

# Simulation methods for stochastic chemical systems that arise from a random time change representation

David F. Anderson  
Department of Mathematics  
University of Wisconsin – Madison

Institute for Mathematics and its Applications  
January 15th, 2008

Model consists of:

1.  $M \geq 1$  chemical species,  $\{X_1, \dots, X_M\}$ , undergoing a series of chemical reactions.
2.  $\nu_k, \nu'_k \in \mathbb{Z}_{\geq 0}^M$  are the vectors representing the number of each species consumed and created in the  $k$ th reaction, respectively.
3. If reaction  $\mu$  occurs at time  $t$ :

$$x(t) = x(t-) + \nu'_\mu - \nu_\mu.$$

4. The  $k$ th reaction has an associated intensity (or propensity) function,  $\lambda_k(x)$ , such that the probability that reaction  $k$  fires in the small time interval  $[t, t + \Delta t)$  is

$$\lambda_k(x(t))\Delta t + o(\Delta t), \quad o(\Delta t)/\Delta t \rightarrow 0, \quad \text{as } \Delta t \rightarrow 0.$$

System is a continuous time Markov chain with transition rates

$$q(x, y) = \begin{cases} \lambda_k(x), & y = x + \nu'_k - \nu_k \\ 0, & \text{else} \end{cases}$$

Typical Goal: Understand the distribution of the state of the system  $x(t)$ .

Kolmogorov's Forward equation (Chemical Master Equation)

$$\frac{d}{dt}P(x, t|x_0) = \sum_k \lambda_k(x - \nu'_k + \nu_k)P(x - \nu'_k + \nu_k, t|x_0) - \sum_k \lambda_k(x)P(x, t|x_0),$$

is extremely difficult to solve or even simulate for many systems.

Solution: simulate sample paths so as to approximate distribution via Monte Carlo methods.

To simulate exact sample paths, 2 questions need answering at each step:

- ▶ How much time will pass before the next reaction takes place.
- ▶ Which reaction takes place at that future time.

Simple to show that if in state  $x$  then (Gillespie 1976/77, Kelly 1979)

$$Prob(\text{time until next firing} > \Delta) = e^{-\lambda_0 \Delta},$$

where  $\lambda_0 = \sum_k \lambda_k(x)$ , and

$$Prob(\text{next reaction is } k^{th}) = \lambda_k(x) / \lambda_0.$$

One step of Gillespie Algorithm:

1. Compute the propensity,  $\lambda_k$ , for each reaction. Let  $\lambda_0 = \sum_k \lambda_k$ .
2. Set  $\Delta = (1/\lambda_0) \ln(1/r_1)$ , where  $r_1 \sim \text{unif}(0, 1)$ .
3. Choose which reaction occurred at time  $t + \Delta$  by solving for  $\mu$  such that

$$\frac{1}{\lambda_0} \sum_{k=1}^{\mu-1} \lambda_k < r_2 \leq \frac{1}{\lambda_0} \sum_{k=1}^{\mu} \lambda_k, \quad (1)$$

where  $r_2 \sim \text{unif}(0, 1)$ .

4. Update the system by  $x(t + \Delta) = x(t) + \nu'_\mu - \nu_\mu$ .

## Random Time Change Representation

If  $R_k(t)$  is the number of times that the  $k$ th reaction has taken place up to time  $t$ ,

$$x(t) = x(0) + \sum_k R_k(t)(\nu'_k - \nu_k).$$

$R_k(t)$  is a counting process with intensity  $\lambda_k(x(t))$  and can be written as

$$R_k(t) = Y_k \left( \int_0^t \lambda_k(x(s)) ds \right),$$

where the  $Y_k$  are independent, unit rate Poisson processes.

$$\implies x(t) = x(0) + \sum_k Y_k \left( \int_0^t \lambda_k(x(s)) ds \right) (\nu'_k - \nu_k).$$

# Internal Time

---

$$x(t) = x(0) + \sum_k Y_k \left( \int_0^t \lambda_k(x(s)) ds \right) (\nu'_k - \nu_k).$$

---

Note:

- ▶ There are actually  $M + 1$  relevant time-frames in the system.
  1.  $t$  is actual or absolute time.
  2. “Internal time” of process  $Y_k$ :

$$T_k(t) = \int_0^t \lambda_k(x(s)) ds$$

## Statistically exact simulation methods

$$x(t) = x(0) + \sum_k Y_k \left( \int_0^t \lambda_k(x(s)) ds \right) (\nu'_k - \nu_k).$$

---

Alternative to the Gillespie algorithm: at time  $t$  let

$$T_k = \int_0^t \lambda_k(x(s)) ds$$

$$P_k = \min\{s > T_k : Y_k(s) > Y_k(T_k)\}.$$

Assuming no other reactions fire first, amount of time until reaction  $k$  fires given by  $\lambda_k \Delta t_k = P_k - T_k$

$$\implies \Delta t_k = \frac{P_k - T_k}{\lambda_k}.$$

# The modified Next Reaction Method

1. Initialize system and set  $T_k = 0$  for all  $k$ .
2. For each  $k$ , set  $P_k = \ln(1/r_k)$ , where  $r_k \sim \text{unif}(0, 1)$ .
3. For each  $k$  set:  $\Delta t_k = (P_k - T_k)/\lambda_k$ .
4.  $\Delta = \min\{\Delta t_k\}$ . Reaction that fires is where min. achieved.
5. Set  $P_\mu = P_\mu + \ln(1/r)$ ,  $r \sim \text{unif}(0, 1)$ .
6. For each  $k$ ,  $T_k = T_k + \lambda_k \Delta$ .
7. Update system and propensities.
8. Return to step 3 or quit.

## Remarks:

- A. Only one random variable needed per iteration after the first.
- B. Similar to generalized semi-Markov processes and stochastic Petri nets.

# Time dependent propensities

Say  $\lambda_k = \lambda_k(x(t), t)$ . What changes? (Not Much)

Now,

1. Find  $\Delta t_k$  by solving:

$$\int_t^{t+\Delta t_k} \lambda_k(x(t), s) ds = P_k - T_k.$$

2.  $\Delta$  is still minimum of  $\Delta t_k$ .
3. Update  $P_\mu = P_\mu + \ln(1/r)$ ,  $r \sim \text{unif}(0, 1)$  (same).
4. Update  $T_k$ :

$$T_k = T_k + \int_t^{t+\Delta} \lambda_k(x(t), s) ds.$$

5. Update system and return.

## Approximate Algorithms: tau-leaping

**Goal:** Substantially increase simulation speed by allowing acceptable amount of error.

Observe: If  $\lambda_k(x(s))$  remains relatively constant in  $[t, t + \tau)$  then, conditioned on  $x(s)$  for  $s \leq t$ ,

$$\begin{aligned}R_k(t + \tau) - R_k(t) &= Y_k \left( \int_0^{t+\tau} \lambda_k(x(s)) ds \right) - Y_k \left( \int_0^t \lambda_k(x(s)) ds \right) \\ &\approx Y_k \left( \lambda_k(x(t))\tau + \int_0^t \lambda_k(x(s)) ds \right) \\ &\quad - Y_k \left( \int_0^t \lambda_k(x(s)) ds \right) \\ &= \text{Poisson}(\lambda_k(x(t))\tau),\end{aligned}$$

## Approximate simulation algorithm: tau-leaping

1. Initialize.
  2. Select  $\tau$  “small enough” so  $\lambda_k$  approx. constant over  $[t, t + \tau)$ .
  3. For each  $k$ , let  $N_k = \text{Poisson}(\lambda_k \tau)$ .
  4. Set  $x(t + \tau) = x(t) + \sum_k N_k (\nu'_k - \nu_k)$ .
- 

**Main Question of tau-leaping methods:** How do we select  $\tau$  to be “small enough”?

**Goal:** Try to ensure one of the following **leap conditions** ( $\epsilon$ 's are small):

$$|\Delta_\tau \lambda_k| \leq \epsilon \lambda_k(x(t))$$

$$|\Delta_\tau x_i| \leq \epsilon_i x_i.$$

**Current Methods (Gillespie, Petzold, Cao):** Select  $\tau$  via a preleap computation to ensure leap condition with some probability.

**New Method:** Unbiased postleap check. Main ideas:

1. By storing information about  $Y_k$ , can reject leaps without biasing sample paths.
2. Will store:

$$T_k(t) = \int_0^t \lambda_k(x(s)) ds$$
$$C_k(t) = Y_k(T_k(t)).$$

3. Given  $Y_k(t)$  and  $Y_k(s)$  with  $s < t$ , for  $r \in (s, t)$

$$Y_k(r) = \text{Binomial} \left( Y_k(t) - Y_k(s), \frac{r - s}{t - s} \right).$$

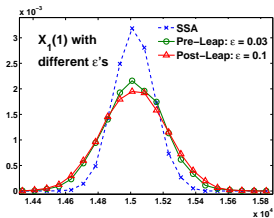
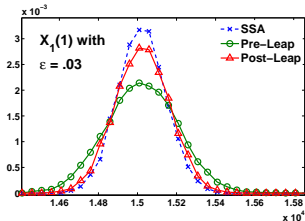
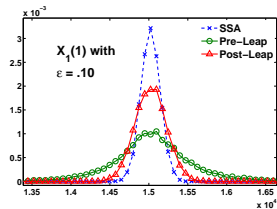
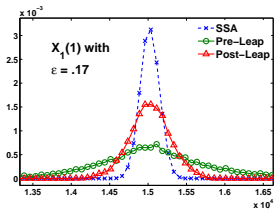
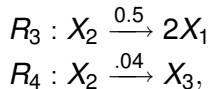
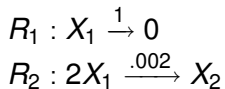
## Shell of new Tau-Leaping Algorithm:

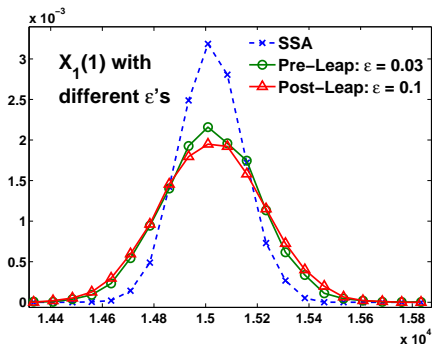
1. Store  $T_k$  and  $C_k$  after each accepted/rejected leap.
2. For a given  $\tau$ , approximate the number of firings per reaction,  $N_k$ , over time interval via Poisson RV (no stored info.) or binomial RV (stored info.) by using that the next internal time will be  $T_k + \lambda_k \tau$ .
3. Accept or reject leap based on leap condition.
4.  $\tau$  can be computed adaptively based upon whether previous attempt succeeded (increase  $\tau$ ) or failed (decrease  $\tau$ ).

## Benefits:

1. Guaranteed that any leap condition can be enforced with probability equal to 1.
2. Easy to implement for any leap conditions.
3. Naturally avoids negative population numbers.
4. Efficient.

# Example: Unstable Dimer





CPU Times	$\epsilon = 0.17$	$\epsilon = 0.10$	$\epsilon = 0.03$
Preleap Computation	31 Min.	46 Min.	120 Min.
Postleap Check	60 Min.	82 Min.	226 Min.

Preleap required 46% more CPU time than postleap.

End of Story

Thanks!