Recycling random numbers in the stochastic simulation algorithm

Christian A. Yates and Guido Klingbeil

Citation: The Journal of Chemical Physics **138**, 094103 (2013); doi: 10.1063/1.4792207
View online: http://dx.doi.org/10.1063/1.4792207
View Table of Contents: http://scitation.aip.org/content/aip/journal/jcp/138/9?ver=pdfcov
Published by the AIP Publishing

Recycling random numbers in the stochastic simulation algorithm

Christian A. Yates[a),b)] and Guido Klingbeil[b),c)]
Center for Mathematical Biology, Mathematical Institute, University of Oxford, 24-29 St Giles', Oxford OX1 3LB, United Kingdom

(Received 8 November 2012; accepted 31 January 2013; published online 1 March 2013)

The stochastic simulation algorithm (SSA) was introduced by Gillespie and in a different form by Kurtz. Since its original formulation there have been several attempts at improving the efficiency and hence the speed of the algorithm. We briefly discuss some of these methods before outlining our own simple improvement, the recycling direct method (RDM), and demonstrating that it is capable of increasing the speed of most stochastic simulations. The RDM involves the statistically acceptable recycling of random numbers in order to reduce the computational cost associated with their generation and is compatible with several of the pre-existing improvements on the original SSA. Our improvement is also sufficiently simple (one additional line of code) that we hope will be adopted by both trained mathematical modelers and experimentalists wishing to simulate their model systems.
 [http://dx.doi.org/10.1063/1.4792207]

I. INTRODUCTION

Stochasticity is an inherent feature of many biological and chemical systems. Recently, in part due to the emergence of the field of systems biology, the modeling of stochastic processes in biology has become increasingly important. When simulating such processes it is common to use the stochastic simulation algorithm (SSA), which was first introduced by Gillespie[1,2] and in a different form by Kurtz.[3] Although, in a biological context, the SSA is not always used to characterize the interactions between chemical species (see Refs. 4 and 5 for example), for consistency with previous works, we will use the terminology of a chemical system. Interacting groups of individuals will be referred to as "species" and their interactions will be known as "reactions." In general we assume that there are $N \geq 1$ species, $\{S_1, \ldots, S_N\}$, interacting stochastically through M reaction channels $\{R_1, \ldots, R_M\}$. The state of the system at time t will be denoted by the vector $\boldsymbol{X}(t) = (X_1(t), \ldots, X_N(t))$, where $X_i(t)$ represents the number of molecules of S_i at time t. The purpose of the SSA is to simulate the time evolution of $\boldsymbol{X}(t)$ given some initial condition $\boldsymbol{X}(t_0) = \boldsymbol{x_0}$. The dynamics of each reaction channel, R_j (for $j = 1, \ldots, M$), are completely characterized by a propensity function, $a_j(\boldsymbol{x})$, which determines the probability of each reaction being the next to fire and a stoichiometric vector, $\boldsymbol{\nu}_j = (\nu_{1j}, \ldots, \nu_{Nj})$ which characterizes the change in state brought about by a single firing of reaction channel j.

The use of the SSA (or refined versions thereof), originally designed to simulate simple chemical systems with statistical exactness, has now become widespread. The success of the SSA in recent years has precipitated its application to much larger systems than was originally anticipated. For example, Arkin *et al.*[6] used the SSA to simulate a stochastic version of the classic model of Shea and Ackers[7] describing the processes which control the switch from lysogenic to lytic modes of growth in a virus, lambda phage, which is now a common experimental model system. The stochastic model contains 75 reactions in 57 chemical species.[8] Kurata *et al.*[9] simulate a 61-reaction model of the heat-shock response of the bacterium *Escherichia coli*. Baker *et al.*[4] employ the SSA in order to model cell migration via a position jump model. Even in their most basic one-dimensional models, species numbers can be of the order of hundreds with the number of reaction channels of a similar order.

For such large systems of reactions it is necessary to make the SSA as efficient as possible in order to maintain feasible simulation times. In some cases it may be necessary to sacrifice the accuracy of the exact SSA in order to accelerate simulations as is done in the "τ-leaping" algorithm.[10–12] However, it is the improvement of the efficiency of the exact SSA on which we focus in this paper.

II. BACKGROUND

One of the original (and most popular) implementations of the mathematically exact SSA is known as the direct method[2] (DM). Given a system at time t in state $\boldsymbol{X}(t)$, a time interval τ, until the next reaction occurs, is generated and along with it a reaction, with index j, is chosen to occur at time $t + \tau$. The changes in the numbers of molecules caused by the firing of reaction channel R_j are implemented, the propensity functions are altered accordingly and the time is updated, ready for the next (τ, j) pair to be selected. A method for the implementation of this algorithm is given below:

1. Initialize the time $t = t_0$ and the species numbers $\boldsymbol{X}(t_0) = \boldsymbol{x}_0$.
2. Evaluate the propensity functions, $a_j(\boldsymbol{x})$, associated with the reaction channels R_j $(j = 1, \ldots, M)$ and their sum $a_0(\boldsymbol{x}) = \sum_{j=1}^{M} a_j(\boldsymbol{x})$.
3. Generate two random numbers $rand_1$ and $rand_2$ uniformly distributed in (0, 1).

[a)]yatesc@maths.ox.ac.uk. URL: http://people.maths.ox.ac.uk/yatesc.
[b)]C. A. Yates and G. Klingbeil contributed equally to this work.
[c)]klingbeil@maths.ox.ac.uk. URL: http://people.maths.ox.ac.uk/klingbeil.

4. Use $rand_1$ to generate a time increment, τ, an exponentially distributed random variable with mean $1/a_0(\boldsymbol{x})$, i.e.,
$$\tau = \frac{1}{a_0} \ln\left(\frac{1}{rand_1}\right).$$
5. Use $rand_2$ to generate a reaction index j with probability $a_j(\boldsymbol{x})/a_0(\boldsymbol{x})$, in proportion with its propensity function, i.e., find j such that[13]
$$\sum_{j'=1}^{j-1} a_{j'}(\boldsymbol{x}) < a_0(\boldsymbol{x}) \cdot rand_2 < \sum_{j'=1}^{j} a_{j'}(\boldsymbol{x}).$$
6. Update the state vector, $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{\nu}_j$ and the time, $t = t + \tau$.
7. If $t < t_{final}$, the desired stopping time, then go to step (2). Otherwise stop.

In an early alternative version of the SSA, the First Reaction Method (FRM), Gillespie[1] proposes to generate a putative time for each reaction. The reaction whose time is first implemented and new putative reaction times are calculated using a freshly generated random number for each reaction, exploiting the memoryless property of Poisson processes, and so the process continues.

Since these early formulations (DM and FRM) there have been several attempts aimed at increasing the speed and efficiency of the SSA: Gibson and Bruck[8] adapted the FRM into their Next Reaction Method (NRM) which reuses the random numbers, originally generated for the reactions whose propensity functions had changed, in order to calculate new putative reaction times, thus saving computational effort on random number generation. They stored the putative reaction times in an "indexed priority queue," using a heap data structure in which the time and index of the next reaction to occur are always easily accessible at the root of the heap. The number of operations required in order to maintain the indexed priority queue at each iteration is $O(\log(M))$. They also increased the efficiency of the algorithm by introducing a dependency graph which lists the propensity functions that depend on the outcome of each reaction, enabling them to identify and alter only those propensity functions which require updating. Such a dependency graph is now implemented as standard in most efficient versions of the SSA. As well as increasing the speed of the SSA, the NRM reformulation is important conceptually. Adaptations of the NRM have been used in order to deal with propensity functions which change during the course of a reaction (due to volume or temperature fluctuations) and to incorporate time delays into the SSA.[14] The concepts underlying the NRM have also been incorporated into an algorithm for efficiently simulating spatially extended systems in the Next Sub-volume Method (NSM).[5]

In response to the NRM, Cao *et al.*[15] introduced the Optimized Direct Method (ODM) an adaptation of Gillespie's original DM with two key alterations. The first borrows the dependency graph of Gibson and Bruck,[8] updating only those reactions whose propensity functions change. The second involves structuring the order of propensity functions in the search list so that reactions occurring more frequently are higher up, reducing the search depth of the linear search. In order to initialize the search list appropriately, Cao *et al.*[15] introduced a pre-simulation step whereby a small number of simulations are run in order to ascertain the relative frequencies of each reaction. The justification for these pre-simulations is based on the premise that, in order to find reliable statistics for the evolution of the species involved in a reaction system, a large number of repeat simulations must be run. Hence a small number of simulations used to initialize the reaction system will account for a negligible proportion of the total time for all repeats. The ODM is accepted as being of comparable efficiency to the NRM in a wide variety of situations,[16] but is expected to outperform the NRM in systems of reactions that are tightly coupled and therefore require a large amount of computational effort to maintain the indexed priority queue of the NRM.[15]

Inspired by the idea of sorting propensity functions, McCollum *et al.*[17] introduced the Sorting Direct Method (SDM). Using a simple bubble-sort-type algorithm the SDM accounts for possible transient changes in reaction frequency by dynamically changing the reaction selection order. This means that reactions which occur more often, at a specific part of the simulation, tend to find themselves higher in the search order, thus reducing the average search depth. The SDM also benefits from not having to implement pre-simulations.

Both the ODM and the SDM rely on reordering the propensity functions in order to reduce search depth. Li and Petzold[18] introduced the logarithmic direct method (LDM) which uses a binary search to determine which reaction is due to fire next. The search effort is therefore independent of the ordering of the propensity functions and depends only on their number. The LDM has shown to be of comparable speed to other efficient versions of the SSA.

An alternative method for the reduction of simulation times in situations which require multiple repeated runs of the SSA is to run these repeats in parallel. With the advent of massively parallel computer architectures such as those associated with graphical processing units (GPUs) it is becoming feasible to run hundreds of instances of the same simulation at the same time. However, there are access barriers to these methods since the implementation of stochastic models on these architectures is far from trivial.[19]

In this paper we work with the premise of Gibson and Bruck[8] that random number generation is expensive and aim to reduce the number of random numbers employed per iteration to one by the statistically acceptable recycling of random numbers. At the same time we hope to provide a method that is a sufficiently straightforward adaptation of the DM that it can be implemented by experimental scientists interested in computational modeling. Our Recycling Direct Method (RDM) is compatible with, and therefore able to build on the efficiency of, the ODM, SDM, and LDM.

III. RECYCLING RANDOM NUMBERS

Our idea is simple but important, as it has the potential to accelerate all stochastic simulations implemented using DM-derived SSAs. We replace steps (3)–(5) of the DM algorithm (see pages 1 and 2) as follows:

3. Generate a random number $rand_1$ uniformly distributed in (0, 1).
4. Use $rand_1$ to generate a reaction index j with probability $a_j(\boldsymbol{x})/a_0(\boldsymbol{x})$, in proportion with its propensity function, i.e., find j such that

$$\sum_{j'=1}^{j-1} a_{j'}(\boldsymbol{x}) < a_0(\boldsymbol{x}) \cdot rand_1 < \sum_{j'=1}^{j} a_{j'}(\boldsymbol{x}).$$

5. Renormalize the random number so that it lies uniformly in (0, 1):

$$rand_2 = \frac{a_0(\boldsymbol{x}) \cdot rand_1 - \sum_{j'=1}^{j-1} a_{j'}(\boldsymbol{x})}{a_j}. \quad (1)$$

6. Use the renormalized random number, $rand_2$, to generate a time increment, τ, from the exponential distribution with mean $1/a_0(\boldsymbol{x})$, i.e.,

$$\tau = \frac{1}{a_0} \ln\left(\frac{1}{rand_2}\right).$$

In order to make the most efficient use of the random numbers generated in the DM we alter the order of the "time-step generation" and the "reaction choice" (steps 4 and 5, respectively, of the original algorithm[20] of page 3). We also incorporate an extra step (step 5 in our revised algorithm) between these. In this step we use the fact that the scaled random number, $a_0 \cdot rand_1$, is uniformly distributed in $[\sum_{j'=1}^{j-1} a_{j'}(\boldsymbol{x}), \sum_{j'=1}^{j} a_{j'}(\boldsymbol{x})]$ to renormalize this random number by the size of this interval (i.e. the size of the propensity function of the chosen reaction, a_j) so that we generate another random number that lies uniformly in (0, 1). We then use this renormalized random number in step 6 in order to generate a new time-step. Because we are able to recycle the random number it is now only necessary to generate one random number per iteration (in step 3). This saves computational effort and can speed up the SSA as demonstrated in Sec. IV.

It can be shown that $rand_2$ is uniformly distributed, independent of the interval into which $rand_1$ falls and as such there are no issues with the independence of the "reaction choice" and "time-step generation." There may, however, be an issue with the accuracy of the time-step generated on rare occasions. If the magnitude of the propensity function chosen is small (i.e., approaching machine precision) then the resulting renormalized random number will have very few digits. For example, if our random numbers are generated initially with k digits of accuracy and the smallest interval is of size 10^{-j} (where $0 < j < k$) then the resulting random number we will use in order to generate the time-step will only have $k - j$ digits of accuracy. This will lead to inaccuracy in time-step, τ, when $j \sim k$.

If it is the case that such small propensity functions account for a large proportion of the total propensity sum, a_0, then it may be argued that the SSA is not the most appropriate strategy for simulating such a reaction system. Alternatively if such small propensity functions make up only a small proportion of the total propensity sum then we can argue that they will not be chosen sufficiently often to have a significant effect on the accuracy of the time-step. In any case, systems with propensity functions which approach machine precision are rarely encountered. We note that this consideration is similar to that of the SDM and the ODM in which reactions with small propensity functions may not fire at all.

In Sec. IV we demonstrate that the random numbers generated by the RDM pass all the stringent tests of randomness dictated by the battery tests of the `TestU01` test suite.[21] We also quantify the increased efficiency of our simulations in comparison to the original DM.

IV. NUMERICAL RESULTS

When carrying out our numerical simulations we are interested in demonstrating two properties of the RDM: its accuracy and the gains in computational efficiency it can provide.

To demonstrate accuracy we test the RDM in two rigorous ways in order to ascertain: (i) that the recycled random numbers are sufficiently random, (ii) that at a functional level, employing recycled random numbers in the stochastic simulation software yields the expected results.

A. Testing for the quality of randomness

The RDM requires the transformation of the uniform random numbers that are employed initially to determine the reaction which fires at each time-step. In order to test the quality of randomness of these recycled uniform random numbers we generate a stream of such recycled random numbers by setting-up a simple isomerisation reaction system consisting of N species and N reactions:

$$\begin{aligned} X_1 &\xrightarrow{c_1} X_2, \\ X_2 &\xrightarrow{c_2} X_3, \\ &\vdots \\ X_N &\xrightarrow{c_N} X_1, \end{aligned} \quad (2)$$

where the reaction rates are chosen to be $c_i = 1$ for $i = 1, \ldots, N$. This reaction systems shuttles molecules between the reservoirs of the N species indefinitely producing a recycled uniform random number in each iteration. We choose $N = 1\,000\,000$ in order to have a small value for the size of the average normalised propensity functions whilst maintaining a feasible simulation time. The initial molecular populations are $X_i = 1\,000\,000$ for $i = 1, \ldots, N$. The pseudocode outlining our transformation scheme is given in Scheme 1. While all our computations are implemented using double precision (64-bit) floating point arithmetic, we run our tests with 32-bit uniform random numbers since, if the random numbers recycled from the original 32-bit random numbers pass the rigorous tests for randomness, then random numbers with more bits (e.g., 64 bits) will also pass the tests.

The stream of recycled uniform random numbers generated by reaction system (2) is then fed into the `bigCrush` battery test suites of the `TestU01` test suite.[21] `TestU01` implements empirical statistical tests of uniform random number generators (RNGs) on the interval (0, 1) or bit sequences

```
1  Let rand_1 = 0, p_0 = 0, p_N rand_1 = 0;
2  Let c_1 = ... = c_N = 1, X_1 = ... = X_N = 1000000;
3  // compute the propensity prefix sums
4  for i = 1 : N
5      p_i = p_{i-1} + c_i X_i;
6  end
7
8  rand_1 =unif();
9  p_N rand_1 = p_N · rand_1;
10
11 // find the index j of the next reaction
12 // which fires using a binary search
13 j = binarySearch();
14
15 // recycle p_N rand_1 to generate rand_2
16 rand_2 = (p_N rand_1 − p_{j-1})/(p_j − p_{j-1}); //extra line of code
17
18 // update the molecular populations
19 if(j = N){
20     X_N = X_N − 1; X_1 = X_1 + 1;
21 } else {
22     X_j = X_j − 1; X_{j+1} = X_{j+1} + 1;
23 }
```

SCHEME 1. Pseudocode of the transformation scheme used to test for the quality of the recycled uniform random numbers, $rand_2$. The code implements a stochastic simulation for the reaction system (2). We use a binary search,[22] as introduced by Li and Petzold[18] for the LDM, in order to find the index of the next reaction which fires, j. For simplicity we assume the binary search is encapsulated in the function `binarySearch` in line 13. The transformation given in Eq. (1) is performed in line 16.

by aggregating and extending well known tests from the literature.[23–26] Our recycled uniform random numbers pass all tests of randomness dictated by `bigCrush`, the most stringent battery test suite.

The SBML (Systems Biology Markup Language) discrete stochastic models test suite (DSMTS) by Evans *et al.*[27] provides test reaction systems as well as the time evolution of their molecular species to rigorously test an exact stochastic simulator. We wrote highly optimised special purpose C-code implementing a subset of the DSMTS. We limit ourselves to the test reaction system requiring a single reaction compartment and without external events which change the course of the simulation since neither is relevant to the accuracy and applicability of the our recycling method. We use the following DSMTS test reaction systems: `dsmts-001-01`, `dsmts-001-03`, `dsmts-001-04`, `dsmts-001-05`, `dsmts-001-07`, `dsmts-002-01`, `dsmts-002-02`, `dsmts-002-04`, `dsmts-002-06`, `dsmts-003-01`, `dsmts-003-02`, `dsmts-004-01`, and `dsmts-004-02`. Evans *et al.*,[27] the creators of the DSMTS test suite, advise the implementation of at least 10 000 repeat simulations of each reaction system in order to detect subtle implementation issues. For our tests we are especially cautious in computing 20 000 realisations for each test case.

The DSMTS compares the values of the mean and standard deviation given by our simulation method against the known mean, μ_t, and known standard deviation, σ_t, of the test reaction systems. For each realisation the species populations are sampled at equidistant points in time. For each time point, t, the sample mean, $\bar{X}_t$, and the sample variance, S_t^2, across all realisations are computed. Appealing to the central limit theorem, our simulator is assumed to be correct if $Z_t = \sqrt{n}(\bar{X}_t - \mu_t)/\sigma_t$ is the range $(-3, 3)$ for each t in each simulation. Similarly, when testing for the correct variance, if each $Y_t = \sqrt{n/2}(S_t^2/\sigma_t^2 - 1)$ is in the range $(-5, 5)$ then our simulation method is considered to be correct.[27]

Evans *et al.*[27] assume that a correct simulator should not fail more than 2 or 3 mean tests and 5 or 6 standard deviation tests in total (i.e., over all reaction systems and all time points). The number of failed data points for both the RDM and the original DM are given in Table I. The RDM fails fewer than the maximum number of tests allowed for a "correct" simulation method, as defined by Evans *et al.*[27] Combined with the previous result pertaining to the quality of

TABLE I. The number of data points failed by the original implementation of the DM[2] and our new RDM. The two methods fail a comparable number of tests. However, in both cases, this is fewer than the number of fails for which the simulation method would be considered to be incorrect.

	Original DM		RDM	
	Mean	Standard deviation	Mean	Standard deviation
DSMTS-001-03	0	2	0	3
DSMTS-002-06	2	0	1	0
DSMTS-004-01	0	0	1	0

our random numbers we can be assured that the RDM is an exact realisation of the SSA.

B. Testing for efficiency

To demonstrate the efficiency of the RDM we simulated three reaction systems: the three species and four reaction dimerisation decay model introduced by Gillespie,[2] the 9 species and 16 reaction circadian cycle model by Vilar *et al.*,[28] and the 11 species and 16 reaction model of the *Escherichia coli* lac operon by Wilkinson[29] (see Tables II and III). The first system is regularly used to benchmark stochastic simulation programmes[2,11] and the latter two are models of biologically important reaction systems.

Double precision (64-bit floating point arithmetic) computations are commonplace in modern computing and, as such, we implement all our computations using double precision. However, we implement two separate efficiency comparisons: one using 32-bit random numbers (the currently accepted standard) and the other using 64-bit random numbers. While our transformation scheme gives a high quality of randomness using 32-bits of randomness, in some situations 64-bits of randomness may be desirable. We use the well known Mersenne twister to generate our initial uniform random numbers.[30] All simulations have been repeated 20 000 times and all times are reported in milliseconds (per realisation). The efficiency comparisons pertaining to simulations of the three example systems using 32-bit and 64-bit random numbers are given in Tables II and III, respectively. In both cases the RDM demonstrates significant gains in speed over the original DM.

We found the benefit of recycling random numbers to be larger when using 64-bit random numbers (as might have been expected given their higher cost of generation). This indicates that recycling random numbers becomes more attractive as the cost of their generation increases.

TABLE II. Numerical results using 32-bit random numbers. All simulations have been repeated 20 000 times and the average taken. All simulation times are given in milliseconds (per realisation).

	Simulation time (ms)		
	Original DM	RDM	Benefit
Dimerisation decay	0.06191	0.05611	9.36%
Circadian cycle	316.3498	292.1804	7.64%
lac-operon	16.6434	14.7323	11.48%

TABLE III. Numerical results using 64-bits of randomness per random number. All simulations have been repeated 20 000 times and the average taken. All simulation times are given in milliseconds (per realisation).

	Simulation time (ms)		
	Original DM	RDM	Benefit
Dimerisation decay	0.08438	0.0719	14.79%
Circadian cycle	437.01778	326.6415	25.26%
lac-operon	18.9685	15.6939	17.26%

There are four main steps in each iteration of the DM algorithm: (i) compute the propensities, (ii) find the next reaction which fires, (iii) compute the time-step, and (iv) update the molecular populations. The time spent computing each of these steps, except for step (iii), depends on the size of the reaction system. The benefit of recycling random numbers over generating another sample in each iteration will, therefore, vary depending on the reaction system. Since other DM-derived SSAs (ODM, SDM, and LDM) work towards mitigating the effect of system size (reducing the cost of (ii) and thus increasing the relative cost of random number generation) we expect the RDM to yield even greater proportional efficiency savings when applied to these methods.

V. DISCUSSION

We have suggested a simple improvement to the random number generation procedure for the DM-derived SSA which allows us to reduce simulation times by up to 25% without significantly reducing the accuracy. This contribution is important since with only one additional line of computer source code the simulation of all of our example reaction systems benefited. By recycling random numbers in a statistically acceptable manner we are able to reduce the number of random numbers used per step of the SSA from two to one. Since the generation of random numbers has often been identified as a costly procedure in the SSA[8] it is unsurprising that our method yields an increase in efficiency. The RDM is versatile enough to be employed in tandem with other efficient implementations of the SSA including the ODM, SDM, and LDM. Furthermore RDM is independent of the underlying uniform RNG used. We are aware, however, that the RDM will not be applicable to the NRM or the FRM, since it relies on the separation of the steps of reaction choice and time-step generation which are implicitly coupled in both of these methods.

It is hoped that the simplicity of the RDM, combined with its potential to yield a significant reduction in simulation time for any reaction system, will lead to its widespread application amongst experimentalists and modelers alike.

ACKNOWLEDGMENTS

C.A.Y. would like to thank Christ Church, Oxford for a Junior Research Fellowship. G.K. would like to thank EPSRC and BBSRC for funding via the Systems Biology Doctoral Training Centre, University of Oxford.

[1]D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," J. Comput. Phys. **22**(4), 403–434 (1976).

[2]D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," J. Phys. Chem. **81**(25), 2340–2361 (1977).
[3]T. G. Kurtz, "The relationship between stochastic and deterministic models for chemical reactions," J. Chem. Phys. **57**(7), 2976–2978 (1972).
[4]R. E. Baker, C. A. Yates, and R. Erban, "From microscopic to macroscopic descriptions of cell migration on growing domains," Bull. Math. Biol. **72**(3), 719–762 (2010).
[5]J. Elf and M. Ehrenberg, "Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases," Syst. Biol. **1**(2), 230–236 (2004).
[6]A. Arkin, J. Ross, and H. H. McAdams, "Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected *Escherichia coli* cells," Genetics **149**(4), 1633–1648 (1998).
[7]M. A. Shea and G. K. Ackers, "The O_R control system of bacteriophage lambda. A physical-chemical model for gene regulation," J. Mol. Biol. **181**(2), 211–230 (1985).
[8]M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," J. Phys. Chem. A. **104**(9), 1876–1889 (2000).
[9]H. Kurata, H. El-Samad, T. M. Yi, M. Khammash, and J. Doyle, "Feedback regulation of the heat shock response in *E. coli*," in *Proceedings of the 40th IEEE Conference on Decision and Control (2001)* (IEEE, 2001), Vol. 1, pp. 837–842.
[10]Y. Cao, D. T. Gillespie, and L. R. Petzold, "Avoiding negative populations in explicit Poisson tau-leaping," J. Chem. Phys. **123**(5), 054104 (2005).
[11]D. T. Gillespie, "Approximate accelerated stochastic simulation of chemically reacting systems," J. Chem. Phys. **115**(4), 1716–1733 (2001).
[12]C. A. Yates and K. Burrage, "Look before you leap: A confidence-based method for selecting species criticality while avoiding negative populations in τ-leaping," J. Chem. Phys. **134**(8), 084109 (2011).
[13]Note that in step 5 of the original DM algorithm and step 4 of our revised algorithm, in the case $j = 1$ we assume $\sum_{j'=1}^{0} = 0$.
[14]D. F. Anderson, "A modified next reaction method for simulating chemical systems with time dependent propensities and delays," J. Chem. Phys. **127**(21), 214107 (2007).
[15]Y. Cao, H. Li, and L. Petzold, "Efficient formulation of the stochastic simulation algorithm for chemically reacting systems," J. Chem. Phys. **121**(9), 4059–4067 (2004).
[16]D. T. Gillespie, "Stochastic simulation of chemical kinetics," Annu. Rev. Phys. Chem. **58**(1), 35–55 (2007).
[17]J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova, "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior," Comput. Biol. Chem. **30**(1), 39–49 (2006).
[18]H. Li and L. Petzold, "Logarithmic direct method for discrete stochastic simulation of chemically reacting systems," Technical report, University of California, Santa Barbara, 2006.
[19]G. Klingbeil, R. Erban, M. Giles, and P. K. Maini, "Fat vs. thin threading approach on GPUs: Application to stochastic simulation of chemical reactions," IEEE Trans. Parallel Distrib. Syst. **23**(2), 280–287 (2012).
[20]Note that the ordering of these steps is interchangeable in the original algorithm, but will not be so in our revised method.
[21]P. L'Ecuyer and R. Simard, "TestU01: A C library for empirical testing of random number generators," ACM Trans. Math. Softw. **33**(4) (2007) (article 22).
[22]D. E. Knuth, *The Art of Computer Programming - Sorting and Searching*, 2nd ed. (Addison-Wesley, 1997), Vol. 3.
[23]D. E. Knuth, *The Art of Computer Programming - Seminumerical Algorithms*, 3rd ed. (Addison-Wesley, 1997), Vol. 2.
[24]G. Marsaglia, The Marsaglia Random Number CDROM, The Diehard Battery of Tests of Randomness, 1995.
[25]G. Marsaglia and W. W. Tsang, "Some difficult-to-pass tests of randomness," J. Stat. Software **7**(3), 1–9 (2002).
[26]A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2001.
[27]T. W. Evans, C. S. Gillespie, and D. J. Wilkinson, "The SBML discrete stochastic models test suite," Bioinformatics **24**(2), 285–286 (2008).
[28]J. M. G. Vilar, H. Y. Kueh, N. Barkai, and S. Leibler, "Mechanisms of noise-resistance in genetic oscillators," Proc. Natl. Acad. Sci. U.S.A. **99**(9), 5988–5992 (2002).
[29]D. Wilkinson, *Stochastic Modelling for Systems Biology* (Chapman & Hall/CRC, 2006).
[30]M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (1998).