**Noname manuscript No.**
(will be inserted by the editor)

# Variance Reduction with Array-RQMC for Tau-Leaping Simulation of Stochastic Biological and Chemical Reaction Networks

**Florian Puchhammer · Amal Ben Abdellah · Pierre L'Ecuyer**

June 8, 2021

**Abstract** We explore the use of Array-RQMC, a randomized quasi-Monte Carlo method designed for the simulation of Markov chains, to reduce the variance when simulating stochastic biological or chemical reaction networks with $\tau$-leaping. The task is to estimate the expectation of a function of molecule copy numbers at a given future time $T$ by the sample average over $n$ sample paths, and the goal is to reduce the variance of this sample-average estimator. We find that when the method is properly applied, variance reductions by factors in the thousands can be obtained. These factors are much larger than those observed previously by other authors who tried RQMC methods for the same examples. Array-RQMC simulates an array of realizations of the Markov chain and requires a sorting function to reorder these chains according to their states, after each step. The choice of sorting function is a key ingredient for the efficiency of the method, although in our experiments, Array-RQMC was never worse than ordinary Monte Carlo, regardless of the sorting method. The expected number of reactions of each type per step also has an impact on the efficiency gain.

**Keywords** Chemical reaction networks · stochastic biological systems · variance reduction · quasi-Monte Carlo · Array-RQMC · tau-leaping · continuous-time Markov chains · Gillespie

## 1 Introduction

We consider systems of chemical species whose molecule numbers dynamically change over time as the molecules react via a set of predefined chemical equations. The evolution of such systems is typically modeled by a *continuous-time Markov chain* (CTMC) (Gillespie, 1977;

F. Puchhammer
Basque Center for Applied Mathematics, Alameda de Mazarredo 14, 48009 Bilbao, Basque Country, Spain; and DIRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada
E-mail: fpuchhammer@bcamath.org

A. Ben Abdellah
DIRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada
E-mail: amal.ben.abdellah@umontreal.ca

P. L'Ecuyer
DIRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada
E-mail: lecuyer@iro.umontreal.ca

Anderson, 1991; Anderson and Kurtz, 2011) whose state is a vector that gives the number of copies of each species. Each transition (or jump) of the CTMC corresponds to the occurrence of one reaction, and the occurrence rate of each potential reaction (also called the *reaction propensity*) is a function of the state of the chain. The probability that any given reaction is the next one that will occur is proportional to its propensity and the time until the next reaction has an exponential distribution whose rate is the sum of these propensities. The *stochastic simulation algorithm* (SSA) of Gillespie (1977) simulates the successive transitions of this CTMC one by one, by generating the exponential time until the next reaction and determining independently which reaction it is. This method is exact (there is no bias). But when the number of molecules is large, simulating all the reactions one by one is often too slow, because their frequency is too high. One popular alternative is to approximate the CTMC by a *discrete-time Markov chain* (DTMC), as follows. Fix a time interval $\tau > 0$. Under the simplifying assumption that the rates of the different reactions do not change during the next $\tau$ units of time, the numbers of occurrences for each type of reaction are independent Poisson random variables with means that are $\tau$ times the occurrence rates (or propensities) of these reactions. Each step (or transition) of the DTMC corresponds to $\tau$ units of time for the CTMC. This DTMC can be simulated by generating a vector of independent Poisson random variables at each step, and updating the state to reflect all the reactions that occurred during this time interval. In the setting of chemical reaction networks, this approach is the *$\tau$-leaping method* of Gillespie (2001), and it is widely used in practice. This is the method we consider in this paper.

There are several other approximation methods, some of them leading to simpler and faster simulations, but the error and/or bias can also be more significant (Gillespie, 2000; Higham, 2008). One simple approach uses a fluid approximation in which the copy numbers are assumed to take real values that vary in time according to a system of deterministic differential equations called the *reaction rate equations* which can be simulated numerically (Gillespie, 2000; Higham, 2008). This type of deterministic model is the primary tool in the field of system dynamics, and it is widely used in many areas. It corresponds to chemical kinetics equations found in textbooks. But this model ignores randomness, so it cannot capture the stochastic variations observed in experiments with real systems (Beentjes and Baker, 2019). Noise can be introduced via a *stochastic differential equations* model, which amounts essentially to approximate the Poisson distribution by a normal distribution, and the denumerable-state CTMC by a continuous-state process. This leads to the *chemical Langevin equation* (Gillespie, 2000; Beentjes and Baker, 2019), which can be simulated efficiently by standard methods for stochastic differential equations (Kloeden and Platen, 1992) and may provide a reasonable approximation, typically when the number of molecules of each type is very large, but can otherwise suffer from bias.

The purpose of the stochastic simulations with $\tau$-leaping could be for example to estimate the probability distribution of the state at a given time $t$, or the probability that the state is in a given subset, or more generally the expectation of some function of the state, at time $t$. The simulations are usually done via Monte Carlo (MC) sampling, using a random number generator that provides a good imitation of independent uniform random variables over the interval $(0, 1)$ (L'Ecuyer, 2012). To estimate the expectation of a random variable such as a function of the state at a given time, standard MC uses the average over $n$ independent simulation samples. The accuracy of this estimate is usually assessed by computing a confidence interval on the true value. Since the width of the interval is proportional to the (estimated) standard deviation, which is the square root of the variance of the sample average, it is of high interest to find alternative estimators with the same expectation, similar computing costs, and a much smaller variance. With standard MC, the variance and the standard deviation of the

sample average converge as $\mathcal{O}(n^{-1})$ and $\mathcal{O}(n^{-1/2})$, respectively, which is rather slow. That is, to obtain one more significant digit of accuracy, as measured by the confidence interval, we need to multiply the number $n$ of simulations by 100. We would like to improve on this.

*Randomized quasi-Monte Carlo* (RQMC) is an alternative sampling approach which under favorable conditions can improve this convergence rate of the variance to $\mathcal{O}(n^{-\alpha+\epsilon})$ for any $\epsilon > 0$, for some constant $\alpha$ that can often reach 2, and even larger values in special situations (Owen, 1997b; L'Ecuyer and Lemieux, 2002; L'Ecuyer, 2009, 2018; L'Ecuyer et al., 2020). *Quasi-Monte Carlo* (QMC) replaces the $n$ independent vectors of uniform random numbers that drive the simulations by $n$ *deterministic* vectors with a sufficient number of coordinates to simulate the system and which cover the space (the unit hypercube) more evenly than typical independent random points (Niederreiter, 1992; Dick and Pillichshammer, 2010). RQMC randomizes these points in a way that each individual point becomes a vector of independent uniform random numbers, while at the same time the set of points as a whole retains its structure and high uniformity. A point set that satisfies these two *RQMC conditions* can provide an unbiased estimator with lower variance than Monte Carlo.

RQMC also has important limitations. Firstly, the $\mathcal{O}(n^{-\alpha+\epsilon})$ convergence rates are proved only under conditions that the integrand is a smooth function of the uniforms, whereas when simulating the CTMC considered here, the sequence of states that are visited is discontinuous in the underlying uniform random variates. Secondly, when the points are high-dimensional and some high-order interactions between the coordinates are important, the variance reduction is usually limited, and this often happens when simulating the CTMCs that model reaction networks via either direct SSA or $\tau$-leaping. Indeed, those simulations require at least one or two random numbers per step of the chain, the number of steps can be very large in real applications, so the dimension of the points, which is the total number of random numbers that are required to simulate one realization of the process, can be very large. Beentjes and Baker (2019) investigated the performance of $\tau$-leaping combined with traditional RQMC and found that the gain from RQMC compared to MC was small. They mentioned the two well-known limitations above as possible explanations for this behavior.

The *Array-RQMC* algorithm (L'Ecuyer et al., 2006, 2008, 2009) has been developed precisely to recapture the power of RQMC when simulating Markov chains over a large number of steps, as in the problem considered here. The empirical variance under Array-RQMC has been observed to converge faster than under MC in several examples from various areas, sometimes at a rate near $\mathcal{O}(n^{-2})$ empirically, even for some examples where the cost function was discontinuous (Demers et al., 2005; L'Ecuyer et al., 2007, 2008, 2009; Dion and L'Ecuyer, 2010; L'Ecuyer et al., 2018; Ben Abdellah et al., 2019). The faster convergence has also been proven theoretically under certain conditions (L'Ecuyer et al., 2008).

Our present work was motivated by Beentjes and Baker (2019) and our aim is to see how Array-RQMC can improve upon MC and classical RQMC, first for the same examples as in their paper, then for a few more elaborate cases. Hellander (2008) also experimented with Array-RQMC, in combination with uniformization of the CTMC and conditional Monte Carlo (CMC) based on the discrete-time conversion method of Fox and Glynn (1990). Their goal was to estimate the probability distribution of the state at a fixed time $t > 0$. In this setting, CMC alone provably reduces the variance. Empirically, with CMC, they obtained variance reductions by factors of about 20 in one example and 45 in another example. With the combination of CMC with Array-RQMC, they observed variance reductions by a factor of about 100 with $n = 10^5$ for both examples. Thus, Array-RQMC provides an additional gain on top of CMC, by a factor of about 2.5 to 5. (The *variance reduction factor* (VRF) for a given method is defined as the variance of the standard MC estimator divided by the variance with the given method, for the same sample size $n$.)

In this paper, we show how to obtain much larger variance-reduction factors with Array-RQMC. We do this in the same setting as Beentjes and Baker (2019), where the $\tau$-leaping method is used to estimate an expectation at a given time $t$. We find empirically that the combination of $\tau$-leaping with the Array-RQMC algorithm can bring not only a significant variance reduction, but also an improved convergence rate, compared with plain MC.

The main idea of the Array-RQMC algorithm is to simulate $n$ copies (or sample paths) of the Markov chain in parallel, in a way that the empirical distribution of the chain's states at any given step is closer to the exact theoretical distribution at that step than with ordinary MC. To achieve this, at each step, the first few coordinates of the RQMC point set are designated to match the points to the states, and the remaining coordinates are used to advance the chains by one step. This matching can be interpreted as sorting the chains in some particular order, to match the ordering of the RQMC points. In the simple case where the state is one-dimensional, it suffices to enumerate the points by increasing order of their first coordinate and sort the chains by increasing order of their state. For higher-dimensional states, one possibility is to use some kind of multivariate sort to order both the points and the states; we will describe some of these sorts in Section 4.2. Another approach is to define an *importance function*, which maps the state to a one-dimensional representative value, and sort the chains by that value. The choice of mapping can have a significant impact on the performance. If the mapping is fast to evaluate, this approach can reduce the computing time significantly, because a one-dimensional sort is usually much faster to execute than a multivariate one. To preserve the power of Array-RQMC, on the other hand, the importance function must provide a good estimate (or forecast) of the expected future value or cost, given the state at which it is evaluated. For this, it must be tailored to the problem at hand. A good tradeoff between simplicity and prediction accuracy is not always easy to achieve, but it is an important ingredient for the performance of Array-RQMC. As a proof of concept that this approach can work for reaction networks, we experiment with a very simple one-step look-ahead importance function, and we find that it works reasonably well in our examples. In our numerical experiments, this approach is often competitive with the best multivariate sorts. We do not mean that this simple heuristic choice of importance function is always good. There are probably situations where a different heuristic would be needed. Our main message is that for a wide range of examples, it is not too hard to design a reasonably simple and effective importance function. Multivariate batch sorts used in previous papers are also competitive in general and offer the best performance in some cases. We also discuss briefly how more elaborate importance functions could be defined.

The remainder is structured as follows. In Section 2 we recall the fixed-step $\tau$-leaping method for the simulation of well-mixed reaction networks in its simplest form. Section 3 gives a short review of RQMC, the main point set constructions, and the underlying theory. In Section 4, we define the Array-RQMC method and discuss some of the most prominent multivariate sorting algorithms. In Section 5, we describe the methodology used for our experiments and provide numerical results, with a discussion. A conclusion follows.

## 2 The CTMC Model and the $\tau$-Leaping Algorithm for Reaction Networks

We consider a system comprised of $\ell \geq 1$ types of chemical species $S_1, \ldots, S_\ell$ that can react via $d \geq 1$ reaction types (or channels) $R_1, \ldots, R_d$. We assume that the species are well-mixed within a volume that does not change over time and whose temperature remains constant. Each reaction $R_k$, $k = 1, \ldots, d$, can be written as

$$\alpha_{1,k} S_1 + \cdots + \alpha_{\ell,k} S_\ell \xrightarrow{c_k} \beta_{1,k} S_1 + \cdots + \beta_{\ell,k} S_\ell, \quad \alpha_{i,k}, \beta_{i,k} \in \mathbb{N}_0,$$

where $c_k > 0$ is the reaction rate constant for $R_k$. Let $\boldsymbol{X}(t) = (X_1(t), ..., X_\ell(t)) \in \mathbb{N}_0^\ell$, where $X_i(t)$ is the copy number (i.e., the number of molecules) of type $S_i$ at time $t$, for $i = 1, \ldots, \ell$ and $0 \le t \le T$. The process $\{\boldsymbol{X}(t),\ t \ge 0\}$ is modeled as a CTMC with fixed initial state $\boldsymbol{X}(0) = \boldsymbol{x}_0$ and for which each jump corresponds to the occurrence of one reaction. The jump rate (or *propensity function*) for reaction $R_k$ is a function $a_k$ of the current state; it is $a_k(\boldsymbol{x})$ when $\boldsymbol{X}(t) = \boldsymbol{x}$. This means that for a small $\delta > 0$, reaction $R_k$ occurs exactly once during the time interval $(t, t+\delta]$ with probability $a_k(\boldsymbol{x})\delta + o(\delta)$ and occurs more than once with probability $o(\delta)$. When $R_k$ occurs, the state changes from $\boldsymbol{x}$ to $\boldsymbol{x} + \boldsymbol{\zeta}_k$, where $\boldsymbol{\zeta}_k = (\beta_{1,k} - \alpha_{1,k}, \ldots, \beta_{\ell,k} - \alpha_{\ell,k})$ is the stoichiometric vector for $R_k$. The standard for $a_k(\boldsymbol{x})$, which we assume in our examples, is the *mass action kinetics* model: $a_k(\boldsymbol{x}) = c_k \hbar_k(\boldsymbol{x})$ where $\hbar_k(\boldsymbol{x}) = \prod_{i=1}^{\ell} \binom{x_i}{\alpha_{i,k}}$ represents the number of ways of selecting the molecules for reaction $R_k$ when in state $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ (Higham, 2008). When in state $\boldsymbol{x}$, the time until the next reaction has an exponential distribution with rate $\lambda(\boldsymbol{x}) = \sum_{k=1}^{d} a_k(\boldsymbol{x})$, the probability that this reaction is $R_k$ is $a_k(\boldsymbol{x})/\lambda(\boldsymbol{x})$, and these random variables are independent. The SSA of Gillespie (1977) simulates this CTMC directly. However, when a large number of reactions occur in the time interval of interest, the direct simulation approach may be too slow.

Gillespie (2001) proposed the *$\tau$-leaping* algorithm as a way to speed up the simulation. This approach discretizes the time into intervals of length $\tau > 0$, and it generates directy the number of occurrences of each type of reaction in each such interval. If $\boldsymbol{X}(t) = \boldsymbol{x}$ at the beginning of an interval, it is assumed (as an approximation) that the rate of each reaction $R_k$ remains equal to $a_k(\boldsymbol{x})$ during the entire interval $[t, t+\tau]$. Under this simplifying assumption, the number $D_k$ of occurrences of $R_k$ during this time interval has a Poisson distribution with mean $a_k(\boldsymbol{x})\tau$, and $D_1, \ldots, D_d$ are independent. These $D_k$ can be simulated easily via the inversion method, by generating independent uniform random numbers over $(0, 1)$ and applying the inverse of the cumulative distribution function (cdf) of the appropriate Poisson distribution (Giles, 2016). The simulated state at time $t+\tau$ is then $\boldsymbol{x} + \sum_{k=1}^{d} D_k \boldsymbol{\zeta}_k$. Repeating this at each step gives an approximating *discrete-time Markov chain* (DTMC) $\{\boldsymbol{X}_j,\ j \ge 0\}$ defined by $\boldsymbol{X}_0 = \boldsymbol{x}_0$ and

$$\boldsymbol{X}_j \;=\; \boldsymbol{X}_{j-1} + \sum_{k=1}^{d} D_{j,k}\boldsymbol{\zeta}_k, \;=\; \boldsymbol{X}_{j-1} + \sum_{k=1}^{d} F_{j,k}^{-1}(U_{j,k})\boldsymbol{\zeta}_k \;\stackrel{\text{def}}{=}\; \varphi(\boldsymbol{X}_{j-1}, \boldsymbol{U}_j), \tag{1}$$

where $D_{j,k} = F_{j,k}^{-1}(U_{j,k})$, $F_{j,k}$ is the cdf of the Poisson distribution with mean $a_k(\boldsymbol{X}_{j-1})\tau$, $\boldsymbol{U}_j = (U_{j,1}, \ldots, U_{j,d})$, and the $U_{j,k}$ are independent uniform random numbers over $(0, 1)$, for $k = 1, \ldots, d$ and $j \ge 1$. If $\tau$ is small enough, $\boldsymbol{X}_j$ has approximately the same distribution as $\boldsymbol{X}(j\tau)$, so this DTMC provides an approximate skeleton of a CTMC sample path.

This $\tau$-leaping approximation has some potential problems, because it introduces bias which can propagate across successive steps, and this bias can be important if $\tau$ is not small enough. It is also possible to obtain negative copy numbers, i.e., some coordinates of some $\boldsymbol{X}_j$ taking negative values. Adaptive strategies and modifications of the algorithm have been designed to prevent or handle this; see, e.g., Anderson (2008); Anderson and Higham (2012); Beentjes and Baker (2019), and the references given there. We do not discuss these techniques in this paper. Our main goal is to explore how Array-RQMC can be effectively combined with $\tau$-leaping with a fixed $\tau$, and we keep the setting simple to avoid distractions. Implementing Array-RQMC in an adaptive setting (with variable $\tau$) would be more complicated; we discuss it briefly at the end of Section 4.2. In our experiments, we took $\tau$ small enough so we did not observe negative copy numbers.

Following Beentjes and Baker (2019), we suppose that the objective is to estimate $\mu = \mathbb{E}[g(\boldsymbol{X}(T))]$ for a given time $T > 0$ and some function $g : \mathbb{N}_0^\ell \to \mathbb{R}$. These authors only

took a coordinate projection for $g$ (i.e., they only estimated expected copy numbers) in their examples, and we do the same for most of our examples, but what we do applies easily to other choices of $g$. In one of our examples, we take $g(\boldsymbol{x})$ as the indicator that $\boldsymbol{x}$ belongs to a given set $A$, so $\mu = \mathbb{P}[\boldsymbol{X}(T) \in A]$. In another example, we also make experiments in which $g(\boldsymbol{x})$ is the square or the cube of one coordinate. We take $\tau = T/s$ where $s$ is a positive integer that represents the number of steps of the DTMC that will be simulated. To estimate $\mu$ with $\tau$-leaping and MC, we simulate $n$ independent realizations of the DTMC via

$$\boldsymbol{X}_{i,0} = \boldsymbol{x}_0, \qquad \boldsymbol{X}_{i,j} = \varphi_j(\boldsymbol{X}_{i,j-1}, \boldsymbol{U}_{i,j}) \quad \text{for } j = 1, \ldots, s \text{ and } i = 0, \ldots, n-1, \tag{2}$$

where the $\boldsymbol{U}_{i,j}$'s are independent uniform random points over $(0,1)^d$. The estimator is

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=0}^{n-1} g(\boldsymbol{X}_{i,s}). \tag{3}$$

We know that $\mathbb{E}[\hat{\mu}_n] = \mathbb{E}[g(\boldsymbol{X}_s)] \approx \mathbb{E}[g(\boldsymbol{X}(T))] = \mu$ (we do not look at the bias $\mathbb{E}[g(\boldsymbol{X}_s)] - \mathbb{E}[g(\boldsymbol{X}(T))]$ in this paper) and $\mathrm{Var}[\hat{\mu}_n] = \mathrm{Var}[g(\boldsymbol{X}_s)]/n$.

To use classical RQMC instead of MC, we simply replace the independent random points by a set of $n$ vectors $\boldsymbol{V}_i = (\boldsymbol{U}_{i,1}, \ldots, \boldsymbol{U}_{i,s})$, $i = 0, \ldots, n-1$, which form an RQMC point set in $sd$ dimensions, as did Beentjes and Baker (2019). The estimator in (3) obtained with these points $\boldsymbol{V}_i$ is the RQMC estimator, denoted $\hat{\mu}_{n,\mathrm{rqmc}}$. In Section 3, we provide a short review of how RQMC point sets are constructed, some theory, and many references.

## 3 Randomized Quasi-Monte Carlo

The two most popular QMC construction methods are lattice rules (usually of rank 1) and digital nets (typically in base 2). For a *lattice rule of rank 1*, with $n$ points in $s$ dimensions, one selects a vector $\boldsymbol{a} = (a_1, \ldots, a_s)$ with coordinates in $\{1, \ldots, n-1\}$, and such that each $a_j$ is relatively prime with $n$. The QMC point set is

$$P_n = \{\boldsymbol{u}_i = (i\boldsymbol{a}/n) \bmod 1,\ i = 0, \ldots, n-1\}\,.$$

This point set has a very regular lattice structure. It can be randomized via a random shift modulo 1: Generate one random vector $\boldsymbol{U}$ uniformly distributed over the unit hypercube $[0,1)^s$ and add this $\boldsymbol{U}$ to each $\boldsymbol{u}_i$, modulo 1. The resulting (random) point set $\tilde{P}_n = \{\boldsymbol{U}_i = (\boldsymbol{u}_i + \boldsymbol{U}) \bmod 1,\ i = 0, \ldots, n-1\}$ is called a randomly-shifted lattice rule. The shift preserves the structure and the conditions for an RQMC point set mentioned in the introduction are satisfied. For more details, pictures, etc., see Hickernell (1998); L'Ecuyer and Lemieux (2000); L'Ecuyer and Munger (2012); Sloan and Joe (1994). The choice of $\boldsymbol{a}$ is important; we will return to this.

A *digital net in base 2* has $n = 2^k$ points for some integer $k$. One selects an integer $w \geq k$ (often, $w = k$) and $s$ generating matrices $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_s$ of dimensions $w \times k$ and of rank $k$, with binary entries. To define the point $\boldsymbol{u}_i = (u_{i,1}, \ldots, u_{i,s})$, for $i = 0, \ldots, 2^k - 1$, we first write $i = a_{i,0} + a_{i,1}2 + \cdots + a_{i,k-1}2^{k-1}$, and then for each $j$ we compute

$$(u_{i,j,1}, \ldots, u_{i,j,w})^{\mathrm{t}} := \boldsymbol{C}_j \cdot (a_{i,0}, \cdots, a_{i,k-1})^{\mathrm{t}} \bmod 2 \quad \text{and} \quad u_{i,j} = \sum_{\ell=1}^{w} u_{i,j,\ell} 2^{-\ell}.$$

Here, the parameters to select are the elements of the matrices $\boldsymbol{C}_j$. A popular way to construct them is to take $\boldsymbol{C}_1$ as the reflected $k \times k$ identity matrix, which gives $u_{i,1} = i/n$

for the first coordinate, then take for the other coordinates the generating matrices for the *Sobol' sequence* (Sobol', 1967; Lemieux, 2009). These are upper triangular invertible $k \times k$ matrices constructed by specific rules, but the bits above the diagonal in the first few columns can be selected arbitrarily, and their values have an impact on the uniformity of the higher-dimensional projections of $P_n$. General-purpose choices are proposed in Joe and Kuo (2008) and Lemieux et al. (2004). Custom constructions can also be made by giving arbitrary weights to the different projections, using the LatNet Builder software (L'Ecuyer et al., 2020).

Applying a random shift modulo 1 to a digital net in base 2 does not preserve its digital net structure, but a random digital shift does and satisfies the RQMC conditions. It consists in generating a single random vector $\boldsymbol{U}$ uniformly over $[0,1)^s$, and doing a bitwise exclusive-or of all the bits of $\boldsymbol{U}$ with the corresponding bits of each $\boldsymbol{u}_i$ to obtain the randomized points $\boldsymbol{U}_i$ (L'Ecuyer and Lemieux, 2002; Dick and Pillichshammer, 2010; L'Ecuyer, 2018). A much more elaborate and costly randomization method for digital nets is the *nested uniform scramble* (NUS) of Owen (1997a,b), described in many places including Dick and Pillichshammer (2010); L'Ecuyer (2018). NUS became popular because Owen (1997b) proved for known point set constructions that for a sufficiently smooth $f$, the RQMC variance with NUS converges as $\mathcal{O}(n^{-3+\epsilon})$ for any $\epsilon > 0$. The same fast convergence rate was later proved by Hickernell et al. (2001) for a less expensive randomization which consists of a *linear matrix scramble* (LMS) followed by a random digital shift (LMS+shift), as proposed by Matoušek (1998). The LMS generates a random non-singular lower-triangular $w \times w$ binary matrix $\boldsymbol{L}_j$ and replaces $\boldsymbol{C}_j$ by $\boldsymbol{L}_j\boldsymbol{C}_j$ mod 2, for each coordinate $j$.

For both the lattice rules and the digital nets, each one-dimensional projection of $P_n$ (truncated to its first $k$ digits in the case of the digital net) is $\{0, 1/n, \ldots, (n-1)/n\}$. Thus, by looking at any one coordinate at a time, the points cover the unit interval $[0,1)$ very evenly, which is already a good start. Beyond this, the quality of $P_n$ must be measured by assessing the uniformity of its higher-dimensional projections, while giving more weights to the projections deemed more important. This is usually done by working in a Hilbert space of functions $f : [0,1)^s \to \mathbb{R}$. The idea is to define a functional ANOVA decomposition of $f$ as $f = \sum_{\mathfrak{u} \subseteq \{1,2,\ldots,s\}} f_{\mathfrak{u}}$ where $f_{\mathfrak{u}}$ depends only on the coordinates of $\boldsymbol{u}$ whose indexes are in the set $\mathfrak{u}$, and $\mathrm{Var}[f(\boldsymbol{U})] = \sum_{\mathfrak{u} \subseteq \{1,2,\ldots,s\}} \mathrm{Var}[f_{\mathfrak{u}}(\boldsymbol{U})]$. The important projections are those for which $\mathrm{Var}[f_{\mathfrak{u}}(\boldsymbol{U})]$ is large. The variance of the RQMC average

$$\hat{\mu}_{n,\mathrm{rqmc}} = \frac{1}{n}\sum_{i=0}^{n-1} f(\boldsymbol{U}_i) \tag{4}$$

is then bounded by a product of two terms, one that depends only on the point set $P_n$ and the other that depends only on the integrand $f$:

$$\mathrm{Var}[\hat{\mu}_{n,\mathrm{rqmc}}] \le \mathscr{D}^2(P_n)\mathscr{V}^2(f), \tag{5}$$

where

$$\mathscr{V}^2(f) = \sum_{\emptyset \ne \mathfrak{u} \subseteq \{1,2,\ldots,s\}} \gamma_{\mathfrak{u}}^{-2}\mathscr{V}^2(f_{\mathfrak{u}}) \tag{6}$$

and

$$\mathscr{D}^2(P_n) = \sum_{\emptyset \ne \mathfrak{u} \subseteq \{1,2,\ldots,s\}} \gamma_{\mathfrak{u}}^{2}\mathscr{D}_{\mathfrak{u}}^2(P_n), \tag{7}$$

where the $\gamma_{\mathfrak{u}} \in \mathbb{R}^+$ are weights assigned to the subsets $\mathfrak{u}$ of coordinates, $\mathscr{V}(f_{\mathfrak{u}})$ measures the *variation* of $f_{\mathfrak{u}}$, and $\mathscr{D}_{\mathfrak{u}}(P_n)$ measures the *discrepancy* (or non-uniformity) of the projection of $P_n$ over the subset $\mathfrak{u}$ of coordinates. For a function $f$ with finite variation $\mathscr{V}(f)$, the RQMC

variance in (5) converges at the same rate as $\mathscr{D}^2(P_n)$, so the goal becomes to construct point sets $P_n$ for which $\mathscr{D}^2(P_n)$ converges to 0 as fast as possible when $n \to \infty$. For the details, see Dick and Pillichshammer (2010); L'Ecuyer (2009); L'Ecuyer et al. (2020); Owen (1998), and references given there.

The decomposition in (5) depends on the choice of function space. The most classical version is the standard Koksma–Hlawka inequality, in which $\mathscr{D}(P_n)$ is the star discrepancy of $P_n$ and $\mathscr{V}(f)$ is the variation in the sense of Hardy and Krause (Niederreiter, 1992). However, the star discrepancy is much too hard to compute to be used as a practical selection criterion. Nowadays, one prefers to construct Hilbert spaces for which $\mathscr{D}_{\mathfrak{u}}(P_n)$ can be computed efficiently for the type of point set construction of interest. For example, for a randomly-shifted lattice rule with point set $P_n$, the most popular measure (called $\mathscr{P}_\alpha$) has

$$\mathscr{D}^2_{\mathfrak{u}}(P_n) = \frac{1}{n}\sum_{i=0}^{n-1}\prod_{j\in\mathfrak{u}}\phi(u_{i,j}), \tag{8}$$

where $\phi(u_{i,j}) = -(-4\pi^2)^{\alpha/2}B_\alpha(u_{i,j})/\alpha!$ for an even integer $\alpha \geq 2$ and $B_\alpha$ denotes the Bernoulli polynomial of degree $\alpha$. For this measure, it is known how to construct lattice point sets $P_n$ such that $\mathscr{D}^2(P_n) = \mathcal{O}(n^{-\alpha+\epsilon})$ for any $\epsilon > 0$, for any $s$ and finite weights (Dick et al., 2006; Sinescu and L'Ecuyer, 2012; L'Ecuyer and Munger, 2016). It is also known that for periodic continuous functions whose mixed partial derivatives up to order $\alpha/2$ are square integrable, the corresponding variation $\mathscr{V}(f)$ is finite. The periodicity condition means that if $\boldsymbol{u}$ has one coordinate $u_j$ at 0 and we replace the value of $u_j$ by 1 (or the limit as $u_j \to 1$), then $f(\boldsymbol{u})$ remains the same. When $f$ is continuous but not periodic, we can easily transform it into an equivalent periodic function by applying a one-dimensional baker (or tent) transformation separately for each coordinate. This transformation stretches each coordinate of each (randomized) point by a factor of 2, then folds back the values by replacing $u$ with $2-u$ when $u > 1$. This is equivalent to compressing the function horizontally by a factor of 2 and making a mirror copy on the other half, which produces a periodic continuous function whose integral is the same as the original one. This can improve the convergence rate, as proved by (Hickernell, 2002), and thus provide huge variance reductions in some cases. On the other hand, it also increases the variation of the integrand, so it may increase the variance (moderately) in other cases.

Similar theory and discrepancy measures have been developed for digital nets with random digital shifts and also for other types of scrambles such as NUS and LMS+shift. The discrepancies that are practically computable usually have the same form as in (7) and (8), with different definitions of $\phi$. For the details, see Dick and Pillichshammer (2010); L'Ecuyer et al. (2020), and the references given there.

## 4 Array-RQMC to Simulate the DTMC

### 4.1 The Array-RQMC Algorithm

We now explain how to apply Array-RQMC to simulate the DTMC via (2) and estimate $\mathbb{E}[g(\boldsymbol{X}_s)] \approx \mu$ again with (3), but with a different sampling strategy for the random numbers. The algorithm simulates the $n$ sample paths of the DTMC in parallel, using an $(l+d)$-dimensional RQMC point set to advance all the chains by one step at a time, for some $l \in \{1,\ldots,\ell\}$. The first $l$ coordinates of the points are used to make a one-to-one pairing between the chains and the points, and the other $d$ coordinates are used to advance the

chains. When $l < \ell$, one must first define a *dimension-reduction mapping* $h : \mathbb{N}_0^\ell \to \mathbb{R}^l$ whose aim is to extract the most important features from the state and summarize them in a lower-dimensional vector which is used for the sort. For $l = 1$, the mapping $h$ has been called an *importance function* or *sorting function* (L'Ecuyer et al., 2006, 2007). At each step, both the RQMC points and the chains are ordered using the same $l$-dimensional sort. Different types of sorts are discussed in Section 4.2.

Specifically, we select a deterministic low-discrepancy (QMC) point set of the form $Q_n = \{(\boldsymbol{w}_i, \boldsymbol{u}_i),\ i = 0, \ldots, n-1\}$, with $\boldsymbol{w}_i \in [0,1)^l$ and $\boldsymbol{u}_i \in [0,1)^d$, whose points are already sorted with respect to their first $l$ coordinates with the multivariate sort that we have selected. At each step $j$, we randomize the last $d$ coordinates of the points of $Q_n$ to obtain the RQMC point set

$$\tilde{Q}_{n,j} = \{(\boldsymbol{w}_i, \boldsymbol{U}_{i,j}) : i = 0, \ldots, n-1\}, \tag{9}$$

in which each $\boldsymbol{U}_{i,j}$ is uniformly distributed in $[0,1)^d$. We also sort the $n$ states $\boldsymbol{X}_{0,j-1}, \ldots, \boldsymbol{X}_{n-1,j-1}$ based on their values of $h(\boldsymbol{X}_{0,j-1}), \ldots, h(\boldsymbol{X}_{n-1,j-1})$, using the same sorting algorithm as for the QMC points, and let $\pi_j$ denote the permutation of the indices $\{0, 1, \ldots, n-1\}$ implicitly defined by this reordering. Then the $n$ chains advance to step $j$ via

$$\boldsymbol{X}_{i,j} = \varphi(\boldsymbol{X}_{\pi_j(i),j-1}, \boldsymbol{U}_{i,j}), \qquad i = 0, \ldots, n-1.$$

It is also possible to use a different sorting method at each step $j$, in which case the QMC points must be sorted differently as well, so this is usually not convenient.

At the end, one computes $\hat{\mu}_n$ as in (3), which gives an unbiased estimator of $\mathbb{E}[g(\boldsymbol{X}_s)]$. The main goal of this procedure is for the empirical distribution of the states $\boldsymbol{X}_{0,j}, \ldots, \boldsymbol{X}_{n-1,j}$ to better approximate the theoretical distribution of $\boldsymbol{X}_j$ at each step $j$, than if the chains were simulated independently with standard MC, and as a result reduce the variance of $\hat{\mu}_n$. The following heuristic argument gives insight on why it works. To simplify, suppose that $l = \ell$ and the state $\boldsymbol{X}_j$ has the uniform distribution over $[0,1)^\ell$, for each $j$. This can be obtained conceptually by a monotone change of variable (which does not have to be known explicitly). At step $j$, for any function $g_j : [0,1)^\ell \to \mathbb{R}$, the algorithm estimates

$$\mathbb{E}[g_j(\boldsymbol{X}_j)] = \mathbb{E}[g_j(\varphi(\boldsymbol{X}_{j-1}, \boldsymbol{U}))] = \int_{[0,1)^{\ell+d}} g_j(\varphi(\boldsymbol{x}, \boldsymbol{u}))\, d\boldsymbol{x}\, d\boldsymbol{u}$$

by the average

$$\frac{1}{n}\sum_{i=0}^{n-1} g_j(\boldsymbol{X}_{i,j}) = \frac{1}{n}\sum_{i=0}^{n-1} g_j(\varphi(\boldsymbol{X}_{i,j-1}, \boldsymbol{U}_{i,j})).$$

This is essentially an RQMC estimate with the point set $\mathcal{Q}_{n,j} = \{(\boldsymbol{X}_{i,j-1}, \boldsymbol{U}_{i,j}),\ 0 \le i < n\}$. We would like $\mathcal{Q}_{n,j}$ to be highly uniform over $[0,1)^{\ell+d}$ but we cannot really choose the $\boldsymbol{X}_{i,j-1}$'s in these points, since they depend on the simulation. What we do instead is select the RQMC point set $\tilde{Q}_{n,j}$ defined above and reorder the states $\boldsymbol{X}_{i,j-1}$ in a way that $\boldsymbol{X}_{i,j-1}$ is close to $\boldsymbol{w}_i$ for each $i$. This is the role of the sorting step. For a more detailed theoretical analysis and empirical evidence, see for example L'Ecuyer et al. (2008, 2009, 2018). To estimate the variance of this Array-RQMC estimator, one can repeat the entire procedure $m$ times, with independent randomizations of the points, and take the empirical variance of the $m$ realizations of $\hat{\mu}_n$ as an unbiased estimator for $\mathrm{Var}[\hat{\mu}_n]$, as with classical RQMC. This Array-RQMC procedure is stated in Algorithm 1.

One may wonder how easily this algorithm can be parallelized. Of course, it is easy to run the $m$ independent replications in parallel, although we usually prefer to use a large $n$ and small $m$ (e.g., 10 to 20) to benefit from the improved convergence rate. On the other

hand, running a single Array-RQMC replication (with $n$ chains) in parallel (e.g., on a GPU card) is more complicated, because of the sorting at each step. This aspect has to be further investigated. Parallelization would not change the variance, but may increase the speed.

**Algorithm 1** Array-RQMC Algorithm

$\boldsymbol{X}_{i,0} \leftarrow \boldsymbol{x}_0$ for $i = 0, ..., n-1$;
**for** $j = 1, 2, ..., s$ **do**
    Sort the states $\boldsymbol{X}_{0,j-1}, \ldots, \boldsymbol{X}_{n-1,j-1}$ by their values of $h(\boldsymbol{X}_{i,j-1})$,
        using the selected sort, and let $\pi_j$ be the corresponding permutation;
        // We assume that the points are sorted in the same way by their first $l$ coordinates;
    Randomize afresh the last $d$ coordinates of the RQMC points, $\mathbf{U}_{0,j}, ..., \mathbf{U}_{n-1,j}$;
    **for** $i = 0, 1, \ldots, n-1$ **do**
        $\boldsymbol{X}_{i,j} = \varphi(\boldsymbol{X}_{\pi_j(i),j-1}, \boldsymbol{U}_{i,j})$ ;
    **end for**
**end for**
Return the estimator $\hat{\mu}_n = (1/n) \sum_{i=0}^{n-1} g(\boldsymbol{X}_{i,s})$.

A user may want to fix an accuracy target and increase $n$ or $m$ adaptively until the target is reached. This is easily implemented by increasing $m$ for a fixed $n$, but again we prefer increasing $n$. Then, we have to re-run the algorithm with the larger $n$, because the result of the sort is different with the larger $n$. A reasonable strategy would be to first run the algorithm say with $(n, m) = (n_1, m_1)$, and estimate from that a pair $(n, m) = (n_2, m_2)$ that would meet the target accuracy. If we find that it would suffice to increase $m$ by a factor of no more than 2 or 3 with the same $n$, we just do that. Otherwise, we re-run the algorithm with the larger $n = n_2$ deemed sufficient, and we can use a linear combination of the two estimators.

### 4.2 Sorting Strategies

In the special case where $l = 1$, the RQMC points are sorted by their first coordinate and the states $\boldsymbol{X}_{i,j-1}$ are simply sorted by their value of $h(\boldsymbol{X}_{i,j-1})$, in increasing order. In this case, one would typically have $\boldsymbol{w}_i = i/n$ and the points are already sorted by construction (this is true for all the point sets used in this paper).

When $\ell > 1$, sorting for good pairing is less obvious. One multivariate sort that gave good results for other applications is the *batch sort*, defined as follows (Lécot and Coulibaly, 1998; El Haddad et al., 2008; L'Ecuyer et al., 2009, 2018). We factor $n \approx n' = n_1 n_2 \cdots n_L$ with $1 \le L \le d$ and $n \le n'$. The approximation is because $n$ is not always easy to factor; it can be a prime number for example. Each time we sort, we split the set of states into $n_1$ batches of size approximately $n/n_1$ such that the first coordinate of every state in one batch is smaller or equal to the first coordinate of every state in the next batch; then we further subdivide each batch into $n_2$ batches of size approximately $n/(n_1 n_2)$ in the same way but now according to the second coordinate of the states. This procedure is repeated $L$ times in total. In practice, $L$ should rarely exceed 3. If $n < n'$, some batches (the last ones) will contain fewer states. Here, the required dimension for $\boldsymbol{w}_i$ in (9) is $l = L$. In our experiments, when we apply the method for several values of $n$, the choices of $n_1, n_2, \ldots, n_L$ must depend on $n$. What we do is select a vector of positive exponents $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_L)$ such that $\alpha_1 + \cdots + \alpha_L = 1$, called the *batch exponents*, and put $n_j = \lceil n^{\alpha_j} \rceil$ for all $j$. We will report those batch exponents.

Another way of sorting is to map the states to the $\ell$-dimensional unit hypercube $[0,1)^\ell$, so we can assume that the state space is now $[0,1)^\ell$ instead of $\mathbb{N}_0^\ell$, and then use a discretized version of a space filling curve for this hypercube. The hypercube is partitioned into a grid of small subcubes so that the event that two states fall in the same small subcube has a very small probability, then the states are sorted in the order that their subcubes are visited by the curve (those in the same subcube can be ordered arbitrarily). With this, we use $(d+1)$-dimensional RQMC points sorted by their first coordinate. This approach is in fact an implicit way to map the states to the one-dimensional real line, and then use a one-dimensional sort (with $l = 1$). This has been suggested in particular with a Z-curve (Wächter and Keller, 2008) and with a Hilbert curve (Gerber and Chopin, 2015). We call the latter a *Hilbert curve sort*. To map $\ell$-dimensional states to $[0,1)^\ell$, Gerber and Chopin (2015) suggest applying a rescaled logistic transformation $\Psi(x_j) = 1/(1+\exp[-(x_j-\mu_j+2\sigma_j)/(4\sigma_j)])$, $1 \le j \le \ell$, to each coordinate. We estimated the means $\mu_j$ and the variances $\sigma_j^2$ of the copy numbers of each species at every step, from data obtained from preliminary experiments (pilot runs). We tried various numbers of pilot runs, from $2^4$ to $2^{19}$, and the results were not significantly better with more pilot runs. This indicates that only very crude estimates are sufficient.

These multivariate sorts can be computationally expensive when $n$ is large. For this reason, we made some efforts in this work to explore ways of defining importance functions $h : \mathbb{N}_0^\ell \to \mathbb{R}$ that can be computed quickly during the simulations and provide at the same time good representations for the value of a state. An appropriate choice of $h$ is certainly problem-dependent and good ones have been constructed for some examples in other settings such as computational finance, queueing, and reliability (L'Ecuyer et al., 2007, 2008, 2018; Ben Abdellah et al., 2019).

We adopt the (partly heuristic) idea that at each step $j$, an ideal importance function $h_j$ should have the property that $h_j(\boldsymbol{x})$ is a good approximation of $\mathbb{E}[g(\boldsymbol{X}_s) \mid \boldsymbol{X}_j = \boldsymbol{x}]$ for all $\boldsymbol{x} \in \mathbb{N}_0^\ell$ and $j = 1,\dots,s$ (L'Ecuyer et al., 2007, 2009). The rationale is that this conditional expectation can be seen as the "value" of the current state $\boldsymbol{x}$, and can be taken as a "summary statistic" in place of the multidimensional state. In particular, two states with equal "value" can be considered equivalent. To really implement this type of approximation, we need to construct a different $h_j$ for each $j$, because the conditional expectation depends on $j$. We will call this a *step-dependent importance function* (SDIF). One *does not* need to use the expensive procedures that we now describe to be able to apply Array-RQMC for a given reaction network. Our goal is rather to see if these elaborate procedures are worthwhile, or if there are much simpler strategies that can perform almost as well. To see how well a general SDIF could perform, we made the following experiment with each of the examples considered in Section 5. First, we generated data by simulating the DTMC for $n = 2^{19}$ independent "pilot" sample paths, and we collected the $n$ pairs $(\boldsymbol{X}_{i,j}, g(\boldsymbol{X}_{i,s}))$, $i = 0,\dots,n-1$, for each $j$. Then, our aim was to find a function $h_j : \mathbb{N}_0^\ell \to \mathbb{R}$ for which $h_j(\boldsymbol{X}_{i,j})$ was a good predictor of $g(\boldsymbol{X}_{i,s})$ conditional on $\boldsymbol{X}_{i,j}$. For this, we selected a parameterized form of function $h_j$, say $h_j(\boldsymbol{\theta},\cdot)$, which depends on a parameter vector $\boldsymbol{\theta}$, and we estimated the best value of $\boldsymbol{\theta}$ by least-squares regression from the data. The general form that we explored for $h_j(\boldsymbol{\theta},\boldsymbol{x})$ was a linear combination of polynomials in the coordinates of $\boldsymbol{x}$, where $\boldsymbol{\theta}$ was the vector of coefficients in the linear combination. The motivation for this choice is that the expected number of molecules of a given type at the next step, given the current state, is an affine function of the expected number of reactions of each type that will occur at that step, and this expected number for reaction type $R_k$ is in turn linear in $a_k(\boldsymbol{x})$, which is a known polynomial in the coordinates of $\boldsymbol{x}$.

Let $\tilde{h}_j$ denote the functions $h_j$ estimated from data as just described, for each $j$. These $\tilde{h}_j$ are noisy estimates, and since they are estimated separately across values of $j$, we can

observe some random variation when looking at their sequence as a function of $j$. To smooth out this variation, we tried fitting a (least-squares) smoothing spline (de Boor, 2001; Pollock, 1993) to this sequence of functions $\tilde{h}_j$ to obtain a sequence of functions $h_j$, $j = 1, \ldots s$, that varies more smoothly across the step number $j$. This yields a *smoothed SDIF*. In our experiments, we never observed a large improvement by doing this, because with $n = 2^{19}$ pilot simulations, the $\tilde{h}_j$ did not vary much already as a function of $j$. But the smoothing might be worthwhile when the number $n$ of pilot simulations is smaller.

A cruder but less expensive strategy uses the same function $h_j = h$ for all $j$. One possibility is to use $h_{s-1}$ at all steps, i.e., take

$$h(\boldsymbol{x}) = h_{s-1}(\boldsymbol{x}) \stackrel{\text{def}}{=} \mathbb{E}[g(\boldsymbol{X}_s) \mid \boldsymbol{X}_{s-1} = \boldsymbol{x}] = \mathbb{E}[g(\boldsymbol{X}_1) \mid \boldsymbol{X}_0 = \boldsymbol{x}].$$

We had some success with this simple version, which we call the *one-step look-ahead importance function* (OSLAIF).

In the special case where $g(\boldsymbol{x})$ is linear in $\boldsymbol{x}$, say $g(\boldsymbol{x}) = \mathbf{b}^t\boldsymbol{x}$ where $\mathbf{b}^t$ is the transpose of a vector of coefficients, then $h(\boldsymbol{x}) = \mathbb{E}[\mathbf{b}^t\boldsymbol{X}_1 \mid \boldsymbol{X}_0 = \boldsymbol{x}]$ is given by a polynomial in $\boldsymbol{x}$, and one can obtain this polynomial exactly, since from (1),

$$\mathbb{E}[\boldsymbol{X}_1 \mid \boldsymbol{X}_0 = \boldsymbol{x}] = \boldsymbol{x} + \sum_{k=1}^{d} \boldsymbol{\zeta}_k \mathbb{E}[D_{1,k} \mid \boldsymbol{X}_0 = \boldsymbol{x}] \;=\; \boldsymbol{x} + \tau \sum_{k=1}^{d} \boldsymbol{\zeta}_k a_k(\boldsymbol{x}), \tag{10}$$

which is a vector of polynomials in $\boldsymbol{x}$ that are easy to calculate. This includes the case of $g(\boldsymbol{x}) = x_i$, the number of molecules of species $i$, which occurs in our examples.

Extending this to more than one step can be more difficult when the $a_k$ are nonlinear. One can write

$$\mathbb{E}[\boldsymbol{X}_2 \mid \boldsymbol{X}_0 = \boldsymbol{x}] = \boldsymbol{x} + \tau \sum_{k=1}^{d} \boldsymbol{\zeta}_k \left[a_k(\boldsymbol{x}) + \mathbb{E}[a_k(\boldsymbol{X}_1) \mid \boldsymbol{X}_0 = \boldsymbol{x}]\right],$$

but when $a_k$ is nonlinear, the quantity in the last expectation is a nonlinear function of a random vector. Extending to more steps leads to even more complicated embedded conditional expectations. This motivated us to try just the OSLAIF rule as a heuristic, and we already obtained satisfactory results with that. Specific illustrations are given in Section 5. This OSLAIF heuristic also works when $g$ is nonlinear, e.g., for higher moments or for an indicator function, as long as we can compute or approximate the expectation. We will give examples of that in Section 5.

Using tau-leaping with a fixed $\tau$ goes along well with Array-RQMC because all the chains are then synchronized in time; they all advance by the same time step $\tau$ at each step and they all reach $T$ after the same number of steps. This does not hold if we simulate the sample paths reaction by reaction, because the times between successive steps are then independent exponential random variables, so the number of steps is random. This lack of synchronization also occurs if the step size $\tau$ is variable and selected adaptively for each sample path. In these cases, it could be a good idea to also include the current clock time $t$ in the list of state variables used for sorting. Array-RQMC still applies when the number of steps is random and differs across the sample paths, as explained in L'Ecuyer et al. (2008), but the variance reduction is typically more modest in that case. This should be explored in future research.

### 4.3 RQMC Point Sets

We report results for the following point sets in this paper (the short names in parentheses are used to identify them in the next section): (1) a randomly-shifted rank-1 lattice rule (Lat+s); (2) a Lat+s with the baker's transformation applied to the points after the shift (Lat+s+b); (3) a Sobol' net with a left random matrix scramble followed by a random digital shift (Sob+LMS). In our experiments, we also tried Sobol' nets with the nested uniform scramble of Owen (1997b) (Sob+NUS), but the variance was about the same for Sob+LMS and the computing times were significantly longer, so the EIF19 was never better. Therefore, we omit these results from the tables. All these methods are explained in Section 3. They are all implemented in SSJ (L'Ecuyer and Buist, 2005; L'Ecuyer, 2016), a general-purpose Java library for stochastic simulation which we used for all our experiments. It provides the required RQMC tools and also implements the sorting methods discussed in Section 4.2 (see the package `umontreal.ssj.util.sort` in SSJ). For the one-dimensional sorts, these methods use the default quicksort implementation available in Java. For classical RQMC, we used the point coordinates sequentially in time, and for each time step we used them in the same order as the reactions are numbered. For Array-RQMC, we used the first coordinates in the same order as they are used for the sort (e.g., for the batch sort), then the other coordinates in the same order as the reactions are numbered. The MRG32k3a random number generator of L'Ecuyer (1999) was used for MC and all the randomizations. The Java code for our examples can be found in a repository available at `https://github.com/FlorianPuchhammer`.

For the lattice rules, the parameters were found with the Lattice Builder tool (L'Ecuyer and Munger, 2016), using the weighted $\mathscr{P}_2$ criterion defined via (7) and (8) with order dependent weights $\gamma_{\mathfrak{u}}^2 = \rho^{|\mathfrak{u}|}$ for $\rho = 0.6$. This choice is certainly not optimal, but it gave reasonably good results for all cases. We tried other values of $\rho$ and smaller values, for which the weight decreases much faster with the dimension, gave better results for some examples (e.g., $\rho = 0.05$ for the example in Section 5.2), but we nevertheless report the results for a single $\rho$, to show that this is already good enough. For the Sobol' points, we used the parameters (direction numbers) from Lemieux et al. (2004) (our preference) for Array-RQMC. But these parameters are given only for up to 360 dimensions, which is often not enough for classical RQMC, so we used the table from Joe and Kuo (2008) in that case.

## 5 Numerical Illustrations

For our numerical illustrations, we use two low-dimensional examples taken from Beentjes and Baker (2019), then a higher-dimensional example from Padgett and Ilie (2016), and one further example taken from Kim et al. (2015) to study the effect of Array-RQMC on quasi-steady state approximation. On these examples, we compare the performances of both classical RQMC and Array-RQMC in combination with $\tau$-leaping.

We repeated each Array-RQMC procedure $m = 100$ times independently to estimate the RQMC variance $\mathrm{Var}[\hat{\mu}_n]$ for $n = 2^{13}, \ldots, 2^{19}$. We then fitted a model of the form $\mathrm{Var}[\hat{\mu}_n] \approx \kappa n^{-\beta}$ to these observations by least-squares linear regression in log-log scale. This gave an estimated convergence rate of $\mathcal{O}(n^{-\hat{\beta}})$ for the variance, where $\hat{\beta}$ is the least-square estimate of $\beta$. We report $\hat{\beta}$ in our results. Ordinary MC gives $\beta = 1$ (exactly), so we can compare. We should keep in mind that the $\hat{\beta}$ are only noisy estimates and the linear model for the RQMC and Array-RQMC methods is only an approximation. We also provide a few plots of $\mathrm{Var}[\hat{\mu}_n]$ as a function of $n$, in log-log scale, to illustrate the typical behavior. The logs are all in base 2, because we use powers of 2 for $n$.

We computed the estimated *variance reduction factor* (VRF) of Array-RQMC compared with MC, which is defined as $\mathrm{Var}[g(\boldsymbol{X}_s)]/(n\mathrm{Var}[\hat{\mu}_n])$ where $\mathrm{Var}[g(\boldsymbol{X}_s)]$ is the MC variance for a single run, which was estimated separately by making $n = 10^6$ independent runs. This is the variance per run for MC divided by the variance per run for Array-RQMC. We call VRF19 this value for $n = 2^{19}$ and we report it in our results. The VRF for other values of $n = 2^k$ can be estimated via $\mathrm{VRF}(k) \approx (\kappa_0 n^{-1})/(\kappa n^{-\beta}) \approx \mathrm{VRF}19/2^{(\beta-1)(19-k)}$.

We also computed an *efficiency ratio* which measures the change in the work-normalized variance (the product of the estimator's variance by its computing cost). It is the VRF multiplied by the CPU time required to compute $n$ realizations with MC and divided by the CPU time to compute the RQMC or Array-RQMC estimator with the same $n$. We call EIF19 its value for $n = 2^{19}$ and we report it as well. This measure takes into account both the gain in variance and the extra cost in CPU time which is required to sort the chains at each step of the Array-RQMC algorithm. Note that using RQMC only is generally not slower than MC; it is often a bit faster. These VRF19's and EIF19's should be understood as only providing noisy estimates, because the VRF19's are only estimates based on $m = 100$ replications, so there could be over 5% error on these estimates. The true variances depend on the selected parameters for the generating vectors or matrices of the RQMC point sets, and the sorts. Also, points generation, randomizations, and the sorts are not necessarily implemented in the best possible way given the hardware. As an illustration, a batch sort with $n_1 = n$ might run slightly faster than defining an importance function that uses only the first coordinate, even though the two are equivalent. On the other hand, the gains we have obtained are sufficiently large to be convincing. The VRF and EIF for smaller values of $n$ can be estimated by using VRF19 and EIF19 together with $\hat{\beta}$. For the timing comparisons, each RQMC or Array-RQMC experiment with one type of point set, one type of sort, all values of $n$, and $m = 100$, was executed as one job on a Lenovo NeXtScale nx360 M5 node with two Intel Xeon E5-2683 v4 cores at 2.1 GHz.

### 5.1 A Reversible Isomerization System

We start with the same simple model of a *reversible isomerization system* as Beentjes and Baker (2019). There are two species, $S_1$ and $S_2$, and $d = 2$ reaction channels with reaction rates $c_1 = 1$ and $c_2 = 10^{-4}$:

$$S_1 \underset{c_2}{\overset{c_1}{\rightleftarrows}} S_2.$$

There are initially $X_1(0) = 10^2$ molecules of type $S_1$ and $X_2(0) = 10^6$ molecules of type $S_2$. Since the total number of molecules is constant over time, it suffices to know the number of molecules of the first type, $X_1(t)$, at any time $t$, so we can define the state of the CTMC as $\boldsymbol{X}(t) = X_1(t)$ only. This gives $\ell = 1$, and we only need a one-dimensional sort for Array-RQMC. We also take $g(\boldsymbol{X}(t)) = X_1(t)$. With our choice of initial state, $\mathbb{E}[X_1(t)] = 10^2$ for all $t > 0$, so we already know the answer for this simple example. There are two possible reactions, so $d = 2$, and we therefore need RQMC points in $2s$ dimensions with classical RQMC and in $\ell + d = 3$ dimensions with Array-RQMC.

Table 1 summarizes our experimental results. Seven cases are reported in the table. The first case (in the upper left) has the same parameters as Beentjes and Baker (2019): $T = 1.6$, and $s = 8$, so $\tau = T/s = 0.2$. Figure 1 illustrates how the variance decreases as a function of $n$ for this case. Notice the steeper slope for the four Array-RQMC variants. Array-RQMC clearly outperforms both MC and classical RQMC in this example.

Table 1: Estimated rates $\hat{\beta}$, VRF19, and EIF19, for the reversible isomerization example, for various choices of $(T, s, \tau)$. MC refers to ordinary MC, RQMC is classical RQMC with Sobol' points and LMS randomization, and the other four rows are for Array-RQMC with different RQMC point sets. "MC Var" is $\mathrm{Var}[g(\boldsymbol{X}_s)]$, the variance per run with MC. For each case, the best value across the sampling methods is in bold.

| $(T, s, \tau) \longrightarrow$ | (1.6, 8, 0.2) | | | (1.6, 128, 0.2/16) | | | (1.6, 1024, 0.2/128) | | |
|---|---|---|---|---|---|---|---|---|---|
| MC Var | 107.8 | | | 96.6 | | | 96.0 | | |
| Point sets | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 |
| MC | 1.00 | 1 | 1 | 1.00 | 1 | 1 | 1.00 | 1 | 1 |
| RQMC | 1.03 | 629 | 1,493 | 1.08 | 79 | 83 | 1.01 | 46 | 68 |
| Lat+s | **1.80** | **27,844** | **14,900** | **1.79** | **16,923** | **5,066** | **1.65** | **7,290** | **2,114** |
| Lat+s+b | 1.61 | 14,431 | 7,026 | 1.64 | 5,583 | 1,629 | 1.42 | 1,970 | 487 |
| Sob+LMS | 1.63 | 14,812 | 7,748 | 1.62 | 8,090 | 2,328 | 1.58 | 4,197 | 1,140 |
| $(T, s, \tau) \longrightarrow$ | (25.6, 128, 0.2) | | | (102.4, 128, 0.8) | | | (819.2, 1024, 0.8) | | |
| MC Var | 111.0 | | | 166.7 | | | 166.6 | | |
| Point sets | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 |
| MC | 1.00 | 1 | 1 | 1.00 | 1 | 1 | 1.00 | 1 | 1 |
| RQMC | 1.06 | 519 | 625 | 1.10 | 2,294 | 2,382 | 1.12 | 2,887 | 3,018 |
| Lat+s | **1.77** | 20,206 | 11,597 | **1.84** | 34,301 | 23,364 | **1.79** | 31,160 | 22,671 |
| Lat+s+b | 1.75 | **32,136** | **16,111** | 1.50 | 39,380 | 29,552 | 1.58 | **43,977** | **31,849** |
| Sob+LMS | 1.65 | 15,709 | 8,990 | 1.66 | **47,713** | **33,388** | 1.56 | 31,959 | 23,705 |

| $(T, s, \tau) \longrightarrow$ | (1.6, 8, 0.2), normal | | |
|---|---|---|---|
| MC Var | 107.8 | | |
| Point sets | $\hat{\beta}$ | VRF19 | EIF19 |
| MC | 1.00 | 1 | 1 |
| RQMC | 1.94 | 3,673,231 | 5,484,012 |
| Lat+s | 1.89 | 56,510 | 8,605 |
| Lat+s+b | 2.01 | **189,471,599** | **28,804,690** |
| Sob+LMS | **2.08** | 5,509,642 | 889,294 |

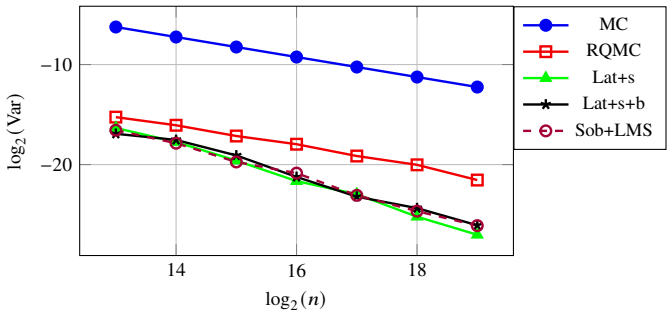


Fig. 1: Estimated $\mathrm{Var}[\hat{\mu}_n]$ as a function of $n$, in log-log scale, for the reversible isomerization system, with $T = 1.6$ and $s = 8$.

We also observe from the first three cases that when we increase $s$ (decrease $\tau$) with $T$ fixed, the factors VRF19 and EIF19 diminish, and the diminution is much more prominent with RQMC. The latter might be no surprise, because increasing $s$ increases the dimension of the RQMC points. But it was unclear a priori if it would also occur with Array-RQMC, and by how much. By doing further experimentation, we found that the decrease of VRF19 is not really due to the increase in the number of steps, but rather to the decrease in $\tau$. To

see this, look at the fourth case, with $(T, s, \tau) = (25.6, 128, 0.2)$. Here we have the same $\tau$ as in the first case, but $s$ is multiplied by 16. For the Array-RQMC methods, the variance reductions and convergence rates are similar to the first case. For RQMC, they are a bit lower, which is not surprising because the dimension has increased. For cases five and six, we have increased $\tau$ to 0.8 and we compare two large values of $s$. The VRF19's are roughly comparable, which means that they really depend on $\tau$ and not much on $s$. Why is that?

Recall that in this example, at each step we generate a pair of Poisson random variables, which are discrete and therefore discontinuous with respect to the underlying uniforms. The mean of each Poisson random variable is proportional to $\tau$, and the larger the mean, the closer it is to a continuous distribution. In fact, as $\tau$ increases, the Poisson converges to a normal distribution, whose inverse cdf is smooth, so the generated values are smooth functions of the underlying uniforms in the limit. That is, we obtain a better VRF19 when $\tau$ is larger because the integrand is closer to a continuous (and smooth) function. When the Poisson distributions have small means, in contrast, the response has larger discontinuities. And it is well known that RQMC is much more effective for smooth functions than for discontinuous functions. This kind of behavior was already pointed out for RQMC in Section 5.2 of Beentjes and Baker (2019). Interestingly, we see that the same effect applies to Array-RQMC as well. To illustrate this effect "in the limit," we made an experiment in which all the Poisson random variables at each step are replaced by normals with the same mean and variance, and the state vector has real-valued components rather than integer components, using the same parameters as in the first case in the table. The results are in the last (bottom) entry of the table and they are stunning. Firstly, for RQMC and all Array-RQMC methods, the rate $\hat{\beta}$ is close to 2, which does not occur for the other cases. Secondly, the VRF19 factor is also very large for RQMC and is huge in particular for Array-RQMC with Lat+s+b. This surprising result for RQMC can be explained as follows. Here the integrand has 16 dimensions, but on a closer look one can see that it is a sum of 16 normal random variables that are almost independent; i.e., almost a sum of one-dimensional functions. This means that the effective dimension is close to 1, and this explains the success of RQMC. Essentially, only the one-dimensional projections of the points are important for classical RQMC, which explains the large gains for this method in this case. For Array-RQMC, the two-dimensional projections are important, because one additional coordinate is used for the sort, and this is why it does not beat classical RQMC for most point sets. The huge gain obtained with Lat+s+b is an exception. It can be explained by the fact that for a smooth one-dimensional function, classical RQMC with Lat+s+b can provide an $\mathcal{O}(n^{-4})$ convergence rate for the variance (Hickernell, 2002). For one-dimensional smooth functions, the baker's transformation produces a locally antithetic effect, so it integrates exactly the piecewise linear approximation and only higher-order error terms remain (L'Ecuyer, 2009). The huge VRF19 indicates that much of this effect carries over to Array-RQMC.

We just saw that as a rough rule of thumb, the RQMC methods bring more gain when the Poisson random variables have larger means. We know (from Section 2) that the mean of the Poisson random variable $D_{j,k}$ is $a_k(\boldsymbol{X}_{j-1})\tau$. This mean can be increased by increasing either $\tau$ or the components of the state vector. For the present example, if we denote $\boldsymbol{X}_{j-1} = (X_{j-1}^{(1)}, X_{j-1}^{(2)})^{\mathrm{t}}$, the number of molecules of each of the two types at step $j-1$, we have $a_k(\boldsymbol{X}_{j-1}) = c_k X_{j-1}^{(k)}$ for $k = 1, 2$, so the Poisson means are increased by a factor $\gamma > 1$ by either multiplying $\tau$ by $\gamma$ or multiplying the vector $\boldsymbol{X}_{j-1}$ by $\gamma$. We made experiments whose results agreed with that when all the components of the state were large enough. But if one component of $\boldsymbol{X}_{j-1}$ is small, and we increase $\tau$ and simulate the system over a few steps, this component has a good chance of getting close to zero at some step, and this increases

the discontinuity. In that situation, a larger $\tau$ can worsen the VRF. To further test the above reasoning, we made another set of experiments in which the initial state $\boldsymbol{X}_0$ had two equal components, exactly $X_0^{(1)} = X_0^{(2)} = (10^2 + 10^6)/2$ molecules of each type, and we adapted the reaction rates to $c_1 = c_2 = 100/X_0^{(1)}$, to keep $\mathbb{E}[X_1(t)] = X_0^{(1)}$ for all $t$. In this case, the problem of one component getting close to 0 does not occur so things remain smoother. We found that the VRFs were larger than in Table 1 for both RQMC and Array-RQMC (we exclude the normal distribution). The VRF for RQMC was also smaller when both $T$ and $s$ were large, but not when $s$ was increased and $T$ remained small. One possible explanation for this is that when $T$ and $s$ are large, the overall change in the state can be large, and then the set of successive changes in the state are less independent, which increases the effective dimension.

### 5.2 The Schlögl System

In this second example, also taken from Beentjes and Baker (2019), we have the three species $S_1$, $S_2$ and $S_3$, and four reaction channels with reaction rates $c_1 = 3 \times 10^{-7}$, $c_2 = 10^{-4}$, $c_3 = 10^{-3}$ and $c_4 = 3.5$, respectively. The model can be depicted as:

$$2S_1 + S_2 \underset{c_2}{\overset{c_1}{\rightleftarrows}} 3S_1, \qquad S_3 \underset{c_4}{\overset{c_3}{\rightleftarrows}} S_1.$$

The propensity functions $a_k$ are given by

$$a_1(\boldsymbol{x}) = c_1 x_1(x_1 - 1)x_2/2, \qquad a_2(\boldsymbol{x}) = c_2 x_1(x_1 - 1)(x_1 - 2)/6,$$
$$a_3(\boldsymbol{x}) = c_3 x_3, \qquad a_4(\boldsymbol{x}) = c_4 x_1.$$

We also take $\boldsymbol{x}_0 = (250,\ 10^5,\ 2 \times 10^5)^{\mathrm{t}}$, $T = 4$, and $\tau = 1/4$, so $s = 16$ steps. This is the same model as in Beentjes and Baker (2019), with the same parameters, except that we took a slightly smaller $\tau$ to avoid negative copy numbers (they had $\tau = 0.4$ also with $T = 4$). As in Beentjes and Baker (2019), we also make the simplifying assumption that the copy numbers of $S_2$ and $S_3$ never change. Only $X_1(t)$, the copy number of $S_1$, is changing when reactions occur. (We will relax this assumption later.) Under this assumption, the Markov chain has a one-dimensional state and the sorting is straightforward, as in the previous example. We want to estimate $\mathbb{E}[X_1(T)]$, the expected number of molecules of $S_1$ at time $T$. Here, this expectation does depend on $T$, and we will see that $\mathrm{Var}[X_1(T)]$ also depends very much on $T$. Our aim is to compare the efficiencies of MC, RQMC, and Array-RQMC. With $d = 4$ possible reactions, the RQMC points must have 5 dimensions for Array-RQMC and $ds = 64$ dimensions for classical RQMC.

Table 2 reports experimental results for this example, first with the parameters just mentioned, then with $(T, s, \tau) = (16, 128, 1/8)$, and finally with $g(\boldsymbol{X}(T)) = X_1(T)$ replaced by $g(\boldsymbol{X}(T)) = \mathbb{I}[X_1(T) > 300]$, so we estimate the probability of having more than 300 molecules of $S_1$ at time $T$ instead of the expected number. In all cases, we see that classical RQMC does not bring much gain, whereas Array-RQMC brings very large variance reductions and efficiency improvements. The gains are larger for the first set of parameters; for classical RQMC, this comes from the smaller dimension, whereas for Array-RQMC, this is due to the larger $\tau$ and smaller $s$ (we made additional experiments and observed that the gains were slightly better when we increased $\tau$ for fixed $s$ or we decreased $s$ for fixed $\tau$).

For the purpose of having a higher-dimensional state, we now consider a slightly different version of this model, in which the copy numbers of all molecule types are assumed to vary.

Table 2: Estimated rates $\hat{\beta}$, VRF19, and EIF19, for the Schlögl system, for various choices of $(T, s, \tau)$ and for two definitions of $g$.

| | $g(\boldsymbol{X}(t)) = X_1(t)$ | | | $g(\boldsymbol{X}(t)) = X_1(t)$ | | | $g(\boldsymbol{X}(t)) = \mathbb{I}[X_1(t) > 300]$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $(T, s, \tau) \longrightarrow$ | (4, 16, 1/4) | | | (16, 128, 1/8) | | | (16, 128, 1/8) | | |
| $\mathbb{E}[g(\boldsymbol{X}_s)]$ | 309.0 | | | 318.3 | | | 0.49 | | |
| MC Var | 44,575 | | | 56,465 | | | 0.25 | | |
| Point sets | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 |
| MC | 1.00 | 1 | 1 | 1.00 | 1 | 1 | 1.00 | 1 | 1 |
| RQMC | 1.10 | 9 | 9 | 1.04 | 3 | 3 | 1.04 | 3 | 5 |
| Lat+s | **1.64** | 2,897 | 2,458 | 1.62 | 1,467 | 1,369 | 1.36 | 1,273 | 1,198 |
| Lat+s+b | 1.24 | 10,147 | 9,318 | 1.09 | 3,427 | 3,341 | 1.07 | 2,709 | 2,535 |
| Sob+LMS | 1.56 | **15,043** | **14,079** | **1.70** | **6,905** | **6,681** | **1.65** | **5,730** | **5,512** |

Since the total number of molecules remains constant over time, the dimension of the state can be taken as $\ell = 2$. We take the state as $\boldsymbol{X} = (X^{(1)}, X^{(2)})^{\mathrm{t}}$, and $X^{(3)}$ can be deduced by $X^{(3)} = N_0 - X^{(1)} - X^{(2)}$ where $N_0$ is the total number of molecules. Given that the model discussed previously can be seen as an approximation of this altered model, we expect $X^{(1)}$ to be the most important variable for the sort in Array-RQMC. Thus, it appears sensible to sort by $X^{(1)}$ alone, and we will try that. We will also try other sorts based on the two-dimensional state and compare. With $d = 4$ possible reactions, the RQMC points for Array-RQMC must be five-dimensional if we construct an importance function $h$ that maps the state to one dimension, and must be six-dimensional otherwise. With classical RQMC, the dimension of the RQMC points is $ds = 64$ for the first case and 512 for the second case.

We now examine how to construct an importance function $h_j : \mathbb{N}_0^2 \to \mathbb{R}$ as discussed in Section 4.2. With the OSLAIF, one can compute the conditional expectation *exactly* by using (10). This gives $h(\boldsymbol{x}) = x_1 + \tau(a_1(\boldsymbol{x}) - a_2(\boldsymbol{x}) + a_3(\boldsymbol{x}) - a_4(\boldsymbol{x}))$, which is a polynomial in $x_1, x_2, x_3$, with coefficients that are easy to compute. To obtain a SDIF for a more general $j$, one possible heuristic could be to assume the same form of polynomial (even if this is not exact) and select the coefficients by least-squares fitting to data obtained from pilot runs as explained in Section 4.2. We did this and we also tried fitting a more general bivariate polynomial that contains all possible monomials $x^{\varepsilon_1} y^{\varepsilon_2}$ with $0 \le \varepsilon_1, \varepsilon_2 \le 3$, but this gave us no improvement over OSLAIF. The other SDIF approches that we tried also did no better than OSLAIF. A plausible explanation is that the functions $h_j$ in this case are based on data obtained from noisy simulations (large variance and dependence on $j$). For the batch sort, we kept the three coordinates in their natural order and we used $n_1 = n_2 = \lceil n^{1/2} \rceil$.

Table 3 summarizes our experimental results with this example. Again, Array-RQMC performs much better than RQMC, with VRF19's in the thousands. All sorting methods reported in the table perform reasonably well. The OSLAIF is very effective for $T = 4$, but somewhat less effective for $T = 32$. The Sobol' points are generally the best performers.

The left panel of Figure 2 shows $\mathrm{Var}[\hat{\mu}_n]$ versus $n$ in log-log scale for the OSLAIF sort, for various point sets. The right panel shows $\mathrm{Var}[\hat{\mu}_n]$ as a function of $n$ under Sob+LMS, in a log-log-scale. The estimated convergence rates $-\hat{\beta}$ are mostly between $-1.3$ and $-1.6$, which beats the MC rate of $-1$. Notice the bump at $n = 2^{18}$ for the lattice rules. It indicates that the selected lattice parameters for this $n$ are not ideal for this specific example. When we made a search for parameters using order-dependent weights with $\rho = 0.05$, as explained in Section 4.3, the bump disappeared and the results were better. Since these weights give more importance to the low-dimensional projections, this suggests that the bump in the figure is due to a point set with one (or more) bad low-dimensional projection for this particular $n$. In

our reported results, we did not want to fine tune the parameters for each example and each $n$, because we think most users will not want to do that and it is not essential.

One important observation is the large difference in MC variance between $T = 4$ and $T = 32$; it is larger at $T = 4$ by a factor of about 100. The mean $\mathbb{E}[\hat{\mu}_n]$ also depends on $T$: it is about 240 at $T = 4$ and about 86 at $T = 32$. What happens is that the trajectories have roughly two very different kinds of transient regimes between $t = 0$ and about $t = 10$. For some trajectories, $X_1(t)$ goes up to somewhere between 400 and 600 at around $t = 4$, then goes down to around the long-term mean, say between 70 and 100. For other trajectories, $X_1(t)$ decreases right away to between 70 and 100 at around $t = 5$. Figure 3 illustrates this behavior, with 16 sample paths. This behavior differs from that of the bistable system discussed before and in Beentjes and Baker (2019). It explains the much larger variance at $T = 4$ than at $T = 32$ and it also shows why it is very hard to predict the state at some larger $T$ from the state at $t = 1/4$, say, hence the difficulty to estimate an "optimal" importance function. Despite this, Array-RQMC performs quite well with simple sorts and brings large efficiency improvements compared with MC and RQMC.

Table 3: Estimated rates $\hat{\beta}$, VRF19, and EIF19 for the Schlögl system, with four types of sorts for Array-RQMC.

| | | $T = 4,\ s = 16$ | | | $T = 4,\ s = 128$ | | | $T = 32,\ s = 128$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbb{E}[g(\boldsymbol{X}_s)]$ | | 243 | | | 239 | | | 86 | | |
| MC Var | | 27,409 | | | 27,471 | | | 270 | | |
| Sort | Sample | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 | $\hat{\beta}$ | VRF19 | EIF19 |
| | MC | 1.00 | 1 | 1 | 1.00 | 1 | 1 | 1.00 | 1 | 1 |
| | RQMC | 1.14 | 11 | 12 | 1.04 | 7 | 8 | 1.29 | 211 | 203 |
| by $S_1$ | Lat+s | 1.54 | 2283 | 2099 | 1.04 | 2003 | 1406 | 1.01 | 255 | 221 |
| | Lat+s+b | 1.24 | 4385 | 4028 | 1.10 | 1596 | 1121 | 1.02 | 189 | 159 |
| | Sob+LMS | 1.41 | 5835 | 5500 | 1.38 | 1760 | 1332 | 1.05 | 268 | 191 |
| OSLAIF | Lat+s | **1.58** | 2686 | 1477 | 1.12 | **3637** | **2201** | 1.08 | 366 | 406 |
| | Lat+s+b | 1.24 | 4385 | 3901 | 1.08 | 1464 | 974 | 1.08 | 442 | 403 |
| | Sob+LMS | 1.35 | 5823 | 5329 | **1.47** | 3215 | 2187 | 1.10 | 666 | 525 |
| Batch | Lat+s | 1.55 | 1283 | 1144 | 1.42 | 906 | 342 | 1.20 | 539 | 573 |
| | Lat+s+b | 1.38 | 4077 | 3633 | 1.23 | 930 | 522 | **1.29** | 1440 | **1582** |
| | Sob+LMS | 1.46 | **6434** | **5760** | 1.41 | 1847 | 1105 | 1.27 | 1569 | 1200 |
| Hilbert | Lat+s | 1.35 | 990 | 818 | 1.17 | 508 | 274 | 1.04 | 1151 | 850 |
| | Lat+s+b | 1.28 | 3157 | 2610 | 0.88 | 337 | 179 | 0.93 | 600 | 438 |
| | Sob+LMS | 1.55 | 3512 | 3138 | 1.23 | 534 | 321 | 1.28 | **1611** | 1221 |

### 5.3 A model of cyclic adenosine monophosphate activation of protein kinase A

This example is a model for the cyclic adenosine monophosphate (cAMP) activation of protein kinase A (PKA), taken from Koh and Blackwell (2012) and Strehl and Ilie (2015). This model is interesting because it has $\ell = 6$ and $d = 6$, which are both larger than in the previous examples. The six molecular species $S_1$ to $S_6$ are (in this order) PKA, cAMP, the partially saturated PKA-cAMP$_2$, the saturated PKA-cAMP$_4$, the regulatory subunit PKAr,

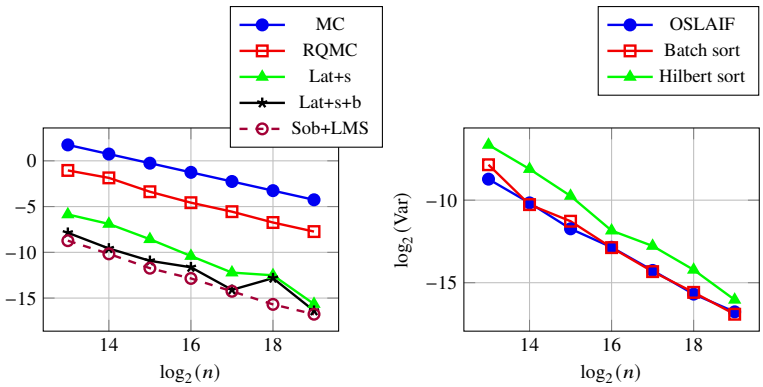


Fig. 2: Empirical variance of the sorting methods vs $n$ in a log-log scale for $T = 4$ and $s = 16$, for the OSLAIF sort and various point sets (left) and for various sorts with Sobol+LMS (right).

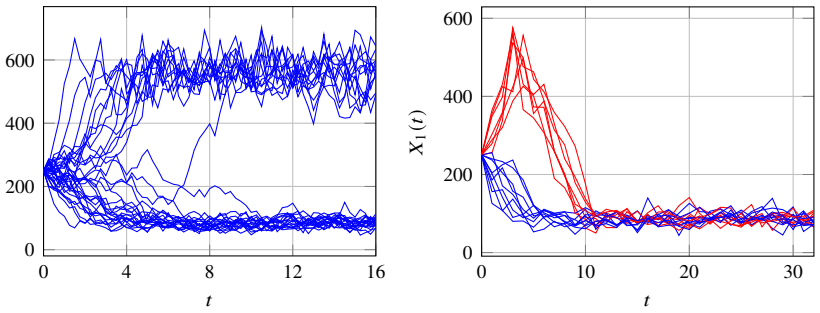


Fig. 3: Trajectories of $X_1(t)$ for $n = 32$ chains for $t \leq 16$ for the Schlögl model in which only $X_1(t)$ varies (left) and for $n = 16$ and $t \leq 32$ when all copy numbers can vary (right).

and the catalytic subunit PKAc. The $d = 6$ possible reactions are depicted here:

$$\mathrm{PKA} + 2\mathrm{cAMP} \underset{c_2}{\overset{c_1}{\rightleftarrows}} \mathrm{PKA\text{-}cAMP_2},$$

$$\mathrm{PKA\text{-}cAMP_2} + 2\mathrm{cAMP} \underset{c_4}{\overset{c_3}{\rightleftarrows}} \mathrm{PKA\text{-}cAMP_4},$$

$$\mathrm{PKA\text{-}cAMP_4} \underset{c_6}{\overset{c_5}{\rightleftarrows}} \mathrm{PKAr} + 2\mathrm{PKAc}.$$

The reaction rates are $c_1 = 2.6255 \times 10^{-6}$, $c_2 = 0.02$, $c_3 = 3.8481 \times 10^{-6}$, $c_4 = 0.02$, $c_5 = 0.016$ and $c_6 = 5.1325 \times 10^{-5}$. We simulate this system with the same parameters as Padgett and Ilie (2016), except that we assume that the molecules are homogeneously distributed in the volume (so the example fits our framework) and we choose a fixed $\tau$ as opposed to changing it adaptively. At time zero there are 33,000 molecules of PKA, 33,030 molecules of cAMP, and 1,100 molecules of each other species. We take $T = 0.05$ and $\tau = T/256$, so we have $s = 256$ steps. This problem requires RQMC points in 7 to 12

dimensions with Array-RQMC, depending on the sort, compared to 1536 dimensions with classical RQMC.

We report experiments with two different objective functions. The first one is $\mathbb{E}[X_1(T)]$, the expected number of molecules of PKA at time $T$, and the second one is $\mathbb{E}[X_5(T)]$, the expected number of molecules of PKAr at time $T$. In each case, we implemented and tested the OSLAIF and SDIF methods to select a mapping $h$ to one dimension. We also tried the multivariate batch sort and the Hilbert sort from Section 4.2. The best performers were the OSLAIF map, the batch sort, and the Hilbert sort.

For $g(\boldsymbol{x}) = x_1$ the OSLAIF is given by the polynomial $h(\boldsymbol{x}) = x_1 + \tau(-c_1 x_1 x_2(x_2-1)/2 + c_2 x_3)$. In this function, $x_1$ outweighs the term $-\tau c_1 x_1 x_2(x_2-1)/2$ on average, followed by $\tau c_2 x_3$. This suggests taking $x_1$ as the most important coordinate for the sort, followed by $x_2$ and $x_3$. For the batch sort, we used these three coordinates in this order, with batch sizes $n_j = \lceil n^{\alpha_j} \rceil$. We first tried $(\alpha_1, \alpha_2, \alpha_3) = (1/2, 3/8, 1/8)$, but $(\alpha_1, \alpha_2) = (1/2, 1/2)$ performed slightly better and is used for our results.

Table 4 summarizes our results for $g(\boldsymbol{x}) = x_1$ (the PKA case). The estimated mean and variance per run with MC are 19663 and 1775, respectively. The batch sort with Sob+LMS gives the largest improvement empirically. Classical RQMC also performs surprisingly well despite the large number of dimensions. With Array-RQMC, we also observe empirical convergence rates $\hat{\beta}$ consistently better than the MC rate of 1.0. This indicates that the VRF should increase further with $n$.

Table 5 gives the results for $g(\boldsymbol{x}) = x_5$ (the PKAr case). The estimated mean and variance per run with MC are about 716 and 47, respectively. The OSLAIF is given by $h(\boldsymbol{x}) = x_5 + \tau(c_5 x_4 - 0.5 c_6 x_5 x_6(x_6 - 1))$. Given that $x_4$, $x_5$, and $x_6$ remain roughly between 500 and 1000 in this model, and that $\tau = 1/5120$, the dominating term in this function is (by far) $x_5$, followed by $-\tau c_6 x_5 x_6^2 \approx -2.5 \times 10^{-3} x_5$. Based on this, for the batch sort, we initially used the coordinates $x_5, x_6, x_4$ in this order. For the reported results, we took $(\alpha_5, \alpha_6, \alpha_4) = (1/2, 1/4, 1/4)$ for the batch exponents. We tried other choices such as $(\alpha_5, \alpha_6, \alpha_4) = (1/2, 3/8, 1/8)$, $(1/3, 1/3, 1/3)$, etc., and similar results were obtained, but with weaker figures for Lat+s.

We also tried SDIF with various types of functions, but it did not really perform better. While doing this, we applied a procedure based on the random forest permutation-based statistical method of Breiman (2001) to detect the most important variables in a noisy function. This procedure told us that $x_6$ was the most important variable for the sort, at all steps. Based on this, we also tried sorting the states by $x_6$ (the number of PKAc molecules) only. This is a degenerate form of batch sort with $n_6 = n$. We call it "By PKAc" in Table 5.

The OSLAIF, Batch, and "By PKAc" sorts perform similarly. They outperform the Hilbert sort and also classical RQMC. Their empirical convergence rates $\hat{\beta}$ are also significantly larger than 1. This example illustrates two facts. First, the dimension of the state is not the ultimate criterion for Array-RQMC to perform well. Secondly, customizing sorting algorithms based on information on the underlying model can improve results significantly.

Following a request from one referee, we performed experiments in which we estimated the second and third moments of the PKA and PKAc copy numbers at time $T$, using both OSLAIF and a batch sort. The OSLAIF is easily computed because the second and third moments of the Poisson distribution are known explicitly. With OSLAIF, the VRF19 with Array-RQMC was around 900 to 1200 with the various point sets. The batch sort with Lat+s gave the best performance, with a VRF19 around 3600. We were also asked to run experiments in which we estimate several expectations simultaneously using the same runs (and therefore the same sort). Our software does not allow this but we "simulated" it by running it for different expectations with the same sort. We estimated the mean as well as the second and

third moments of PKA copy numbers with the OSLAIF for $g(\boldsymbol{x}) = x_1$ and obtained the same VRF19 as in Table 4. When we estimated the expected copy numbers of all six types of molecules using the same sort, for this example, the average VRF19 was reduced by a factor of about 2 compared with the case where we have a sort adapted to each expectation. So this is still reasonably effective. For some coordinates, the VRF19 remained the same while it dropped by factors between 10 to 30 for the worst ones. This can certainly be improved by selecting the sort more carefully, but we intentionally restrained ourselves to simple methods that are easy to apply. The sorts we used were an "average OSLAIF", which takes the average of the six OSLAIF functions $h$ adapted to each of the six expectations, and a batch sort with batch exponents all equal to 1/6 with the state variables kept in their default order.

Table 4: Estimated rates $\hat{\beta}$, VRF19, and EIF19, for PKA with $T = 0.05$, $s = 256$.

| Sort | Sample | $\hat{\beta}$ | VRF19 | EIF19 |
|---|---|---|---|---|
| | MC | 1.00 | 1 | 1 |
| | RQMC | 1.08 | 464 | 603 |
| OSLAIF | Lat+s | **1.44** | 1141 | 671 |
| | Lat+s+b | 1.25 | 830 | 460 |
| | Sob+LMS | 1.28 | 1112 | 762 |
| Batch | Lat+s | 1.30 | 1535 | 801 |
| | Lat+s+b | 1.09 | 1446 | 681 |
| | Sob+LMS | 1.17 | **1979** | **1146** |
| Hilbert | Lat+s | 1.25 | 1054 | 400 |
| | Lat+s+b | 1.17 | 855 | 305 |
| | Sob+LMS | 1.19 | 1258 | 545 |

Table 5: Estimated rates $\hat{\beta}$, VRF19, and EIF19, for PKAr with $T = 0.05$, $s = 256$.

| Sort | Sample | $\hat{\beta}$ | VRF19 | EIF19 |
|---|---|---|---|---|
| | MC | 1.03 | 1 | 1 |
| | RQMC | 1.17 | 39 | 45 |
| OSLAIF | Lat+s | 1.42 | 3634 | 1727 |
| | Lat+s+b | 1.38 | 1491 | 673 |
| | Sob+LMS | 1.47 | 2062 | 1163 |
| Batch | Lat+s | 1.54 | **3961** | **2104** |
| | Lat+s+b | 1.44 | 1416 | 728 |
| | Sob+LMS | **1.65** | 1224 | 811 |
| By PKAc | Lat+s | 1.33 | 2470 | 1513 |
| | Lat+s+b | 1.36 | 1364 | 779 |
| | Sob+LMS | 1.45 | 1856 | 1386 |
| Hilbert | Lat+s | 1.17 | 135 | 54 |
| | Lat+s+b | 1.12 | 88 | 27 |
| | Sob+LMS | 1.24 | 126 | 60 |

### 5.4 Quasi-steady state approximation examples

Quasi-steady state approximation (QSSA) is a simplification approach to reduce the size of a model so that it can be simulated much faster (Cao et al., 2005; Kim et al., 2015; Rao

and Arkin, 2003; Thomas et al., 2012). It applies in situations where some of the reaction types occur at a slow time scale, whereas other reaction types occur at a much faster time scale. In the simplified model, one assumes that after each slow-type reaction, a very large number of the fast-type reactions occur in an infinitesimal time period, so that the system reaches steady-state very quickly with respect to those reactions. One then assumes that until the next slow-type reaction, the state of the vector of variables that are affected only by the fast-type reactions is distributed according to its steady-state conditional distribution given the other variables (which we call the slow-type variables and are assumed fixed). Under these assumptions, only the slow reactions need to be simulated, using propensities that are functions of the states of the slow-type variables only. These functions are often non-polynomial. The validity of this type of approximation is studied in many papers, including Kim et al. (2015), Thomas et al. (2012), and other references cited there. Here, we focus on assessing how Array-RQMC can improve the statistical efficiency for simplified models in which the reaction rate functions depart from the mass action kinetics. To fit our framework, we combined QSSA with tau-leaping.

One of the reviewers suggested the following model of *cooperative enzyme kinetics*, given in Eq. (9) of Kim et al. (2015). Its simplified version has reaction rates given in their Eq. (10), with two state variables $S$ and $P$ which correspond to our $S_1$ and $S_2$. The reactions can be depicted as $\emptyset \xrightarrow{c_1} S_1$ and $S_1 \xrightarrow{c_1} S_2$, with propensities $a_1(\boldsymbol{x}) = 1$ and $a_2(\boldsymbol{x}) = x_1^2/(K_m^2 + x_1^2)$, respectively, and constants $c_1 = 0.5, c_2 = 1$, and $K_m = 2.02 \times 10^5$. We also took $T = 2^{17}$ with $s = 1024$, so $\tau = 2^7 = 128$, and we started with an empty system.

We consider two cases: (1) when we want to estimate $\mathbb{E}[X_1(T)]$ and (2) when we want to estimate $\mathbb{E}[X_2(T)]$. In case (1), the expectation does not depend on the current number of molecules of $S_2$, so we have a one-dimensional chain only, and the sorting is easy. In case (2) the expectation depends on both numbers of molecules, so the state is two-dimensional. For this second case, the OSLAIF gives $h(\boldsymbol{x}) = x_2 + \tau a_2(\boldsymbol{x}) = x_2 + \tau k_p x_1^2/(K_m^2 + x_1^2)$ and we use batch exponents $\boldsymbol{\alpha} = (1/2, 1/2)$ for the batch sort. The results are reported in Table 6. We see that Array-RQMC can provide very large gains, much larger than classical RQMC. For case (2), we find that $x_1$ is the most important variable for the sort: sorting by the copy number of $S_1$, or a batch sort that takes $x_1$ as the first variable, give the best results. The OSLAIF is not competitive in this case because as time goes on, $x_2$ increases, and $h(\boldsymbol{x})$ does not give enough weight to $x_1$ compared with $x_2$. In this system, it takes a very large pool of $S_1$ to start producing $S_2$ at a significant rate, and this is why $x_1$ is important.

Table 6: Estimated rates $\hat{\beta}$, VRF19, and EIF19, for the enzyme kinetics example, with $T = 2^{17}$ and $s = 2^{10}$, for $g(\boldsymbol{x}) = x_1$ (left) and for $g(\boldsymbol{x}) = x_2$ with Lat+s (right).

| $\mathbb{E}[X_1(T)]$ | 61,512 | | |
|---|---|---|---|
| MC Var | 55,398 | | |
| Point sets | $\hat{\beta}$ | VRF19 | EIF19 |
| MC | 1.00 | 1 | 1 |
| RQMC | 1.05 | 4,532 | 4,857 |
| Lat+s | **1.92** | 57,267 | 30,499 |
| Lat+s+b | 1.51 | 75,809 | 42,319 |
| Sob+LMS | 1.55 | **129,414** | **79,531** |

| $\mathbb{E}[X_2(T)]$ | 4,024 | | |
|---|---|---|---|
| MC Var | 4,479 | | |
| Point sets | $\hat{\beta}$ | VRF19 | EIF19 |
| MC | 1.00 | 1 | 1 |
| RQMC | 1.04 | 365 | 387 |
| OSLAIF | 1.43 | 1,469 | 708 |
| Batch | **1.81** | 14,570 | 7,250 |
| by $S_1$ | 1.70 | **20,153** | **12,344** |
| by $S_2$ | 1.00 | 273 | 165 |

## 6 Conclusion

We have studied the combination of the fixed step $\tau$-leap algorithm with Array-RQMC for well-mixed chemical reaction networks and found that in this way, we can reduce the variance in comparison to MC significantly. In contrast to the simulation with traditional RQMC, this approach could often also improve the convergence rate of the variance. Array-RQMC requires to sort the chains by their states at each step of the chain. This can be done with a multivariate sort. But we also showed that one can construct sorts by mapping the states into the real numbers via a simple importance function, and then the sorting is trivial. Some basic knowledge of how the model behaves is of course useful to identify the important state variables that should be retained for a batch sort or to build a better importance function, which in turn can improve the convergence rate of the variance. In our experiments, Array-RQMC was never worse than MC, for all sorting methods. In follow-up work, it would be interesting to explore how automatic learning methods could be used to find better importance functions.

**Acknowledgements** This work has been supported by a Canada Research Chair, an IVADO Research Grant, and an NSERC Discovery Grant number RGPIN-110050 to P. L'Ecuyer. F. Puchhammer was also supported by Spanish and Basque governments fundings through BCAM (ERDF, ESF, SEV-2017-0718, PID2019-108111RB-I00, PID2019-104927GB-C22, BERC 2018e2021, EXP. 2019/00432, ELKARTEK KK-2020/00049), and the computing infrastructure of i2BASQUE academic network and IZO-SGI SGIker (UPV).

## References

Anderson D, Higham D (2012) Multilevel Monte Carlo for continuous-time Markov chains, with applications in biochemical kinetics. Multiscale Modeling & Simulation 10(1):146–179, DOI 10.1137/110840546

Anderson DF (2008) Incorporating postleap checks in tau-leaping. The Journal of Chemical Physics 128(5):054103, URL https://doi.org/10.1063/1.2819665

Anderson DF, Kurtz TG (2011) Continuous time Markov chain models for chemical reaction networks. In: Koeppl H, Densmore D, Setti G, di Bernardo M (eds) Design and analysis of biomolecular circuits, vol 117, Springer, New York, pp 3–42

Anderson WJ (1991) Continuous-Time Markov Chains: An Applications-Oriented Approach. Springer-Verlag, New York

Beentjes CHL, Baker RE (2019) Quasi-Monte Carlo methods applied to tau-leaping in stochastic biological systems. Bulletin of Mathematical Biology 81:2931–2959

Ben Abdellah A, L'Ecuyer P, Puchhammer F (2019) Array-RQMC for option pricing under stochastic volatility models. In: Proceedings of the 2019 Winter Simulation Conference, IEEE Press, pp 440–451, URL https://www.informs-sim.org/wsc19papers/429.pdf

Breiman L (2001) Random forests. Machine learning 45(1):5–32

Cao Y, Gillespie DT, Petzold LR (2005) The slow-scale stochastic simulation algorithm. The Journal of Chemical Physics 122(1):014116, DOI 10.1063/1.1824902

de Boor C (2001) A Practical Guide to Splines, 2nd edn. Springer-Verlag, New York

Demers V, L'Ecuyer P, Tuffin B (2005) A combination of randomized quasi-Monte Carlo with splitting for rare-event simulation. In: Proceedings of the 2005 European Simulation and Modeling Conference, EUROSIS, Ghent, Belgium, pp 25–32

Dick J, Pillichshammer F (2010) Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration. Cambridge University Press, Cambridge, U.K.

Dick J, Sloan IH, Wang X, Woźniakowski H (2006) Good lattice rules in weighted Korobov spaces with general weights. Numerische Mathematik 103:63–97

Dion M, L'Ecuyer P (2010) American option pricing with randomized quasi-Monte Carlo simulations. In: Proceedings of the 2010 Winter Simulation Conference, pp 2705–2720

El Haddad R, Lécot C, L'Ecuyer P (2008) Quasi-Monte Carlo simulation of discrete-time Markov chains on multidimensional state spaces. In: Keller A, Heinrich S, Niederreiter H (eds) Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer-Verlag, Berlin, pp 413–429

Fox BL, Glynn PW (1990) Discrete-time conversion for simulating finite-horizon Markov processes. SIAM Journal on Applied Mathematics 50:1457–1473

Gerber M, Chopin N (2015) Sequential quasi-Monte Carlo. Journal of the Royal Statistical Society, Series B 77(Part 3):509–579

Giles MB (2016) Algorithm 955: approximation of the inverse Poisson cumulative distribution. ACM Transactions on Mathematical Software 42:1–22

Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry 81(25):2340–2361, DOI 10.1021/j100540a008

Gillespie DT (2000) The chemical Langevin equation. The Journal of Chemical Physics 113(1):297–306

Gillespie DT (2001) Approximate accelerated stochastic simulation of chemically reacting systems. The Journal of Chemical Physics 115(4):1716–1733, DOI 10.1063/1.1378322

Hellander A (2008) Efficient computation of transient solutions of the chemical master equation based on uniformization and quasi-Monte Carlo. The Journal of Chemical Physics 128:154109

Hickernell FJ (1998) Lattice rules: How well do they measure up? In: Hellekalek P, Larcher G (eds) Random and Quasi-Random Point Sets, Lecture Notes in Statistics, vol 138, Springer-Verlag, New York, pp 109–166

Hickernell FJ (2002) Obtaining $O(N^{-2+\epsilon})$ convergence for lattice quadrature rules. In: Fang KT, Hickernell FJ, Niederreiter H (eds) Monte Carlo and Quasi-Monte Carlo Methods 2000, Springer-Verlag, Berlin, pp 274–289

Hickernell FJ, Hong HS, L'Ecuyer P, Lemieux C (2001) Extensible lattice sequences for quasi-Monte Carlo quadrature. SIAM Journal on Scientific Computing 22(3):1117–1138

Higham DJ (2008) Modeling and simulating chemical reactions. SIAM Review 50(2):347–368, URL https://doi.org/10.1137/060666457

Joe S, Kuo FY (2008) Constructing Sobol sequences with better two-dimensional projections. SIAM Journal on Scientific Computing 30(5):2635–2654

Kim JK, Josić K, Bennett MR (2015) The relationship between stochastic and deterministic quasi-steady state approximations. BMC Systems Biology 9(87):1–13, URL https://doi.org/10.1186/s12918-015-0218-3

Kloeden PE, Platen E (1992) Numerical Solutions of Stochastic Differential Equations. Springer-Verlag, Berlin

Koh W, Blackwell KT (2012) Improved spatial direct method with gradient-based diffusion to retain full diffusive fluctuations. The Journal of Chemical Physics 137(15):154111, DOI 10.1063/1.4758459

Lécot C, Coulibaly I (1998) A quasi-Monte Carlo scheme using nets for a linear Boltzmann equation. SIAM Journal on Numerical Analysis 35(1):51–70

L'Ecuyer P (1999) Good parameters and implementations for combined multiple recursive random number generators. Operations Research 47(1):159–164

L'Ecuyer P (2009) Quasi-Monte Carlo methods with applications in finance. Finance and Stochastics 13(3):307–349

L'Ecuyer P (2012) Random number generation. In: Gentle JE, Haerdle W, Mori Y (eds) Handbook of Computational Statistics, 2nd edn, Springer-Verlag, Berlin, pp 35–71

L'Ecuyer P (2016) SSJ: Stochastic simulation in Java, http://simul.iro.umontreal.ca/ssj/

L'Ecuyer P (2018) Randomized quasi-Monte Carlo: An introduction for practitioners. In: Glynn PW, Owen AB (eds) Monte Carlo and Quasi-Monte Carlo Methods: MCQMC 2016, Springer, Berlin, pp 29–52

L'Ecuyer P, Buist E (2005) Simulation in Java with SSJ. In: Proceedings of the 2005 Winter Simulation Conference, IEEE Press, Piscataway, NJ, pp 611–620

L'Ecuyer P, Lemieux C (2000) Variance reduction via lattice rules. Management Science 46(9):1214–1235

L'Ecuyer P, Lemieux C (2002) Recent advances in randomized quasi-Monte Carlo methods. In: Dror M, L'Ecuyer P, Szidarovszky F (eds) Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications, Kluwer Academic, Boston, pp 419–474

L'Ecuyer P, Munger D (2012) On figures of merit for randomly-shifted lattice rules. In: Woźniakowski H, Plaskota L (eds) Monte Carlo and Quasi-Monte Carlo Methods 2010, Springer-Verlag, Berlin, pp 133–159

L'Ecuyer P, Munger D (2016) Algorithm 958: Lattice builder: A general software tool for constructing rank-1 lattice rules. ACM Transactions on Mathematical Software 42(2):Article 15

L'Ecuyer P, Lécot C, Tuffin B (2006) Randomized quasi-Monte Carlo simulation of Markov chains with an ordered state space. In: Niederreiter H, Talay D (eds) Monte Carlo and Quasi-Monte Carlo Methods 2004, Springer-Verlag, Berlin, pp 331–342

L'Ecuyer P, Demers V, Tuffin B (2007) Rare-events, splitting, and quasi-Monte Carlo. ACM Transactions on Modeling and Computer Simulation 17(2):Article 9, 45 pages

L'Ecuyer P, Lécot C, Tuffin B (2008) A randomized quasi-Monte Carlo simulation method for Markov chains. Operations Research 56(4):958–975

L'Ecuyer P, Lécot C, L'Archevêque-Gaudet A (2009) On array-RQMC for Markov chains: Mapping alternatives and convergence rates. In: L'Ecuyer P, Owen AB (eds) Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer-Verlag, Berlin, pp 485–500

L'Ecuyer P, Munger D, Lécot C, Tuffin B (2018) Sorting methods and convergence rates for Array-RQMC: Some empirical comparisons. Mathematics and Computers in Simulation 143:191–201

L'Ecuyer P, Marion P, Godin M, Fuchhammer F (2020) A tool for custom construction of QMC and RQMC point sets. In: Monte Carlo and Quasi-Monte Carlo Methods: MCQMC 2020, submitted manuscript, available at http://www.iro.umontreal.ca/~lecuyer/myftp/papers/mcqmc20latnet.pdf

Lemieux C (2009) Monte Carlo and Quasi-Monte Carlo Sampling. Springer-Verlag

Lemieux C, Cieslak M, Luttmer K (2004) RandQMC User's Guide: A Package for Randomized Quasi-Monte Carlo Methods in C. Software user's guide, available at http://www.math.uwaterloo.ca/~clemieux/randqmc.html

Matoušek J (1998) On the $L_2$-discrepancy for anchored boxes. J of Complexity 14:527–556

Niederreiter H (1992) Random Number Generation and Quasi-Monte Carlo Methods, SIAM CBMS-NSF Reg. Conf. Series in Applied Mathematics, vol 63. SIAM

Owen AB (1997a) Monte Carlo variance of scrambled equidistribution quadrature. SIAM Journal on Numerical Analysis 34(5):1884–1910

Owen AB (1997b) Scrambled net variance for integrals of smooth functions. Annals of Statistics 25(4):1541–1562

Owen AB (1998) Latin supercube sampling for very high-dimensional simulations. ACM Transactions on Modeling and Computer Simulation 8(1):71–102

Padgett JMA, Ilie S (2016) An adaptive tau-leaping method for stochastic simulations of reaction-diffusion systems. AIP Advances 6(3):035217, DOI 10.1063/1.4944952

Pollock DSG (1993) Smoothing with cubic splines. Tech. rep., University of London, Queen Mary and Westfield College, London

Rao CV, Arkin AP (2003) Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm. The Journal of Chemical Physics 118(11):4999–5010, DOI 10.1063/1.1545446

Sinescu V, L'Ecuyer P (2012) Variance bounds and existence results for randomly shifted lattice rules. Journal of Computations and Applied Mathematics 236:3296–3307

Sloan IH, Joe S (1994) Lattice Methods for Multiple Integration. Clarendon Press, Oxford

Sobol' IM (1967) The distribution of points in a cube and the approximate evaluation of integrals. USSR Comput Math and Math Phys 7(4):86–112

Strehl R, Ilie S (2015) Hybrid stochastic simulation of reaction-diffusion systems with slow and fast dynamics. The Journal of Chemical Physics 143(23):234108, DOI 10.1063/1.4937491

Thomas P, Straube AV, Grima R (2012) The slow-scale linear noise approximation: an accurate, reduced stochastic description of biochemical networks under timescale separation conditions. BMC Systems Biology 6(39), DOI 10.1186/1752-0509-6-39

Wächter C, Keller A (2008) Efficient simultaneous simulation of Markov chains. In: Keller A, Heinrich S, Niederreiter H (eds) Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer-Verlag, Berlin, pp 669–684