Vectorised simulations for stochastic differential equations

P. M. Burrage[*]

(Received 8 August 2003, revised 16 February 2004)

Abstract

Often when solving stochastic differential equations numerically, many simulations must be generated. For example, this approach is required when computing the statistics of the numerical solution, or when verifying the strong order of convergence of a numerical method (when a range of step sizes is also required). Such computational effort can be very slow, and this paper discusses an approach to vectorise the simulation calculations and hence produce an efficient implementation. The numerical simulations here were performed in Matlab but the techniques are equally applicable in a high performance computing environment using, for example, Fortran 90.

[*]Advanced Computational Modelling Centre, University of Queensland, Brisbane, Queensland 4072, Australia. mailto:pmb@maths.uq.edu.au

See http://anziamj.austms.org.au/V45/CTAC2003/Burr for this article, © Austral. Mathematical Soc. 2004. Published June 3, 2004. ISSN 1446-8735

Contents

1 Introduction

Stochastic differential equations (SDEs) model physical systems where there are random effects. For example, application areas include genetic regulation, chemical kinetics and hydrology. Because such systems of differential equations cannot usually be solved analytically, numerical methods are required and these must be designed to perform with a certain order of accuracy (this concept is defined shortly).

In the following, $W(t)$ is a Wiener process which satisfies the properties

$$\begin{aligned} &W(0) = 0\,, \quad E\left[W(t)\right] = 0\,, \quad \text{for all } t \\ &\operatorname{Var}\left[W(t) - W(s)\right] = t - s\,, \quad t > s\,, \end{aligned}$$

and has independent increments on non-overlapping intervals. Thus $W(t)$ is normally-distributed with mean 0 and variance t, written $N(0,t)$. A Wiener process (named after N. Wiener) is sometimes called Brownian Motion, which is a term used to describe the phenomenon of the erratic behaviour of a particle in a liquid, acted on by random impulses, in the absence of friction.

A general SDE driven by d Wiener processes is

$$dy = g_0(y)\,dt + \sum_{j=1}^{d} g_j(y)\,dW_j(t)\,, \quad y(t_0) = y_0\,, \quad t \in [t_0, T]\,, \quad y \in \mathbb{R}^m. \quad (1)$$

This is written in integral form as

$$y(t) = y_0 + \int_{t_0}^{t} g_0(y(s))\,ds + \sum_{j=1}^{d} \int_{t_0}^{t} g_j(y(s))\,dW_j(s)\,. \quad (2)$$

The d integrals in (2) are stochastic integrals with respect to a Wiener process, but the integrals are interpreted differently depending on the underlying rules of calculus being used. In particular, equation (1) can be interpreted in Itô form or Stratonovich form, depending both on modelling considerations and the choice of calculus. Note that it is possible to switch from one representation to the other using the relationship (given here for $d = 1$)

$$\bar{g}_0(y) = g_0(y) - \frac{1}{2}\frac{\partial g_1}{\partial y}(y) g_1(y)\,.$$

Then equation (1) is in Stratonovich form when $\bar{g}_0$ is used in place of g_0.

In some application areas it is important that the sample paths of the numerical approximation be close to the strong solution of an SDE; such trajectories can provide considerable insight into the dynamics and qualitative behaviour of the SDE. In other situations weak solutions are required, and so many trajectories are computed and statistics/moments of the distribution of the solution are obtained. This is discussed in some detail in [2].

An interesting example of an SDE is in filtering theory, where a signal is sent through a noisy channel and has to be filtered from noisy measurements (see [2] for further details). An important FM demodulator is the phase-locked loop (PLL) [5]. The system of equations for the PLL model is

$$\begin{aligned} dy_1 &= -\left(\sqrt[3]{c}y_1 + \sin y_2\right) dt + \sqrt{c}(dW_1 - dW_2)\,, \\ dy_2 &= \left(\frac{1}{2}y_1 - \sin y_2\right) dt - \sqrt{c}\,dW_2\,, \end{aligned}$$

where y_1 represents a phase error and y_2 represents a frequency error. In the stochastic case, as c increases, cycle slips occur and the trajectories can cross multiple boundaries before re-locking—but then the PLL is no longer an efficient demodulator.

In designing numerical methods for SDEs there are two types of order—strong and weak. In this paper, only strong order is relevant. Formally, if $\bar{y}_N$ is the numerical approximation to $y(t_N)$ after N steps with constant step size $h = (t_N - t_0)/N$, then $\bar{y}_N$ is said to converge strongly to y with strong global order p if there exists $C > 0$ (independent of h) and $\delta > 0$ such that

$$E\left[\|\bar{y}_N - y(t_N)\|\right] \leq Ch^p\,, \quad h \in (0,\delta)\,.$$

Note that p can be fractional since the root mean-square order of the Wiener process is $h^{1/2}$.

A numerical method of a certain order of convergence is derived by expanding both the exact solution and the numerical solution of an SDE in a stochastic Taylor series and comparing the terms—the order of accuracy of the method depends on the number of terms that match.

For the SDE (1), the Stratonovich Taylor series expansion begins

$$y_0 + \sum_{j=0}^{d} J_j g_j(y_0) + \sum_{i=0}^{d}\sum_{j=0}^{d} J_{ij}\left(g_j' g_i\right)(y_0) + \cdots\,, \tag{3}$$

where J_i and J_{ij} (using $\circ$ to denote Stratonovich calculus) are defined by

$$J_i^{(n)} = \int_{t_n}^{t_{n+1}} \circ dW_i(s)\,, \quad J_{ij}^{(n)} = \int_{t_n}^{t_{n+1}}\int_{t_n}^{s_2} \circ dW_i(s_1) \circ dW_j(s_2)\,.$$

Equation (3) could be used directly as a numerical procedure to approximate the solution of a SDE but requires the evaluation of derivatives. A derivative free approach is developed through the construction of a general stochastic

Runge-Kutta (SRK) method, defined by

$$\begin{aligned} Y_i &= y_n + \sum_{k=0}^{d} \sum_{j=1}^{s} Z_{ij}^{(k)} g_k(Y_j)\,, \quad i = 1, \ldots, s\,, \\ y_{n+1} &= y_n + \sum_{k=0}^{d} \sum_{j=1}^{s} z_j^{(k)} g_k(Y_j)\,. \end{aligned} \tag{4}$$

Here $Y_1, \ldots, Y_s$ represent the internal stages of the method, y_{n+1} represents the update of the numerical solution at the end of the current step, and $Z_{ij}^{(k)}$ and $z_j^{(k)}$ are random variables based on these Stratonovich integrals. Comparing the expansion of the method with (3) yields order conditions that the method must satisfy to achieve the required accuracy.

A 2-stage SRK method (designated R2) of strong order 1 is

$$\begin{aligned} Z^{(0)} &= h \begin{pmatrix} 0 & 0 \\ \frac{2}{3} & 0 \end{pmatrix}, \quad Z^{(1)} = J_1 \begin{pmatrix} 0 & 0 \\ \frac{2}{3} & 0 \end{pmatrix}, \\ z^{(0)\top} &= h \begin{pmatrix} \frac{1}{4} & \frac{3}{4} \end{pmatrix}, \quad z^{(1)\top} = J_1 \begin{pmatrix} \frac{1}{4} & \frac{3}{4} \end{pmatrix}, \end{aligned} \tag{5}$$

and just requires the random variable J_1 (the Wiener increment), while a 4-stage SRK (E1, requiring J_1 and J_{10}) has strong local order 1.5:

$$\begin{aligned} & Z^{(0)} = h \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & 0 & 0 \\ \frac{3}{2} & -\frac{1}{3} & 0 & 0 \\ \frac{7}{6} & 0 & 0 & 0 \end{pmatrix} \qquad \begin{aligned} z^{(0)\top} &= h\left(\tfrac{1}{4}, \tfrac{3}{4}, -\tfrac{3}{4}, \tfrac{3}{4}\right) \\ z^{(1)\top} &= J_1\left(-\tfrac{1}{2}, \tfrac{3}{2}, -\tfrac{3}{4}, \tfrac{3}{4}\right) \\ z^{(2)\top} &= \tfrac{J_{10}}{h}\left(\tfrac{3}{2}, -\tfrac{3}{2}, 0, 0\right) \end{aligned} \\ & Z^{(1)} = J_1 \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{6} & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix} \qquad Z^{(2)} = \frac{J_{10}}{h} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{2}{3} & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{2} & 0 & 0 \end{pmatrix}. \end{aligned} \tag{6}$$

To implement these methods, J_1 and J_{10} must be simulated at each step. In general, note that $J_i \sim N(0,h)$ and so is sampled as $\sqrt{h}u$ where $u \sim$

$N(0,1)$. Also, J_i and J_{i0} are correlated, and so J_{i0} is computed efficiently using the formula $J_{i0} = \frac{h}{2}\left(J_i + \sqrt{h/3}\,u_1\right)$ where $u_1 \sim N(0,1)$. However, for higher order methods with representations from J_{ij} $(i \neq j)$, the simulation of such stochastic integrals (at each step) is much more complex, and this is discussed in Section 3.

For general d, methods R2 and E1 both reduce to strong order 0.5, because the methods need a representation from all the J_{ij} to perform with higher order. So the natural extension to arbitrary d (by replicating R2 and E1 across all the stochastic components) does not maintain order unless the noise is commutative, that is, $(g_i' g_j)(y) = \left(g_j' g_i\right)(y)$, $i, j = 1, \ldots, d$. This is because, for commutative noise,

$$J_{ij} g_j' g_i + J_{ji} g_i' g_j = (J_{ij} + J_{ji})\, g_j' g_i = J_i J_j g_j' g_i\,,$$

and so the method does not need representation from either J_{ij} or J_{ji}.

To verify that a numerical method is performing with the correct order of accuracy, many trajectories must be computed and then all the trajectories are repeated but with half the step size; the entire step size halving process is repeated several times. The numerical results at the endpoint are then compared, for each step size used, so that the order of accuracy of the method is determined.

This procedure can be extremely time consuming and this has provided the motivation for developing a vectorised implementation of stochastic numerical methods. Although this paper details just the vectorised MATLAB implementation, the paradigm extends to a high performance computing environment. In section 2 we describe the particular functions used in the MATLAB implementation of the above SRK methods and describe the vectorisation of these methods; numerical results are presented. In section 3 we discuss issues that arise in a multi-Wiener process SDE system $(d > 1)$ and present an alternative method (CD) that is appropriate for any d; the vectorisation procedure is extended to allow a vectorised implementation of this style of method, and numerical results are presented. Some issues that arise in approximating the higher order stochastic integrals (for example, J_{ij}) are discussed. In section 4, we present a large-scale SDE system and demonstrate the performance of sequential and vectorised implementations on this problem. Section 5 presents conclusions as well as suggestions for future research.

2 A vectorised implementation

A non-vectorised implementation of the above SRK methods proceeds step-by-step, for each trajectory. For the vectorised approach, we still implement the method step-by-step but now we do this for all N trajectories simultaneously. We also include vectorisation across the dimension m of the SDE in this implementation (that is, all dimensions for all trajectories are computed simultaneously). An alternative approach is to vectorise across the method itself, but that is left for future research.

Thus, for $i = 1, \ldots, s$, the implementation computes $[Y_{i,1}, \ldots, Y_{i,N}]^\top$ and then calculates the update stage:

$$\begin{pmatrix} y_{n+1,1} \\ \vdots \\ y_{n+1,N} \end{pmatrix} = \begin{pmatrix} y_{n,1} \\ \vdots \\ y_{n,N} \end{pmatrix} + \sum_k \sum_j z_j^{(k)} \begin{pmatrix} g_k(Y_{j,1}) \\ \vdots \\ g_k(Y_{j,N}) \end{pmatrix}.$$

In the coding, we need to use several vectorisation functions that are available in MATLAB. Firstly there is $K = \mathrm{kron}(X, Y)$, which returns the Kronecker tensor product of matrices X and Y. If X is $m \times n$ and Y is $p \times q$, then $K = \mathrm{kron}(X, Y)$ is the $mp \times nq$ matrix

$$\begin{pmatrix} x_{11}Y & x_{12}Y & \cdots \\ x_{21}Y & x_{22}Y & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

The `reshape` function is used frequently in this vectorised implementation to rearrange the elements in the work arrays. For example, a 3×2 matrix A is reshaped into a 2×3 matrix B, where the elements of B are taken columnwise from A:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}, \quad B = \texttt{reshape(A,2,3)} = \begin{pmatrix} a_{11} & a_{31} & a_{22} \\ a_{21} & a_{12} & a_{32} \end{pmatrix}.$$

The colon operator is used extensively for processing arrays. The notation $A(:,j)$ denotes the jth column of A, whereas $A(i,:)$ is the ith row. The default stride between elements is 1, but this can be overwritten to skip intermediate elements when required.

The `sum` function is also used in this implementation. Specifying `sum(A,n)` will sum the elements of A in the nth dimension. For example,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \texttt{sum(A,2)} = \begin{pmatrix} a_{11}+a_{12} \\ a_{21}+a_{22} \end{pmatrix}.$$

In the vectorised implementation of the above SRK methods (see equation (4)), the vectors now have length $N\times m$ (that is, the number of trajectories times the dimension of the SDE). The work array $g_0(Y)$ has size $(N\times m)\times s$ where s is the number of stages in the SRK method, while $g(Y)$ represents all the stochastic components and so is a 3D array $(N\times m)\times d\times s$. The implementation commences by obtaining all the random samples required (for all the steps) using the smallest step size h and storing them in 2D arrays (of dimension $Nd\times$ #steps):

$$J_1 = \sqrt{h}\,\texttt{randn}(Nd, \#\text{steps})\,, \quad J_{10} = \frac{h}{2}\left(J_1 + \sqrt{\frac{h}{3}}\,\texttt{randn}(Nd, \#\text{steps})\right).$$

All the trajectories of the numerical approximation are calculated with this h-value. Then the trajectories are repeated but with step size $2h$ (and

Table 1: Timings for Example 1

Implementation	method	time taken
sequential	R2	219.00 seconds
vectorised	R2	1.77 seconds
sequential	E1	377.00 seconds
vectorised	E1	2.28 seconds

then $4h$ and so on, depending on how many sets of trajectories are required). The same Brownian path must be followed for each step size used; starting with the smallest h and successively doubling the step size leads to an efficient way of "consolidating" the random samples while maintaining the path:

$$\begin{aligned} J_{10}\left(:,\tfrac{i}{2}\right) &= J_{10}\left(:,i-1\right) + J_{10}\left(:,i\right) + hJ_1\left(:,i-1\right)\,, \\ J_1\left(:,\tfrac{i}{2}\right) &= J_1\left(:,i-1\right) + J_1\left(:,i\right)\,. \end{aligned}$$

The consolidated values are stored in the first $\frac{1}{2}$(#steps) columns. Note that these relationships preserve the mean and variance of J_1 and J_{10} between the Brownian paths.

To verify the order of convergence of a numerical method, we compare the errors at the endpoint of the integration interval for each of the step sizes used. The ratio of these errors provides an estimate of the order of convergence. When the actual solution of the SDE is not known, then it is approximated by using the numerical method with an even smaller step size (for example, $\frac{1}{4}\times$smallest h, for the purposes of this exercise). The speed-ups obtained are demonstrated by the following example.

Example 1 The following SDE appears in [4].

$$\begin{aligned} dy &= -\alpha(1-y^2)\,dt + \beta(1-y^2)\circ dW\,, \quad y(0)=y_0\,, \\ y(t) &= \frac{(1+y_0)\exp\left(-2\alpha t + 2\beta W(t)\right) + y_0 - 1}{(1+y_0)\exp\left(-2\alpha t + 2\beta W(t)\right) + 1 - y_0}\,, \\ y_0 &= 0\,, \quad \alpha = 1\,, \quad \beta = 2\,, \quad t\in[0,1]\,. \end{aligned}$$

The step sizes used were $h = \frac{1}{200}, \frac{1}{100}, \frac{1}{50}$ and $\frac{1}{25}$ with 500 simulations each, and the timings (produced on a Dell Precision 530, Pentium IV, 1.7 GHz) are presented in Table 1.

3 Multi-Wiener process noncommutative SDEs

The SRK methods R2 and E1 described in Section 2 are suitable for SDEs with one Wiener process or, in the multi-Wiener process case, for SDEs with commutative noise. However, when the noise is non-commutative, these methods only perform with strong order 0.5 as the methods do not have any representation from the higher order stochastic integrals J_{ij}. So a new style of method is required; the method used here is based on the Taylor series expansion but with the derivatives approximated by central differences. The method (CD) (see [1]) has strong order 1 and is suitable for arbitrary dimension d.

However an implementation issue here is the sampling of the J_{ij}, $i, j = 1, \ldots, d$. One approach is to approximate J_{ij} by a Lévy area, using 2^k subintervals (the larger k, the better the approximation but the calculations are more computationally intensive). The approach used here is to approximate J_{ij} by a truncated Fourier series — but how many terms (p) should be included in the series? Kloeden and Platten [4] suggest that p should be proportional to $1/h$, but Wiktorsson [6] (by looking at the truncated terms) can justify p being proportional to $1/\sqrt{h}$. So depending on the step size required, there can be significant computational effort also in the Fourier series approach.

The method CD is (with $J_0 = h$):

$$\begin{aligned}
Y_1 &= y_n\,,\\
Y_{i+1} &= y_n + J_0 g_0(Y_1) + \sqrt{h}\frac{\theta_i}{2} g_i(Y_1)\,, \quad i = 1, \ldots, d\,,\\
Y_{i+d+1} &= y_n + J_0 g_0(Y_1) - \sqrt{h}\frac{\theta_i}{2} g_i(Y_1)\,, \quad i = 1, \ldots, d\,,\\
y_{n+1} &= y_n + \sum_{i=0}^{d} \left(J_i g_i(Y_1) + \sum_{j=1}^{d} \frac{J_{ji}}{\sqrt{h}\theta_j} \left[g_i(Y_{j+1}) - g_i(Y_{j+d+1}) \right] \right).
\end{aligned}$$

For implementation with $d = 2$, we used $\theta_1 = 1\,,\ \theta_2 = \frac{1}{2}$. We describe next the vectorisation of this method but note also that there is further scope for parallelism as all the internal stages depend only on Y_1. The vectorisation procedure described in Section 2 applies here also, but there is the extra requirement of vectorising the J_{ij} and their consolidation. This is achieved by storing the J_{ij} in blocks within the large array JJ:

$$JJ = \begin{pmatrix} (J_{11}) & \cdots & \cdots \\ (J_{21}) & (J_{22}) & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

Each J_{ij} block is of dimension N (#trajectories) by #steps. During the implementation, for the kth step, we need to access the kth column of each J_{ij} block—hence the requirement for the stride to equal the number of steps rather than the default value 1. Because of the relationship $J_{ij} + J_{ji} = J_i J_j$, the block upper triangular part of JJ does not need to be simulated.

Note that the definition of the SDE must be spread across all the trajectories so that its size and shape matches that of the work arrays used in the implementation of the method. This is where the use of the Kronecker tensor product and reshape functions are required.

Example 2 To demonstrate the speed-up obtained by vectorisation of the method CD, the following 2D linear SDE with two Wiener processes was used:

$$\begin{aligned}
&dy = G_0 y\, dt + G_1 y \circ dW_1 + G_2 y \circ dW_2 \,,\\
&y(0) = \begin{pmatrix} 1 & 1 \end{pmatrix}^\top , \quad t \in [0,1]\,, \quad h = \tfrac{1}{32}, \tfrac{1}{16}, \tfrac{1}{8}, \tfrac{1}{4},\\
&G_0 = \begin{pmatrix} -0.9 & 0.0 \\ 0.25 & -0.5 \end{pmatrix}, \quad G_1 = \begin{pmatrix} 0.75 & 0.0 \\ 0.0 & -0.75 \end{pmatrix}, \quad G_2 = \begin{pmatrix} 0.0 & 0.9 \\ 0.9 & 0.0 \end{pmatrix}.
\end{aligned}$$

The SDE is non-commutative, and an approximation to the solution was obtained numerically using $h = \frac{1}{128}$. The sequential implementation of CD required 40 seconds, while the vectorised implementation took 6 seconds. The speed-ups are more substantial for larger problems.

4 A large-scale problem

We present here an example that integrates (in both sequential and vectorised mode) a large-scale stochastic partial differential equation (SPDE) system which has diagonal multiplicative noise in both space and time:

$$\frac{\partial U}{\partial t}(t,x) = -\gamma U(t,x) + \beta \frac{\partial^2 U}{\partial x^2}(t,x) + \sigma \sqrt{1 - U(t,x)^2}\, \zeta(t,x)\,.$$

The constants γ, β and σ are positive, and the space variable x is not restricted. Although x can be in $\mathbb{R}^3$ for this problem, we assume only one spatial dimension—see [3]. With $x \in [0,L]$ and $\Delta x = \frac{L}{N+1}$, and using a second order central difference scheme for the second order partial derivative, the SPDE is spatially discretised. Given $U(t,0) = U_0$ and $U(t,1) = U_{N+1}$, the solution for the N interior points $U = [U_1(t), \ldots, U_N(t)]^\top$ is written in N-dimensional vector form as

$$\frac{dU}{dt} = \left(-\gamma I + \frac{\beta}{(\Delta x)^2} A\right) U + \frac{\beta}{(\Delta x)^2}\left(U_0 e_1 + U_{N+1} e_N\right) + \frac{\sigma}{\sqrt{\Delta x}} B(U)\, \xi(t)\,.$$

Here the $\xi(t)$ is a vector of independent white noise processes, matrix A is $\operatorname{tridiag}(1,-2,1)$, $B(U)$ is the diagonal matrix with entries $B_{kk} = \sqrt{1 - U_k^2}$,

TABLE 2: Timings for the large scale problem of Section 4.

N	R2-sequential	R2-vectorised
2	12.25 seconds	0.172 seconds
4	23.64 seconds	0.360 seconds
8	46.28 seconds	0.672 seconds
16	92.11 seconds	1.390 seconds

and the e_j are the unit basis vectors for $\mathbb{R}^N$. As this large-scale problem has commutative noise, we use method R2 (for example) and replicate it across the N stochastic components, while still maintaining strong order of convergence 1. In fact this is an example of an SDE with diagonal noise.

In this implementation, we set $\gamma = 0$, $\beta = \sigma = 1$ and $L = 1$. The boundary conditions are $U(t,0) = 0$, $U(t,1) = \frac{1}{2}$, and we choose $N = 2, 4, 8$ and 16. There were 500 simulations. See Table 2.

When working with SPDEs (and semi-discretisation), it is essential to use sparse matrices in the vectorised implementation; this will also allow sparse matrix calculations to be performed. For very large problems, memory constraints may affect the results (even in sparse matrix mode); at this stage, then, the implementation should be ported to a high performance computing environment. Comparing the sequential implementation with the vectorised implementation, we see in Table 2 consistent speed-up factors of 65–70.

5 Conclusions

In this paper we demonstrated that substantial speed-ups are possible by vectorising the implementation of the numerical methods. Careful structuring of both the method and the representation of the SDE leads to an efficient vectorised implementation of the numerical methods described in this paper, and this approach is a useful tool for generating many simulations of the numerical solution. Another area for investigation is the technique of parallelisation across the method itself, and this will be the subject of a future paper.

Acknowledgment: I thank Kevin Burrage for his helpful comments.

References

[1] K. Burrage and P. M. Burrage, Derivation of a central differencing method for the numerical solution of SDEs, in preparation, 2004. C359

[2] K. Burrage, P. M. Burrage and T. Tian, Numerical methods for strong solutions stochastic differential equations: an overview, *Proc. R. Soc. Lond. A* **460**, 373–402, 2004. C352

[3] C. R. Doering, Nonlinear parabolic stochastic differential equations with additive coloured noise on $\mathbb{R}^d \times \mathbb{R}_+$: a regulated stochastic quantization, *Commun. Math. Phys.* **109**, 537–561, 1987. C361

[4] P. E. Kloeden and E. Platen, *The Numerical Solution of Stochastic Differential Equations*, Springer-Verlag, 1992. C358, C359

[5] Z. Schuss, *Theory and Applications of Stochastic Differential Equations*, John Wiley and Sons, New York, 1980. C352

[6] M. Wiktorsson, Joint characteristic function and simultaneous simulation of iterated Itô integrals for multiple independent Brownian motions, *Annals of Applied Probability*, **11**, 470–487, 2001. C359