Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational and Applied Mathematics 205 (2007) 1002–1018
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

SDELab: A package for solving stochastic differential equations in MATLAB

Hagen Gilsing[a,1], Tony Shardlow[b,*,2]

[a] *Institut für Mathematik, Humboldt Universität zu Berlin, Unter den Linden 6, Berlin Mitte 10099, Germany*
[b] *School of Mathematics, The University of Manchester, M13 9PL, UK*

Received 22 August 2005

Abstract

We introduce SDELab, a package for solving stochastic differential equations (SDEs) within MATLAB. SDELab features explicit and implicit integrators for a general class of Itô and Stratonovich SDEs, including Milstein's method, sophisticated algorithms for iterated stochastic integrals, and flexible plotting facilities.

MSC: 65C30

Keywords: Stochastic differential equations; MATLAB; Software; Numerical solution; Computations

1. Introduction

MATLAB is an established tool for scientists and engineers that provides ready access to many mathematical models. For example, ordinary differential equations (ODEs) are easily examined with tools for finding, visualising, and validating approximate solutions [22]. The main aim of our work has been to make stochastic differential equations (SDEs) as easily accessible. We introduce SDELab, a package for solving SDEs within MATLAB. SDELab features explicit and implicit integrators for a general class of Itô and Stratonovich SDEs, including Milstein's method and sophisticated algorithms for iterated stochastic integrals. Plotting is flexible in SDELab and includes path and phase plane plots that are drawn as SDELab computes. SDELab is written in C. SDELab and installation instructions are available from either

http://www.ma.man.ac.uk/~sdelab
http://wws.mathematik.hu-berlin.de/~gilsing/sdelab

This article is organised as follows: Section 2 introduces SDEs and the examples we work with. Section 3 describes the numerical integrators implemented in SDELab, including methods for Itô and Stratonovich equations, and the

* Corresponding author. Tel.: +44 161 2755821; fax: +44 161 2755819.
E-mail addresses: gilsing@informatik.hu-berlin.de (H. Gilsing), shardlow@maths.man.ac.uk, Tony.Shardlow@manchester.ac.uk (T. Shardlow).
[1] This work was supported by funding secured by U. Küchler, Institut für Stochastik, HU Berlin and by J. Jacod, Laboratoire de Probabilité, Paris VI/VII.
[2] This work was partially supported by EPSRC grant GR/R78725/01.

0377-0427/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2006.05.037

special case of small noise. Section 4 discusses approximation of iterated integrals. Section 5 shows how SDELab is used and includes the code necessary to approximate geometric Brownian motion. Section 6 uses the explicit solution for geometric Brownian motion to test the SDELab integrators. We also show that SDELab is much faster when dynamic libraries are used to specify the SDE rather than m-files. Section 7 uses SDELab to investigate the bifurcation behaviour of the van der Pol Duffing system.

2. Stochastic differential equations (SDEs)

Consider a drift function $f\colon \mathbb{R}\times\mathbb{R}^d \to \mathbb{R}^d$ and a diffusion function $g\colon \mathbb{R}\times\mathbb{R}^d \to \mathbb{R}^{d\times m}$. Let $W(t)$ be an $\mathbb{R}^m$-valued Brownian motion and $y_0 \in \mathbb{R}^d$ be deterministic initial data. We will consider the Itô SDE

$$\mathrm{d}Y = f(t,Y)\,\mathrm{d}t + g(t,Y)\,\mathrm{d}W, \quad Y(t_0) = y_0, \tag{1}$$

and also the Stratonovich SDE

$$\mathrm{d}Y = f(t,Y)\,\mathrm{d}t + g(t,Y)\circ\mathrm{d}W, \quad Y(t_0) = y_0, \tag{2}$$

where Y and W are evaluated at time t. We will assume that f, g are sufficiently regular that the SDEs have a unique solution $Y(t)$ on $[t_0, T]$ for each y_0. This can be hard to establish, though in specific cases theory is available. For example, if f and g are continuous in (t, Y) and globally Lipschitz in Y, there is a unique strong solution [19]. Stronger conditions will be necessary for statements on the order of accuracy to be correct.

We test SDELab with the following examples.

Van der Pol Duffing: Consider the van der Pol Duffing system [1] ($d = 2,\ m = 1$) where $Y = (Y_1, Y_2)^{\mathrm{T}}$,

$$\mathrm{d}Y = \begin{pmatrix} Y_2 \\ aY_1 + bY_2 - AY_1^3 - BY_1^2Y_2 \end{pmatrix}\mathrm{d}t + \begin{pmatrix} 0 \\ \sigma Y_1 \end{pmatrix}\mathrm{d}W, \tag{3}$$

where a, b, A, B are parameters and σ is noise intensity. This second order system is typical of many problems in physics where the noise impinges directly only on Y_2, which represents the momentum of an oscillator. In constant temperature molecular dynamics, the Langevin equation [18] has this character. The Itô notation is used, but in this case Stratonovich and Itô interpretations are the same.

This system does not have an explicit solution and numerical approximations are required. There are two types of approximation that we may be interested in. The first is *pathwise or strong approximation*: for a given sample of the Brownian path $W(t)$, compute the corresponding $Y(t)$. This is of interest in understanding how varying a parameter affects behaviour. The second is *weak approximation*: for a given test function $\phi\colon \mathbb{R}^d \to \mathbb{R}$, compute the average of $\phi(Y(t))$. For example, we may like to know the average kinetic energy at time t, case $\phi(Y) = \frac{1}{2}Y_2^2$. Release 1 of SDELab focuses on strong approximations and we illustrate its use in understanding the dependence of trajectories on parameters in Section 7.

Geometric Brownian motion in $\mathbb{R}^d$: To demonstrate the convergence of the methods in SDELab in Section 6, we use the following generalisation of geometric Brownian motion to d dimensions [8, p. 151]:

$$\mathrm{d}Y = AY\,\mathrm{d}t + \sum_{i=1}^{m} B_i Y\,\mathrm{d}W_i, \quad Y(0) = y_0, \tag{4}$$

where $A, B_i \in \mathbb{R}^{d\times d}$ and W_i are independent scalar Brownian motions for $i = 1, \ldots, m$. If the matrices A, B_i all commute (so that $AB_i = B_iA$ and $B_iB_j = B_jB_i$ for $i, j = 1, \ldots, m$), the solution is known to be

$$Y(t) = \exp\left(\left(A - \frac{1}{2}\sum_{i=1}^{m} B_i^2\right)t + \sum_{i=1}^{m} B_i W_i(t)\right) y_0. \tag{5}$$

3. Integrators

We introduce the integrators used in SDELab and briefly describe their theory. A full theoretical development is available in [15,8].

We use $\|\cdot\|$ to denote the Euclidean norm on $\mathbb{R}^d$ and the Frobenius norm on $\mathbb{R}^{d\times m}$. $\mathbf{E}$ denotes the expectation over samples of the Brownian motion. $\mathcal{O}(\Delta t^p)$ is a quantity bounded by $K\Delta t^p$, where K is independent of Δt but dependent on the differential equation, the time interval, and initial data.

3.1. Itô SDEs—Euler- and ϑ-methods

One important class of integrators for the ODE $\mathrm{d}Y/\mathrm{d}t = f(t, Y)$ are the ϑ-methods

$$Z_{n+1} = Z_n + [(1-\vartheta)f(t_n, Z_n) + \vartheta f(t_{n+1}, Z_{n+1})]\Delta t, \quad Z_0 = y_0,$$

where ϑ is a parameter in $[0, 1]$, Δt is the time step, and $t_n = t_0 + n\Delta t$. It is well known that this method converges to the exact solution on $[t_0, T]$. If f is sufficiently smooth, $\|Y(t_n) - Z_n\| = \mathcal{O}(\Delta t^p)$ when $t_0 \leqslant t_n \leqslant T$ for $p = 1$ (respectively, $p = 2$) if $\vartheta \neq \frac{1}{2}$ (resp., $\vartheta = \frac{1}{2}$).

These methods are extended to Itô SDEs as follows:

$$Z_{n+1} = Z_n + [(1-\vartheta)f(t_n, Z_n) + \vartheta f(t_{n+1}, Z_{n+1})]\Delta t + g(t_n, Z_n)\Delta W_n, \tag{6}$$

where initial data $Z_0 = y_0$ and $\Delta W_n = W(t_{n+1}) - W(t_n)$. The method includes the stochastic version of explicit Euler ($\vartheta = 0$), which is often called the Euler–Maruyama method following [12], the trapezium rule ($\vartheta = \frac{1}{2}$), and the implicit Euler method ($\vartheta = 1$). This method is implemented in SDELab and referred to as the *Strong Ito Euler method with parameter* ϑ. These methods provide accurate pathwise solutions for small time steps if the drift and diffusion are well behaved. Suppose [8, Theorem 10.2.2] for a constant $K > 0$ that f and g obey

$$\begin{aligned}
&\|f(t, Y_1) - f(t, Y_2)\| + \|g(t, Y_1) - g(t, Y_2)\| \leqslant K\|Y_1 - Y_2\|,\\
&\|f(t, Y)\| + \|g(t, Y)\| \leqslant K(1 + \|Y\|),\\
&\|f(s, Y) - f(t, Y)\| + \|g(s, Y) - g(t, Y)\| \leqslant K(1 + \|Y\|)|s - t|^{1/2}
\end{aligned} \tag{7}$$

for $t_0 \leqslant t \leqslant T$ and $Y, Y_1, Y_2 \in \mathbb{R}^d$. Then, the solution Z_n of (6) converges to the solution $Y(t)$ of (1) and has strong order $\frac{1}{2}$; i.e., $(\mathbf{E}\|Y(t_n) - Z_n\|^2)^{1/2} = \mathcal{O}(\Delta t^{1/2})$ for $t_0 \leqslant t_n \leqslant T$. The conditions (7) are restrictive and do not apply for instance to the van der Pol Duffing system. Theory is available [7] for systems with locally Lipschitz f if the moments can be controlled, but it is hard to characterise completely when the methods will converge. The user should be aware that integrators in SDELab may fail if asked to approximate an SDE with poor regularity.

There are two main issues in implementing this class of method: the generation of random numbers and solution of nonlinear equations. To generate the increments ΔW_n, we must take m independent samples from the Gaussian distribution $N(0, \Delta t)$. SDELab employs the Ziggurat method [11]. This method covers the Gaussian distribution curve with a set of regions, comprising rectangles and a wedge shaped area for the tail. By careful choice of the covering, a Gaussian sample is generated by choosing a region from the uniform distribution and rejection sampling on the chosen region. We use the efficient implementation provided in [11] that uses 255 rectangles and includes its own uniform random number generator. The algorithm is able to generate a Gaussian sample using only two look up table fetches and one magnitude test 99% of the time. To allow efficient linking from our C implementation of the algorithms, the method is implemented within SDELab, rather than calling MATLAB's own random number generators (which use the same basic algorithm [16]).

For $\vartheta \neq 0$, the integrator requires solution of a system of nonlinear equations for all but the most trivial drift functions. We employ Minpack [17], a library of FORTRAN routines freely available through http://www.netlib.org/, to solve the nonlinear equations. Minpack implements a variation of the Powell hybrid method [20] that can be used with exact or numerical derivatives.

3.2. Milstein methods

The basic tool for developing integration methods of higher order is Taylor expansions. Taylor expansions for Itô equations may be developed as follows: expand both drift and diffusion terms in (1) using the Itô Formula:

$$\begin{aligned} \mathrm{d}f(t,Y) &= f_t(t,Y)\,\mathrm{d}t + f_Y(t,Y)f(t,Y)\,\mathrm{d}t + f_Y(t,y)g(t,Y)\,\mathrm{d}W \\ &\quad + \frac{1}{2}\sum_{i,j=1}^{d}\sum_{k=1}^{m} f_{Y_iY_j}(t,Y)g_{ik}(t,Y)g_{jk}(t,Y)\,\mathrm{d}t \end{aligned}$$

and similarly for $g(t,Y)$. Substituting these expressions back into (1) yields

$$\begin{aligned} Y(T) - y_0 &= \int_{t_0}^{\mathrm{T}}\left[f(t_0,y_0) + \int_{t_0}^{t} f_t(s,Y)\,\mathrm{d}s + \cdots\right]\mathrm{d}t + \int_{t_0}^{\mathrm{T}}\Bigg[g(t_0,y_0) \\ &\quad + \int_{t_0}^{t} g_t(s,Y) + g_Y(s,Y(s))f(s,Y(s)\,\mathrm{d}s + \int_{t_0}^{t} g_Y(s,Y(s))g(s,Y(s))\,\mathrm{d}W(s) \\ &\quad + \frac{1}{2}\int_{t_0}^{t}\sum_{i,j=1}^{d}\sum_{k=1}^{m} g_{Y_iY_j}(s,Y(s))g_{ik}(s,Y(s))g_{jk}(s,Y(s))\,\mathrm{d}s\Bigg]\,\mathrm{d}W(t). \end{aligned}$$

Further iteration yields an expansion akin to the Taylor expansion that can be truncated to find new integrators in terms of iterated integrals

$$\int_0^{\Delta t}\int_0^{s_1}\cdots\int_0^{s_{p-1}} \mathrm{d}W_{i_p}(s_p)\,\mathrm{d}W_{i_{p-1}}(s_{p-1})\ldots\mathrm{d}W_{i_1}(s_1),$$

where $\mathrm{d}W_0 = \mathrm{d}t$ and $i_k \in \{0,1,\ldots,m\}$. These terms have order $\Delta t^{(p+q)/2}$, where q is the number of $i_j = 0$, and are the generalisation of the building blocks Δt^p of the deterministic Taylor expansion. It is difficult to compute these quantities. Usually, the work involved outweighs the benefits of high order convergence and `SDELab` provides integrators that depend on the first level of iterated integrals only. The basic example is the Milstein method [13], which is implemented in `SDELab` as the *Strong Itô Milstein method with parameter* ϑ:

$$\begin{aligned} Z_{n+1} &= Z_n + [(1-\vartheta)f(t_n,Z_n) + \vartheta f(t_{n+1},Z_{n+1})]\Delta t + g(t_n,Z_n)\Delta W_n \\ &\quad + \sum_{j=1}^{m}\frac{\partial}{\partial y}g_j(t_n,Z_n)(g(t_n,Z_n)\xi_j), \quad Z_0 = y_0, \end{aligned} \tag{8}$$

where $g_j(t,Z)$ is the jth column of $g(t,Z)$, $\xi_j = (I_{1j,n},\ldots,I_{mj,n})^{\mathrm{T}}$, and

$$I_{ij,n} = \int_{t_n}^{t_{n+1}}\int_{t_n}^{r}\mathrm{d}W_i(s)\,\mathrm{d}W_j(r).$$

We discuss how `SDELab` approximates ξ_j in Section 4. To implement this method without requiring the user to specify the derivative of g, we include derivative free versions,

$$\begin{aligned} Z_{n+1} &= Z_n + [(1-\vartheta)f(t_n,Z_n) + \vartheta f(t_{n+1},Z_{n+1})]\Delta t + g(t_n,Z_n)\Delta W_n \\ &\quad + \sum_{j=1}^{m} Dg(n,j)\xi_j, \quad Z_0 = y_0, \end{aligned} \tag{9}$$

where $Dg(n,j) = (g(t_n,Z^{\mathrm{aux}}_{n,j}) - g(t_n,Z_n))/\Delta t^{1/2}$ and the support vectors $Z^{\mathrm{aux}}_{n,j}$ can be set in `SDELab` as $Z^{\mathrm{aux}}_{n,j} = Z_n + \Delta t^{1/2}g(t_n,Z_n)e_j$ or $Z^{\mathrm{aux}}_{n,j} = Z_n + \Delta t f(t_n,Z_n) + \Delta t^{1/2}g(t_n,Z_n)e_j$ (where e_j denotes the jth standard basis function in

$\mathbb{R}^m$). The Milstein methods converge with order 1, more rapidly than the order $\frac{1}{2}$ convergence of methods (6). Further regularity on f and g is required, but details are not given here; see [8, p. 345].

3.3. Small noise

Many SDEs of interest in science and engineering feature small noise and have the form

$$\mathrm{d}Y = f(t, Y)\,\mathrm{d}t + \varepsilon g(t, Y)\,\mathrm{d}W, \quad Y(t_0) = y_0, \tag{10}$$

for a small parameter ε. Certain methods are especially useful in this context, as they give an improvement over the Euler–Maruyama method when $\varepsilon \ll \Delta t$ and this improvement does not depend on iterated integrals and therefore is efficient. This is true of (6) with $\vartheta = \frac{1}{2}$ (the trapezium rule). SDELab also provides the second order backward differentiation formula (*Strong Itô BDF2*):

$$Z_{n+1} = \tfrac{4}{3}Z_n - \tfrac{1}{3}Z_{n-1} + \tfrac{2}{3}f(t_{n+1}, Z_{n+1})\Delta t + g(t_n, Z_n)\Delta W_n - \tfrac{1}{3}g(t_{n-1}, Z_{n-1})\Delta W_{n-1} \tag{11}$$

for $n \geqslant 2$ and with starting values given by

$$Z_1 = Z_0 + [\tfrac{1}{2}f(t_0, Z_0) + \tfrac{1}{2}f(t_1, Z_1)]\Delta t + g(t_0, Z_0)\Delta W_0, \quad Z_0 = y_0.$$

The solution Z_n from either (11) or (6) with $\vartheta = \frac{1}{2}$ satisfies $(\mathbf{E}\|Y(t_n) - Z_n\|^2)^{1/2} = \mathcal{O}(\Delta t^2 + \varepsilon\Delta t + \varepsilon^2\Delta t^{1/2})$ for $t_0 \leqslant t_n \leqslant T$. See [2,14] for further details. In the small noise case $\varepsilon \ll \Delta t$, the $\mathcal{O}(\varepsilon^2\Delta t^{1/2})$ term becomes negligible and the error is $\mathcal{O}(\varepsilon\Delta t + \Delta t^2)$. The methods look like they have order 1 for a range of Δt even though in the limit $\Delta t \to 0$ they are order $\frac{1}{2}$. This is illustrated in Section 6.

3.4. Stratonovich SDEs

The Itô methods can be used to approximate Stratonovich SDEs by converting to the Itô formulation. To work directly with the Stratonovich SDE, SDELab provides the Euler–Heun and Stratonovich Milstein methods. The Heun method for the ODE $\mathrm{d}Y/\mathrm{d}t = f(t, Y)$ is

$$Z_{n+1} = Z_n + [f(t_n, Z_n) + f(t_{n+1}, Z_{n+1}^{\mathrm{aux}})]\Delta t,$$

where a predicted value $Z_n^{\mathrm{aux}} = Z_n + f(t_n, Z_n)\Delta t$ is used. In the Euler–Heun method for SDEs, we approximate the diffusion with the Heun method and the drift with the ϑ method as in (6). We choose the simplest value of the predictor that results in convergence to the Stratonovich SDE (2). In SDELab, the *Strong Stratonovich Euler–Heun method with parameter* $\vartheta \in [0, 1]$ is

$$\begin{aligned} Z_{n+1} = Z_n &+ [(1-\vartheta)f(t_n, Z_n) + \vartheta f(t_{n+1}, Z_{n+1})]\Delta t \\ &+ \tfrac{1}{2}[g(t_n, Z_n) + g(t_n, Z_n^{\mathrm{aux}})]\Delta W_n, \quad Z_0 = y_0 \end{aligned} \tag{12}$$

with predicted value $Z_n^{\mathrm{aux}} = Z_n + g(t_n, Z_n)\Delta W_n$. This method converges to the solution of (2) with order 1 when the noise is commutative and order $\frac{1}{2}$ otherwise.

SDELab provides *the Strong Stratonovich Milstein method with parameter* ϑ:

$$\begin{aligned} Z_{n+1} = Z_n &+ [(1-\vartheta)f(t_n, Z_n) + \vartheta f(t_{n+1}, Z_{n+1})]\Delta t \\ &+ g(t_n, Z_n)\Delta W_n + \sum_{j=1}^{m} \frac{\partial}{\partial y} g_j(t_n, Z_n)(g(t_n, Z_n)\xi_j), \quad Z_0 = y_0, \end{aligned} \tag{13}$$

where $\xi_j = (J_{1j,n}, \ldots, J_{mj,n})^{\mathrm{T}}$ for the iterated Stratonovich integral $J_{ij,n} = \int_{t_n}^{t_{n+1}} \int_{t_n}^{r} \circ\,\mathrm{d}W_i(s) \circ \mathrm{d}W_j(r)$. The Stratonovich Milstein method converges to the solution of (2) with order 1. Again SDELab includes versions that do not require user-supplied derivatives.

4. Iterated stochastic integrals

We look at how SDELab generates second order iterated integrals. We work with Stratonovich iterated integrals on $[0, \Delta t]$ and use the notation

$$J_i = \int_0^{\Delta t} \mathrm{d}W_i(s), \quad J_{ij} = \int_0^{\Delta t} \int_0^r \circ \mathrm{d}W_i(s) \circ \mathrm{d}W_j(r).$$

SDELab computes the second order Itô integrals from the Stratonovich version by $I_{ij} = J_{ij}$ for $i \neq j$ and $I_{ii} = J_{ii} - \frac{1}{2}\Delta t$ for $i = 1, \ldots, m$. There are a number of important special cases that are used by SDELab to improve efficiency. If the diffusion $g(t, Y)$ is *diagonal*, J_{ij} are not required for $i \neq j$. If

$$\frac{\partial}{\partial x_i} g_{kj}(t, Y) g(t, Y) = \frac{\partial}{\partial x_j} g_{ki}(t, Y) g(t, Y), \tag{14}$$

for $k = 1, \ldots, d$ and $i, j = 1, \ldots, m$, the diffusion is said to be *commutative* and the identity $J_{ij} + J_{ji} = J_i J_j$ is used to simplify the Milstein method. In particular, we compute only the product $J_i J_j$ and avoid approximating J_{ij} directly. If $g(t, Y)$ does has not have the above structures, we must approximate each J_{ij}. There are a number of efficient methods [21,5] for sampling J_{ij} with $m = 2$. Unfortunately, J_{ij} cannot be generated pairwise for $m > 2$ because correlations are significant. Such specialist methods are not included in SDELab as we prefer algorithms that are widely applicable. SDELab generates samples using a truncated expansion of the Brownian bridge process with a Gaussian approximation to the tail.

The Brownian bridge process $W_i(t) - (t/\Delta t) W_i(\Delta t)$ for $0 \leqslant t \leqslant \Delta t$ has Fourier series

$$W_i(t) - \frac{t}{\Delta t} W_i(\Delta t) = \frac{1}{2} a_{i0} + \sum_{r=1}^{\infty} a_{ir} \cos \frac{2\pi r t}{\Delta t} + b_{ir} \sin \frac{2\pi r t}{\Delta t}, \tag{15}$$

for $i = 1, \ldots, m$, where (by putting $t = 0$)

$$a_{i0} = -2 \sum_{r=1}^{\infty} a_{ir} \tag{16}$$

and the coefficients b_{ir}, a_{ir} are independent random variables with distributions $N(0, \Delta t/2\pi^2 r^2)$ for $r = 1, 2, \ldots$. This representation was developed [9] to express numerically computable formulae for iterated stochastic integrals and in particular J_{ij} by truncating the expansions to p terms. By integrating (15) over $[0, \Delta t]$ with respect to $\mathrm{d}t$, we find $J_{i0} = \frac{1}{2}\Delta t (J_i + a_{i0})$, and using the symmetry relation $J_{0i} + J_{i0} = J_i J_0$, we see $J_{0i} = \frac{1}{2}\Delta t (J_i - a_{i0})$. Integrating (15) over $[0, \Delta t]$ with respect to $W_j(t)$,

$$J_{ij} = \tfrac{1}{2} J_i J_j - \tfrac{1}{2}(a_{j0} J_i - a_{i0} J_j) + \Delta t A_{ij}, \quad i, j = 1, \ldots, m, \tag{17}$$

where $A_{ij} = (1/\Delta t) \sum_{r=1}^{\infty} \zeta_{ir}^* \eta_{jr}^* - \eta_{ir}^* \zeta_{ji}^*$ and $\eta_{jr}^* = \sqrt{\pi r} a_{jr}$ and $\zeta_{jr}^* = \sqrt{\pi r} b_{jr}$. Because ζ_{jr}^*, η_{jr}^*, and $W_j(\Delta t)$ are independent, we easily find ζ_{jr}^* and η_{jr}^* by sampling from $N(0, \Delta t/2\pi r)$. We approximate A_{ij} by truncating the sum to p terms,

$$A_{ij}^p = \frac{1}{\Delta t} \sum_{r=1}^{p} \zeta_{ir}^* \eta_{jr}^* - \eta_{ir}^* \zeta_{ji}^*, \tag{18}$$

and define the approximate iterated integral $J_{ij}^p = \frac{1}{2} J_i J_j - \frac{1}{2}(a_{j0}^p J_i - a_{i0}^p J_j) + \Delta t A_{ij}^p$, where from (16)

$$a_{i0}^p = -\sum_{r=1}^{p} \frac{2}{\sqrt{\pi r}} \zeta_{ir}^*. \tag{19}$$

To understand the importance of the tail correction, consider the estimate

$$(\mathbf{E}[|\tilde{J}_{ij}^p - J_{ij}|^2])^{1/2} \leqslant \frac{\Delta t}{\sqrt{2p\pi}},$$

which holds for approximations $\tilde{J}_{ij}^p = \frac{1}{2}J_iJ_j - \frac{1}{2}(\tilde{a}_{j0}^p J_i - \tilde{a}_{i0}^p J_j) + \Delta t A_{ij}^p$ that include a higher order correction to $\tilde{a}_{i0}^p$,

$$\tilde{a}_{i0}^p = a_{i0}^p - 2\sqrt{\Delta t \rho_p}\mu_{jp}, \tag{20}$$

where $\mu_{jp} = (1/\sqrt{\Delta t \rho_p})\sum_{r=p+1}^{\infty} a_{jr}$ and $\rho_p = (\frac{1}{12}) - (\frac{1}{2}\pi^2)\sum_{r=1}^{p} 1/r^2$. To use this approximation in the Milstein scheme and retain order 1 convergence, $\tilde{J}_{ij}^p$ must approximate $\tilde{J}_{ij}$ with error $\mathcal{O}(\Delta t^{3/2})$ and the number of terms, p, in the expansions must be at least $\mathcal{O}(1/\Delta t)$.

Wiktorsson [23] introduced a technique that reduces the number, p, of terms necessary to achieve an $\mathcal{O}(\Delta t^{3/2})$ error. Recall the Levy area

$$\mathscr{A}_{ij} = \tfrac{1}{2}(J_{ij} - J_{ji}) = a_{i0}J_j + a_{j0}J_i + \Delta t A_{ij}.$$

Wiktorsson uses the conditional (on $W_j(\Delta t)$) joint characteristic function of the Levy areas to derive a Gaussian approximation to the tail of A_{ij}. Sampling from this Gaussian provides a small correction to J_{ij}^p that improves the rate of convergence. `SDELab` implements the following algorithm:

(1) Fix a constant C. Choose the smallest number p such that

$$p \geqslant \frac{1}{C\pi}\sqrt{\frac{m(m-1)}{24\Delta t}}\sqrt{m + 4\sum_{j=1}^{m} W_j(\Delta t)^2/\Delta t}. \tag{21}$$

Note that the number, p, of terms grows like $1/\sqrt{\Delta t}$, not $1/\Delta t$ as in the first method, and that p depends on the path.

(2) Using (18)–(19), compute approximations $J_{ij}^p = \frac{1}{2}J_iJ_j - \frac{1}{2}(a_{j0}^p J_i - a_{i0}^p J_j) + \Delta t A_{ij}^p$.

(3) We now define the tail approximation $\mathscr{A}^{p,\mathrm{tail}}$. Let $x, y \in \mathbb{R}^m$, $M = \frac{1}{2}m(m-1)$, and e_i^m denote the ith standard basis element in $\mathbb{R}^m$. We introduce $P_m\colon \mathbb{R}^{m^2} \to \mathbb{R}^M$, the linear operator defined by $P_m(x \otimes y) = y \otimes x$, and $K_m\colon \mathbb{R}^{m^2} \to \mathbb{R}^M$, the linear operator defined by $K_m(e_i^m \otimes e_j^m) = e_{k(i,j)}^M$ and $K_m(e_j^m \otimes e_i^m) = 0 = K_m(e_j^m \otimes e_j^m)$, where $i < j$ and $k(i, j)$ is the position of (i, j) in the M term sequence

$$(1,2), (1,3), \ldots, (1,m), (2,3), \ldots, (2,m), \ldots, (m-1,m).$$

Denote by I_m the $m \times m$ identity matrix. The tail approximation is

$$\mathscr{A}^{p,\mathrm{tail}} = (I_{m^2} - P_m)\,K_m^{\mathrm{T}}\,\frac{\Delta t}{2\pi}\,a_p^{1/2}\sqrt{\Sigma_\infty}G_p, \tag{22}$$

where $a_p = \sum_{k=p+1}^{\infty} k^{-2}$, $G_p \in \mathbb{R}^M$ is chosen from the distribution $N(0, I_M)$,

$$\Sigma_\infty = 2I_M + \frac{2}{\Delta t}K_m(I_{m^2} - P_m)(I_m \otimes W(\Delta t)W(\Delta t)^{\mathrm{T}})(I_{m^2} - P_m)K_m^{\mathrm{T}},$$

and $W(\Delta t) = (W_1(\Delta t), \ldots, W_m(\Delta t))^{\mathrm{T}}$. To compute (22), we use the following expression for the square root of Σ_∞ [23]:

$$\sqrt{\Sigma_\infty} = \frac{\Sigma_\infty + 2\alpha I_M}{\sqrt{2}(1+\alpha)} \quad \text{where } \alpha = \sqrt{1 + \sum_{j=1}^{m} W_j(\Delta t)^2/\Delta t}.$$

(4) Add the correction term to J_{ij}^p to define $J_{ij}^{p+\mathrm{tail}} = J_{ij}^p + \mathscr{A}_{ij}^{p,\mathrm{tail}}$.

Under the truncation condition (21), [23] proves that

$$\max_{ij} \mathbf{E}[|J_{ij} - J_{ij}^{p+\text{tail}}|^2 | W(\Delta t)] \leqslant C^2 \Delta t^3,$$

where $\mathbf{E}[\Upsilon | W(\Delta t)]$ denotes the expectation of Υ conditioned on $W(\Delta t)$. In terms of Gaussian samples, the tail expansion is justified in the limit $\Delta t \to 0$. The Euler methods (6) required $\mathcal{O}(1)$ Gaussians per time step, Milstein (8) with $\tilde{J}_{ij}$ requires $\mathcal{O}(1/\Delta t)$ Gaussians, and Milstein with $\tilde{J}_{ij}^{p+\text{tail}}$ requires $\mathcal{O}(\Delta t^{-1/2})$. On the other hand, rates of convergence are $\mathcal{O}(\Delta t^{1/2})$ for Euler and $\mathcal{O}(\Delta t)$ for Milstein. Hence, to achieve a particular level of accuracy ε both Euler and Milstein with $\tilde{J}_{ij}$ require ε^{-2} samples, whilst Milstein with $\tilde{J}_{ij}^{p+\text{tail}}$ requires only $\varepsilon^{-3/2}$ samples. Asymptotically in $\Delta t \to 0$, the use of the tail approximation means fewer Gaussian samples are required.

In practice, the method is expensive for large m, because the covariance matrix Σ_∞, which is an $M \times M$ matrix where $M = \frac{1}{2}m(m-1)$, is treated as a dense matrix with $\mathcal{O}(m^4)$ entries. This is impractical for very high m as it is hard to take Δt sufficiently small to see its benefits. To give some understanding, the table compares the two methods J_{ij}^p and $J_{ij}^{p+\text{tail}}$ for different values of m. The time to compute 10^6 samples is given (in seconds) and the error in computing the variance (again using 10^6 samples) of $J_{12}/\mathrm{d}t$, which is known to equal $\frac{1}{2}$, is given.

m	Δt	J_{ij}^p:	time	error	$J_{ij}^{p+\text{tail}}$:	time	error
5	0.1		1.9	0.1		3.5	3.7e − 3
	0.01		4.8	2.3e − 2		3.5	5.5e − 4
10	0.1		4.2	0.9e − 1		13.65	6.5e − 3
	0.001		66.04	2e − 3		12.39	6.3e − 5
100	0.1		114	0.1		3420	0.04

5. The use of `SDELab`

We describe the most important features of `SDELab`; extensive documentation is provided online. To start using `SDELab` within MATLAB, type `sdelab_init`. To find approximate paths for (1) or (2), one of the following is used

```
[t,y] =sdelab_strong_solutions(fcn, tspan, y0, m, opt, ...);

sdelab_strong_solutions(fcn, tspan, y0, m, opt, ...);
```

The return values give approximate solutions `y(:,i)` at times `t(i)`. If `[t, y]` is omitted, a MATLAB figure appears and the approximate paths are plotted as they are computed. The arguments are

fcn: `SDELab` provides the single structure `fcn` for specifying the drift f and diffusion g. The `fcn` fields may point to a variety of implementations, including m-files, mex files, and dynamic library routines. This flexibility allows users to prototype quickly using m-files and develop efficient code by linking to dynamic libraries of C or FORTRAN routines.

When using m-files, the fields `drift` and `diff_noise`, and optional fields `drift_dy` and `diff_noise_dy` contain the names of the m-files. An example is given later. When using dynamic libraries, the fields `drift`, `diff_noise`, etc. each have subfields `Libname` (name of dynamic library) and `Init_fcn`, `Exec_fcn`, and `Cleanup_fcn` (names of functions in the dynamic library that initialise, compute, and clean up).

tspan is a vector that indicates the time interval for integration $[t_0, T]$. If `tspan` has more than two points, it specifies the times at which $Y(t)$ is approximated and is returned in `t`.

y0 is the initial condition.

m equals the dimension of the Brownian motion W.

opt is a structure whose fields set `SDELab` options.

... is an optional list of model parameters.

The following are specified by setting the corresponding field in `opt`.

MaxStepSize: The time step Δt is the largest value less or equal to **MaxStepSize** such that an integer multiple of steps fits into $[t_0, T]$. A default value of $(T - t_0)/100$ is used.

IntegrationMethod specifies the type of equation (Itô /Stratonovich) as well as the integrator. The default is `StrongItoEuler` and the options are

`StrongItoMilstein, StrongItoBDF2,`

`StrongStratoEulerHeun, StrongStratoMilstein.`

The parameter ϑ in (6), (8), (12), and (13) is controlled by the following:

`StrongItoEuler.Alpha, StrongItoMilstein.Alpha,`

`StrongStratoEulerHeun.Alpha, StrongStratoMilstein.Alpha.`

RelTol is the relative error used by Minpack as a termination criterion.

MaxFeval controls the behaviour of the nonlinear Minpack solve. If positive, `MaxFeval` is the maximum number of function evaluations allowed by Minpack. If -1, the Minpack default value is chosen.

Stats controls the reporting of number of function calls and Minpack information. It should be set to `on` or `off`.

NoiseType indicates the structure of the diffusion term. If `NoiseType = 1`, the diffusion is considered to be unstructured and second order iterated integrals are approximated using Wiktorsson's method. If `NoiseType = 2`, the diffusion is treated as diagonal and if `NoiseType = 3` as commutative (see (14)).

MSIGenRNG.SeedZig is the seed for the random number generator.

OutputPlot is set to `1` if plots are required; `0` otherwise.

OutputPlotType specifies the type of plot. The possibilities are

`sdelab_path_plot` (path plot; default);
`sdelab_phase_plot` (two dimensional phase plot);
`sdelab_time_phase` (two dimensional path plots against time);
`sdelab_phase3_plot` (three dimensional phase plot).

OutputSel controls which components of `y` are used in the plots.

We show how to approximate geometric Brownian motion (4) with `SDELab` using m-files to specify drift and diffusion functions. The drift is defined by the following m-file:

```
function [z] = drift(t, y, varargin)
A = varargin{2}.A; % Extract parameters
z = A*y;           % Compute drift
```

The specification of the diffusion is more involved. `SDELab` requires that we specify the diffusion in two ways: (1) as the product of the matrix $g(t, Y)$ with the Brownian increment, which is beneficial for sparse diffusion matrices, and (2) as the matrix $g(t, Y)$. `SDELab` uses the two ways in its implementation of the Milstein methods to reduce the number of function calls.

```
function z = diff_noise(t, y, dw, flag, varargin)
 B = varargin{2}.B; % Extract parameters
 m = length(dw);
 d = length(y);
 B2 = zeros(d,m); % Compute the diffusion
 for (i = 1:m)
  B2(:,i) = B(:,:,i) * y;
 end;
 if (flag == 0)
  z = B2 * dw;
 else
  z = B2;
 end;
```

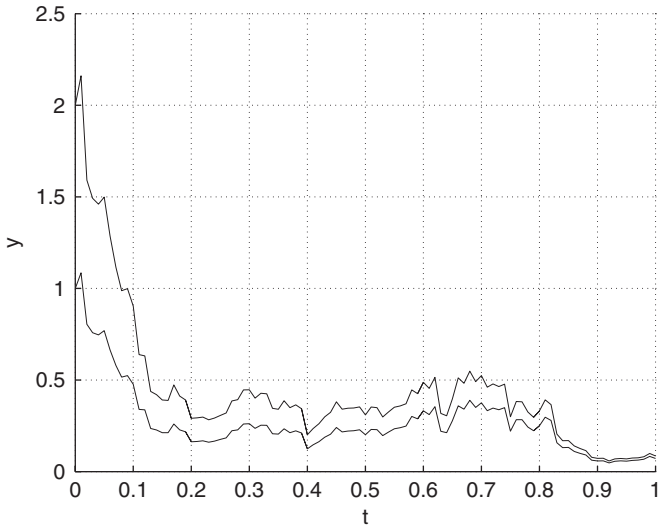

Fig. 1. The two components of a realisation of $Y(t)$ satisfying (23).

We now use SDELab to approximate paths of

$$\mathrm{d}Y = \begin{pmatrix} -0.5 & 0 \\ 0 & -1 \end{pmatrix} Y\,\mathrm{d}t + Y\,\mathrm{d}W_1(t) + Y\,\mathrm{d}W_2(t), \quad Y(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}. \tag{23}$$

Set up the problem dimensions, time interval and initial data:

```
d = 2;              % dimension of y
m = d;              % dimension of W(t)
tspan = [0, 1];     % time interval
y0 = [1; 2];        % initial condition
```

Define the drift and diffusion functions with their parameters:

```
fcn.drift = 'drift';                % name of MATLAB m-files
fcn.diff_noise = 'diff_noise';
params.A = [-0.5, 0; 0, -1];        % parameters
params.B = zeros(d,d,m);
params.B(:, :, 1) = diag([1; 1]);
params.B(:, :, 2) = diag([1; 1]);
```

Finally, run SDELab with the default method, Itô Euler with $\vartheta = 0$.

```
opt.IntegrationMethod = 'StrongItoEuler';
opt.MSIGenRNG.SeedZig = 23; sdelab_init;
sdelab_strong_solutions(fcn, tspan, y0, m, opt, params);
xlabel('t'); ylabel('y');
```

A window pops up and you see the path plotted as it is computed. See Fig. 1.

To assist the nonlinear solver, the user may provide derivatives of the drift function. SDELab can utilise a function drift_dy that returns the Jacobian matrix of f with entries $\partial f_i(t, Y)/\partial Y_j$ for $i, j = 1, \ldots, d$. For (4), the Jacobian is A.

```
function z = drift_dy(t, y, varargin)
A = varargin{2}.A;
z = A;
```

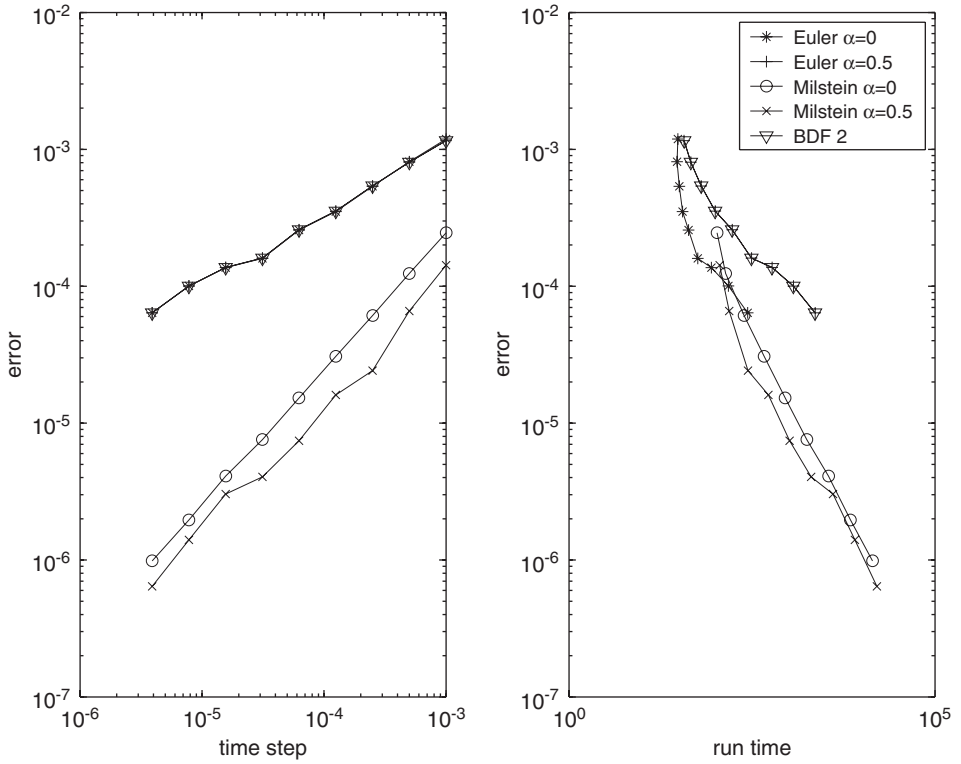

Fig. 2. Plots of error (computed with 2000 samples) against time step and run time for Itô geometric Brownian motion. Notice Milstein methods have order 1 convergence, and the Euler and BDF 2 methods have order $\frac{1}{2}$ convergence.

Let $g_{ij}(t, Y)$ denote the (i, j) entry of $g(t, Y)$ for $i = 1, \ldots, d$ and $j = 1, \ldots, m$. `SDELab` can utilise a function `diff_noise_dy` that returns the derivative of the jth column of $g(t, Y)$ with respect to Y in the direction $\mathrm{d}y \in \mathbb{R}^d$; that is, $\mathrm{d}g_j\,\mathrm{d}y \in \mathbb{R}^d$, where $\mathrm{d}g_j$ is the $d \times d$ matrix with (i, k) entry $\partial g_{ij}/\partial Y_k$. For (4), $\mathrm{d}g_j\,\mathrm{d}y = B_j\,\mathrm{d}y$.

```
function z =diff_noise_dy(t, y, dy, j, varargin)
B=varargin{2}.B;
z=B(:,:,j)*dy;
```

Finally define the `fcn` structure and compute the solution using Itô Milstein with $\vartheta = 1$. Rather than plot, we store the results in `[t, y]`.

```
fcn.drift='drift';
fcn.drift_dy='drift_dy';
fcn.diff_noise='diff_noise';
fcn.drift_noise_dy='diff_noise_dy';
opt.IntegrationMethod='StrongItoMilstein';
opt.StrongItoMilstein.Alpha=1.0;
[t, y]=sdelab_strong_solutions(fcn, tspan, y0, m, opt, params);
```

6. Geometric Brownian motion

We consider the behaviour of five of `SDELab`'s Itô methods for approximating geometric Brownian motion (4):

- Eq. (6) with $\vartheta = 0$: Euler–Maruyama.
- Eq. (6) with $\vartheta = \frac{1}{2}$: trapezium rule with explicit diffusion.
- Eq. (8) with $\vartheta = 0$: Milstein with explicit drift.

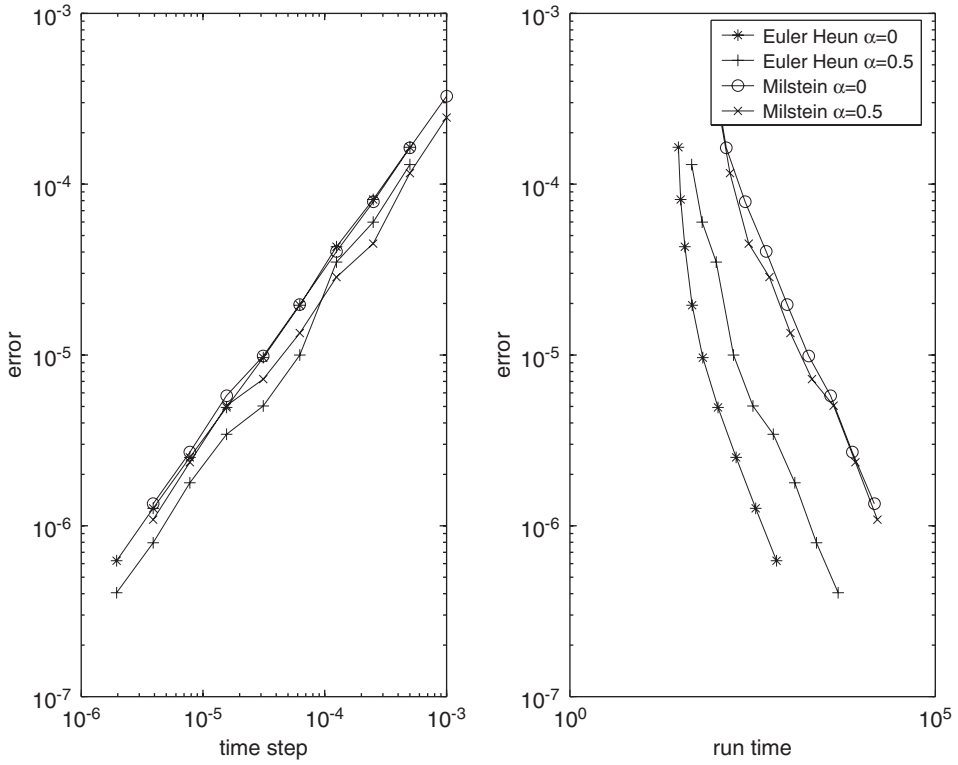

Fig. 3. Plots of error (computed with 2000 samples) against time step and run time for Stratonovich geometric Brownian motion. As the noise is commutative, the Euler–Heun method has the same order 1 convergence as the Milstein methods. The Milstein method is slower to compute, as this test was done *without* the commutative noise flag set.

- Eq. (8) with $\vartheta = \frac{1}{2}$: Milstein with trapezium rule for drift.
- Eq. (11): BDF2.

The following matrices are used

$$A = -2I, \quad B_1 = \begin{pmatrix} 0.3106 & 0.1360 \\ 0.1360 & 0.3106 \end{pmatrix}, \quad B_2 = \begin{pmatrix} 0.9027 & -0.0674 \\ -0.0674 & 0.9027 \end{pmatrix}.$$

These matrices are commutative and the exact solution (5) is available. Using the exact solution, we compute the strong error with L samples by

$$\left(\frac{1}{L} \sum_{L \text{ samples}} \|Y(T) - Z_N\|^2 \right)^{1/2}, \quad N\Delta t = T.$$

Fig. 2 plots error against time step and run time (with drift and diffusion functions implemented as C dynamic library functions), with $L = 2000$. To test the approximations to the iterated integrals, we set `opt.NoiseType = 1` during these calculations (rather than take advantage of the commutative structure). We see the order 1 convergence of the Milstein methods and the order $\frac{1}{2}$ convergence of the Euler methods. Even allowing for the extra time to compute a single time step, it is more efficient to use the Milstein methods in this case. Fig. 3 shows the same plot for the Stratonovich version of geometric Brownian motion. Here the Euler–Heun method has order 1 because the matrices are commutative. Fig. 4 shows the same plot for the Itô equation with small noise; specifically, (4) with the diffusion matrices B_i replaced by εB_i with $\varepsilon = 10^{-3}$. We clearly see the benefits of choosing the method carefully and the BDF 2 and trapezium rule ((6) with $\vartheta = \frac{1}{2}$) are most efficient. Fig. 5 shows how the CPU time depends on problem

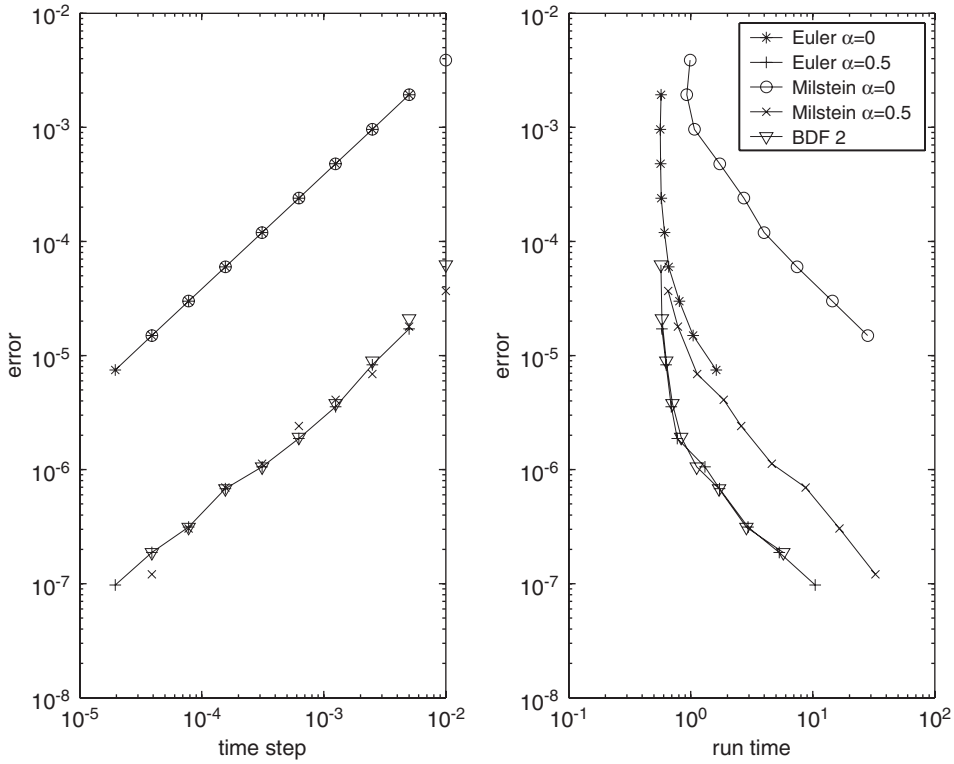

Fig. 4. Plots of error (computed with 2000 samples) against time step and run time for the Itô equation (4) with small noise. Each method appears to have order 1.

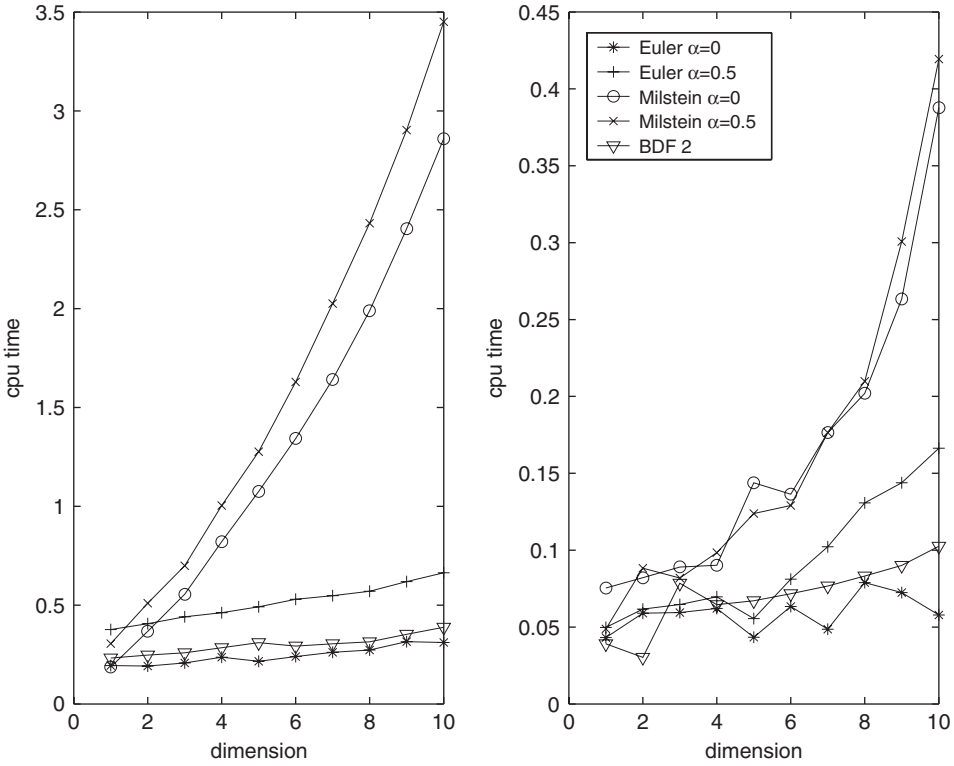

Fig. 5. Plots of CPU time against problem dimension $m = d$ for the Itô equation (4) for both MATLAB m-files (left) and dynamic library functions (right).

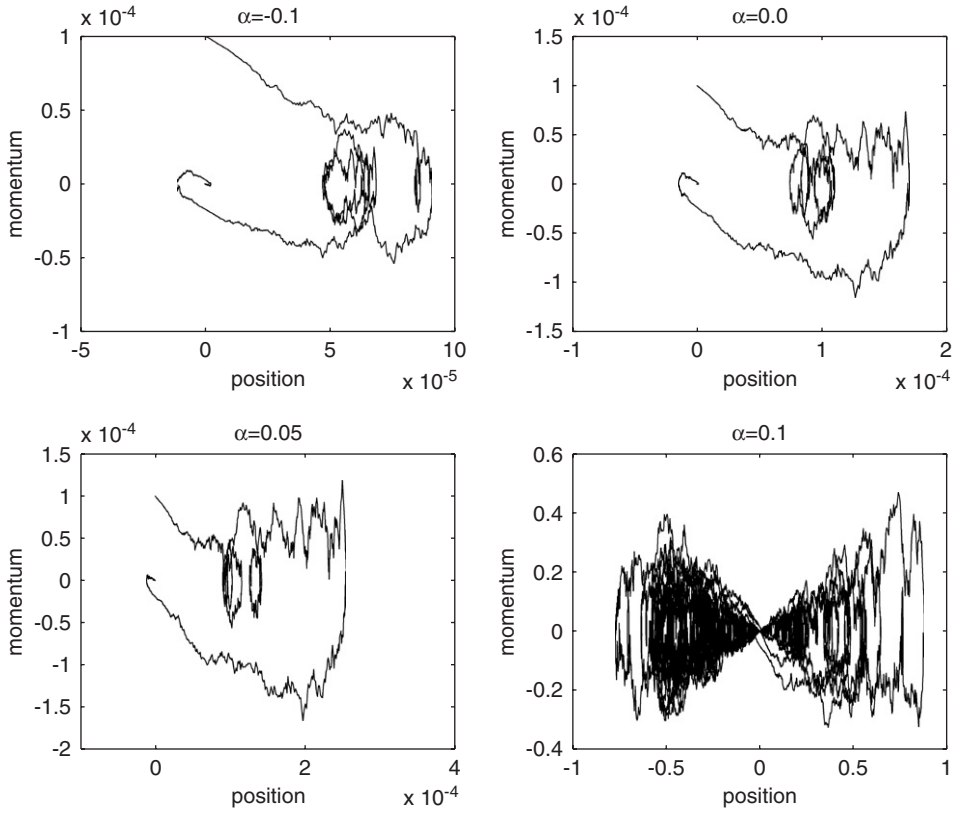

Fig. 6. Paths of (3) for $a = -0.2, 0, 0.05, 0.1$ with $b = -1.0$ over time interval [500, 1000] with initial data [0, 0.001] specified at $t_0 = 0$. Notice the change of scale in the bottom right plot.

dimension m. Matrices $A, B_1, \ldots, B_m$ are chosen and approximations computed using both the m-file and dynamic library implementation of the drift and diffusion functions. We see the cost of using Milstein methods scales badly with m due to the difficulties of computing the stochastic integrals. We also notice considerable speed improvements in using a dynamic library implementation.

7. Van der Pol Duffing

Consider the van der Pol Duffing system (3) with parameters $A = B = 1$. We use the plotting facilities of `SDELab` to illustrate two bifurcations in this system. In order to use the same Brownian path for each plot, we set the seed of the random number generator at the start of each simulation. This is effective if we fix the time step for all our simulations. In the long run, we would like to add functionality to decrease the time step and refine the same Brownian motion.

The MATLAB m-files are given in Appendix A. Set the `fcn` structure as in the previous example, and set the dimensions, initial data, and time interval for the van der Pol Duffing system:

```
d=2; m=1;          %problem dimensions
tspan=[0, 500];    %time interval
y0=[0.0, 0.0001]; %initial condition
```

Define the problem parameters:

```
params.a=-1.0;          params.b=0.1;
params.A=1.0;           params.B=1.0;
params.sigma=0.1;
```

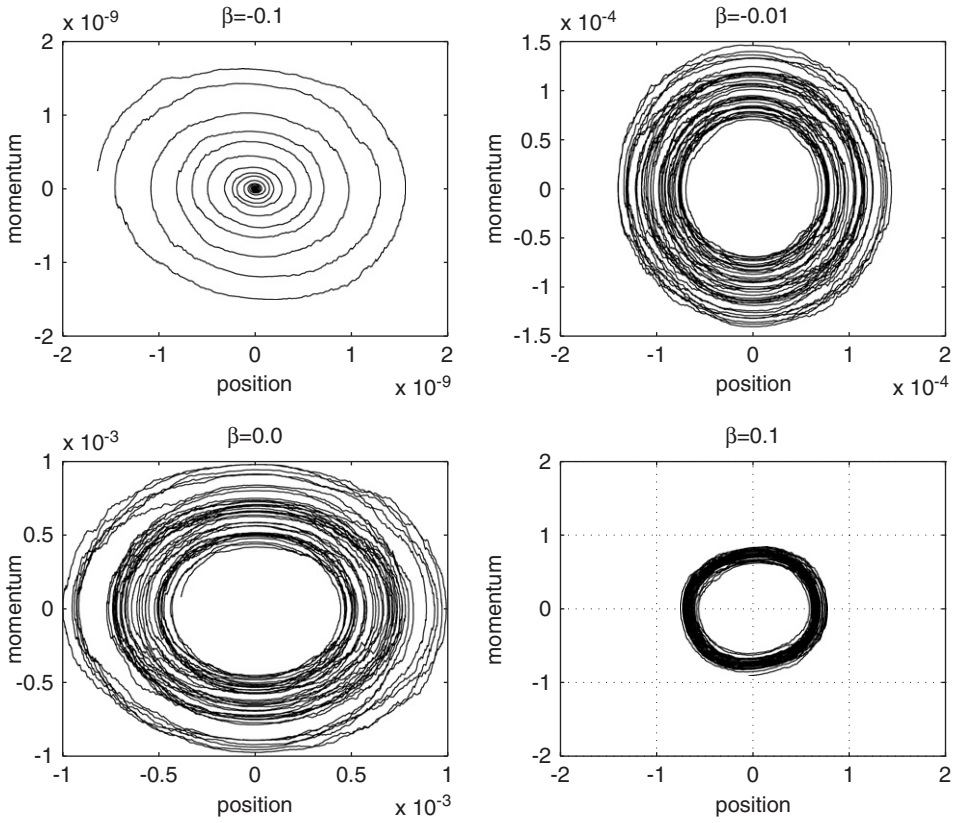

Fig. 7. Paths of (3) for $b = -0.1, -0.01, 0, 0.1$ with $a = -1.0$ over time interval [250, 500] with initial data [0, 0.001] specified at $t_0 = 0$. Notice the change of scales in the plots.

A phase plot with seed 23 and maximum time step 0.01 can be produced as follows:

```
opt.MaxStepSize = 1e-2;
opt.OutputPlotType = 'sdelab_phase_plot';
opt.MSIGenRNG.SeedZig = 23;
sdelab_strong_solutions(fcn, tspan, y0, m, opt, params);
```

It is now easy to explore the dynamical behaviour of the system and its response to varying a and b. Without noise (case $\sigma = 0$) and with $\beta < 0$, the system experiences a pitchfork bifurcation (when a fixed point loses its stability and gives rise to two stable fixed points) as the parameter a crosses 0. We explore this situation for $\sigma = 0.4$ in Fig. 6. We see the dynamics do not settle down to fixed points when there is noise, but oscillate near two meta stable states. Only the last plot shows the two meta stable states created by the bifurcation (notice the change in scale on the plots), even though three of the figures have parameter $a \geqslant 0$. This is a well known phenomenon [1]: noise delays a pitchfork bifurcation. In this case, the bifurcation is delayed until $a \approx 0.1$.

If $a < 0$, a Hopf bifurcation (or creation of a periodic orbit) can be found in the deterministic system as the parameter b crosses 0. With noise present, the bifurcation point is known to occur for $b < 0$. Fig. 7 illustrates the behaviour of (3) with $a = -1$ and $\sigma = 0.1$ for values of $b = -0.1, -0.01, 0, 0.1$. We see the trajectories are focused on a circle for $b \geqslant -0.01$, which shows that the bifurcation occurs for negative b.

8. Further directions

There are a number of ways we would like to develop SDELab.

(1) SDELab does not provide special algorithms for computing averages of $\phi(Y(t))$ for a test function ϕ. This is an important problem and will be dealt with in future releases of SDELab.

(2) One of the key features of the MATLAB ODE suite is its use of error estimation to select time step sizes. The theory of error estimation for SDE integrators is not well developed (see [10,3] for recent work) and we are unaware of any technique robust enough for inclusion in a software package for (1) or (2). We hope the algorithms will mature and eventually be included in SDELab.
(3) It is frequently of interest to determine times at which certain events happen, such as $Y(t)$ crossing a barrier. At this time, no such algorithms are included in SDELab.
(4) Some classes of SDEs deserve special attention, such as Langevin equations, geometric Brownian motion, and the Ornstein–Uhlenbeck process, and we would like to address this within SDELab.

9. Final remarks

For some complementary reading, we refer to [4,6]. We thank the referees and the guest editors for their interest in our work and their helpful observations.

Appendix A. Van der Pol Duffing

```
function z = drift(t, y, varargin)
a = varargin{2}.a; % Extract parameters
b = varargin{2}.b;
A = varargin{2}.A;
B = varargin{2}.B;
 z = [y(2); ...
 (b-B*y(1)*y(1))*y(2)+ (a-A*y(1)*y(1))*y(1) ];

function z = drift_dy(t, y, varargin)
a = varargin{2}.a; % Extract parameters
b = varargin{2}.b;
A = varargin{2}.A;
B = varargin{2}.B;
 z = [         0              1; ...
 a-(3*A*y(1)+2*B*y(2))*y(1) b-B*y(1)*y(1)];

function z = diff_noise(t, y, dw, flag, varargin)
sigma = varargin{2}.sigma;
if (flag)
 z = [0; sigma*y(1)];        % Return g(t,y)
else
 z = [0; sigma*y(1)*dw];     % Compute g(t,y) * dw
end;

function z = diff_noise_dy(t, y, dw, j, varargin)
sigma = varargin{2}.sigma;
z = [0; sigma * dw(1)];
```

References

[1] L. Arnold, Random Dynamical Systems, Springer, Berlin, 1998.
[2] E. Buckwar, R. Winkler, Multi-step methods for SDEs and their applications to problems with small noise, Technical Report, Humboldt-Universität zu Berlin, 2004.
[3] P.M. Burrage, R. Herdiana, K. Burrage, Adaptive stepsize based on control theory for stochastic differential equations, J. Comput. Appl. Math. 170 (2) (2004) 317–336.
[4] S. Cyganowski, L. Grüne, P.E. Kloeden, Maple for stochastic differential equations, in: J.F. Blowey, J.P. Coleman, A.W. Craig (Eds.), Theory and Numerics of Differential Equations, Springer, Berlin, 2001, pp. 127–178.
[5] J. Gaines, T. Lyons, Random generation of stochastic area integrals, SIAM J. Appl. Math. 54 (4) (1994) 1132–1146.

[6] D.J. Higham, An algorithmic introduction to numerical simulation of stochastic differential equations, SIAM Rev. Edu. Sec. 43 (2001) 525–546.
[7] D.J. Higham, X. Mao, A.M. Stuart, Strong convergence of Euler-type methods for nonlinear stochastic differential equations, SIAM J. Numer. Anal. 40 (3) (2002) 1041–1063.
[8] P.E. Kloeden, E. Platen, Numerical solution of stochastic differential equations, Applications of Mathematics, vol. 23, Springer, Berlin, 1992.
[9] P.E. Kloeden, E. Platen, I.W. Wright, The approximation of multiple stochastic integrals, Stochastic Anal. Appl. 10 (4) (1992) 431–441.
[10] H. Lamba, An adaptive timestepping algorithm for stochastic differential equations, J. Comput. Appl. Math. 161 (2) (2003) 417–430.
[11] G. Marsaglia, W.W. Tsang, The Ziggurat method for generating random variables, J. Statist. Software 5 (8) (2000) 7 ⟨http://www.jstatsoft.org/v05/i08/ziggurat.pdf⟩.
[12] G. Maruyama, Continuous Markov processes and stochastic equations, Rend. Circ. Mat. Palermo 4 (2) (1955) 48–90.
[13] G.N. Milstein, Approximate integration of stochastic differential equations, Theory Probab. Appl. 19 (1974) 557–562.
[14] G.N. Milstein, M.V. Tretyakov, Mean-square numerical methods for stochastic differential equations with small noises, SIAM J. Sci. Comput. 18 (4) (1997) 1067–1087.
[15] G.N. Milstein, M.V. Tretyakov, Stochastic numerics for mathematical physics, Scientific Computation, Springer, Berlin, 2004.
[16] C. Moler, Normal behavior: Ziggurat algorithm generates normally distributed random numbers, Technical Report, MATLAB News & Notes, Spring 2001.
[17] J.J. Moré, B.S. Garbow, K.E. Hillstrom, User guide for MINPACK-1, Technical Report ANL-80-74, Argonne National Laboratory, Argonne, IL, USA, 1980.
[18] A. Neumaier, Molecular modeling of proteins and mathematical prediction of protein structure, SIAM Rev. 39 (3) (1997) 407–460.
[19] B. Øksendal, Stochastic Differential Equations, sixth ed., Universitext, Springer, Berlin, 2003.
[20] M.J.D. Powell, A FORTRAN subroutine for solving systems of nonlinear algebraic equations, in: Numerical Methods for Nonlinear Algebraic Equations (Proceedings of the Conference, University of Essex, Colchester, 1969), Gordon and Breach, London, 1970, pp. 115–161.
[21] T. Rydén, M. Wiktorsson, On the simulation of iterated Itô integrals, Stochastic Process. Appl. 91 (1) (2001) 151–168.
[22] L. Shampine, M. Reichelt, The MATLAB ODE suite, SIAM J. Sci. Comput. 18 (1997) 1–22.
[23] M. Wiktorsson, Joint characteristic function and simultaneous simulation of iterated Itô integrals for multiple independent Brownian motions, Ann. Appl. Probab. 11 (2) (2001) 470–487.