

# From ODEs to ABMs: stochastic simulation software for a broad class of models in ecology and other fields

Edward B. Baskerville  
Department of Ecology and Evolutionary Biology  
University of Michigan  
ebaskerv@umich.edu

9 May 2009

## 1 Introduction

For ecologists and scientists in many other fields, differential equations are the simplest place to start when formulating models, but they quickly become insufficient for many problems. For example, in ecology, the discrete, unpredictable nature of individual organisms can fundamentally alter model results. Local interactions of organisms in space can have similarly drastic effects, and heterogeneity among individuals may also be important. The first problem can be tackled with the same simulation approaches used in chemical kinetics (the Gillespie algorithm and approximations thereof), and implementation is relatively straightforward. The second and third problems require the development of much more complex code, but the software engineering challenges can be solved with a general-purpose toolkit.

In essence, stochastic models of this kind can be thought of as a number of simultaneous Poisson processes, where the rates of some subset of the processes change after every event. The simulation problem is thus reduced to two basic steps for each event: (1) calculating the time until the next event and the identity of the event based on the rates of all the processes; and (2) updating rates that are affected by the state change. While simple conceptually, it is quite difficult to efficiently implement this process for spatial, individual-based models, where each possible state transition of each individual in the system is a different Poisson process. However, given a description of the possible transitions and the relationship of transition rates to densities of different states, a software toolkit can handle all the difficulties of actually simulating the system.

Here I describe a new software toolkit, **bookfour**, for simulating this class of model, using ecological processes as guiding examples. The toolkit provides a natural way to

successively relax assumptions, starting with the continuous, well-mixed structure of mean-field ODE models, moving to stochastic models with discrete individuals, and finally adding spatial structure and heterogeneous individuals. Simple models are implemented in a straightforward declarative style that closely mirrors the mathematical formulation, and spatial structure can easily be added to non-spatial models. In general, the toolkit makes it possible to represent many other kinds of behavior, thus providing a framework for a broad class of stochastic agent-based models.

## 2 The mathematical framework

Many ecological models can be formulated as sets of simple state transitions akin to chemical reactions, where objects of one type are transformed into objects of another type. For example, the simple SIR disease model can be defined as two transitions between three states, susceptible (S), infected/infectious (I), and recovered (R):



That is, susceptible individuals (state  $S$ ) are converted to infected/infectious individuals (state  $I$ ) at rate  $\beta N_I/N$ , where  $N_I$  is the number of individuals in state  $I$ , and  $N$  is the total number of individuals in the system; and infected/infectious individuals recover at constant rate  $\nu$ .

The famous Lotka-Volterra predator-prey model can be represented in a similar fashion, but rather than individuals transitioning between distinct states, the representation includes the creation and destruction of both prey ( $R$ , for “resource”) and predator ( $C$ , for “consumer”) individuals:



A natural first step is to realize these state transitions in systems of ordinary differential equations (ODEs).

### 2.1 Realization as systems of ODEs

In ODE form, the SIR model becomes

$$\frac{dN_S}{dt} = -\beta \frac{N_I(t)}{N} N_S(t) \qquad \frac{dN_I}{dt} = \beta \frac{N_I(t)}{N} N_S(t) - \nu N_I(t) \qquad \frac{dN_R}{dt} = \nu N_I(t)$$

the equation for  $N_R$  being redundant, since  $N_S + N_I + N_R$  remains constant. Similarly, the Lotka-Volterra system becomes

$$\frac{dN_R}{dt} = \alpha N_R(t) - \beta N_R(t) N_C(t) \qquad \frac{dN_C}{dt} = \delta N_R(t) N_C(t) - \gamma N_C(t)$$

Using an ODE representation implicitly makes a number of assumptions:

1. The population behaves as if it is infinite, or, equivalently, the transition, creation, and destruction events happen in infinitesimal chunks over infinitesimal periods of time, that is, the population is *continuous*.
2. The rates depend only on the global number of individuals in different states, not on local interactions in some spatial or network topology. That is, the population is *well-mixed*.
3. The rates do not depend on characteristics of individuals. That is, the population is *homogeneous*.

In some systems, these assumptions may be reasonable, and the ODE model may be enough. In others, it is clear that the assumptions are violated, but there is still substantial value in starting with an ODE representation to keep things simple and more amenable to analysis. These assumptions can then be systematically relaxed to create more complex, computationally intensive models that can be naturally compared to simpler versions.

## 2.2 Making things discrete

In many cases, the discrete nature of individuals in the system is of fundamental importance to dynamics (Durrett and Levin 1994) [1], and the appropriate formalism is a discrete-event stochastic process in continuous time that tracks the number of individuals in different states. Instead of a description of rates of change of the numbers, we can write a set of transition probabilities over time, for example, the probability that an individual becomes infected in a small period of time in the SIR model.

The SIR model simply becomes this set of transition probabilities, which maps perfectly onto the ODE representation:

$$\begin{aligned}
 P[N_S(t + dt) = N_S(t) - 1, N_I(t + dt) = N_I(t) + 1] &= \beta \frac{N_I(t)}{N} N_S(t) dt \\
 P[N_I(t + dt) = N_I(t) - 1, N_R(t + dt) = N_R(t) + 1] &= \nu N_I(t) dt
 \end{aligned}$$

Likewise, the Lotka-Volterra model becomes this:

$$\begin{aligned}
 P[N_R(t + dt) = N_R(t) + 1] &= \alpha N_R(t) dt \\
 P[N_R(t + dt) = N_R(t) - 1] &= \beta N_R(t) N_C(t) dt \\
 P[N_C(t + dt) = N_C(t) + 1] &= \delta N_R N_C dt \\
 P[N_C(t + dt) = N_C(t) - 1] &= \gamma N_C dt
 \end{aligned}$$

The probability of the system being in different states over time can sometimes be solved explicitly (see Kot 2001 [2] for a readable treatment of master equations for stochastic birth-death processes). Whether an analytical solution is possible or not, it is often desirable to simulate realizations of the process, as described in section 3.

## 2.3 Making things spatial

Well-mixedness may also be an appropriate assumption for some situations, but in many cases processes happen on a scale much smaller than the system under consideration. A simple extension of the stochastic SIR model is to assume that individuals are connected to each other via some network topology, either a regular lattice or a more complex structure, and that infections happen only between connected individuals—you can only get sick from your neighbors. Now, the state of every site must be tracked individually, and transition probabilities between different states are different at every site, depending on the state of that site's neighbors. Transition probabilities for some site  $i$  are thus

$$P[s_i(t + dt) = I | s_i(t) = S] = \beta \frac{N_{i,I}(t)}{N_i} dt$$
$$P[s_i(t + dt) = R | s_i(t) = I] = \nu dt$$

where  $s_i(t)$  is the state of site  $i$  at time  $t$ , and  $N_{i,I}(t)$  represents the number of neighbors of site  $i$  that are infected at time  $t$ . This type of model is sometimes known as an interacting particle system or a stochastic cellular automaton.

The Lotka-Volterra example does not map as neatly onto a spatial structure—depending on the question being explored, one of many possible representations may make sense. One simple approach is to use a similar representation as the SIR model, where sites can be in three states, prey ( $R$ ), predator ( $C$ ), or empty ( $E$ ), and rather than removing prey to represent death from predation, prey sites are converted directly into predators ( $R \rightarrow C$ ). Another approach is to use higher-order interactions: the conversion of an empty site to a predator site (predator reproduction) can depend not on the number of neighboring predators, but rather on the total number of potential predator-prey interactions of neighboring predators, which can be written

$$P[s_i(t + dt) = C | s_i(t) = E] = \delta \sum_{j \in n_{i,C}} N_{j,R}$$

where  $n_{i,C}$  is the set of neighbors of site  $i$ .

## 2.4 Adding heterogeneity

Having come this far, it is straightforward to relax the final assumption, that individuals in the system are homogeneous. Continuing with the SIR model, different sites on the lattice may have higher probabilities of transmitting the disease than others. Nothing in the basic representation of the system changes, however: the value of the coefficients  $\beta$  and  $\nu$  are simply allowed to vary across the lattice:

$$P[s_i(t + dt) = I | s_i(t) = S] = \beta_i \frac{N_{i,I}(t)}{N_i} dt$$
$$P[s_i(t + dt) = R | s_i(t) = I] = \nu_i dt$$

It is not difficult to also have the parameters can also vary with time:

$$P[s_i(t + dt) = I | s_i(t) = S] = \beta_i(t) \frac{N_{i,I}(t)}{N_i} dt$$
$$P[s_i(t + dt) = R | s_i(t) = I] = \nu_i(t) dt$$

As long as the timescale of parameter fluctuations (say, seasonal fluctuations in infection rates) is fast compared to the rate of events (infections), the simulation procedure can remain essentially the same. In fact, all the models described here fit comfortably into a single computational framework.

### 3 The computational framework

Gillespie (1976) [3] was the first to describe an exact simulation algorithm for stochastic simulation of chemical kinetics, and, in abstract form, the algorithm works for all the models described above. A cartoon of the Gillespie algorithm goes like this:

1. Choose what will happen next and when it will happen.
2. Make it happen.
3. Repeat.

Gillespie described two ways of accomplishing the first step. The first, typically referred to the “direct method” or just the “Gillespie algorithm,” separates the problem of deciding what will happen and when it will happen into two steps. Which event will happen is simply distributed according to a discrete probability distribution weighted by the probabilities per unit time of each event. The waiting time until the next event is exponentially distributed with a rate parameter equal to the sum of all the probabilities per unit time. To pick an event, then, you simply draw from the weighted discrete distribution, and to pick when it happens, you draw from the exponential distribution. The second method, called the “first reaction method,” maintains a list of future times for each event based on current probabilities (drawn for each event from an exponential distribution), and proceeds by always selecting the the event with the lowest time and then updating the times of changed probabilities.

In practice, using either of these algorithms for complex models requires careful consideration of how to avoid unnecessary computation. A more concrete description of a generalized Gillespie algorithm is as follows:

1. Draw the time of the next event from an exponential distribution with rate parameter equal to the sum of the probabilities per unit time of all possible events.

2. Draw the identity of the event from a discrete probability distribution weighted according to the individual probabilities per unit time. Do this in an efficient manner, taking into consideration that the next step must also be done efficiently.
3. Update the data structures used to generate the identity of the event in an efficient manner, making sure to only update values that actually need to be changed.

Implementing the code to make these things happen efficiently is not a trivial task for the typical modeler. However, the code can be written in a general way and abstracted into an interface that supports all the types of models described above in a natural way.

## 4 The software toolkit: `bookfour`

I am currently developing a new software toolkit, called `bookfour`, that makes it straightforward to implement all the types of stochastic models described here. The design of `bookfour` is guided by two main goals: (1) natural model implementation and (2) efficient simulation. Both of these goals are best met by a design that makes implementing models in a declarative style that closely mirrors the mathematical formulation. By abstracting away the simulation algorithm, not only is the modeler freed from many software engineering complexities, but the toolkit can also be improved without requiring the modification of model code.

The implementation of a simple non-spatial model consists of the following:

1. A set of states, represented as a Java enumeration.
2. A set of transitions between states and their associated rates, defined by one method call for each transition.

Adding spatial structure simply involves setting up one more object and adding it to the model. Heterogeneity is easily added by replacing discrete states with objects of a small class with locally defined rate parameters. Support is provided for other kinds of spatial relationships, such as decoupling the space from the individuals, so that movement and multiple occupancy are possible. Furthermore, models with arbitrarily complex spatial and non-spatial rate relationships can be constructed, but may require the implementor to explicitly specify what events are dependent on what other events. In short, any model that can be expressed as a set of objects and a set of possible events is supported, from simple interacting particle systems to complex stochastic agent-based models.

Listing 1 shows a minimal implementation of the spatial SIR model described above. Note that implementation details will likely change before the initial release, and that more advanced features, such as loading parameters from the command line and callbacks triggered by model events, are supported but not shown. For more complex models, it will make more sense to subclass `StochasticModel` rather than using it directly.

A prototype version of `bookfour` includes Gillespie direct method implementations based on the binary search tree approach of Wong and Easton (1980) [4], the basic rejection method, and an improved rejection method along the lines of Rajasekaran and Ross (1993) [5]. An improved version of the first-reaction method (the “next-reaction” method), due to Gibson and Bruck (2000) [6], is also included. In a future version, I will add the constant-time algorithm due to Matias, Vitter, and Ni (200), which should scale better for more complex models. A number of approximation algorithms for stochastic simulation have been developed (Li et al. 2008) [7] and I will consider including them, although their current form they are only be useful for non-spatial models.

## 5 Summary

A natural way to formulate many models is by defining a set of continuous-time stochastic processes over a set of discrete objects. A natural place to start in the realization such models is a set of ordinary differential equations (ODEs), but a number of assumptions are implicit in such a representation that will not hold for many systems. Successive relaxation of these assumptions quickly leads to discrete-event, continuous-time stochastic models with explicit spatial structure, which present a number of computational challenges. In order to reduce the difficulties of implementing such models, I am developing a new software toolkit, `bookfour`, which abstracts away the details of simulation algorithms with a straightforward Java interface. Furthermore, `bookfour` supports even more complex models, with heterogeneous individuals and arbitrarily complex dependencies among events, making it possible to implement a wide variety of stochastic agent-based models.

## References

- [1] Durrett, Richard and Simon Levin. 1994. The importance of being discrete (and spatial). *Theoretical Population Biology* 46: 363–394.
- [2] Kot, Mark. 2001. *Elements of Mathematical Ecology*. Cambridge, UK: Cambridge University Press.
- [3] Gillespie, Daniel T. 1976. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics* 22: 403.
- [4] Wong, C. K. and M. C. Easton. 1980. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing* 9(1): 111–113.
- [5] Rajasekaran, Sanguthevar and Keith W. Ross. 1993. Fast algorithms for generating discrete random variates with changing distributions. *ACM Transactions on Modeling and Computer Simulation* 3(1): 1–19.

- [6] Gibson, Michael A. and Jehoshua Bruck. 2000. Efficient exact simulation of chemical systems with many species and many channels. *J. Phys. Chem. A* 104(9): 1876–1889.
- [7] Li, Hong, Yang Cao, Linda R. Petzold and Daniel T. Gillespie. 2008. Algorithms and software for stochastic simulation of biochemical reacting systems. *Biotechnology Progress* 24(1): 56–61.

---

**Listing 1** Implementation of the spatial SIR model using bookfour.

---

```
import bookfour.*;
import bookfour.space.*;
import cern.jet.random.engine.*;

public class SIRModel
{
    private static double beta = 1.0;
    private static double nu = 1.0;
    private static int size = 100;
    private static double maxTime = 100.0;
    private static double timeStep = 1.0;
    private enum State { Susceptible, Infected, Recovered }

    public static void main(String[] args)
    {
        RandomEngine rng = new MersenneTwister(new Date());

        model = new StochasticModel<State>(rng);
        model.addTransitionEvent(State.Susceptible, State.Infected,
                                new EasyList<State>(State.Infected), beta);
        model.addTransitionEvent(State.Infected, State.Recovered, null, nu);
        model.setRateScaling(RateScaling.DivideByNeighborhoodSize);

        EnumLattice<State> space = new EnumLattice<State>(size, size,
                BoundaryCondition.Periodic, NeighborhoodType.VonNeumann);
        space.initializeRandom(rng);
        model.setSpace(space);

        double time = 0.0;
        System.err.printf("t\tS\tI\tR\n");
        while(time < maxTime)
        {
            time = model.runFor(timeStep);
            System.err.printf("%f\t%d\t%d\t%d\n", time,
                    model.getCount(State.Susceptible),
                    model.getCount(State.Infected),
                    model.getCount(State.Recovered));
        }
    }
}
```

---