. volume or temperature). This can also be used to introduce variation in the properties of the cells created by NCellsGen.

Lua can also be inserted to calculate the number of reactants consumed or products produced by a reaction by placing the Lua code in square brackets before the species name. This is mostly useful to set these values in global parameters. For example, the reaction

```
A + B --[c_react]--> [n]C
```

allows easy manipulation of both the reaction's rate constant and number of C's produced from a central Lua block.

5.7 Delayed Reactions

By default, products of a reaction are released immediately. However, in several cases it has been shown that the reaction is better modelled by delaying the release by a certain amount of time. The delay for a given product is specified by placing the delay time in parentheses after the product. This can be used to efficiently simulate multi-step reactions without including all intermediate reactions. For example, the time it takes for a gene to be transcribed by an RNA Polymerase, spliced, translated by a Ribosome, and folded can be simplified into:

```
Promoter + RNAp --[k]--> Promoter(2) + RNAp(20) + P(2000)
```

In this example, the time it takes for the `RNAp` to transcribe the entire gene and be available to bind to another gene is 20 seconds, while it only takes 2 seconds for the gene's `Promoter` to be released so other `RNAp`s can bind and begin transcribing. The end product of the reaction is a protein and is released long after the `RNAp` finishes transcribing the gene to account for the time it needs to be translated and folded.

If nothing but a number is specified as the delay, the time delay will be assumed to be constant. However, many delays are modelled best when sampled from a distribution, especially when modeling multi-step reactions. For example, when transcribing a gene, RNA Polymerase molecules do not always move at a constant speed across the gene, resulting in a distribution of time delays for the release of the proteins. In this case, the time delay for a given product can be generated from several available distributions. The distribution is specified in the form `(`*distribution*`:`*parameters*`)`. The distribution can be any of the following:

Name	Parameters	Description
`delta`, `const`	d	A constant delay of d
`gaussian`, `gaus`, `normal`	μ, σ	A normal distribution with mean μ and standard deviation σ
`exponential`, `exp`	λ	An exponential distribution with a mean of $1/\lambda$

For example, the reaction $A+B \rightarrow C$ where C's release delay follows a Gaussian distribution with mean 10 and standard deviation 2 would be input as:

```
A + B --[k]--> C(gaussian:10,2)
```

Additionally, distributions that allow negative delays to be generated (such as the `gaussian` distribution) will have their negative part truncated, so that negative delays are impossible.

Since the time delay parameters are Lua enabled, they can, similar to rates constants, be defined by global variables as well as vary from cell to cell.

5.8 Virtual Substrates and Reaction Rates

Sometimes, especially when building a reaction system from a system of ODEs, the propensities of the reactions must vary by some non-linear function of the populations of some other species. For example, when the production of one species is suppressed by another.

CellLine like its predecessor, *SGNSim* calculates a reaction's propensity from the following formula:

$$a_i = c_i \prod_{s \in S_i} r_s([s]) \tag{1}$$

Where c_i is the reaction's stochastic rate constant, S_i is the set of substrates of the reaction, and r_s is the substrate's rate function (described below). That is, the propensity of the reaction is the product of the reaction's constant rate and all the rates calculated from the substrates' rate functions.

A rate function can be assigned to a substrate in a manner similar to assigning a delay distribution to a product. It is specified in the form `(`*function*`:`*parameters*`)`. Unlike delay distributions, there is no default rate function if no *function* is given. If a parameter is omitted, it is assumed to be `1`. Available rate functions (where a and b are the parameters, and X is the population of the substrate) are:

Name	Function
`gilh`	$\prod_{j=1}^{a} \frac{X-j+1}{j}$
`const`	a
`linear`	aX
`square`, `sqr`	aX^2
`cube`	aX^3
`pow`	bX^a
`hill`	$\frac{X^b}{a^b+X^b}$
`invhill`	$\frac{a^b}{a^b+X^b}$
`max`	$b \cdot max(a, X)$
`min`	$b \cdot min(a, X)$

If no special rate function is given, substrates are assumed to have the `gilh` rate where a is the number of molecules of the substrate that are consumed. `gilh` corresponds to the calculation of H given on p. 5 of [1].

Sometimes, it is necessary to not have any molecules of a given substrate consumed by a reaction. In this case, the substrate becomes a catalyst or inhibitor of the reaction without itself being affected by it, and becomes a 'virtual' substrate. This scenario can be input either by specifying 0 as the number of molecules consumed, or by placing a `*` before the species name. Virtual substrates that are not explicitly given a rate are assumed to have the `linear` rate function. For example, the ODEs representing a toggle switch with cooperative binding,

$$\frac{d[A]}{dt} = \frac{k_p}{1+[B]^c} - k_d[A] \tag{2}$$

$$\frac{d[B]}{dt} = \frac{k_p}{1+[A]^c} - k_d[B] \tag{3}$$

can be represented with the following reactions:

```
*B(invhill:1,c) --[k_p]--> A
*A(invhill:1,c) --[k_p]--> B
A --[k_d]-->
B --[k_d]-->
```

While the ability to override the usual rate functions greatly enhances the flexibility of the simulator, it is necessary to use them with caution so that populations do not become negative. In this case, the simulation behavior is undefined.

6 Lua

Most numeric fields allow for Lua code to be inserted. Lua code encountered where a number is expected is compiled as though "`return`" was prepended to it. All Lua code is compiled and run immediately when it is encountered, in the order it appears.

A block of generic Lua code can be executed using the `lua` identifier. It is recommended that the data block be delimited with `!{ ··· }!` so that semicolons and braces can be used inside the code block without having the block split into several chunks and compiled separately.

A global function "`parse`" is also provided to feed a string from Lua into the parsing system. The function takes one or two parameters. When two parameters are given, the first is interpreted as a string containing the *identifier*, and the second is the *data*. A single string passed to the function will be parsed as though it was from a file.

Documentation for the Lua 5.1 language can be found at
http://www.lua.org/manual/5.1/.

6.1 Generating random numbers from a Poisson distribution

We have included a random number generator that follows a Poisson distribution, that is implemented in a file called "poisson.lua" (provided by us). It can be called at runtime from within a Lua block.

Here, we use this to create DNA double strand breaks. To do this, first poisson.lua file must be in the same directory as the reactions file, and it must be included by executing `require ("poisson");`.

Poisson.lua uses `math.random` to produce uniform random numbers. Therefore `math.randomseed` should be used to seed this generator.

Poisson.lua defines a single function called `poisson`. This function takes a single parameter: λ, the desired mean of the Poisson distribution.

7 Output

The simulation will produce an output file that is tab-delimited spreadsheet in text format. Each row corresponds to a sampling of the system at a given time and contains the sample time (*Time*), number of times the system time was updated (*Steps*), number of reactions completed (*Num Reactions*), number of elements in the waiting list (*WaitList Length*) and the populations of all the molecular species at that time. Species are output in the order that they are discovered in the input. The following columns will invariantly appear in the output file (X_i represent the various species):

Time, Steps, Num Reactions, WaitList Length, X_1, ..., X_n

The fourier spectrum of the resulting time series can also be obtained as mentioned above. The "Fourier.xls" is a tab delimited text file, which can be viewed in EXCEL.

8 Runtime Complexity

Module "DynSim.exe" algorithm's best-case runtime is $\Omega(R \cdot logR)$ and worst-case runtime is $O(R^2)$, where R is the number of reactions in the system. The worst case occurs when the most commonly used substrate is found in $O(R)$ reactions.

Depending on the number of cells one desires to model with NCellsGen, this time grows linearly with N. CellLineGen grows exponentially with the number of generations.

References

[1] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*

[2] D.T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comput. Phys.*, 22:403–434, 1976.

[3] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[4] Andre S. Ribeiro and Jason Lloyd-Price. Sgn sim, a stochastic gene networks simulator. *Bioinformatics*, 10.1093/bioinformatics/btm004, 2007.

[5] M.R. Roussel and Rui Zhu. Validation of an algorithm for delay stochastic simulation of transcription and translation in prokaryotic gene expression. *Phys Biol.*, 3(4):274–84, 2006.