24th International Congress on Modelling and Simulation, Sydney, NSW, Australia, 5 to 10 December 2021
mssanz.org.au/modsim2021

INVITED PAPER

Simulating population-size-dependent birth-and-death processes using CUDA and piecewise approximations

S. Hautphenne [a] and **B. Patch** [a]

[a] *School of Mathematics and Statistics, The University of Melbourne, Parkville, Victoria, 3010, Australia*

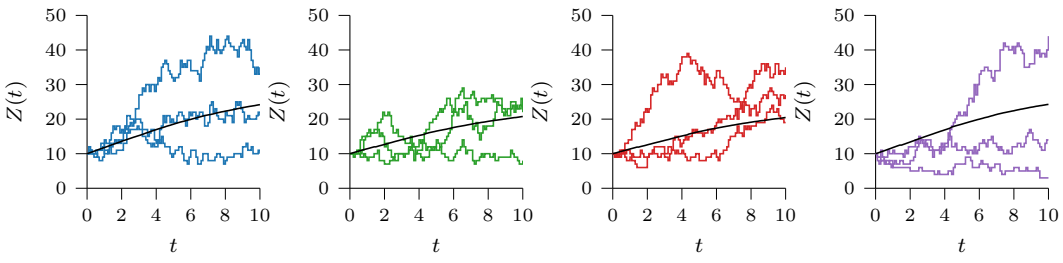
Email: patch@unimelb.edu.au

Abstract: Birth-and-death processes (BDPs) are widely used to model stochastically evolving populations in ecology, genetics, and epidemiology, among others. Population-size-dependent BDPs (PSDBDPs) allow the rate at which individuals give birth and die at any moment in time to depend on the total number of individuals present in the population at that time.

Explicit expressions for the probability distribution and expected value of the future population size at a particular time, conditional on the current population size, are not available in general for PSDBDPs. Simulation is a viable avenue for estimating the probability distribution and expected value of interest. Due to the large number of samples that may need to be simulated in order for accurate estimates to be obtained, simulation may, at first sight, appear to be prohibitively computationally expensive. In this paper we compare the classic exact simulation algorithm with more recent piecewise (or "tau-leaping") approximations that are designed to speed up the simulation process. We also introduce a novel piecewise approximation to PSDBDPs based on linear BDPs. Sample paths generated using each algorithm are displayed in Figure 1, which is discussed in more detail in the paper.

We show that for CPU-only implementations, our new algorithm compares favourably against both exact simulation and other, less accurate, piecewise approximations. In addition to this we investigate a graphics processing unit (GPU) implementation of the exact algorithm. This GPU implementation is able to use CUDA to output estimates of these quantities at a tiny fraction of the time taken by CPU-only implementations.

Our experiments indicate that the CUDA version of exact simulation is the fastest of the approaches we tried. When only a CPU is available, in our experiments our new piecewise approximation is as accurate as exact simulation, but noticeably faster. In particular, the computational burden of our approximation is unchanged by an increase in population size; while on the other hand exact simulation consistently becomes more time consuming as the population increases. Our piecewise approximation is of comparable speed to, but far more accurate than, the classic piecewise approximations we compared it with. It is also robust to increases in the length of the subintervals making up the piecewise approximation — it only suffers from a minor decrease in accuracy as the subintervals are taken to be larger.



(a) Alg. 1 (CPU) (b) Alg. 2 (c) Alg. 3 (d) Alg. 4

Figure 1. Sample paths $z_{10,l}(t)$ (coloured) and the average of 10^4 sample paths $\hat{m}_{10}^{10^4}(t)$ (black) of an SIS model (Verhulst model with $\alpha = 1/N$ and $\beta = 0$) with $\gamma = 0.75$, $\nu = 0.5$, and $N = 100$, generated using different algorithms.

Keywords: *Birth-and-death processes, tau-leaping, simulation, piecewise approximation, CUDA*

1 INTRODUCTION

Birth-and-death processes (BDPs) are useful and widely applied models for stochastically evolving populations in ecology, genetics, and epidemiology, among others (Allen 2008, Novozhilov et al. 2006, Nåsell 2002). Population-size-dependent BDP (PSDBDPs) are BDPs where the rate at which individuals give birth and die at any moment in time depends on the total number of individuals present in the population at that time. These processes are able to capture realistic stochastic population dynamics, such as changes in the transmission rate of a virus in response to the number of infected individuals and the decreasing fertility of animals as their populations approach a carrying capacity.

When individual birth and death rates do not depend upon population size, the BDP is called a *linear BDP* (LBDP). For LBDPs explicit expressions are available for the probability distribution and expected value of the future population size, conditional on the current population size (Guttorp 1991). For PSDBDPs such expressions are not, in general, available. Approaches based on matrix exponentials and other matrix operations (van Dijk et al. 2018, Mandjes & Sollie 2021), Laplace transforms (Murphy & O'donohoe 1975), and diffusion approximations (Kurtz 1971, Pollett 2001) provide approximate solutions.

Simulation is another viable avenue from which to obtain estimates of probability distributions and expected values for PSDBDPs. Due to the large number of samples that may need to be simulated in order for accurate estimates to be obtained, simulation may, at first sight, appear to be prohibitively computationally expensive. In this paper we compare the classic exact simulation algorithm first used by Kendall (1950) with more recent piecewise (or "tau-leaping") approximations stemming from the work of Gillespie (2001). Next, we introduce a novel piecewise approximation to PSDBDPs based on LBDPS. We will see that, when both are implemented on a CPU, this new algorithm compares favourably against both exact simulation and other, less accurate, piecewise approximations. In addition to this, we investigate a graphics processing unit (GPU) implementation of the exact algorithm. Due to the parallelisable nature of estimating probability distributions and expected values using simulation, this GPU implementation is able to use CUDA (Nvidia 2021) to output estimates of these quantities at a fraction of the time needed by traditional CPU-only implementations.

This paper mathematically describes PSDBDPs in Section 2, introduces algorithms that can be used to simulate them (including our novel approach) in Section 3, contains a simulation study comparing the algorithms in Section 4, and provides concluding remarks in Section 5.

2 MODELS

A PSDBDP with birth rate λ_z and death rate μ_z is a continuous-time Markov process which spends an exponentially distributed amount of time in state $z \in (0, 1, \ldots, N)$, where N may be infinite, with mean $(\lambda_z + \mu_z)^{-1}$ before increasing to state $z+1$ with probability $\lambda_z/(\lambda_z + \mu_z)$ or decreasing to state $z-1$ with probability $\mu_z/(\lambda_z + \mu_z)$. Some PSDBDP birth and death rate functions are summarised in Table 1. Note that linear birth-and-death processes (LBDPs), in the first row of Table 1, do not exhibit true population-size dependence: in populations evolving according to a LBDP, individuals give birth and die, respectively, at rates λ and μ, independently of each other and of the current population size z (note the distinction between the LBDP individual birth and death rates λ and μ and the functions λ_z and μ_z). In contrast to this, for the five other models in Table 1, the per-individual birth and death rates depend on z. Since these models take the birth/death rate to be a decreasing/increasing function of z, they are suitable for modelling populations subject to logistic growth (S-shaped trajectories), which have population sizes that tend to fluctuate around a threshold value called the *carrying capacity*.

In this paper, we use the Verhulst model with $\alpha = 1/N$ and $\beta = 0$, which corresponds to the well-known susceptible-infectious-susceptible (SIS) model, as our main example. This model assumes a total population size of N and tracks the number of infectious individuals present at time t. Contact times of infectious individuals with other individuals are assumed to occur according to a Poisson process with rate γ. If there are z infectious individuals, then the probability an infectious individual meets a susceptible individual at a contact time is $1 - \frac{z}{N}$. Susceptible individuals who come into contact with infectious individuals become infectious and z increments to $z+1$. This leads to a birth rate of $\lambda_z = \gamma z \left(1 - \frac{z}{N}\right)$. Infections are assumed to last for an exponentially distributed amount of time with mean ν^{-1}, implying a death rate of $\mu_z = \nu z$. In this case the carrying capacity is $N\left(1 - \frac{\nu}{\gamma}\right)$.

Let $Z(t)$ denote the state (or population size) of a PSDBDP at time $t \geq 0$, and suppose

$$p_{i,j}(t) = \mathbb{P}(Z(t) = j \mid Z(0) = i) \quad \text{and} \quad m_i(t) = \mathbb{E}[Z(t) \mid Z(0) = i], \tag{1}$$

Table 1. Transition rates of a selection of PSDBDs and their parameters for $z \geq 0$; negative values are modified to be equal to 0.

Model	λ_z	μ_z	Parameters
Linear	λz	μz	λ, μ
Verhulst	$\gamma z(1-\alpha z)$	$\nu\,(1+\beta z)\,z$	γ, ν, α, β
Ricker	$\gamma z \exp\left(-(\alpha z)^c\right)$	νz	γ, ν, α, c
Hassell	$\frac{\gamma z}{(1+\alpha z)^c}$	νz	γ, ν, α, c
MaynardSmith-Slatkin	$\frac{\gamma z}{1+(\alpha z)^c}$	νz	γ, ν, α, c
Moran	$\frac{N-z}{N}\left(\frac{\alpha z(1-u)+\beta(N-z)v}{N}\right)$	$\frac{z}{N}\left(\frac{\beta(N-z)(1-v)+\alpha z u}{N}\right)$	α, β, u, v, N

are of interest to a practitioner. In general, closed-form expressions for $p_{i,j}(t)$ and $m_i(t)$ are not available. One of the notable exceptions to this is for LBDPs, for which it is known (see Guttorp (1991)) that the probability of any particular individual present at time 0 and all of its descendants "dying" before time t is given by

$$\alpha(t) = \begin{cases} \mu\{\exp\big((\lambda-\mu)t\big)-1\}/\{\lambda\exp\big((\lambda-\mu)t\big)-\mu\} & \text{if } \lambda \neq \mu, \\ \lambda t/(1+\lambda t) & \text{if } \lambda = \mu. \end{cases} \tag{2}$$

Each surviving individual results in H individuals being present at time t, where the distribution of H, for $h \geq 0$, is given $\mathbb{P}(H=h) = (1-\beta(t))\beta(t)^{h-1}$ with

$$\beta(t) = \begin{cases} \lambda\alpha(t)/\mu & \text{if } \lambda \neq \mu, \\ \alpha(t) & \text{if } \lambda = \mu. \end{cases} \tag{3}$$

Furthermore, using this it can be shown (for the LBDP model) that $p_{i,j}(t) = \alpha(t)^i$ and, for $j \geq 1$,

$$p_{i,j}(t) = \sum_{k=\max(0,i-j)}^{i-1} \binom{i}{k}\binom{j-1}{i-k-1}\alpha(t)^k\Big[\big\{1-\alpha(t)\big\}\big\{1-\beta(t)\big\}\Big]^{j-k}\beta(t)^{j-i+k}. \tag{4}$$

Moreover, $m_i(t) = i\exp\big((\lambda-\mu)t\big)$.

3 ALGORITHMS

For many PSDBDP models it may be necessary to resort to simulation to compute $p_{i,j}(t)$ and $m_i(t)$. In this section we present four algorithms that can be used to generate k realisations $(z_{i\ell}(t),\, \ell = 1,\dots,k)$ which approximate $Z(t)$ conditional on $Z(0) = i$. Using these algorithms we can estimate $p_{i,j}(t)$ and $m_i(t)$ using, respectively,

$$\hat{p}^k_{i,j}(t) = \frac{1}{k}\sum_{\ell=1}^{k} 1_{\{z_{i\ell}(t)=j\}} \quad \text{and} \quad \hat{m}^k_i(t) = \frac{1}{k}\sum_{\ell=1}^{k} z_{i\ell}(t),$$

where $1_{\{A\}}$ is the indicator function of event A.

We first consider an algorithm that generates exact samples of $Z(t)$ using sequences of randomly generated holding times and jump directions throughout the entire interval $[0,t]$. **Alg. 1** generates $(z_{i\ell}(t),\, \ell = 1,\dots,k)$ by repeating the following steps for $\ell \in (1,\dots,k)$:

1. Set $z = i$ and $s = \Delta_z$ where $\Delta_z \sim \mathsf{Exp}(\lambda_z + \mu_z)$.
2. While $s < t$:
   (i) Generate $\mathcal{U} \sim \mathsf{Uniform}([0,1])$.
   (ii) If $\mathcal{U} \leq \lambda_z/(\lambda_z+\mu_z)$ set $z = z+1$; otherwise set $z = z-1$.
   (iii) Set $s = s + \Delta_z$ where $\Delta_z \sim \mathsf{Exp}(\lambda_z+\mu_z)$.

3. Set $z_{i\ell}(t) = z$.

The other three algorithms are all piecewise approximations where the interval $[0, t]$ is partitioned into subintervals of length $0 < \tau < t$. These algorithms approximate the evolution of $Z(t)$ over each subinterval by a combination of random variables that approximately correspond to the total number of births and deaths that occur within the subinterval. Alg. 2 is the tau-leaping algorithm of Gillespie (2001), which uses Poisson distributed random variables to approximate the number of births and deaths. The parameters of these Poisson distributed random variables depend on the state of the process at the start of the subinterval. **Alg. 2** generates $(z_{i\ell}(t),\, \ell = 1, \ldots, k)$ by repeating the following steps for $\ell \in (1, \ldots, k)$:

1. Set $z = i$ and $s = \tau$.
2. While $s < t$:
   (i) Generate $\mathcal{C} = \mathcal{L} - \mathcal{M}$ where $\mathcal{L} \sim \mathsf{Poisson}(\lambda_z \tau)$ and $\mathcal{M} \sim \mathsf{Poisson}(\mu_z \tau)$.
   (ii) Set $z = z + \mathcal{C}$ and $s = s + \tau$.
3. Set $z_{i\ell}(t) = z$.

The authors of Anderson et al. (2011) consider another tau-leaping algorithm, which also uses Poisson random variables to approximate the number of births and deaths, but makes them depend upon an approximation to the expected mid-point of the subinterval. **Alg. 3** proceeds exactly the same as Alg. 2, except that in Step 2(i):

$$\mathcal{L} \sim \mathsf{Poisson}\big(\lambda_{z+\frac{1}{2}\tau(\lambda_z-\mu_z)}\tau\big) \quad \text{and} \quad \mathcal{M} \sim \mathsf{Poisson}\big(\mu_{z+\frac{1}{2}\tau(\lambda_z-\mu_z)}\tau\big).$$

Alg. 2 and Alg. 3 generate sample paths by taking the difference of the number of events in piecewise Poisson processes. Since the rate of a Poisson process is constant, this is equivalent to holding the birth and death rates of the PSDBDP constant within each subinterval; as such, the resulting sample paths can be considered piecewise zero-order approximations.

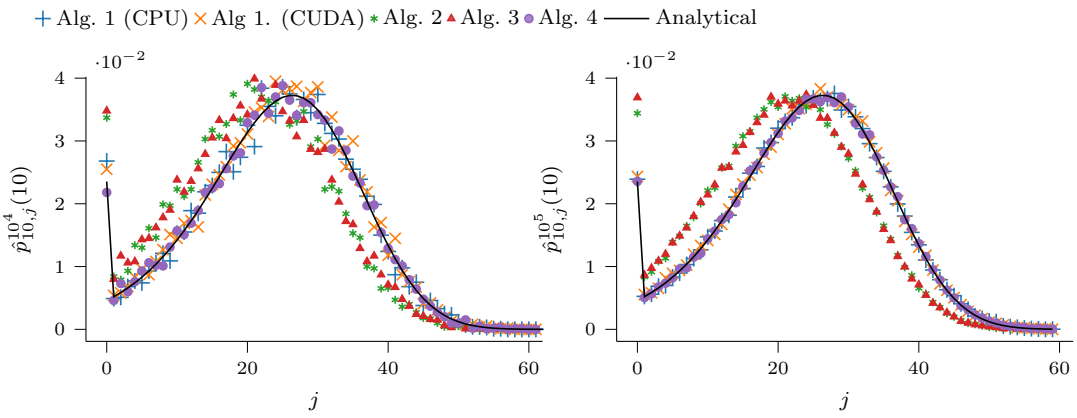
We now introduce a novel piecewise approximation in which a LBDP $\tilde{Z}(\tau)$ is used to approximate the evolution of the PSDBDP $Z(\tau)$ of interest within each subinterval. Using this approximation, the non-linear birth and death rates of the PSDBDP are approximated by linear rates (as opposed to the constant rates in Alg. 2 and Alg. 3); as such, the resulting sample paths can be considered as piecewise first-order approximations. Since the distribution of $\tilde{Z}(\tau)$ is known, as given in Equation (4), the change in population over the interval can be sampled directly without needing to consider sequences of holding times and jump directions. In particular, using (2)–(4), **Alg. 4** proceeds the same as Alg. 2 and Alg. 3, but in Step 2(i) a Binomially distributed random variable $\mathcal{B}$ with number of trials z and success probability $1 - \alpha(\tau)$ is generated, and then $\mathcal{C}$ is a negative Binomially distributed random variable with parameters $\mathcal{B}$ and $1 - \beta(\tau)$. More specifically,

$$\mathbb{P}(\mathcal{C} = k \mid \mathcal{B}) = \binom{k + \mathcal{B} - 1}{k} \beta(\tau)^{\mathcal{B}} (1 - \beta(\tau))^k. \tag{5}$$

where $\mathcal{B}$ is a $\mathsf{Binomial}(z, 1 - \alpha(\tau))$ distributed random variable, and $\alpha(\tau)$ and $\beta(\tau)$ are given by (2) and (3), using $\lambda = \lambda_z / z$ and $\mu = \mu_z / z$ (where λ_z and μ_z correspond to the birth and death rates of the PSDBDP). This approximating piecewise LBDP has its rates set equal to that of the PSDBDP at the beginning of each subinterval. For example, if the PSDBDP is a Verhulst model with $\beta = 0$, then $\lambda = \nu(1 - \alpha z)$ and $\mu = \nu$ are used to compute $\alpha(\tau)$ and $\beta(\tau)$ in (2) and (3).

4 RESULTS

In this section we explore the application of the four algorithms described in Section 3 to the PSDBDP models detailed in Section 2. Our numerical experiments are performed using our Python package BirDePy (v0.0.7) (Hautphenne & Patch 2021*a*,*b*). This package includes two functions that can be used to generate simulation-based transition probability density estimates for PSDBDPs: `birdepy.probability()` and `birdepy.gpu_functions.probability()`. The first of these implements all four algorithms described in Section 3. They can be accessed by setting the argument of `method` equal to `"sim"`, and then changing the argument of `sim_method` between `"exact"`, `"ea"`, `"ma"`, and `"gwa"` switches the function, respectively, between Alg. 1, Alg. 2, Alg. 3, and Alg. 4. The function `birdepy.gpu_functions.probability()` only implements Alg. 1, which it uses by default. This



(a) $k = 10^4$ (b) $k = 10^5$

Figure 2. Estimates $\hat{p}^k_{10,j}(10)$ for an SIS model (Verhulst model with $\alpha = 1/N$ and $\beta = 0$) with $\gamma = 0.75$, $\nu = 0.5$, and $N = 100$, using different algorithms with (a) $k = 10^4$, and (b) $k = 10^5$.

Table 2. Computation time (seconds) to produce estimates in Figure 2.

k	Alg. 1 (CUDA)	Alg. 1 (CPU)	Alg. 2	Alg. 3	Alg. 4
10^4	0.0090	24.4842	6.6010	17.4398	16.8830
10^5	0.0606	254.4340	68.1310	182.4107	171.3088

function performs simulations in parallel on a GPU using CUDA. When Alg. 1 is implemented using this function we refer to it as Alg. 1 (CUDA), otherwise we refer to it as Alg. 1 (CPU). Throughout the section we also refer to an 'analytical' approximation to $p_{i,j}(t)$; this is a matrix exponential based approach generated by `birdepy.probability()` with `method` set to `"expm"`. Our experiments used Python 3.8.10 on a computer running Windows 10 with an Intel i5-10400F CPU, 16GB RAM, and a Nvidia GTX 1070 GPU. The code used to perform these experiments is available at https://github.com/bpatch/modsim_2021.

As our main example we consider the SIS model (which corresponds to the Verhulst model with $\alpha = 1/N$ and $\beta = 0$) with $\gamma = 0.75$, $\nu = 0.5$, and $N = 100$. Figure 1 illustrates sample paths $z_{10,\ell}(t)$ and mean estimates $m^{10^4}_{10}(t)$ for this model generated using each of the algorithms. In this figure, and elsewhere in this section unless specified otherwise, $\tau = 0.1$ is the subinterval length used by Alg. 2, Alg. 3 and Alg. 4. Promisingly, all four algorithms generate sample paths which appear reasonably similar to the naked eye. Notably, despite having the same seed, the generated sample paths are not identical; this highlights the fact that each algorithm generates a sample path according to a distinct stochastic process. The exclusion of Alg. 1 (CUDA) from Figure 1 allows us to underline a key drawback of this algorithm: it is not suited to generating $(z_{10,\ell}(t_j))_{j=0}^n$ where $t_0 < t_1 < \cdots < t_n$, it is only concerned with the generation of $Z(t)$ at a single time t.

Moving on, Figure 2 displays estimates $\hat{p}^k_{10,j}(10)$ of $p_{10,j}(t)$, for the Verhulst model we are interested in, generated using each of the algorithms. Unlike the previous figure, Alg. 1 (CUDA) is also included in this figure. It is immediately apparent from this figure that Alg. 2 and Alg. 3 output highly biased estimates of $p_{10,j}(10)$ for this example. Interestingly the midpoint algorithm Alg. 3 does not appear to improve upon Alg. 2. Note, however, that our approach, Alg. 4, is nearly indistinguishable from the analytical truth. The difference in the amount of noise between the left hand panel, where $k = 10^4$, and the right hand panel, where $k = 10^5$, is noticeable but not very large. Table 2 indicates a substantial difference, however, in the computation time needed to generate the estimates. Indeed, the tenfold increase in k appears to result in approximately a tenfold increase in computation time for all but Alg. 1 (CUDA). The other aspect specific to Alg. 1 (CUDA) relative to the other approaches is its miniscule computation time. Alg. 2 is the next fastest algorithm, but of the other algorithms which appear to produce accurate estimates (Alg. 1 (CPU) and Alg. 4), our approach, Alg. 4, is approximately 33% faster.

Figure 3(a) and Figure 3(d) allow us to explore in more detail the effect of k on accuracy and efficiency.

Panels (a)–(c) in this figure report the Jensen–Shannon distance between the distributions $(p^k_{10,j}(10),\ j \in (0,1,\ldots,N))$ and $(p_{i,j}(10),\ j \in (0,1,\ldots,N))$. This metric for comparing discrete probability distributions is implemented for Python in `scipy.spatial.distance.jensenshannon()`; all mentions of error in this paper use this metric. Figure 3(a) demonstrates the well known fact that Alg. 1 provides asymptotically accurate estimates of $p_{i,j}(t)$. Interestingly, the panel also provides evidence Alg. 4 is converging, as k increases, to an estimate that is also highly accurate. This is in contrast to Alg. 2 and Alg. 3, which appear to converge to an estimate that exhibits a moderate amount of error. The comparative efficiency between the algorithms, displayed in Figure 3(d), matches that previously seen in Table 2; exact simulation on the CPU is slowest, but exact simulation using CUDA is incredibly fast, while the other algorithms lie inbetween.

The subinterval length τ of the piecewise approximation also affects the accuracy and efficiency of Alg. 2 and Alg. 3 for this example. Figure 3(b) shows a clear increase in the error of the estimates produced by Alg. 2 and Alg. 3 as τ increases. Alg. 4 on the other hand, remarkably, does not have a noticeable increase in error as τ increases over the considered range. Figure 3(e) shows that the CPU time needed to generate estimates decreases as τ increases for all three of these algorithms; this follows from the fact that larger subintervals implies fewer subintervals, which means fewer random variables need to be generated overall for these algorithms (since changes in population during each subinterval are approximated by a fixed number of random variables). Combining these facts demonstrates that a value of $\tau > 0.1$ could be used in Alg. 4 with the same accuracy as displayed in Figure 2, but a much smaller CPU time than reported in Table 2. Similarly, τ could be increased by a user of Alg. 2 or Alg. 3 to reduce computation time at the cost of higher error.

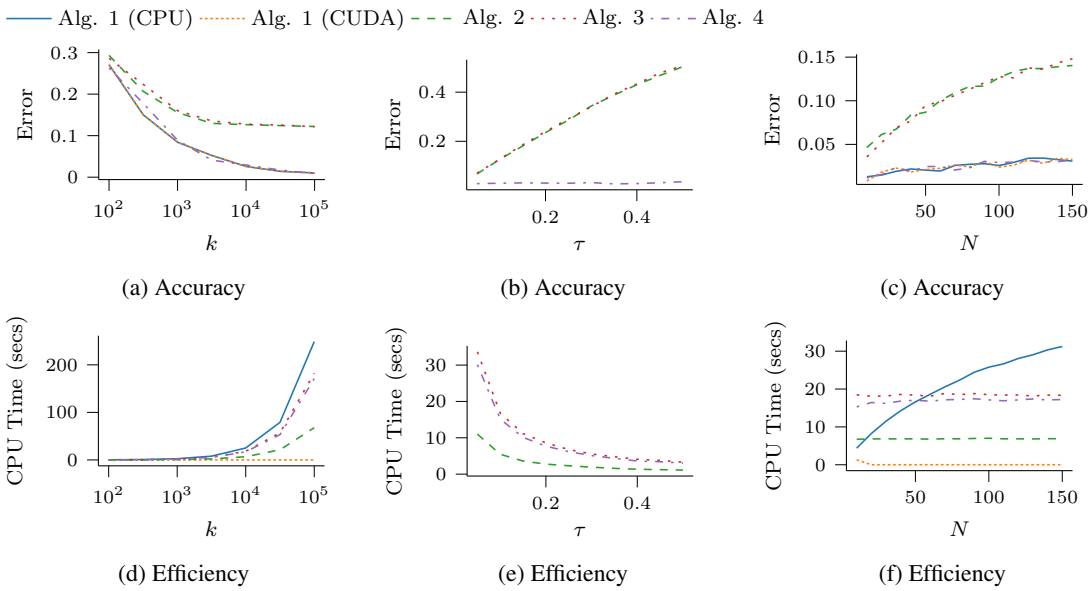
The accuracy and efficiency of a simulation algorithm can depend greatly on the parameters of the model being simulated. Figure 3(c) shows that, for this example, the error of all four algorithms increases as the population size N goes up. This makes intuitive sense because a higher population means: (i) that the process being studied is likely to have more transitions in any particular subinterval (especially if z is small), and (ii) the number of points in the support of $Z(t)$ is higher, so consequently there are more $p_{i,j}(t)$ values to be estimated, implying each estimate effectively relies on fewer samples for fixed k. Notably, however, the error of our new approach, Alg. 4, tracks closely with the error of the exact approach, Alg. 1, as N increases; while the other approximations Alg. 2 and Alg. 3 experience a much greater increase in error as N increases. This fact is even more noteworthy when it is combined with the observation that, in panel (f) of Figure 3, the CPU time of our approach does not increase with N, while the CPU time of the exact algorithm on the CPU does. It is conceivable that, for larger N, Alg. 4 generates estimates which are nearly as accurate as exact simulation, but at a fraction of the CPU time. A tempering observation to these discoveries about our new approach, is that Alg. 1 implemented using CUDA continues (as it has with all of the other experiments) to take an incredibly small relative amount of time to complete for the entire range of N.

5 CONCLUDING REMARKS

This paper has investigated methods for simulating PSDBDPs. Implementations of the standard exact simulation algorithm (Alg. 1) using a standard desktop configuration and a specialised configuration including a CUDA enabled GPU were considered. In addition to this, two classic approaches which rely on zero-order piecewise approximations were included in our study (Alg. 2 and Alg. 3) . We also introduced a novel first-order piecewise approximation (Alg. 4). We explored the performance of these algorithms using an SIS model.

We saw that if a CUDA enabled GPU is available, then using the exact approach combined with this hardware can be much more efficient than all other considered approaches. Our new piecewise approximation was highly accurate for the example we used. It was, promisingly, comparable in accuracy to exact simulation. The same cannot be said for Alg. 2 and Alg. 3, which exhibited error when estimating a probability distribution, even as the number of samples became very large. Our algorithm was also robust against increases in the size of the subintervals of the piecewise approximation, meaning these could be taken larger without much loss in accuracy; resulting in even more potential for our approach to generate efficiency. At the same time we observed that Alg. 4 was more efficient than exact simulation — it returned output using much less CPU time than exact simulation, comparable CPU time to Alg. 3 and slightly more CPU time than Alg. 2. A particular strength of our algorithm when compared to exact simulation was that it did not become less efficient as the population size N was increased, while maintaining comparable accuracy to exact simulation (which suffered from a substantial increase in CPU time as the population size increased).

In further research we are currently investigating ways to quantify the difference in accuracy between exact simulation and our new approach analytically. Our future analytical result will, hopefully, be generalisable across a range of models and parameter values.



(a) Accuracy (b) Accuracy (c) Accuracy

(d) Efficiency (e) Efficiency (f) Efficiency

Figure 3. Accuracy and efficiency of simulation algorithms applied to an SIS model (Verhulst model with $\alpha = 1/N$ and $\beta = 0$) with $\gamma = 0.75$, $\nu = 0.5$, and $N = 100$.

ACKNOWLEDGMENTS

This research is funded by the Australian Government through the ARC (DP200101281).

REFERENCES

Allen, L. J. (2008), An introduction to stochastic epidemic models, *in* 'Mathematical Epidemiology', Springer, pp. 81–130.

Anderson, D. F., Ganguly, A. & Kurtz, T. G. (2011), 'Error analysis of tau-leap simulation methods', *The Annals of Applied Probability* **21**(6), 2226–2262.

Gillespie, D. T. (2001), 'Approximate accelerated stochastic simulation of chemically reacting systems', *The Journal of Chemical Physics* **115**(4), 1716–1733.

Guttorp, P. (1991), *Statistical inference for branching processes*, Vol. 122, Wiley-Interscience.

Hautphenne, S. & Patch, B. (2021*a*), 'Birth-and-death processes in Python'.
URL: *https://birdepy.github.io/*

Hautphenne, S. & Patch, B. (2021*b*), 'Birth-and-death processes in Python: The BirDePy package'. arXiv.2110.05067.

Kendall, D. G. (1950), 'An artificial realization of a simple "birth-and-death" process', *Journal of the Royal Statistical Society. Series B (Methodological)* **12**(1), 116–119.

Kurtz, T. (1971), 'Limit theorems for sequences of jump Markov processes', *Journal of Applied Probability* **8**(2), 344–356.

Mandjes, M. & Sollie, B. (2021), 'A numerical approach for evaluating the time-dependent distribution of a quasi birth-death process', *Methodology and Computing in Applied Probability* pp. 1–23.

Murphy, J. & O'donohoe, M. (1975), 'Some properties of continued fractions with applications to Markov processes', *IMA Journal of Applied Mathematics* **16**(1), 57–71.

Nåsell, I. (2002), 'Stochastic models of some endemic infections', *Mathematical Biosciences* **179**(1), 1–19.

Novozhilov, A. S., Karev, G. P. & Koonin, E. V. (2006), 'Biological applications of the theory of birth-and-death processes', *Briefings in Bioinformatics* **7**(1), 70–85.

Nvidia (2021), 'CUDA Toolkit: Develop, optimize and deploy gpu-accelerated apps'.
URL: *https://developer.nvidia.com/cuda-toolkit*

Pollett, P. (2001), Diffusion approximations for ecological models, *in* 'Proceedings of the international congress on modelling and simulation', Vol. 2, Citeseer, pp. 843–848.

van Dijk, N. M., van Brummelen, S. P. J. & Boucherie, R. J. (2018), 'Uniformization: Basics, extensions and applications', *Performance Evaluation* **118**, 8–32.