Chapter 1
Algorithms for Linear Stochastic Delay Differential Equations

Harish S. Bhat

Abstract Models consisting of linear, N-dimensional stochastic delay differential equations present a particular set of challenges for numerical simulation. While the user often seeks the probability density function of the solution, currently available methods rely on Monte Carlo sampling to generate sample paths, from which a density function must be estimated statistically. In the present work, we derive a new algorithm to compute the density function of the solution with no sampling. By discretizing the stochastic equation in time, we bypass closure issues that prevent a Fokker-Planck approach. We carry out a number of numerical tests to compare the algorithm against Monte Carlo methods, to judge its behavior as the time step decreases, and to check its capabilities of handling fully vectorial systems, both with and without time delays. The results indicate that the algorithm is a fast, accurate alternative to existing methods.

1.1 Introduction

We consider the stochastic delay differential equation (SDDE)

$$d\mathbf{X}_t = A\mathbf{X}_t dt + B\mathbf{X}_{t-\tau} dt + C d\mathbf{W}_t. \tag{1.1}$$

Here A, B, and C are $N \times N$ constant coefficient matrices, the time delay $\tau > 0$ is constant, $\mathbf{W}_t$ is N-dimensional Brownian motion, and the unknown $\mathbf{X}_t$ is an $\mathbb{R}^N$-valued stochastic process.

System (1.1) models phenomena in neuroscience [7] and mechanics [10, 4], among several other fields. For each $t \geq 0$, let $p(\mathbf{x}, t)$ denote the probability density function of $\mathbf{X}_t$. In many scientific contexts, the quantities of interest are functionals of p—for example, the mean and variance of the solution of (1.1). In these con-

Harish S. Bhat
University of California, Merced, USA, e-mail: hbhat@ucmerced.edu

texts, the sample paths $\mathbf{X}_t$ of (1.1) are of interest only to the extent that they help to compute p or functionals of p.

Let † denote matrix transpose. If we remove the time delay term, say by setting $B = 0$, then we can solve for $p(\mathbf{x},t)$ directly via the partial differential equation

$$\frac{\partial p}{\partial t} + \operatorname{trace}(A)p + (A\mathbf{x})^{\dagger}\nabla p = \frac{1}{2}CC^{\dagger}\nabla^2 p \tag{1.2}$$

This equation is known as either the Fokker-Planck or Kolmogorov equation associated with the stochastic differential equation $d\mathbf{X}_t = A\mathbf{X}_t dt + Cd\mathbf{W}_t$.

The Fokker-Planck equation associated with the time-delayed equation (1.1) suffers from a closure problem [6]. This problem prevents the application of deterministic methods from numerical analysis to solve for the density function $p(\mathbf{x},t)$. As a result, Monte Carlo methods are commonly employed; in this framework, one simulates a sufficiently large number of sample paths of (1.1) in order to estimate the density function or functionals thereof.

In this note, we develop a new algorithm to directly solve for the density function of (1.1). By first discretizing (1.1) in time, we bypass the closure issues of Fokker-Planck approaches. The resulting scheme involves no sampling, and is thus capable of computing the density function of (1.1) faster than Monte Carlo methods, for the same desired level of accuracy.

1.2 Algorithm

Starting from (1.1), we apply the Euler-Maruyama time-discretization. Specifically, let ℓ denote a positive integer, and set $h = \tau/\ell$. Let $\mathbf{Y}_n$ denote the numerical approximation to $\mathbf{X}_{nh}$. Then, by definition, $\mathbf{Y}_{n-\ell}$ is the numerical approximation to $\mathbf{X}_{nh-\tau}$. Set I equal to the $N \times N$ identity matrix, and let $\{\mathbf{Z}_n\}_{n\geq 1}$ denote an i.i.d. sequence of $\mathscr{N}(\mathbf{0}, I)$ random variables—here $\mathscr{N}(\mu, \Sigma)$ denotes the multivariate Gaussian with mean vector μ and covariance matrix Σ. Then the Euler-Maruyama discretization of (1.1) is

$$\mathbf{Y}_{n+1} = (I + Ah)\mathbf{Y}_n + Bh\mathbf{Y}_{n-\ell} + Ch^{1/2}\mathbf{Z}_{n+1}. \tag{1.3}$$

Thus far we have not mentioned initial conditions. For the original differential equation (1.1), the initial conditions consist of the segment $\{\mathbf{X}_t \mid -\tau \leq t \leq 0\}$. Discretizing this segment yields $\mathscr{I} = \{\mathbf{Y}_n \mid -\ell \leq n \leq 0\}$, where $\mathbf{Y}_n = \mathbf{X}_{nh}$. In what follows, we assume that $\mathscr{I}$ is given and that each $\mathbf{Y}_n \in \mathbb{R}^N$ is a constant (deterministic) vector.

With (1.3) together with the initial segment $\mathscr{I}$, we can certainly generate sample paths $\{\mathbf{Y}_n\}_{n\geq 1}$. Note that this involves sampling the random variables $\{\mathbf{Z}_n\}_{n\geq 1}$. See [2] for numerical analysis of this approach.

Let us now give a convenient representation of the solution of (1.3):

Theorem 1. *For each $n \geq -\ell$, there exist $N \times N$ matrices $\{\alpha_m^n\}_{m=-\ell}^{0}$ and $\{\beta_r^n\}_{r=1}^{n}$ such that*

$$\mathbf{Y}_n = \sum_{m=-\ell}^{0} \alpha_m^n \mathbf{Y}_m + \sum_{r=1}^{n} \beta_r^n \mathbf{Z}_r. \tag{1.4}$$

Proof. When $-\ell \leq n \leq 0$, the statement is true by definition: in this range, the β_r^n matrices are all zero, $\alpha_n^n = I$, and $\alpha_m^n = 0$ for $m \neq n$.

The rest of the proof is by induction. For the $n = 1$ case, we note that (1.3) implies

$$\mathbf{Y}_1 = (I + Ah)\mathbf{Y}_0 + Bh\mathbf{Y}_{-\ell} + Ch^{1/2}\mathbf{Z}_1.$$

Hence $\alpha_0^1 = I + Ah$, $\alpha_{-\ell}^1 = Bh$, and $\alpha_m^n = 0$ for $-\ell < m < 0$. Setting $\beta_1^1 = Ch^{1/2}$, we see that (1.4) holds for $n = 1$.

Next assume that (1.4) holds for $1 \leq n \leq n'$. For $-\ell \leq m \leq 0$, set

$$\alpha_m^{n'+1} = (I + Ah)\alpha_m^{n'} + Bh\alpha_m^{n'-\ell}. \tag{1.5}$$

For $1 \leq r \leq n' + 1$, set

$$\beta_r^{n'+1} = \begin{cases} (I + Ah)\beta_r^{n'} + Bh\beta_r^{n'-\ell} & 1 \leq r \leq n' - \ell \\ (I + Ah)\beta_r^{n'} & n' - \ell + 1 \leq r \leq n' \\ Ch^{1/2} & r = n' + 1. \end{cases} \tag{1.6}$$

A calculation now shows that $\mathbf{Y}_{n'+1}$ defined by (1.4), (1.5), and (1.6) satisfies the $n = n' + 1$ case of (1.3). □

The system (1.5)-(1.6) is an algorithm for determining the solution of the discretized equation (1.3). This algorithm does not involve sampling any random variables. There are several points we wish to make about this algorithm:

1. The α equation (1.5) is decoupled from the β equation (1.6). The equations can be stepped forward in time independently of one another.
2. The dynamics of (1.5-1.6) are independent of the initial conditions $\mathscr{I}$. Once we have computed α and β, we can evaluate the solution (1.4) for any choice of initial conditions.
3. Once we have the solution in the form (1.4), it is simple to determine the distribution of $\mathbf{Y}_n$. Each $\beta_r^n \mathbf{Z}_r$ has a $\mathscr{N}(\mathbf{0}, \beta_r^n (\beta_r^n)^\dagger)$ distribution. Using the independence of each $\mathbf{Z}_r$ and the fact that the initial vectors $\{\mathbf{Y}_m\}_{m=-\ell}^{0}$ are constant, we have

$$\mathbf{Y}_n \sim \mathscr{N}\left(\sum_{m=-\ell}^{0} \alpha_m^n \mathbf{Y}_m, \sum_{r=1}^{n} \beta_r^n (\beta_r^n)^\dagger \right). \tag{1.7}$$

   The upshot is that the α and β coefficients describe, respectively, the mean and the variance/covariance of the computed solution.
4. The α equation (1.5) can be derived in a much more direct fashion. Let us first take the expected value of both sides of (1.1) to derive the deterministic DDE (delay differential equation):

$$\frac{d}{dt} E[\mathbf{X}_t] = AE[\mathbf{X}_t] + BE[\mathbf{X}_{t-\tau}].$$

Applying the standard Euler discretization to this equation yields (1.5). Numerous prior works have studied Euler discretizations of a deterministic DDE. Therefore, for the empirical convergence tests described below, we consider problems where $E[\mathbf{X}_t]$ is zero and focus our attention on (1.6).

5. Several methods exist to approximate SDDE by Markov chains [8, 9, 1]. Such methods necessarily involve creating a number of discrete states to approximate the continuous state space of (1.1); often the number of such states scales with ℓ, the discrete delay. While the Markov chain method of [1] is accurate and fast for delayed random walks where ℓ is small and fixed, the number of states scales like 4^ℓ. Hence the method breaks down when τ is large; in this case, in order for the time step $h = \tau/\ell$ to be acceptable, we must choose a large value of ℓ. Algorithm (1.5-1.6) does not discretize the state space of (1.1), and it is much less sensitive to the magnitude of the time delay τ than Markov chain methods.

1.3 Implementation and Tests

We have implemented algorithm (1.5-1.6) in R, an open-source framework for statistical computing. The implementation simplifies considerably in the case of a scalar equation, i.e., when $N = 1$. We therefore separate our discussion into scalar and vector cases.

1.3.1 Scalar Case ($N = 1$)

When $N = 1$, the coefficients A, B, and C in (1.1) and the coefficients $\{\alpha^n_m\}$ and $\{\beta^n_r\}$ in (1.4) are all scalars. Then $\alpha^n = (\alpha^n_{-\ell}, \ldots, \alpha^n_0)$ and $\beta^n = (\beta^n_1, \ldots, \beta^n_n)$ are vectors, of respective dimension $\ell + 1$ and n. With this notation, (1.5-1.6) can be written in matrix-vector form as

$$\alpha^{n+1} = (1 + Ah)\alpha^n + Bh\alpha^{n-\ell} \tag{1.8}$$

$$\beta^{n+1} = \begin{bmatrix} (1+Ah)\beta^n \\ 0 \end{bmatrix} + \begin{bmatrix} Bh\beta^{n-\ell} \\ \mathbf{0} \end{bmatrix} + Ch^{1/2}\mathbf{e}_{n+1}. \tag{1.9}$$

Here $\mathbf{0}$ is the zero vector in $\mathbb{R}^{\ell+1}$, and $\mathbf{e}_{n+1} = (0, \ldots, 0, 1) \in \mathbb{R}^{n+1}$.

As explained above, algorithm (1.8-1.9) yields the exact probability density function of the stochastic delay *difference* equation (1.3). To explore the practical benefits of this fact, we compare our algorithm against the following Monte Carlo procedure: (i) fix a value of the time step $h = \tau/\ell$, (ii) sample the random variables $\{Z_n\}_{n\geq 1}$ and step forward in time using (1.3), (iii) stop when we obtain a sample of Y_n at a fixed final time $T > 0$. Running this procedure many times, we obtain a corpus of samples of Y_n at time T. In what follows, we will compare the variance of these samples against the variance computed using (1.9).

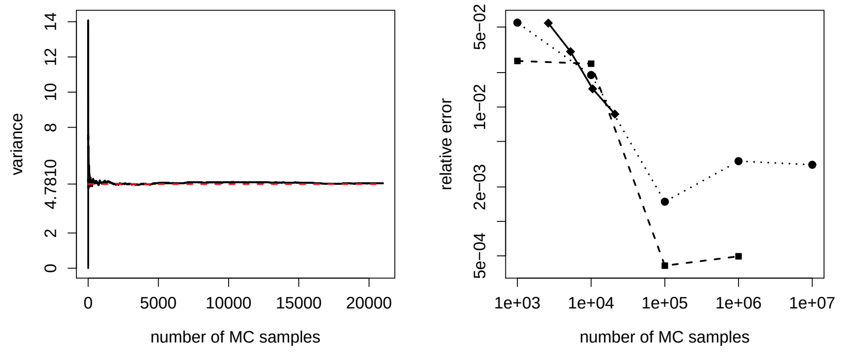

Fig. 1.1 On the left, we fix $h = 10^{-4}$, and plot in solid black the variance of the first N Monte Carlo (MC) samples of (1.3) as a function of N, together with the variance computed using (1.9) in dashed red. Convergence to the reference variance is not monotonic, seen more clearly on the right log-log plot. Here we show results from runs with $h = 10^{-2}$ (circles, dotted), $h = 10^{-3}$ (squares, dashed), and $h = 10^{-4}$ (diamonds, solid). Each point is the relative error between the computed MC variance and the reference variance. In general, a very large number of MC samples may be necessary to achieve the accuracy of (1.9).

For concreteness, let us fix the parameters $\tau = 1$, $T = 5$, $A = -0.2$, $B = 0.3$, and $C = 1.0$. Recall that the time step h is determined by $h = \tau/\ell$ where ℓ is a fixed positive integer. In the left half of Figure 1.1, we set $\ell = 10^4$ (so that $h = 10^{-4}$) and plot in solid black the variance of the first N Monte Carlo samples as a function of N. The total number of samples computed here is $N = 21000$. We also plot in dashed red the variance computed using (1.9), which to four decimals is 4.7810.

In the right half of Figure 1.1, we show three sets of numerical tests. Each point here is the relative error between the computed Monte Carlo variance and the reference variance computed using (1.9), plotted on a log-log scale. In circles (dotted line), we have data for $h = 10^{-2}$. In squares (dashed line), we have data for $h = 10^{-3}$. In diamonds (solid line), we have data for $h = 10^{-4}$. The main point that we take from this plot is that the convergence of the Monte Carlo method to the solution computed using (1.9) is likely to be slow and non-monotonic. This implies that algorithm (1.8-1.9) can be used to significantly speed up simulations of linear SDDE. Algorithm (1.8-1.9) computes a solution with an accuracy that can only be approached by Monte Carlo methods with an extremely large number of samples.

In terms of convergence results, what we are most interested in is the $h \to 0$ convergence of the algorithm (1.5-1.6) or its scalar variant (1.8-1.9), without regard to any Monte Carlo scheme. In the left half of Figure 1.2, we plot the variance computed using (1.9) as a function of h, the time step. The horizontal axis has been scaled logarithmically. The convergence shown is consistent with first-order convergence, i.e., an error that scales likes h. This comes as no surprise; the Euler-Maruyama method used to derive (1.3) exhibits first-order weak convergence. To

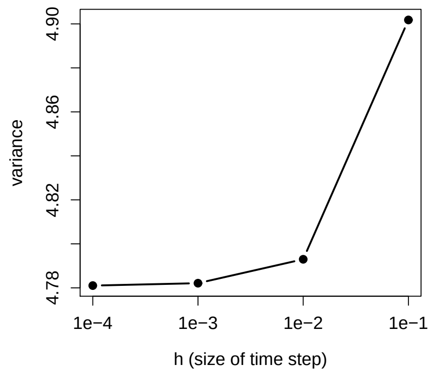

Fig. 1.2 We compute the variance using (1.9) at $T = 5$ using $h = 10^{-j}$ for $j = 1,2,3,4$. The variance appears to converge as h decreases, and the rate is consistent with first-order convergence. Note that the horizontal axis is logarithmically scaled.

state this more formally, let $\mathscr{C}_P^k(\mathbb{R}^N)$ denote the space of k times continuously differentiable real-valued functions on $\mathbb{R}^N$, such that the functions and their derivatives have polynomial growth [5]. Then it is known [3] that there exist $0 < H < 1$ and C (independent of h) such that for all $0 < h < H$ and all $g \in \mathscr{C}_P^{2(\gamma+1)}(\mathbb{R}^N)$,

$$|E(g(\mathbf{X}_T)) - E(g(\mathbf{Y}_{T/h}))| \leq Ch. \tag{1.10}$$

In future work, we aim to build on this result to prove convergence of (1.5-1.6).

1.3.2 Vector Case ($N > 1$)

Now we return to the fully vectorial algorithm (1.5-1.6). Let $N = 2$ and define

$$A = \begin{bmatrix} -0.8 & -1.25 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -0.05 & -0.21 \\ 0.19 & -0.36 \end{bmatrix}, \quad C = \begin{bmatrix} 0.014 & 0.028 \\ 0.042 & 0.014 \end{bmatrix}. \tag{1.11}$$

We fix $\tau = 1$ and set the initial conditions $\mathbf{X}_t = [1,0]$ for $-\tau \leq t \leq 0$. We then seek the solution $\mathbf{X}_t$ of (1.1).

Using algorithm (1.5-1.6), we compute the α and β matrices up to $T = 10$ using a time step of $h = 10^{-2}$. We then use (1.7) to compute the mean vector and variance-covariance matrix of the solution at each point in time from $t = 0$ to $t = T$. In the left half of Figure 1.3, we plot using a black solid line the time evolution of the mean vector. At each point in time at which the solution is computed, we also plot a red line segment whose total length is twice the spectral norm of the variance-covariance matrix at that time. These segments are intended to help visualize the uncertainty in the mean solution, and they are plotted orthogonally to the tangent vectors of the black line.

We see from (1.11) that if B and C were instead equal to zero, the dynamics of (1.1) would be governed by A. The resulting linear system has a globally attracting spiral-type equilibrium at $[0,0]$. This stable spiral dynamic can be seen in the left plot of Figure 1.3. The width of the red band is due entirely to the C matrix in (1.11). If we solve (1.1) with the noise matrix shut off (i.e., $C = 0$) and A and B as in (1.11), the solution would be given by the black line.

To analyze the effect of the time-delay term governed by B, we solve the system again using algorithm (1.5-1.6), but this time with $B = 0$. The solution in this case is plotted in the right half of Figure 1.3. Though the attracting fixed point at $[0,0]$ remains, the dynamics are noticeably different. In this case, we can see that the time delay term acts to slow the system's approach to equilibrium.

Note that producing both plots in Figure 1.3 requires less than 15 minutes on a single core of a desktop computer with a 2.0 GHz Intel Xeon chip. To produce plots of a similar quality using Monte Carlo simulations of (1.1) would require much more computational effort.

Earlier we remarked that the scalar case was simpler than the vector case. In the scalar case, we compute *all* "1×1" matrices β_r^{n+1} at once. Thus in the scalar algorithm (1.8-1.9), the only loop variable is n, discrete time. In the vector case, we must loop over both n and r, since each β^n is now a collection of n different $N \times N$ matrices. This last fact further complicates matters: $\beta^{n-\ell}$ contains a different number of matrices than β^n. At the moment, we use the list data structure in R to store all these objects. In ongoing work, we seek large performance gains by reimplementing the algorithm using more efficient data structures in C++.

1.4 Conclusion

In this paper, we have derived, implemented, and tested a new algorithm for the numerical simulation of linear N-dimensional SDDE of the form (1.1). The algorithm does not involve sampling any random variables, nor does it compute sample paths. Instead, the algorithm computes matrices that yield the full probability density function of the solution. Overall, the results indicate that the new algorithm produces accurate solutions much more efficiently than existing Monte Carlo approaches. Specific features of the algorithm include (i) the ability to generate solutions for many different initial conditions after running the algorithm only once, and (ii) the decoupling of the mean and the variance portions of the algorithm. Future work shall involve establishing the convergence and stability of the algorithm, and applying the algorithm to realistic modeling problems.

Acknowledgements This work was partially supported by a grant from UC Merced's Graduate Research Council.

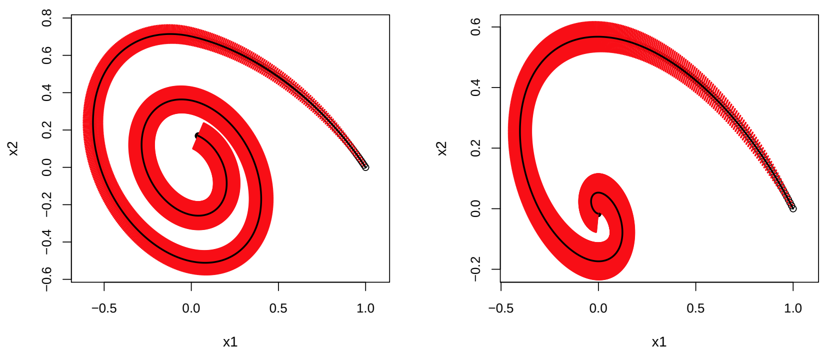

Fig. 1.3 We illustrate vector-valued solutions of (1.1) using algorithm (1.5-1.6) with $\tau = 1$, $h = 10^{-2}$, and initial conditions fixed at $[1,0]$. For both plots, the black line gives the evolution of the mean vector $E[\mathbf{X}_t]$; at each point in time, the red band has total width equal to twice the spectral norm of the variance-covariance matrix $\mathrm{Var}[\mathbf{X}_t]$. For the plot on the left, all three matrices A, B, and C are nonzero and given by (1.11). For the plot on the right, we retain the A and C matrices, but shut off the time delay by setting $B = 0$. These plots demonstrate the utility of algorithm (1.5-1.6).

References

1. Bhat, H.S., Kumar, N.: Spectral solution of delayed random walks. Physical Review E **86**(4), 045701 (2012)
2. Buckwar, E.: Introduction to the numerical analysis of stochastic delay differential equations. Journal of Computational and Applied Mathematics **125**, 297–307 (2000)
3. Buckwar, E., Kuske, R., Mohammed, S.E., Shardlow, T.: Weak convergence of the Euler scheme for stochastic differential delay equations. LMS J. Comput. Math **11**, 60–99 (2008)
4. Crawford III, J.H., Verriest, E.I., Lieuwen, T.C.: Exact statistics for linear time delayed oscillators subjected to Gaussian excitation. Journal of Sound and Vibration **332**(22), 5929–5938 (2013)
5. Kloeden, P.E., Platen, E.: Numerical Solution of Stochastic Differential Equations. Applications of Mathematics: Stochastic Modelling and Applied Probability. Springer, Berlin, Heidelberg (1992)
6. Longtin, A.: Stochastic delay-differential equations. In: F.M. Atay (ed.) Complex Time-Delay Systems, pp. 177–195. Springer, Berlin, Heidelberg (2009)
7. Milton, J.G., Townsend, J.L., King, M.A., Ohira, T.: Balancing with positive feedback: the case for discontinuous control. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **367**(1891), 1181–1193 (2009)
8. Ohira, T., Milton, J.G.: Delayed random walks: Investigating the interplay between delay and noise. In: D.E. Gilsinn, T. Kalmár-Nagy, B. Balachandran (eds.) Delay Differential Equations, pp. 305–335. Springer, New York (2009)
9. Sun, J.Q.: Finite dimensional Markov process approximation for stochastic time-delayed dynamical systems. Communications in Nonlinear Science and Numerical Simulation **14**(5), 1822–1829 (2009)
10. Sun, J.Q., Song, B.: Solutions of the FPK equation for time-delayed dynamical systems with the continuous time approximation method. Probabilistic Engineering Mechanics **27**(1), 69–74 (2012)