

Comparing three different simulation methods for solving chemical reaction systems

Luca Cacchiani

Dipartimento di Informatica, University of Torino*

June 27, 2007

Abstract

In this paper I introduce a simulator for chemical reaction systems which computes kinetic trajectories using stochastic differential equations (SDEs). This simulator is implemented in Java as a part of Dizzy [Ram05], the chemical kinetics stochastic simulation software developed by the Institute for System Biology. I give a brief survey of the basic concepts of SDEs theory and their numerical solutions, and I explain a method of numerically evaluating these stochastic differential equations in order to compute the solution of some chemical reaction systems. After that I compare this simulator with more traditional ways for solving chemical reactions systems, like using ordinary differential equations (ODEs) or the Gillespie stochastic simulation algorithm (SSA). In addition I show some examples of solved models with the application of these simulators, including the Michaelis-Menten, Schlögl and the Lotka-Volterra model.

1 Introduction

Models of chemically reacting systems have traditionally been simulated either by solving a set of ordinary differential equations (ODEs) or by using the stochastic simulation algorithm (SSA) of Gillespie. Traditional ODE-based approaches are adequate when dealing with large numbers of molecules, when discreteness and noise have no macroscopic effects, but in general they are not able to provide a fully physically accurate representation of the noise amplification caused by the essential stochastic processes in living cells. At the same time SSA approaches can simulate accurately all those dynamics by using a discrete-space Markov process, but their drawback remain the great amount of computation time that is often required to simulate a model.

From this setting we arrive to stochastic differential equations (SDEs), a combination between a strictly deterministic approach and a stochastic one.

*Currently visiting Laboratory for Foundations of Computer Science, University of Edinburgh

SDEs represent a position on the border between different branches of science and engineering, from mathematical analysis to probability calculus. In particular regarding stochastic processes: the application of SDEs is of interest to different subjects like physics, financial mathematics and biology.

SDEs, used to solve chemical reactions, are Markov processes described by the Chemical Langevin Equation (CLE). We assume $Y(t) \in \mathbb{R}^N$ the state vector of a continuous time, real-valued stochastic process at the time t : so $Y_i(t)$ is a real-valued random variable representing the number of molecules of the i th species. The stoichiometric or state-change vector is described by $v_j \in \mathbb{R}^N$, whose i th component is the change in the number of S_i molecules due to the j th reaction. Finally $a_j(Y(t))$ is the propensity function, which gives us the probability this reaction will take place in the infinitesimal interval time $[t, t + dt)$. Having made this assumption, CLE takes the Itô form

$$dY(t) = \sum_{j=1}^M v_j a_j(Y(t)) dt + \sum_{j=1}^M v_j \sqrt{a_j(Y(t))} dW_j(t) \quad (1)$$

where the $W_j(t)$ are independent Wiener processes. A Wiener process is a stochastic process satisfying

$$E(W(t)) = 0, \quad E(W(t)W(s)) = \min\{t, s\}$$

The Wiener increments are independent Gaussian processes with mean 0 and variance $|t - s|$. Specifically the Wiener increment $\Delta W(t) \equiv W(t + \tau) - W(t)$ is a Gaussian random variable $N(0, \tau) = \sqrt{\tau}N(0, 1)$. A numerical solution to solve these kinds of SDEs comes from the Euler-Maruyama method and, discretizing the Brownian path and applying the Euler-Maruyama method to the linear SDE, it takes the form

$$Y(t + \tau) = Y(t) + \tau \sum_{j=1}^M v_j a_j(Y(t)) + \sqrt{\tau} \sum_{j=1}^M v_j \sqrt{a_j(Y(t))} Z_j$$

We have produced a Java implementation of the Euler-Maruyama method as an extension to a simulator for chemical reactions *Dizzy*, the chemical kinetics stochastic simulation software of the Institute for Systems Biology. In this paper I am going to show three examples of models solved with SDEs: Michaelis-Menten, Lotka-Volterra and Schlögl. For all these models I have produced different charts that can help to compare results of the SDEs simulator with ODEs and SSA simulators. Although SDEs are capable of capturing the major aspects of a chemical reaction, we can find this approach not suitable for multi-scale models in which we manage an exponentially large (or small) population variables or we have to deal with exponentially unlikely events. The Schlögl model is one of these models for which SDEs do not give satisfactory results.

This article is organised as follows. Section 2 explains the code I have implemented on *Dizzy*. Afterwards, Section 3 exposes to three different models solved with the Euler-Maruyama algorithm with *Dizzy*. Finally, in Section 4 I point the reader to final conclusions, problems and future works.

2 Numerical solution

The aim of this section is to give the reader a concrete understanding of the Euler-Maruyama algorithm[Lam06]. To achieve this target, I present the Java code I have written for the stochastic simulator Dizzy. The code is structured in two classes, *SimulatorSDEBase* and *SimulatorSDEEulerMaruyama*.

The first class is a basic class for possible further SDEs simulators, extends *Simulator* and gathers all information regarding model parameters and input data. Moreover *SimulatorSDEBase* provides the main cycle wherein a specific SDEs simulator computes each iteration, supplying all the essential environment.

SimulatorSDEEulerMaruyama implements the interface *ISimulator* and extends *SimulatorSDEBase*, which provides all the mandatory methods for every simulator of chemical reactions in Dizzy. The most important method in this class is *iterate()*, called directly from the main cycle in *SimulatorSDEBase* in every iteration. The code below is an extract of the method *iterate()* of the class *SimulatorSDEEulerMaruyama*:

```
1  /* SimulatorSDEEulerMaruyama.java
   * Implementation of Euler-Maruyama algorithm for Dizzy
   */

   computeReactionProbabilities();
6
   int numReactions = mReactions.length;
   int numSymbols = pNewDynamicSymbolValues.length;
   double reactionRate = 0.0;

11 DoubleVector.zeroElements(derivative);

   Object []dynamicSymbolAdjustmentVectors = mDynamicSymbolAdjustmentVectors;

   for(int i = 0; i < numReactions; i++) {
16   reactionRate = mReactionProbabilities[i];
      drift = reactionRate * stepSize;
      diffusion = Math.sqrt(Math.abs(drift)) * mScratchPad.nextRand();
      diffusion = drift + diffusion;
      double []symbolAdjustmentVector =
21   (double[])dynamicSymbolAdjustmentVectors[i];
      DoubleVector.scalarMultiply(symbolAdjustmentVector, diffusion, k);
      DoubleVector.add(k, derivative, derivative);
   }

26 double sumScale = 0;

   for(int i = 0; i < numSymbols; i++) {
      scale[i] = Math.abs(pNewDynamicSymbolValues[i]) + Math.abs(derivative[i]);
      sumScale += scale[i];
31 }

   if(sumScale == 0)
      throw new AccuracyException("unable to determine any scale at time: " +
          mSymbolEvaluator.getTime());
```

```

DoubleVector.add(pNewDynamicSymbolValues,derivative,pNewDynamicSymbolValues);
mSymbolEvaluator.setTime(mSymbolEvaluator.getTime() + stepSize);

return(mSymbolEvaluator.getTime());

```

Listing 1: SimulatorSDEEulerMaruyama.java

First of all I recall the stochastic differential equation with the discretized Brownian path.

$$Y(t + \tau) = Y(t) + \tau \sum_{j=1}^M v_j a_j(Y(t)) + \sqrt{\tau} \sum_{j=1}^M v_j \sqrt{a_j(Y(t))} Z_j \quad (2)$$

Such an equation can be divided in four main components: the previous result, the drift and the diffusion term, and the white noise.

$$Y(t + \tau) = \underbrace{Y(t)}_{prev} + \tau \underbrace{\sum_{j=1}^M v_j a_j(Y(t))}_{drift} + \sqrt{\tau} \underbrace{\sum_{j=1}^M v_j \sqrt{a_j(Y(t))}}_{diffusion} \cdot \underbrace{Z_j}_{noise} \quad (3)$$

In line 5, I compute the probability rate for each reaction in the model: `computeReactionProbabilities()` is a method inherited from the class *Simulator*. This probability is described in (3) as the $a_j(Y(t))$ term.

Once probabilities are evaluated, for each simulation (line 15) I begin to determine the drift (line 17) and the diffusion term (line 18), referring to τ as the *stepsize*. The diffusion term is furthermore multiplied by the white noise: such latter term, which in the code is described as `mScratchPad.nextRand()`, is a double random number generated by a Normal distribution. I have used the package provided by *Colt Project*¹ embedded in *Dizzy* to generate those random values.

Finally in line 22 I multiply the drift and the diffusion term for the stoichiometric matrix v_j described in the array *symbolAdjustmentVector* and I add the value to the other values obtained with other reactions.

3 Experiments

I apply the methodology implemented in the previous section to three models: the Michaelis-Menten model, the Schlögl model and the Lotka-Volterra model. All these models are solved² in three different ways: with Ordinary Differential Equations (ODEtoJava adaptive [Spi05]), with Stochastic Simulation Algorithms (Gillespie's direct method, Gibson - Bruck algorithm and Gillespie's τ -leap [Gil01]) and with Stochastic Differential Equations proposed in Section 2.

¹<http://dssd.lbl.gov/~hoschek/colt/>

²All the experiments were performed using P4 Dell 512Mb RAM using a Fedora Linux with GUI Gnome 2.14

3.1 The Michaelis-Menten model

The Michaelis-Menten model is widely used in biology for studying the kinetics of many non-allosteric enzymes: enzyme (E) and substrate (S) can combine themselves in complex (ES), which then eventually dissociates to free enzyme and product (P).

```
E = 100;
S = 100;
P = 0;
ES = 0;
enzyme_substrate_combine, E + S -> ES, 1.0;
enzyme_substrate_separate, ES -> E + S, 0.1;
make_product, ES -> E + P, 0.01;
```

Listing 2: Michaelis-Menten model

As described in the model, the initial concentration of the enzymes and substrates is 100; the rate to combine enzymes and substrates is 1.0; to separate them is 0.1 and to create products is 0.01.

Comparing these three charts, we can observe a very similar behaviour of all four species trends. In particular, around the time 60 - 80, the concentration of P (and E) exceeds the number of ES independent of the solution algorithm.

3.2 The Schlögl model

The Schlögl model is a famous artificial chemical system used to describe bistable behaviour in the state variable for certain parameters.

```
X = 250;
c1 = 3E-7;
c2 = 1E-4;
c3 = 1E-3;
c4 = 3.5;
N1 = 1E5;
N2 = 2E5;
R1, -> X, [(c1/2)*N1*X*(X-1)];
R2, X -> , [(c2/6)*X*(X-1)*(X-2)];
R3, -> X, [c3*N2];
R4, X -> , [c4*X];
```

Listing 3: Schlögl model

As the charts showed below, the trajectory of X can be accurately simulated only with stochastic algorithms. During these simulations the state variable has always taken values above the initial condition (250), either using the SDEs and the SSA simulator. Assuming bistable states, the mean and the variance do not have a physically significant meaning: so both SDEs and SSA simulation were computed only a single time.

Unlike the Michaelis-Menten model, SSA Gibson Bruck simulator is slower than SDEs computing the Schlögl model.

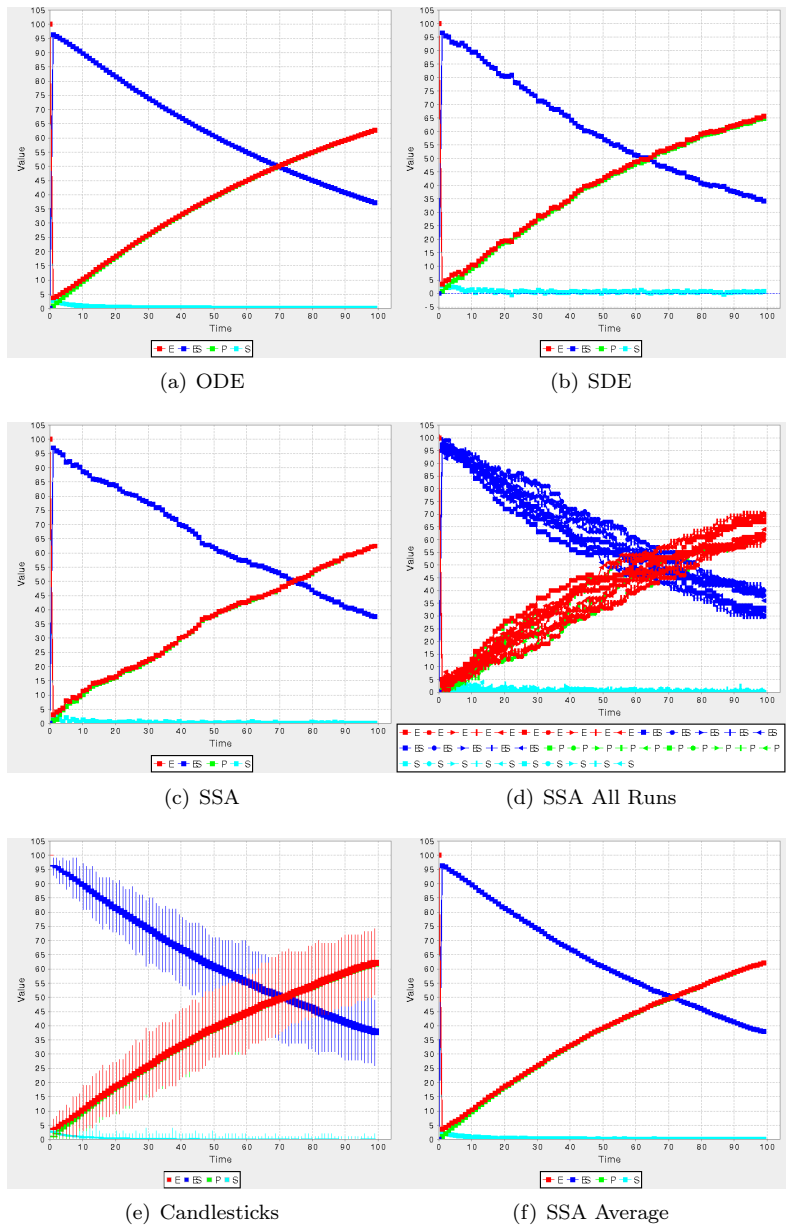


Figure 1: Michaelis-Menten Charts

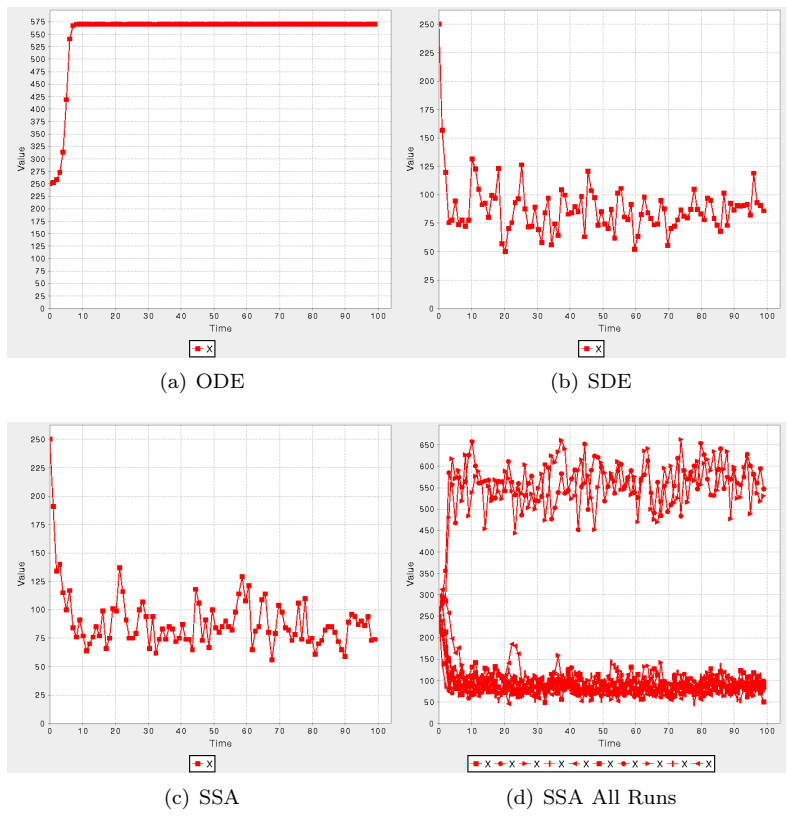


Figure 2: Schlögl Charts

The Schlögl model, due to its rare events that arise in multi-scale systems when we calculate exponentially large or small observables, has a bad description if we simulate it with the diffusion approximation. This issue resides in the diffusion approximation of the SDE process (drift term + diffusion term), which leads to a wrong result simulating exponential values. In order to understand the problem, first of all we can give a deterministic description of the problem, defining the concentration $x = X/V$, assuming V the system volume, with the ODE $\frac{dx}{dt} = u(x) - d(x)$, with $u(x) = c1 * x^2 + c4$ and $d(x) = c2 * x^3 + c4 * x$. In this form 1 can be written as

$$dx = [u(x) - d(x)]dt + \sqrt{u(x) + d(x)}dW \quad (4)$$

Passing to an intensive quantity $x = n/V = \varepsilon n$ in order to analyse large system size limits, we can rewrite 4 in this form

$$dx = [u(x) - d(x) + \varepsilon(u_1(x) - d_1(x))]dt + \sqrt{\varepsilon(u(x) + d(x))}dW \quad (5)$$

that correspond to the original SDE only with the rates rescaled by ε . Applying to 5 the large deviations principle leads us to a wrong Partial Differential Equation (PDE).

A satisfactory description of the issue can be found in [Sar05]

3.3 The Lotka-Volterra model

The Lotka-Volterra model is one of the simplest models in which predators and prey interact together, in a one-prey-group and one-predator-group scenario. In this example, I set the quantity of food at 1, the number of predators (pred) and prey (prey) at 1000.

```

food = 1;
prey = 1000;
pred = 1000;
natc = 0;
r1, food + prey -> prey + prey + food, 10;
r2, prey + pred -> pred + pred, 0.01;
r3, pred -> natc, 10;

```

Listing 4: Lotka-Volterra model

Three simple reactions describe the model: food and prey generate two prey and release a food with rate 10; a prey and a predator react with rate 0.01 and become two predators; finally a predator dies of natural causes (natc) with rate 10.

Both charts are characterised by oscillations in the population size of both preys and predators, as predicted: using the SDEs simulator we can save a large amount of time computing these oscillations.

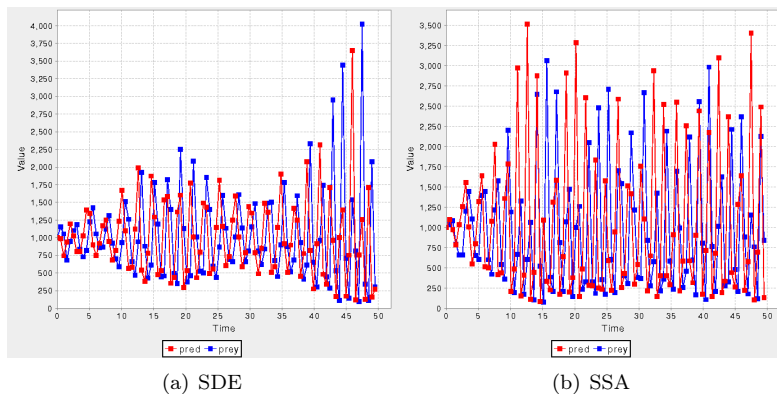


Figure 3: Lotka-Volterra Charts

3.4 Final results

In the table below I show the average time (in seconds) spent by each algorithm for simulating the three models described above. Each model was simulated ten times with all the algorithms:

Algorithm/Model	Michaelis-Menten	Schlögl	Lotka-Volterra
ODEs	0.93	0.18 ³	0.25 ³
SDEs	2.81	4.78	2.83
Gillespie's SSA	0.15	3.72	7.38
Gibson-Bruck Next Reaction	0.16	4.01	7.65
Gillespie's τ -leap	0.12	3.69	7.12

As we can see, Gillespie's τ -leap method is the fastest way to solve either Michaelis-Menten model and Schlögl model. In these examples, either in the Schlögl model and in the Lotka-Volterra model, ODEs are not good enough to describe the behaviour of the species trends. Instead SDEs and SSAs algorithms, because of their stochastic nature, are suitable for all three models: moreover SDEs are faster in the Lotka-Volterra model, and a little bit slower in the Michaelis-Menten model.

4 Concluding remarks

I presented a simulator which uses stochastic differential equations to solve chemical reactions, and I compared this simulator with other two standard ways to solve chemical reaction systems. After making several tests, a few shown in Section 3, I can assert this simulator is good enough for models when we treat

³ODEs became stable after a very short amount of time, so this method is not relevant

chemical reaction systems with non-exponential size variables (i.e. without multi-scale problems). This simulator has also a better performance than a traditional stochastic simulator algorithm in some models, and can represent a valid alternative when these become too slow to compute the given model.

5 Related works

Higham [Hig01, Hig07] proposed a Matlab implementation of SDEs, in order to compare the Michaelis-Menten model with Reaction Rate Equations and a Stochastic Simulation Algorithm.

F. Petruccione and P. Biechele [Pet01] presented a very exhaustive description of SDEs and their implementation for solving physical problems. Namely in Section 7 they proposed an implementation of the Euler-Maruyama method applied to the Ornstein-Uhlenbeck process and to a SDE with multiplicative noise.

Sargsyan [Sar05], as described in Section 3.2, discussed the systematic error of the SDEs approach to the Schlögl model, introducing also the Kinetic Monte Carlo model (essentially the Gillespie's simulation algorithm) as an answer for computing trajectories of the Schlögl model.

K. Burrage, T. Tian and P. Burrage [Bur04], after an interesting discussion comparing ODEs and SDEs, defined a strong solution of stochastic differential equations, either with explicit and implicit methods.

Acknowledgements

I would like to thank Prof. Stephen Gilmore for his helpful corrections and comments at this paper during the visit to the University of Edinburgh.

References

- [Gil00] Daniel Gillespie. *The Chemical Langevin Equation*. Journal of Chemical Physics (2000).
- [Bur04] Kevin Burrage, Tianhai Tian, Pamela Burrage. *Numerical Methods for Strong Solutions of Stochastic Differential Equations: an Overview*. Proceedings: Mathematical, Physical and Engineering, Royal Society of London (2004).
- [Sar05] Khachik Sargsyan. *Fluctuations in chemical reactions in a large volume*. GFD Program Proceedings (2005).
- [Bur04-1] Kevin Burrage, Tianhai Tian, Pamela Burrage. *A multi-scaled approach for simulating chemical reaction systems*. Prog. Biophys. Mol. Biol. (2004).

- [Hig07] Desmond J. Higham. *Modeling and Simulating Chemical Reactions*. Uni. of Strathclyde Research Report 02 (2007).
- [Hig01] Desmond J. Higham. *An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations*. SIAM Rev 43, 525 (2001).
- [Pet01] F. Petruccione, P. Biechele. *Stochastic methods for Physics using Java: An Introduction*. (2001)
- [Gil01] Daniel Gillespie. *Approximate accelerated stochastic simulation of chemically reacting systems*. J. Chem. Phys. 115 (2001).
- [Lam06] H. Lamba, J. C. Mattingly, A. M. Stuart. *An adaptive Euler-Maruyama scheme for SDEs: convergence and stability. Preprint*. Available online via <http://front.math.ucdavis.edu/math.NA/0601029> (2006).
- [Spi05] Raymond Spiteri. *Introducing odeToJava: A problem-solving environment for initial-value problems*. Abstract at SciCADE (2005).
- [Ram05] Ramsey S., Orrell D., Bolouri H. *Dizzy: stochastic simulation of large-scale genetic regulatory networks*. J. Bioinformatics Computational Biology (2005).