



GillespieSSA: Implementing the Stochastic Simulation Algorithm in R

Mario Pineda-Krch

University of California, Davis

Abstract

The deterministic dynamics of populations in continuous time are traditionally described using coupled, first-order ordinary differential equations. While this approach is accurate for large systems, it is often inadequate for small systems where key species may be present in small numbers or where key reactions occur at a low rate. The Gillespie stochastic simulation algorithm (SSA) is a procedure for generating time-evolution trajectories of finite populations in continuous time and has become the standard algorithm for these types of stochastic models. This article presents a simple-to-use and flexible framework for implementing the SSA using the high-level statistical computing language R and the package **GillespieSSA**. Using three ecological models as examples (logistic growth, Rosenzweig-MacArthur predator-prey model, and Kermack-McKendrick SIRS metapopulation model), this paper shows how a deterministic model can be formulated as a finite-population stochastic model within the framework of SSA theory and how it can be implemented in R. Simulations of the stochastic models are performed using four different SSA Monte Carlo methods: one exact method (Gillespie's direct method); and three approximate methods (explicit, binomial, and optimized tau-leap methods). Comparison of simulation results confirms that while the time-evolution trajectories obtained from the different SSA methods are indistinguishable, the approximate methods are up to four orders of magnitude faster than the exact methods.

Keywords: Gillespie's exact method, Kermack-McKendrick SIRS model, logistic growth, metapopulation model, Rosenzweig-MacArthur predator-prey model, tau-leaping.

1. Introduction

It is well known that stochasticity in finite populations can generate dynamics profoundly different from the predictions of the corresponding deterministic model. For example, demographic stochasticity can give rise to regular and persistent population cycles in models that

are deterministically stable (e.g., Nisbet and Gurney 1976; Dushoff *et al.* 2004; McKane and Newman 2005; Pineda-Krch *et al.* 2007) and can give rise to molecular noise and noisy gene expression in genetic and chemical systems where key molecules are present in small numbers or where key reactions occur at a low rate (e.g., Arkin *et al.* 1998; Fedoroff and Fontana 2002; McAdams and Arkin 2005). Because analytical solutions to stochastic time-evolution equations for all but the simplest systems are intractable, while numerical solutions are often prohibitively difficult, stochastic simulations have become an invaluable tool for studying the dynamics of finite biological, chemical, and physical systems.

In the 1970s, Daniel T. Gillespie (1977, 1976) developed an exact stochastic simulation approach for chemical kinetics, the Gillespie stochastic simulation algorithm (SSA). The SSA is a procedure for generating time-evolution trajectories of finite populations in continuous time and has since its introduction become the standard algorithm for these types of models. The development of the SSA was also the first effort to accelerate stochastic simulations beyond what is possible using the basic algorithm by Gillespie. Although the SSA and its various exact and accelerated Monte Carlo implementations have largely been developed for models of chemical kinetics and molecular dynamics, the procedures are applicable to any continuous time system that can be described using coupled first-order ordinary differential equations. The examples used in this paper are selected to illustrate how to implement the SSA for different types of ecological models.

This article presents a simple-to-use and flexible framework for implementing the SSA using the high-level statistical computing language R (R Development Core Team 2007a). Using three ecological models as examples, I show how the deterministic model can be formulated as a finite-population stochastic model within the framework of the SSA theory and how it can be implemented in R using the **GillespieSSA** package which provides a simple-to-use and intuitive interface to several SSA Monte Carlo procedures. Section 2 provides a brief background introduction to the theory underlying the SSA and several of its Monte Carlo implementations. Section 3 describes the basic strategy of implementing the SSA using the **GillespieSSA** package. Section 4 demonstrates the implementation of **GillespieSSA** using three biological example models. Section 5 provides a discussion of the computational accuracy and performance of the different Monte Carlo methods in light of the examples. Finally, Section 6 provides a brief summary of future developments of the package.

2. The Gillespie stochastic simulation algorithm

The Gillespie stochastic simulation algorithm (SSA) is a procedure for generating statistically correct trajectories of finite well-mixed populations in continuous time. The trajectory that is produced is a stochastic version of the trajectory that would be obtained by solving the corresponding stochastic differential equations. While the original SSA (Gillespie 1976, 1977) is numerically exact, it is generally too slow for most practical applications. Over the years, numerous approximate SSA methods have been developed to improve the computational efficiency of the procedure. The SSA methods described in this paper are the ones implemented in the R package **GillespieSSA** (i.e., Gillespie’s exact method and three accelerated approximate methods). Parameters and variables that are formal arguments in the package are indicated by typewriter font. Unless otherwise indicated they are followed by the default value (e.g., `epsilon` = 0.03).

The SSA assumes a population consisting of a finite number of individuals distributed over a finite set of discrete states. Changes in the number of individuals in each state occur due to reactions between interacting states. Given an initial time t_0 and initial population state $\mathbf{X}(t_0)$, the SSA generates the time evolution of the state vector $\mathbf{X}(t) \equiv (X_1(t), \dots, X_N(t))$ where $X_i(t)$, $i = 1, \dots, N$, is the population size of state i at time t and N is the number of states. The states interact through M reactions \mathcal{R}_j where $j = 1, \dots, M$ denotes the j th reaction. A reaction is defined as any process that instantaneously changes the population size of at least one state. Each reaction \mathcal{R}_j is characterized by two quantities. The first is its state-change vector $\boldsymbol{\nu}_j = (\nu_{1j}, \dots, \nu_{Nj})$, where ν_{ij} is the population change in state i caused by one \mathcal{R}_j reaction. In other words, if the system is in state \mathbf{x} , assuming $\mathbf{x} = \mathbf{X}(t)$, and one \mathcal{R}_j reaction occurs, the system instantaneously jumps to state $\mathbf{x} + \boldsymbol{\nu}_j$. The second component of \mathcal{R}_j is its propensity function $a_j(\mathbf{x})$ which is the probability of one \mathcal{R}_j reaction occurring in the infinitesimal time interval $[t, t + dt)$.

To illustrate the basic SSA theory, consider the simple radioactive decay model (also known as the irreversible isomerization reaction set, see [Gillespie 1977](#))

$$\frac{dX}{dt} = -cX$$

where X is the population density at time t and c is the decay parameter. This system consists of a single state X_1 ($N = 1$) and a single reaction, $X \rightarrow \emptyset$ ($M = 1$). Assuming $X = 1000$ and $c = 0.5$ the SSA can now be applied by setting the initial population state $X_1 = 1000$, the state-change vector $\boldsymbol{\nu} = -1$, and the propensity function $a_1 = c$. This model is available in the package as a demonstration model and can be invoked using `demo("radioactiveDecay")`. Figure 1 shows the output from one realization.

2.1. Exact SSA

There are several mathematically equivalent Monte Carlo procedures for constructing exact numerical realizations of the SSA. Perhaps the simplest procedure is the original method by Gillespie, the so-called direct method ([Gillespie 1977, 1976](#)). The direct method proceeds by drawing two random numbers r_1 and r_2 from the uniform distribution in the unit interval. The time step to the next reaction is then determined as $\tau = \frac{1}{a_0(\mathbf{x})} \ln(1/r_1)$ where $a_0(\mathbf{x}) = \sum a_j(\mathbf{x})$ and the index of the next reaction to execute \mathcal{R}_j is the smallest integer j satisfying $j = \sum_{i=1}^j a_i(\mathbf{x}) > r_2 a_0(\mathbf{x})$. The reaction is then executed by replacing $t \leftarrow t + \tau$ and $\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\nu}_j$.

When $a_0(\mathbf{x})$ becomes large, due to large population sizes or high reaction rates, the time increment between reactions decreases, slowing down the simulation. Although computationally more efficient exact methods have been developed ([Gibson and Bruck 2000](#); [Cao et al. 2004b](#)), any procedure that simulates one reaction at a time will inevitably be too slow for most practical applications.

2.2. Approximate SSAs

Several approximate Monte Carlo procedures have been developed which provide better computational performance than the exact methods. These accelerated procedures sacrifice the exactness of the exact methods for potentially large improvements in computational efficiency.

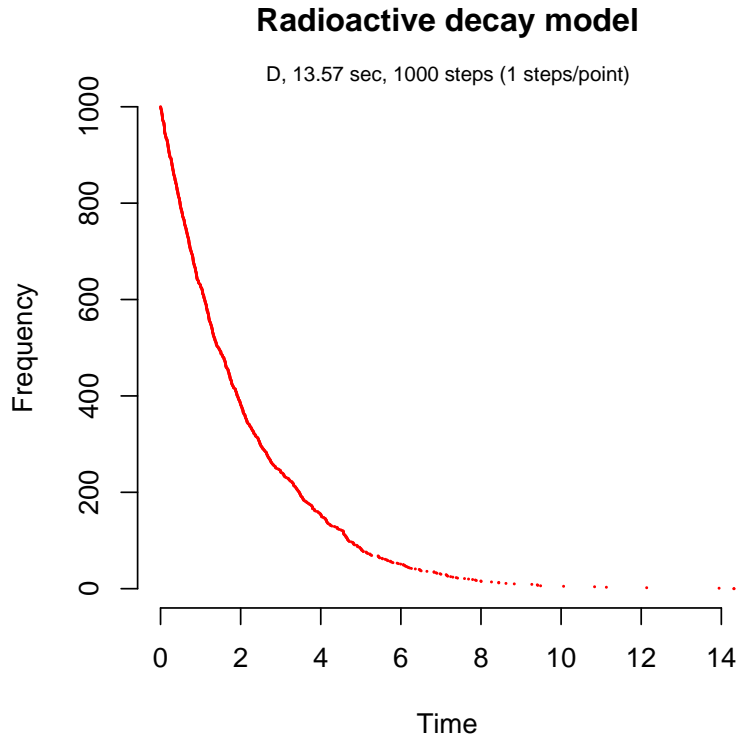


Figure 1: Example of a single realization of the radioactive decay model using `demo("radioactiveDecay")`. Each data point represents a single time step.

Instead of simulating a single reaction per time step, reactions are "bundled" in coarse-grained time increments. As one time step leaps over many reactions, these methods are known as "leap" or "accelerated" methods. Under the right conditions, speed-ups of several orders of magnitude can be achieved.

An important assumption in all accelerated SSA procedures is that the Leap Condition is satisfied. The Leap Condition requires that the time leap be small enough that the change in the propensity functions are negligible (Gillespie 2007). In other words, $a_j(\mathbf{x}) \approx \text{constant}$ in $[t, t + \tau]$ for all j . Three approximate SSA methods are presented in this paper: the explicit tau-leap method (ETL); binomial tau-leap method (BTL); and optimized tau-leap method (OTL).

2.3. Explicit tau-leap (ETL)

In the explicit tau-leap (ETL) method (Gillespie 2001), the number of firings for each reaction \mathcal{R}_j during the time step τ is sampled independently from a Poisson distribution. In the original ETL formulation, the procedure for selecting the step size is computationally inefficient (both in terms of speed and accuracy) (Gillespie 2001). In order to keep the algorithm as simple and as fast as possible the ETL step-size selection procedure is simplified in the **GillespieSSA** package by assuming a constant user-defined step size (`tau = 0.3`).

The ETL method proceeds by generating, for each reaction $j = 1, \dots, M$, the number of firings k_j of reaction \mathcal{R}_j in the time step τ as a sample of a Poisson random variable with mean (and variance) $a_j(\mathbf{x})\tau$. Assuming $\mathbf{x} = \mathbf{X}(t)$, the leap is executed by replacing $t \leftarrow t + \tau$ and $\mathbf{x} \leftarrow \mathbf{x} + \sum_{j=1}^M k_j \boldsymbol{\nu}_j$.

While this version of the ETL method has the advantage of computational simplicity, it does require the user to supply a predefined step size. A heuristic procedure for selecting an appropriate step size is to evaluate the distribution of step sizes obtained from short exploratory simulations using the direct method (this assumes that every time step is recorded, i.e., `censusInterval` = 0). As a rule of thumb, the step size should be at least a few multiples of the expected time to the next reaction, $1/a_0(\mathbf{x})$ for the ETL method to be worthwhile implementing (Gillespie 2001). Although the ETL procedure can significantly speed up the simulation, it is not as foolproof as the direct method. Due to the unbounded nature of Poisson random variables and the lack of coordination between reactions during a time step, the population sizes of individual states can become negative.

2.4. Binomial tau-leap (BTL)

To address the problem of negative population sizes arising in the ETL method, Tian and Burrage (2004) and Chatterjee *et al.* (2005) independently developed the binomial tau-leap (BTL) method. The BTL method described here is a slightly modified version of the procedure proposed by Chatterjee *et al.* (2005).

In the ETL method negative population sizes can arise in two ways. Either a reaction can fire more times than the number of available reactants (i.e., individuals) in one of the states, or several reactions firing simultaneously during a time step, can drive the number of individuals in one or several states below zero (Gillespie 2007). By coordinating the number of firings among different reactions during a time step the maximum number of firings per reaction \mathcal{R}_j can be bounded by the number of individuals in the limiting state (i.e., the state that would be completely depleted if its consuming reaction were to go to completion). To coordinate the reactions, the maximum number of firings for a given reaction k_j^* is updated between each subsequent firing during a time step.

The time increment τ is determined by scaling the expected time to the next reaction $1/a_0(\mathbf{x})$ by a coarse graining factor, f (assuming $f > 1$) ($f = 10$), $\tau = f/a_0(\mathbf{x})$. The number of firings of reaction \mathcal{R}_j during time step τ is then sampled from the binomial random variable $k_j = \mathcal{B}(k_j^*, p)$, where k_j is the number of successes in k_j^* independent Bernoulli trials, where each trial has the probability of success (i.e., it fire) of $p = a_j(\mathbf{x})\tau/k_j^*$. If $a_j(\mathbf{x})\tau > k_j^*$ (i.e., $p > 1$) the resulting probability is coerced to unity resulting in $k_j = k_j^*$ and the extinction of the limiting state. This occurs when a too high coarse-graining factor, f , has been used.

One difference here from the original BTL method of Chatterjee *et al.* (2005) is that if reaction \mathcal{R}_j does not have any limiting populations, the number of firings is sampled from a Poisson distribution with mean $a_j(\mathbf{x})\tau$.

2.5. Optimized tau-leap (OTL)

The optimized tau-leap (OTL) method was introduced by Cao *et al.* (2006). It is an improved procedure that efficiently estimates the largest possible step size, while providing more accurate results than previous approximate methods (Gillespie 2007).

The OTL method partitions the reactions into set of critical and noncritical reactions with \mathcal{J}_c and \mathcal{J}_{nc} being the sets of indices in j of all critical and noncritical reactions ($j = 1, \dots, M$). A reaction is defined as critical if it is within n_c (`nc = 10`) number of firings of depleting any one of its reacting states.

The step size selection procedure in the OTL method is based on bounding the relative change in the propensity functions of the noncritical reactions by the same amount (in contrast to the Leap Condition which evaluates the absolute change in each propensity function separately). The OTL method proceeds by calculating a candidate time step τ_{nc} , estimating the time step to the next noncritical reaction, for which the estimated fractional change $a_j(\mathbf{x})/a_j(\mathbf{X}(t + \tau))$ of noncritical reactions, $j \in \mathcal{J}_{nc}$, is bounded by a user-specified accuracy-control parameter ϵ ($0 < \epsilon \ll 1$) (`epsilon = 0.03`). The largest permissible τ_{nc} is selected indirectly by bounding the estimated fractional change in each state i by an amount $\varepsilon_i = \varepsilon_i(\epsilon, x_i)$ where the function ε_i is chosen based on the highest order of the reaction in which state i is involved (see [Cao et al. 2006](#), for further details).

If the candidate time step τ_{nc} is less than the D multiple (`dtf = 10`) of $1/a_0(\mathbf{x})$, the OTL method is temporarily suspended and the next n_D steps (`nd = 100`) are executed as single reaction time steps using the direct method. If the tau-leaping proceeds, i.e., $\tau_{nc} \geq D/a_0(\mathbf{x})$, a second candidate time step τ_c , estimating the time to the next critical reaction, is generated as a sample of the exponential random variable with mean $1/a_c(\mathbf{x})$, where $a_c = \sum_{j \in \mathcal{J}_c} a_j(\mathbf{x})$. The smaller of the two candidate time steps, τ_{nc} and τ_c , is then chosen as the actual time step τ .

The number of firings of each noncritical reaction k_j is sampled as a Poisson random variable with mean $a_j(\mathbf{x})\tau$, where $j \in \mathcal{J}_{nc}$. If $\tau_c \leq \tau_{nc}$ a single critical reaction j_c fires once during the current time step. The reaction j_c is selected as a sample of the integer random variable with point probabilities $a_j(\mathbf{x})/a_c(\mathbf{x})$ where $j \in \mathcal{J}_c$. If $\tau_{nc} < \tau_c$ no critical reactions are allowed to fire during the current time leap.

3. Implementing the SSA in R

The SSA framework described in this paper is implemented using the R package **GillespieSSA** (Gillespie’s Stochastic Simulation Algorithm, [Pineda-Krch 2008](#)), which is available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=GillespieSSA>. The package is designed to be simple and intuitive to use and is aimed at facilitating rapid prototyping, deployment, and evaluation of continuous time stochastic models while being extensible and flexible enough to allow the development of larger and more complex models.

Usually Monte Carlo implementations of the SSA are developed using low-level compiled language such as C, C++ or Fortran, e.g., **StochKit** ([Cao et al. 2004a](#)). While this has the advantage of being computationally efficient and enables true code parallelization it also requires the end user to have a high level of technical skill to successfully utilize these routines. Implementing the SSA in a higher level programming language like R, in particular the functions that provide the end-user interface, has the advantage of allowing faster model prototyping, simpler and shorter model code, platform independence, and a more interactive development cycle ([Petzoldt and Rinke 2007](#)). While the simulations will inevitably be slower than in low-level implementations, under certain circumstances, it may be possible to parallelize the

code by distributing independent computations across nodes of a computational cluster, e.g., by using the **snow** (Simple Network of Workstations) package (Rossini *et al.* 2007). Another advantage of implementing the package in R is that it is required to pass validation checks that require a consistent documentation giving the package a certain “permanence” and making it citable (R Development Core Team 2007b).

The main interface function in **GillespieSSA** is `ssa()` which is the function most users will use to launch simulations. The `ssa()` function first checks the consistency of the defined system. Then the function evaluates the propensity functions, sets appropriate default values to SSA arguments that have not been defined by the user, and runs a single Monte Carlo realization using the requested SSA method, `ssa.[method name]()`. When the simulation terminates `ssa()`, returns a list object containing the generated time series, a sub-list of the initial system definitions, and a sub-list of various simulation statistics. For a full description of all the options see Pineda-Krch (2008).

It is possible to directly invoke the method functions `ssa.[method name]()`. If all of the functionality of the higher-level interface function `ssa()` are not required it may be possible to optimize the performance of the simulation code by directly accessing the necessary method function. Because the method functions execute one time step at a time, events taking place between subsequent time steps (e.g., incrementing the time, updating the state vector, re-evaluating the propensity functions, and recording the current state of the system) have to be implemented separately by the user. The package manual (Pineda-Krch 2008) provides examples of how such simulations can be set up.

Usually the first step in implementing a stochastic model within the framework of SSA theory is to formulate the deterministic model, the set of coupled ordinary differential equations that define the instantaneous rate of change of the population densities in the different states (see for example Equations 1, 2 and 4 below). Although it is possible to formulate the stochastic model without first having a deterministic model, the advantage of starting out with the deterministic formulation is that one can generate predictions for the qualitative dynamics of the model. For example, a local stability analysis can determine parameter regions that would be interesting to explore using the stochastic model.

The next step is to identify the reactions and derive their associated quantities — the propensity functions and the state-change vectors — both of which can be derived from the deterministic model. In the propensity functions, the state variables are no longer densities, but the actual (finite) numbers of individuals at any given point in time. Once these quantities have been defined, the stochastic model can be specified within the framework of **GillespieSSA** with the desired parameters and initial conditions.

4. Example models

To illustrate how to define, set up, and run different types of models using the **GillespieSSA** package, three different biological models are used as examples. Two of the models are classical ecological models, the logistic-growth model (Kot 2001) and the Rosenzweig-MacArthur predator-prey model (Pineda-Krch *et al.* 2007) while the third model is an epidemiological metapopulation model based on the Kermack-McKendrick SIRS model (Brown and Rothery 1994). The models span a broad range of complexity, ranging from the simple logistic model to the dynamically complex predator-prey model and the more complex formulation of the

metapopulation SIRS model.

4.1. Logistic growth

The classical logistic-growth model assumes that the growth of a population decreases with increasing population size and is given by the following equation,

$$\frac{dN}{dt} = rN \left(1 - \frac{N}{K}\right) \quad (1)$$

where N is the population density, K is the carrying capacity of the environment, and r is the intrinsic per capita growth rate of the population. The model consists of two reactions, birth (\mathcal{R}_1) and death (\mathcal{R}_2). Given $r = b - d$ where b and d are the birth and death rates and assuming $\mathbf{x} = \mathbf{X}(t)$, the propensity functions for the two reactions are defined from Equation 1 as $a_1(\mathbf{x}) = bN$ and $a_2(\mathbf{x}) = (d + rN/K)N$ (Pineda-Krch *et al.* 2007), and the state-change vector is given by $\boldsymbol{\nu} = (+1, -1)$.

Assuming $b = 2$, $d = 1$, $K = 1000$ and $\mathbf{X}(0) = (500)$, the initial state vector, the state-change matrix, and the vector of propensity functions can now be defined in the **GillespieSSA** framework as

```
R> parms <- c(b = 2, d = 1, K = 1000)
R> x0     <- c(N = 500)
R> nu     <- matrix(c(+1, -1), ncol = 2)
R> a      <- c("b*N", "(b+(b-d)*N/K)*N")
```

Note that the elements in the initial state vector `x0` are named using the same notation as the state variable in the propensity vector `a` (in this case `N`). Additionally, the state-change matrix `nu` is defined as a matrix even for one-dimensional systems, i.e., systems having a single species or a single reaction, or both. The rows in `nu` correspond to the initial state vector `x0` and the columns correspond to the elements in the vector of propensity functions `a`. The elements of `a` are the individual propensity functions for the M reactions and are defined as character elements using the same notation for the state variables as in `x0`.

Perhaps the simplest way to run this model using the direct method, `method = "D"`, for 10 time units, `tf = 10`, is by invoking `ssa()` as follows

```
R> tf      <- 10
R> method  <- "D"
R> simName <- "Logistic growth"
R> out     <- ssa(x0, a, nu, parms, tf, method,
                  simName, verbose = TRUE, consoleInterval = 1)
```

which generates the following output (middle part is omitted),

```
Running D method with console output every 1 time step
Start wall time: 2007-09-05 15:57:25...
t=0 : 500
(0.37s) t=1.000318 : 764
(1s) t=2.000523 : 870
```



```

...cut...
(8.32s) t=9.001437 : 912
t=10.00018 : 991
-----
tf: 10.00018
TerminationStatus: finalTime
Duration: 9.7 seconds
Method: D
Nr of steps: 36615
Mean step size: 0.0002731169+/-0.0002866836
End wall time: 2007-09-05 15:57:34

```

By default the `ssa()` runs in silent mode, without displaying output on the console during the run. Setting `verbose = TRUE` enables console output during the simulation while `consoleInterval = 1` sets the interval (in simulation time) between updates.

The return value from `ssa()` is a list object, named `out` in the previous example, containing the time series of the population ($t, \mathbf{X}(t)$) in the list `out$data`, various simulation statistics in `out$stats`, and the formal arguments to the function `ssa()` in `out$args`.

Figure 2A shows a time series for the logistic model using the different SSA methods. The results show that while there are substantial difference between the many small time steps in the direct method and the few and large time leaps of the approximate methods, all methods generate results that agree well with the predicted trajectory of the deterministic model (solid line in Figure 2A). The distributions of population sizes from 10,000 realizations of each method are virtually indistinguishable (Figure 2B) and are all centered on the predicted equilibrium population size ($K = 1000$).

4.2. Predator-prey model

The Rosenzweig-MacArthur predator-prey model has density-dependent growth in the prey (just as in the logistic-growth model of Equation 1) and a nonlinear type-2 functional response in the predator where the number of prey consumed by the predator per time unit saturates at high prey densities (Rosenzweig and MacArthur 1963; Kot 2001). The deterministic model is given by the following pair of coupled differential equations,

$$\begin{aligned}
 \frac{dN}{dt} &= rN \left(1 - \frac{N}{K}\right) - \frac{\alpha N}{1 + wN} P \\
 \frac{dP}{dt} &= P \left(c \frac{\alpha N}{1 + wN} - g\right)
 \end{aligned}
 \tag{2}$$

Here N and P are the densities of prey and predator, respectively, b and d are the intrinsic per capita birth and death rates of the prey, K is the carrying capacity of the prey, α is the predation efficiency, c is the conversion efficiency of the predator (given by the average number of predator offspring produced per consumed prey) and g is the per capita death rate of the predator. The parameter w measures the degree of predator saturation. When $w > 0$, the rate of prey consumption by the predator gradually increases as prey density increases, exhibiting a diminishing return before eventually levelling off at α/w . The limit $w = 0$ corresponds to a linear functional response, allowing the consumption rate to increase indefinitely in proportion to prey density.

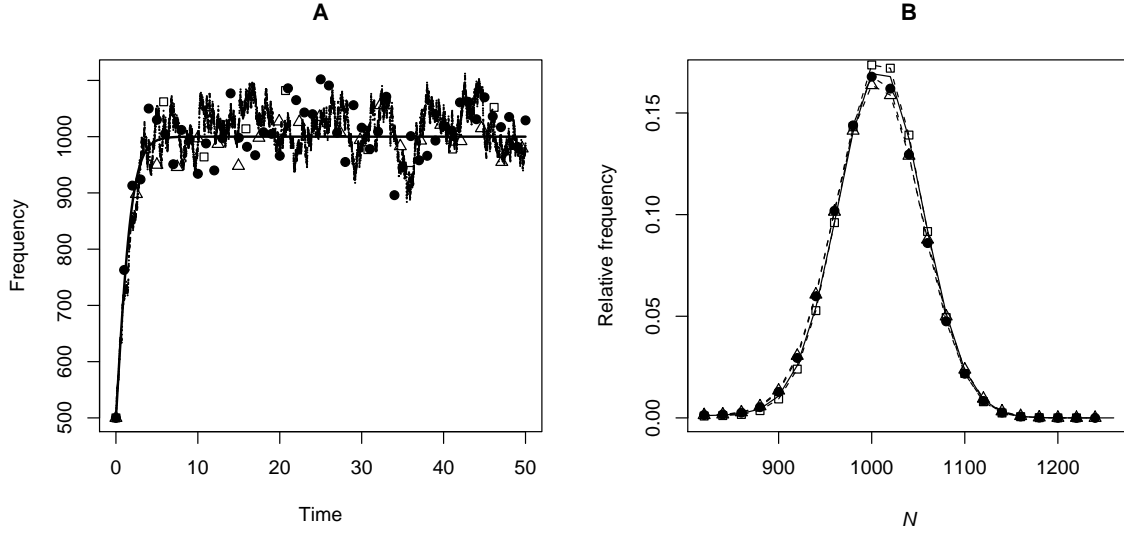


Figure 2: A. Individual realizations of the logistic-growth model using the direct method (\cdot), explicit tau-leap method (\bullet), binomial tau-leap method (\square), and optimized tau-leap method (\triangle). The solid black line is the deterministic trajectory. Each point represents 20 time steps. B. Distribution of relative population sizes from 10000 realizations for each of the SSA methods (each realization ran for 100 time units). Same markers as in panel A except for the direct method that is indicated with a solid line without markers.

This model consists of five reactions: prey birth (\mathcal{R}_1); prey death due to non-predatory events (\mathcal{R}_2); prey death due to predation (\mathcal{R}_3); predator birth (\mathcal{R}_4); and predator death (\mathcal{R}_5). Assuming $r = b - d$ (see logistic-growth model), the propensity functions are defined as $a_1(\mathbf{x}) = bN$, $a_2(\mathbf{x}) = d + r(N/K)N$, $a_3(\mathbf{x}) = \alpha/(1 + wN)NP$, $a_4(\mathbf{x}) = c\alpha/(1 + wN)NP$, and $a_5(\mathbf{x}) = gP$ (Pineda-Krch *et al.* 2007) and the state-change matrix,

$$\boldsymbol{\nu} = \begin{bmatrix} +1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & +1 & -1 \end{bmatrix}. \quad (3)$$

Assuming $b = 2$, $d = 1$, $K = 1000$, $\alpha = 0.007$, $w = 0.0035$, $c = 2$, $g = 2$, and $\mathbf{X}(0) = (1000, 100)$ the model can be defined as,

```
R> parms <- c(b = 2, d = 1, K = 1000, alpha = 0.007,
+           w = 0.0035, c = 2, g = 2)
R> x0 <- c(N = 1000, P = 100)
R> nu <- matrix(c(+1, -1, -1, 0, 0,
+               0, 0, 0, +1, -1),
+             nrow = 2, byrow = TRUE)
R> a <- c("b*N",
+        "(d+(b-d)*N/K)*N",
+        "alpha/(1+w*N)*N*P",
+        "c*alpha/(1+w*N)*N*P",
+        "g*P")
```

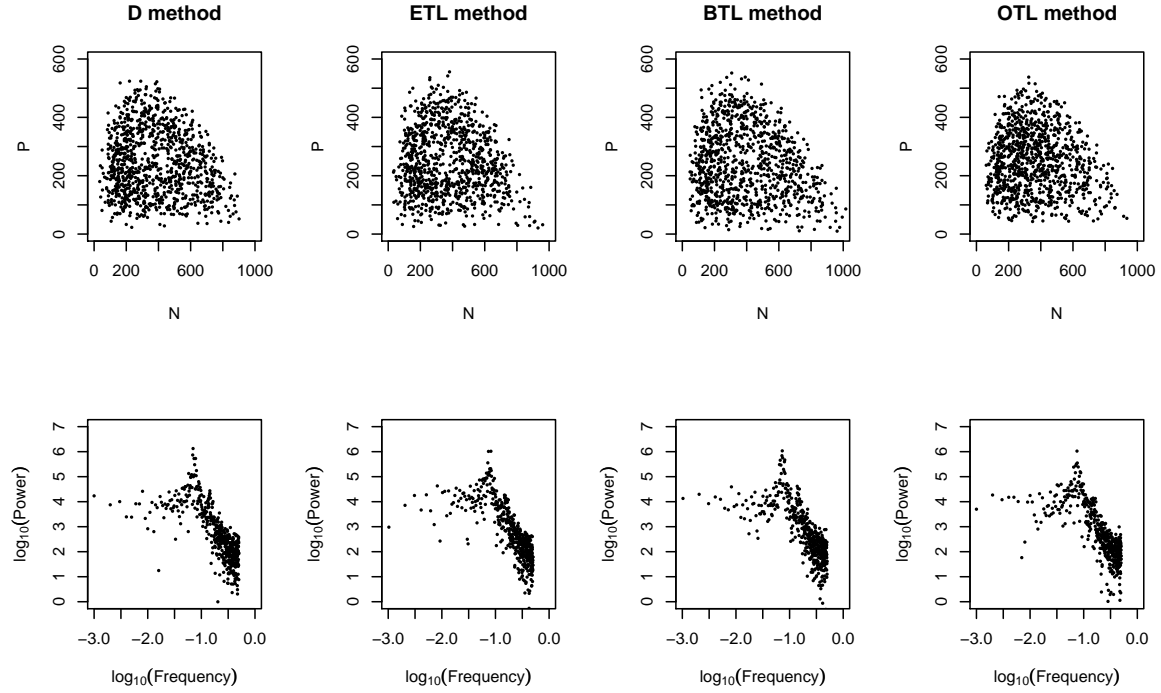


Figure 3: Top panels: Phase plane trajectory of the Rosenzweig-MacArthur predator-prey model for different SSA methods (D – direct method, ETL – explicit tau-leap method, BTL – binomial tau-leap method, OTL – optimized tau-leap method). Bottom panels: Corresponding power spectra for the predator populations (the power spectra for the prey populations are identical). The simulations were run for 1000 time units and were censused every time unit (`censusinterval = 1`). Data are same as in Table 1.

The following command runs this model for 100 time units using the direct method with console output every 10 time units and a limit of the simulation duration to a maximum wall time of 30 seconds:

```
R> out <- ssa(x0, a, nu, parms, tf = 100, method,
+           simName = "Predator-prey model",
+           verbose = TRUE,
+           consoleInterval = 10,
+           maxWallTime = 30)
```

Figure 3 shows the phase planes and power spectra of four realizations of the predator-prey model, one for each SSA method. The distribution of sample points in the phase plane and the spectral distribution of the dominant frequency are indistinguishable among all the SSA methods. Because the populations were sampled every time unit (rather than every time step as in Figure 2), the different method have the same number of data points (1000) while the actual number of time steps executed spans three orders of magnitude with a corresponding difference in the execution times (Table 1).

4.3. SIRS metapopulation model

The classical Kermack-McKendrick SIRS model (Susceptible, Infectious, Recovered, Susceptible) is a compartment model of the number of individuals infected with a contagious illness in a closed population over time (Kermack and McKendrick 1927). It assumes that: population size is constant (i.e., no births or deaths); there is no incubation period of the infectious agent; the duration of infectivity is same as length of the disease; and individuals losing immunity join the susceptible class again. In this version of the model, inspired by the linear chain system by Cao *et al.* (2004b), the population is structured into a number of patches, each one of which has a constant population size P . Disease transmission occurs both within patches (intra-patch) and between patches (inter-patch), and individuals in the immune compartment lose their immunity at a certain rate and become susceptible again. Assuming that transmission only occurs between adjacent patches in the clock-wise direction, with the last patch in the chain infecting the first patch, this model can be described by the following coupled differential equations,

$$\begin{aligned} \frac{dS_i}{dt} &= -\beta S_i [(1-\epsilon)I_i + \epsilon I_{(i \bmod U+1)}] + \rho(P - S_i - I_i) \\ \frac{dI_i}{dt} &= \beta S_i [(1-\epsilon)I_i + \epsilon I_{(i \bmod U+1)}] - \gamma I_i. \end{aligned} \quad (4)$$

Here S_i is the number of susceptible individuals and I_i is the number of infected individuals in patch i , $i = (1, \dots, U)$, ϵ is the probability of inter-patch contact, $(1-\epsilon)$ is the probability of intra-patch contact, β is the transmission coefficient, γ is the disease clearance rate, and ρ is the rate at which immunity is lost. Because each patch has a constant population size, the recovered state (R) does not have to be treated explicitly.

Each patch consists of two states, $N = 2$ (susceptible and infectious), and four reactions, $M = 4$. Thus, in a system with U patches, there are $2U$ number of states and $4U$ number of reaction. Each patch v , $v = (1, \dots, U)$, has the following four reactions: intra-patch transmission ($\mathcal{R}_{1+(v-1)U}$); inter-patch transmission ($\mathcal{R}_{2+(v-1)U}$); loss of infectiousness ($\mathcal{R}_{3+(v-1)U}$); and loss of immunity ($\mathcal{R}_{4+(v-1)U}$). The corresponding propensity functions are $a_{1+(v-1)U}(\mathbf{x}) = (1-\epsilon)\beta S_i I_i$, $a_{2+(v-1)U}(\mathbf{x}) = \epsilon\beta S_i I_{(i \bmod U+1)}$, $a_{3+(v-1)U}(\mathbf{x}) = \gamma I_i$, and $a_{4+(v-1)U}(\mathbf{x}) = \rho(P - S_i - I_i)$.

The state-change matrix for a system with U patches is a $2U \times 4U$ block diagonal matrix

$$\boldsymbol{\nu} = \begin{bmatrix} \tilde{\boldsymbol{\nu}}_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \tilde{\boldsymbol{\nu}}_U \end{bmatrix} \quad (5)$$

where the submatrices on the diagonal are the state-change matrices for each individual patch,

$$\tilde{\boldsymbol{\nu}}_v = \begin{bmatrix} -1 & -1 & 0 & +1 \\ +1 & +1 & -1 & 0 \end{bmatrix}. \quad (6)$$

If the dimensions $n_{\boldsymbol{\nu}} \times m_{\boldsymbol{\nu}}$ of the state-change matrix $\boldsymbol{\nu}$ are smaller than the dimensions of the state vector $n_{\mathbf{x}}$ and the propensity vector $m_{\mathbf{a}}$, `ssa()` checks if $\boldsymbol{\nu}$ is a diagonal block matrix. If it is, a separate set of SSA methods are invoked, `ssa.[method].diag()`. These methods define a virtual full sized $NU \times MU$ state-change matrix by mapping reaction j and state i to the smaller $N \times M$ state-change matrix $\tilde{\boldsymbol{\nu}}$. This procedure reduces the number of elements in the

state-change matrix by a factor of U^2 which simplifies the definition of the matrix and reduces memory requirements, particularly for large systems. This virtualization of the state-change matrix has (to my best knowledge) previously not been described in the literature. Further work is required to establish the extent to which it improves the computational efficiency of different Monte Carlo implementations of the SSA.

Assuming $U = 100$, $P = 500$, and a single infectious individual at $t = 0$, the initial state vector can be defined in a vectorized form, first by assigning the population sizes for the $2U$ states and then by assigning each element the corresponding name,

```
R> U          <- 100
R> N          <- 500
R> x0         <- c((N-1), 1, rep(0,(2*U)-2))
R> names(x0)  <- c(paste(c("S", "I"), floor(seq(1, (U+0.5), 0.5))), sep = ""))
```

Similarly the propensity functions are created iteratively,

```
R> a <- NULL
R> for (patch in (seq(U))) {
+   i <- patch          # Intra-patch index
+   if (patch == 1) j <- U # Inter-patch index
+   else j <- patch-1
+   a_patch <- c(paste("(1-epsilon)*beta*S", i, "*I", i, "", sep = ""),
+               paste("epsilon*beta*S", i, "*I", j, sep = ""),
+               paste("gamma*I", i, sep = ""),
+               paste("rho*(N-S", i, "-I", i, ")", sep = ""))
+   a <- c(a, a_patch)
+ }
```

Finally the parameters are defined and the simulation is run for 10 time units using the explicit tau-leap method (`method="ETL"`),

```
R> parms <- c(beta = .001, gamma = .1, rho = .005, epsilon = .01, N = 500)
R> out    <- ssa(x0, a, nu, parms, tf = 10, method = "ETL",
+               simName = "SIRS patch model")
```

Figure 4 illustrates the spread of the outbreak through the chain of patches using the different SSA methods. While there is no perceptible difference in the rate of spread and the extent of the outbreak, there are as expected substantial differences in the computational timings (Table 1) with the binomial tau-leap method running two orders of magnitude faster than the direct and optimized tau-leap methods. The slowest method was the optimized tau-leap method, even though it had 7% fewer time steps than the direct method. A closer inspection of the simulation statistics reveals that the majority of time steps were suspended tau-leaps. Evaluating

```
R> out$stats$nSuspendedTauLeaps / out$stats$nSteps
```

```
[1] 0.99075
```

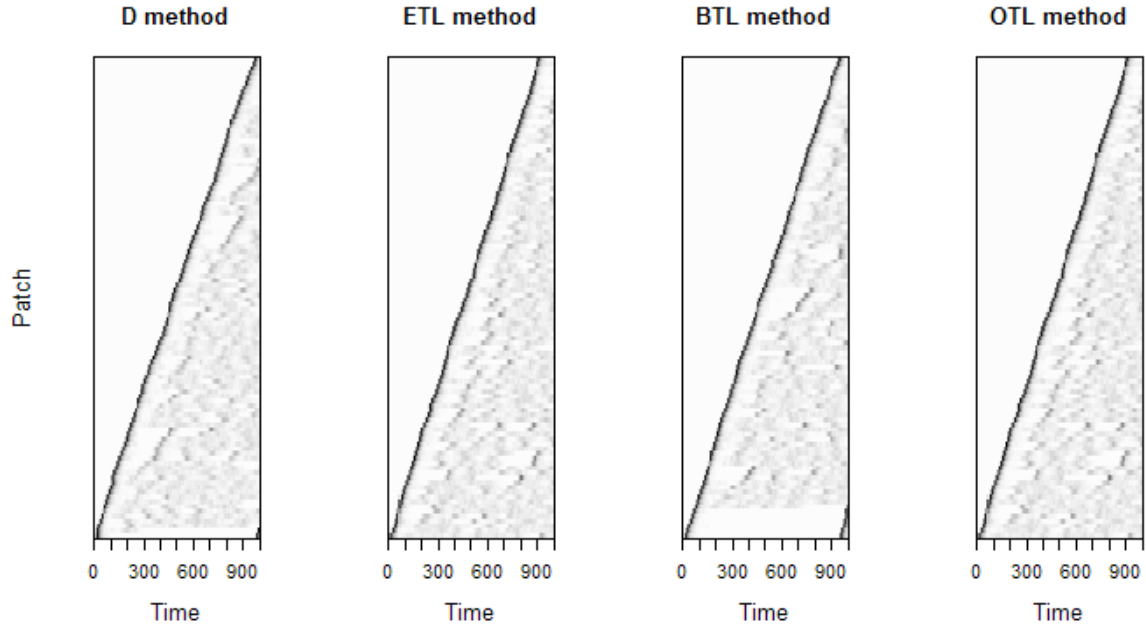


Figure 4: Individual realizations of the SIRS metapopulation model using the four different SSA methods (same abbreviations as in Figure 3). The intensity of the grey shading indicates the relative number of individuals that are in the infectious class. The data are the same as in Table 1 (here showing only the first 1000 time units).

reveals that 99% of the time steps were performed using the direct method, which occurs when $\tau_{nc} < D/a_0(x)$ (see description of the OTL method in Section 2.5).

5. Discussion

Using three biological models as examples, this paper shows how to define stochastic models within the framework of SSA theory and implement them in R using the **GillespieSSA** package. The results show that while all implemented SSA methods generate consistent and virtually identical results they exhibit large differences in computational efficiency. The simulation results confirm that the approximate methods can be several orders of magnitude faster than the exact method, while generating results that are indistinguishable from its results. This suggests that approximate methods are good alternatives to the direct method, even for models exhibiting complex dynamics such as large-amplitude population cycles.

The relative timing performance of the approximate methods, however, is not as consistent. While the binomial tau-leap method outperforms the other methods in the logistic-growth model, it performs relatively poorly in the predator-prey and metapopulation model. Similarly, the optimized tau-leap method performs very well in the predator-prey model but is slower than the direct method for the metapopulation model (Table 1). This suggests that while approximate methods will perform better than the direct method the majority of the

	Logistic-growth model			
	D	ETL	BTL	OTL
Duration scaled to D	1	13969	63904	899
Elapsed WT	60070	4.3	0.94	66.8
Mean step size	0.25	50	251	15
Nr of steps	3,980,814	20001	3980	67267

	Predator-prey model			
	D	ETL	BTL	OTL
Duration scaled to D	1	95	111	5019
Elapsed WT	25096	264	227	5
Mean step size	0.5	50	52	110
Nr of steps	1,915,828	200,001	192,213	3,153

	Metapopulation SIRS model			
	D	ETL	BTL	OTL
Duration scaled to D	1	53	5	1
Elapsed WT	12560	241	2478	12699
Mean step size	2.2	100	2.2	2.4
Nr of steps	909,254	20,001	89,040	850,164

Table 1: Simulation statistics of the example models using the different SSA method (same abbreviations as in Figure 3). A single realization was run for each model-method combination. The logistic-growth model and the predator-prey model were run for 1000 time units (same data as in Figure 3), and the SIRS metapopulation model was run for 2000 time units with $U = 100$ (same data as in Figure 4). Elapsed wall time (WT) is in seconds, mean step size is in 1/1000 simulation time unit, and step duration is in 1/1000 second. All methods were run with their default SSA arguments (see [Pineda-Krch 2008](#)).

time, selecting the most efficient approximate method is not straight forward. To date, few studies have addressed the significance of appropriate method selection for models exhibiting a wide range of dynamical behaviours.

All of the simulations in this paper have used the default arguments of the `ssa()` function. Several of the methods, however, have additional formal arguments that can be used to optimize the performance of the method, both in terms of speed and accuracy (see [Pineda-Krch 2008](#), for more details). For example, increasing the step size (`tau`) in the explicit tau-leap method will invariably result in fewer total steps. The corresponding increase in performance is bought, however, at the expense of accuracy in the results. This raises the important question of how to assess the accuracy of simulation results obtained from approximate SSA methods. One possible approach is to compare the results from the stochastic simulations to the predictions of the corresponding deterministic formulation, i.e., the numerical or analytical solutions of the set of coupled ordinary differential equations (ODEs) (see Figure 2). This approach may not, however, be feasible in large-scale systems with complex dynamics, such as the SIRS metapopulation model, and it also assumes that the dynamics of the stochastic model are the same as those of the deterministic formulation, which is not always the case

(Pineda-Krch *et al.* 2007). An alternative method commonly used is to obtain an ensemble of trajectories by running many individual realizations (each one with a different seed for the random number generator) and comparing the probability density functions of the state variables to results obtained using an exact method (see Figure 2B). Although the comparison of the distributions is often visual (Gillespie 2001; Chatterjee *et al.* 2005; Cao *et al.* 2006) quantitative comparisons of the distance between the distributions is also possible (Cao *et al.* 2004a).

6. Future development

Future developments of the package will occur in two main directions. Firstly, recoding of the method functions in a lower-level compiled language to improve the computational efficiency. Secondly, the addition of new method functions, particularly multiscale SSA methods optimized for stiff systems, i.e., a continuous time system characterized by well-separated fast and slow dynamical modes, the fastest of which is stable (Gillespie 2007). As part of this process user contributions and improvements are welcomed, e.g., improvements to existing methods as well as the addition of new methods (stone soup).

Computational details

The logistic-growth and the SIRS metapopulation models were run on a 20-node Linux cluster (SUSE Linux 9.3), on which each node has dual 2.4GHz AMD Opteron CPUs and 12GB RAM. The predator-prey model was run using Windows XP on a Lenovo laptop with a 1.99GHz Intel Core 2 processor and 2GB RAM. All simulations were performed using R 2.5.1 (R Development Core Team 2007a) with the Mersenne-Twister random number generator. The example models are available in the package as demonstration models. Individual realizations can be run using

```
R> demo("logisticGrowth", package = "GillespieSSA")
R> demo("rma", package = "GillespieSSA")
R> demo("epiChain", package = "GillespieSSA")
```

Note that some of the parameter values in the demonstration models differ from those presented in the paper and were chosen to make the simulations shorter and more manageable. An R script that employs exactly the same parameter settings as for the paper is available along with this paper. Both R and **GillespieSSA** are available from the Comprehensive R Archive Network at <http://CRAN.R-project.org/>.

Acknowledgments

Heinrich zu Dohna, Ben Bolker, Josh O'Brien, Thomas Petzoldt, and three anonymous reviewers for constructive feedback on the **GillespieSSA** package and comments on the manuscript. Parts of the simulations were performed on the Center for Animal Disease Modeling and Surveillance Linux cluster at University of California, Davis.

References

- Arkin A, Ross J, McAdams HH (1998). “Stochastic Kinetic Analysis of a Developmental Pathway Bifurcation in Phage- λ E. coli Cell.” *Genetics*, **149**, 1633–1648.
- Brown D, Rothery P (1994). *Models in Biology: Mathematics, Statistics and Computing*. John Wiley & Sons.
- Cao Y, Gillespie DT, Petzold LR (2006). “Efficient Step Size Selection for the Tau-Leaping Simulation Method.” *Journal of Chemical Physics*, **124**, 044109.
- Cao Y, Hall A, Li H, Lampoudi S, Petzold L (2004a). *User’s Guide to **StochKit***. Computational Science and Engineering, UCSB.
- Cao Y, Li H, Petzold L (2004b). “Efficient Formulation of the Stochastic Simulation Algorithm for Chemically Reacting Systems.” *Journal of Chemical Physics*, **121**, 4059–4067.
- Chatterjee A, Vlachos DG, Katsoulakis MA (2005). “Binomial Distribution Based τ -leaping Accelerated Stochastic Simulation.” *Journal of Chemical Physics*, **122**(024112), 1–7.
- Dushoff J, Plotkin JB, Levin SA, Earn DJD (2004). “Dynamic Resonance Can Account for Seasonality in Influenza Epidemics.” *Proceedings of the National Academy of Sciences (USA)*, **101**, 16915–16916.
- Fedoroff NV, Fontana W (2002). “Small Numbers of Big Molecules.” *Science*, **297**, 1129–1131.
- Gibson MA, Bruck J (2000). “Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels.” *Journal of Physical Chemistry*, **105**, 1876–1889.
- Gillespie DT (1976). “A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions.” *Journal of Computational Physics*, **22**, 403–434.
- Gillespie DT (1977). “Exact Stochastic Simulation of Coupled Chemical Reactions.” *Journal of Physical Chemistry*, **81**, 2340–2360.
- Gillespie DT (2001). “Approximate Accelerated Stochastic Simulation of Chemically Reacting Systems.” *Journal of Chemical Physics*, **115**, 1716–1733.
- Gillespie DT (2007). “Stochastic Simulation of Chemical Kinetics.” *Annual Review of Physical Chemistry*, **58**, 35–55.
- Kermack WO, McKendrick AG (1927). “A Contribution to the Mathematical Theory of Epidemics.” *Proceedings of the Royal Society, London A*, **115**, 700–721.
- Kot M (2001). *Elements of Mathematical Ecology*. Cambridge University Press.
- McAdams HH, Arkin A (2005). “Stochastic Mechanisms in Gene Expression.” *Proceedings of the National Academy of Sciences (USA)*, **94**, 814–819.
- McKane AJ, Newman TJ (2005). “Predator-Prey Cycles From Resonant Amplification of Demographic Stochasticity.” *Physics Review Letters*, **94**, 218102.

- Nisbet RM, Gurney WSC (1976). “A Simple Mechanism for Population Cycles.” *Nature*, **263**, 319–320.
- Petzoldt T, Rinke K (2007). “**simecol**: An Object-Oriented Framework for Ecological Modelling in R.” *Journal of Statistical Software*, **22**(9), 1–31. URL <http://www.jstatsoft.org/v22/i09/>.
- Pineda-Krch M (2008). *GillespieSSA: Gillespie’s Stochastic Simulation Algorithm (SSA)*. R package version 0.5-2, URL <http://CRAN.R-project.org/package=GillespieSSA>.
- Pineda-Krch M, Blok H, Dieckmann U, Doebeli M (2007). “A Tale of Two Cycles: Distinguishing Quasi-Cycles and Limit Cycles in Finite Predator-Prey Populations.” *Oikos*, **116**, 53–64.
- R Development Core Team (2007a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org/>.
- R Development Core Team (2007b). *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-11-9, URL <http://www.R-project.org/>.
- Rosenzweig ML, MacArthur RH (1963). “Graphical Representation and Stability Conditions of Predator-Prey Interactions.” *The American Naturalist*, **97**, 209–223.
- Rossini AJ, Tierney L, Li N (2007). “Simple Parallel Statistical Computing in R.” *Journal of Computational and Graphical Statistics*, **16**, 399–420.
- Tian T, Burrage K (2004). “Binomial Leap Methods for Simulating Stochastic Chemical Kinetics.” *Journal of Chemical Physics*, **121**, 10356–10364.

Affiliation:

Mario Pineda-Krch
 Center for Animal Disease Modeling and Surveillance
 Department of Medicine and Epidemiology
 School of Veterinary Medicine
 University of California, Davis
 95616, CA, United States of America
 E-mail: mpineda@ucdavis.edu
 URL: <http://pineda-krch.com/>