Performing Parallel Monte Carlo and Moment Equations Methods for Itô and Stratonovich Stochastic Differential Systems: R Package **Sim.DiffProc**

Arsalane Chouaib Guidoum

University of Science and Technology
Houari Boumediene

Kamal Boukhetala

University of Science and Technology
Houari Boumediene

Abstract

We introduce **Sim.DiffProc**, an R package for symbolic and numerical computations on scalar and multivariate systems of stochastic differential equations (SDEs). It provides users with a wide range of tools to simulate, estimate, analyze, and visualize the dynamics of these systems in both forms, Itô and Stratonovich. One of **Sim.DiffProc** key features is to implement the Monte Carlo method for the iterative evaluation and approximation of an interesting quantity at a fixed time on SDEs with parallel computing, on multiple processors on a single machine or a cluster of computers, which is an important tool to improve capacity and speed-up calculations. We also provide an easy-to-use interface for symbolic calculation and numerical approximation of the first and central second-order moments of SDEs (i.e., mean, variance and covariance), by solving a system of ordinary differential equations, which yields insights into the dynamics of stochastic systems. The final result object of Monte Carlo and moment equations can be derived and presented in terms of $\text{\LaTeX}$ math expressions and visualized in terms of $\text{\LaTeX}$ tables.

Furthermore, we illustrate various features of the package by proposing a general bivariate nonlinear dynamic system of Haken-Zwanzig, driven by additive, linear and nonlinear multiplicative noises. In addition, we consider the particular case of a scalar SDE driven by three independent Wiener processes. The Monte Carlo simulation thereof is obtained through a transformation to a system of three equations. We also study some important applications of SDEs in different fields.

Keywords: stochastic differential equation, stochastic calculus, moment equations, Monte Carlo simulation, parallel computing, statistical sampling, statistical analysis, nonparametric, transition density, reproducible research, $\text{\LaTeX}$, R.

1. Introduction and motivation

Differential equations are used to describe the evolution of a system. In most applications where such ordinary differential equations (ODEs) intervene, the trajectories measured experimentally are only rarely in conformity with the analytical solution of the equation. Random effects are superimposed on the ideal trajectory, therefore it seems reasonable to disrupt these equations. The purpose of stochastic differential equations (SDEs) is to provide a mathematical model, for describing the process by which the perturbations of a considered phenomenon are diffused until an equilibrium state is established, i.e., until the differences that drive the process are minimized. SDEs are of fundamental importance in many disciplines: see, e.g., [Weberruß \(1993\)](#), [Klebaner \(2005, p. 375\)](#), [Allen \(2007, p. 156\)](#), [Jedzejewski \(2009\)](#) and [Freidlin and Wentzell \(2012\)](#) for applications in physics. In quantitative finance and financial mathematics, we have [Øksendal \(2003, p. 261\)](#), [Klebaner \(2005, p. 287\)](#), [Racicot and Théoret \(2006\)](#), [Allen \(2007, p. 174\)](#), [Rolski, Schmidli, Schmidt, and Teugels \(2008\)](#), [Platen and Bruti-Liberati \(2010\)](#), [Iacus \(2011\)](#), [Gushchin \(2015\)](#) and [Mackevicius \(2016\)](#). In biology and life sciences, we mention [Hadel, de Mottoni, and Schumacher \(1980\)](#), [Helland \(1983\)](#), [Klebaner \(2005, p. 351\)](#), [Allen \(2007, p. 145\)](#), [Fuchs \(2013a\)](#), [Fuchs \(2013b\)](#), [Han and Kloeden \(2017\)](#) and [Panik \(2017\)](#), and more specifically in genetics, see, e.g., [Lange \(2002\)](#). For applications in science and engineering, see [Soong \(1973\)](#), [Sobczyk \(1991\)](#), [Kloeden and Platen \(1999, p. 253\)](#), [Primak, Kontorovich, and Lyandres \(2004\)](#) and [Henderson and Plaschko \(2006\)](#). Readers wishing to go further into the theory of stochastic calculus should consult the classical works of [Gikhman and Skorokhod \(1972\)](#), [Karatzas and Shreve \(1988\)](#), [Rogers and Williams \(1994, 2000\)](#) and [Itô and McKean, Jr. \(1996\)](#). For a historical perspective on the development of the theory, see the original articles of [Itô \(1944, 1946, 1987\)](#), [Stratonovich \(1966\)](#) and also the little book of [McKean, Jr. \(1969\)](#) which contributed to popularize these works.

Before we give a rigorous definition of SDEs, we show how they arise as randomly perturbed ODEs and give a physical interpretation. If X_t is a differentiable function defined for $t \geq 0$, $f(t, x)$ is a function of time t and state x , the following relation is satisfied for all t ($0 \leq t \leq T$):

$$\frac{dX_t}{dt} = X'_t = f(t, X_t), \quad \text{with } X_0 = x_0. \quad (1)$$

Then X_t is a solution of the ODE with the initial condition $X_0 = x_0$. The above equation can be written in other forms (by continuity of X'_t):

$$X_t = X_0 + \int_0^t f(s, X_s) ds.$$

Although the Wiener process (Brownian motion) is not differentiable with respect to time, we write the White noise ξ_t as its fictitious derivative of the Wiener process W_t , i.e., $\xi_t = dW_t/dt = W'_t$, just with the purpose to mimic standard ODE calculus. This will allow us to introduce the Itô stochastic integral in the following form: if $g(t, x)$ is the intensity of the White noise, then we define the [Itô \(1944\)](#) integral as:

$$\int_0^T g(t, X_t) \xi_t dt = \int_0^T g(t, X_t) W'_t dt = \int_0^T g(t, X_t) dW_t,$$

where the middle term $\int_0^T g(t, X_t) W'_t dt$ has no mathematical meaning. Indeed, SDEs are just a symbolic writing for the mathematically correct stochastic integral representation.

SDEs arise when the coefficients of (1) are perturbed by White noise. For example, if X_t denotes the population density, then the population growth can be described by the ODE:

$$\frac{dX_t}{dt} = aX_t(1 - X_t),$$

the growth is exponential with birth rate a . Random perturbation of the birth rate results in the equation:

$$\frac{dX_t}{dt} = (a + \sigma\xi_t)X_t(1 - X_t)$$

or the SDE:

$$dX_t = aX_t(1 - X_t)dt + \sigma X_t(1 - X_t)dW_t, \quad X_0 = x_0.$$

There are two widely used types of stochastic calculus: Itô (1944, 1946) and Stratonovich (1966), differing with respect to the stochastic integral used. The Itô integral takes the left end point of each subinterval as the evaluation point, whereas the Stratonovich integral uses the midpoint of each subinterval. Under certain conditions, the Itô integral has the most convenient property of being a martingale (see Gikhman and Skorokhod 2007), but has a transformation rule, called the Itô's formula, which is more complicated than the chain rule of deterministic calculus. In contrast, the Stratonovich integral meets the transformation rules of classical calculus (see Kloeden and Platen 1991a,b). Practical considerations typically dictate which version is appropriate. The following classical example demonstrates how the Itô integral differs from Stratonovich¹ integral:

$$\begin{aligned} \text{Itô stochastic calculus:} \quad & \int_{t_0}^t W_s \, dW_s = \frac{1}{2} (W_t^2 - W_{t_0}^2) - \underbrace{\frac{1}{2}(t - t_0)}_{\text{additional term}} \\ \text{Stratonovich stochastic calculus:} \quad & \int_{t_0}^t W_s \circ dW_s = \frac{1}{2} (W_t^2 - W_{t_0}^2) \end{aligned}$$

Let $\mathbf{X}(t) = \{X_1(t), X_2(t), \dots, X_d(t)\}$, with $d \geq 1$ be an d -dimensional random process with components satisfying the Itô SDEs:

$$dX_j(t) = f_j(t, \mathbf{X}(t))dt + \sum_{k=1}^m g_{jk}(t, \mathbf{X}(t))dW_{jk}(t). \quad (2)$$

$W_{jk}(t)$ is a standard Wiener process (Gaussian random variables with means zero, initial value zero with probability one and variance equal to t at time t), which we assume to be independent, where $j = 1, 2, \dots, d$ and $k = 1, 2, \dots, m$. The drift $f_j(t, x)$ and diffusion $g_{jk}(t, x)$ are known functions which are assumed to be sufficiently regular (Lipschitz, bounded growth) for the existence and uniqueness of the solution (see, e.g., Øksendal 2003). The same solution process $\mathbf{X}(t)$ can be obtained from the Stratonovich SDEs denoted by:

$$dX_j(t) = \underline{f}_j(t, \mathbf{X}(t))dt + \sum_{k=1}^m g_{jk}(t, \mathbf{X}(t)) \circ dW_{jk}(t). \quad (3)$$

¹To distinguish Stratonovich integral from the Itô integral we insert a symbol $\circ$ before the differential dW_s in Stratonovich (1966) form.

It is possible to switch between the two stochastic approaches by using a modified j th drift coefficient:

$$\underline{f}_j(t, \mathbf{X}(t)) = f_j(t, \mathbf{X}(t)) - \frac{1}{2} \sum_{i=1}^d \sum_{k=1}^m \left(g_{ik}(t, \mathbf{X}(t)) \frac{\partial}{\partial X_i} g_{jk}(t, \mathbf{X}(t)) \right). \quad (4)$$

(4) is called the drift correction formula. Note that the diffusion coefficients are the same in both representations. Such a Stratonovich SDE (3) is equivalent to an Itô SDE:

$$dX_j(t) = \left(\underline{f}_j(t, \mathbf{X}(t)) + \frac{1}{2} \sum_{i=1}^d \sum_{k=1}^m g_{ik}(t, \mathbf{X}(t)) \frac{\partial}{\partial X_i} g_{jk}(t, \mathbf{X}(t)) \right) dt + \sum_{k=1}^m g_{jk}(t, \mathbf{X}(t)) dW_{jk}(t). \quad (5)$$

Thus, we can write the Stratonovich SDE in the form of an Itô SDE. Whichever interpretation of an SDE is deemed appropriate in a particular situation, we can always switch to the corresponding SDE in the other interpretation when this is advantageous. In other words, (2) and (3) have the same solution. Notice that if the noise is additive (g independent of x), then the two representations are equivalent.

The current development of computing tools (software and hardware) has allowed solving many theoretical problems on SDEs that have become the object of practical research. This has also enabled many researchers in different domains to use these equations for modeling and analyzing practical problems. In response to these many uses, there already exist some open source packages that can perform stochastic calculus in the R software environment (R Core Team 2020); see the freely available packages **sde** (Iacus 2008, 2016) and **yuima** (Brouste *et al.* 2014; Iacus and Yoshida 2018). They provide functions for simulation and inference for multidimensional SDEs, driven by Wiener process, fractional Brownian motion and Lévy processes, with or without jumps specified, as well as continuous time autoregressive moving average (CARMA), continuous time generalized autoregressive conditional heteroskedasticity (COGARCH) and point processes. The packages **DiffusionRgqd**, **DiffusionRimp** and **DiffusionRjgqd** (Pienaar and Varughese 2016a,b,c) constitute a handy collection of tools for performing inference and analysis using a data-imputation scheme and the method of lines on scalar and bivariate time-inhomogeneous diffusion and jump-diffusion processes with quadratic drift and diffusion. The **QPot** package (Moore, Stieha, Nolting, Cameron, and Abbott 2016) covers advanced techniques for quasi-potential analysis and visualizes the dynamics of two-dimensional systems of SDEs. The **diffeqr** package (Rackauckas and Nie 2017) is useful to solve SDE problems using the **DifferentialEquations.jl** package from the Julia programming language (Bezanson, Edelman, Karpinski, and Shah 2017).

In general, simulation allows to study and experiment with a given system where the complex interactions are known, to measure the effects of certain changes in the interactions on the behavior of the system, to experiment with new situations. It should be noted that if one looks for a faithful representation of the phenomena observed, one is quickly confronted with difficulties due to the calculations not being available in an explicit way. Monte Carlo (MC) techniques will allow us to approach these calculations numerically. There exists a well-developed literature on general Monte Carlo methods, we might mention Fishman (1996); Glasserman (2004); Liu (2004); Thomopoulos (2013); Graham and Talay (2013), and for computer vision with the R language see, e.g., Robert and Casella (2010). Methodological advances have led to much more computational demand in statistical computing, such as Markov chain Monte Carlo (MCMC) algorithms, bootstrapping, cross-validation, Monte

Carlo simulation, etc. Many areas of statistical applications are experiencing rapid growth in the size of data sets (see, e.g., [Mishra, Dehuri, Kim, and Wang 2016](#)). Parallel computing is a common approach to resolve these problems. This is available in R since 2011 (version 2.14.0) through the base package **parallel**. This provides parallel facilities previously contained in packages **multicore** for parallel computation on shared memory (unix) platforms (multiprocessor and multi-core computers, [Urbanek 2014](#)) and **snow** for parallel computation on distributed memory platforms, with slight revisions (simple network of workstations, [Tierney, Rossini, Li, and Sevcikova 2018](#)). Several approaches to parallel computing are available in R, see [Schmidberger, Morgan, Eddebuettel, Yu, Tierney, and Mansmann \(2009\)](#); [Eugster, Knaus, Porzelius, Schmidberger, and Vicedo \(2011\)](#); [McCallum and Weston \(2011\)](#), and an extensive and updated list of R packages is reported in the Comprehensive R Archive Network (CRAN) Task View on *High-Performance and Parallel Computing with R* at <https://CRAN.R-project.org/view=HighPerformanceComputing> ([Eddebuettel 2020](#)).

The determination of response moments in SDEs is one of the fundamental problems in stochastic analysis and is much more complicated in the case of nonlinear equations. The main reason is the fact that the widest class of nonlinear SDEs subjected to additive or multiplicative random excitation do not have exact solutions in analytical forms. In many cases, therefore, it is necessary to adopt approximate solutions. In the moment equations method (MEM) approach (see, e.g., [Bover 1978](#); [Socha 2008](#); [Alibrandi and Ricciardi 2012](#)), the response statistical characterization is given by the moments. This method consists in writing a system of ODEs for the response statistical moments of any order, taking advantage of the Fokker-Planck equations or of the Itô differential rule. The MEM can be easily solved for linear systems. Unfortunately, for nonlinear systems in general, these equations constitute an infinite hierarchy. In fact the moment differential equations up to a given order contain moments of higher order and, consequently, this system is not solvable.

This has motivated us to take an interest in this particular field by introducing the R package **Sim.DiffProc** (simulation of diffusion processes, [Guidoum and Boukhetala 2020](#)). For the R package developed in the present paper we focus on two problems: Firstly, we develop a module for performing Monte Carlo methods whose essence is the use of repeated experiments to iteratively evaluate and approximate a quantity of interest (replicates of a statistic(s) to SDEs) at a fixed time t for scalar and multivariate Itô and Stratonovich SDEs, with the techniques for parallel computing on computer clusters, and multi-core systems using the **parallel** package. Secondly, we develop a module for symbolic and numerical computations by analytical determination of the dynamics of first and central second-order moments of an SDE (i.e., mean, variance and covariance of the dynamical variables). We also provide the built-in functions for random number generation based on the simulated trajectories at a fixed time t , and the evaluation of transition densities for SDEs in both interpretations, with nonparametric techniques. In addition, the simulation and estimation outputs can be graphically represented in 2D and 3D plots (like [Figures 10](#)). Our package is structured according to the S3 object-oriented system of classes and methods. **Sim.DiffProc** is currently the only package on CRAN, available at <http://CRAN.R-project.org/package=Sim.DiffProc>, that uses parallel processing on multiple processors and a cluster of computers in stochastic computing.

The paper is organized as follows: [Section 2](#) we explain the implementation and describe the core functions of the package, and present the usage and options of the parallel Monte Carlo method, also we briefly review some theoretical concepts concerning the moment equa-

tions method for an analytical approximation of first and central second-order moments. In Section 3 we give illustrations with simulations in different domains using scalar and multivariate Itô and Stratonovich SDEs. In Section 4 we give an overview of using the package to approximate the European options pricing for Black-Scholes (1D) and Heston models (2D) with an application using real data. The presentation of the results of a simulation study and moment equations with $\text{\LaTeX}$ tables/math expressions is demonstrated in Section 5. Concluding remarks discussing the package and its limitations, as well as future goals, are provided in Section 6.

2. Sim.DiffProc, an open source R package

2.1. Obtaining, installation and documentation

As **Sim.DiffProc** is an R package, it requires R version 3.0.0 or higher to be installed. It is distributed as open source software under the GPL-2/GPL-3 license. The package is available from CRAN at <https://CRAN.R-project.org/package=Sim.DiffProc> (stable version), or from GitHub at <https://github.com/acguidoum/Sim.DiffProc> (development version). To download, install and load the current release, just type the code below in your current R session:

```
R> install.packages("Sim.DiffProc")
R> library("Sim.DiffProc")
```

or,

```
R> install.packages("devtools")
R> devtools::install_github("acguidoum/Sim.DiffProc")
R> library("Sim.DiffProc")
```

In addition to the help pages required by the R packaging system, the package includes vignettes and demonstration scripts. These can be obtained using:

```
R> vignette(package = "Sim.DiffProc")
```

and

```
R> demo(package = "Sim.DiffProc")
```

at the R prompt.

2.2. Outline of the package structure

Sim.DiffProc consists of a set of functions that allow the user to perform simulation and stochastic analysis on scalar and multivariate Itô and Stratonovich SDEs. In this package, the user can specify:

- The type of SDEs to be simulated: Itô (2) and Stratonovich (3).

- The R expression of drift and diffusion coefficients.
- The correlation structure between the Wiener processes (for any 2D and 3D).
- The number of trajectories to be simulated.
- The initial value of state variables.
- The time interval $[t_0, T]$ to be considered.
- Integration step size (mesh size) Δ to be used for the numerical approximation of the SDEs solution. We assume that step size satisfies the following assumptions:
 1. *Large sample asymptotic*: Δ fixed, $T = n\Delta \rightarrow \infty$ as $n \rightarrow \infty$.
 2. *High frequency*: $T = n\Delta$ fixed, $\Delta \rightarrow 0$ as $n \rightarrow \infty$.
 3. *Rapidly increasing design*: $T = n\Delta \rightarrow \infty$, $\Delta \rightarrow 0$ as $n \rightarrow \infty$ under the additional condition $n\Delta^k \rightarrow 0$ for $k > 1$.

Assumptions 1 and 3 require stationarity and/or ergodicity, and for assumption 2 neither stationarity nor ergodicity is necessary.

- The numerical integration methods: Euler-Maruyama (Maruyama 1955), predictor-corrector (Platen 1995), Milstein and second Milstein schemes (Milstein 1975, 1979), Taylor scheme of strong order 1.5 (Platen 1982; Kloeden and Platen 1989, 1991b, 1999), Heun order 2.0 (McShane 1974; Saito and Mitsui 1993); Runge-Kutta 1-, 2- and 3-stage (Kasdin 1995; Rössler 2003, 2004, 2007, 2010). For a numerical comparison between these methods see Platen (1999); Kloeden and Platen (1999); Kloeden, Platen, and Schurz (2003); Burrage, Burrage, Higham, Kloeden, and Platen (2006). These numerical methods are usually based on discrete approximations of the continuous solution to an SDE. To assess the quality of the approximation, we define the order of convergence.

Let X^δ be a time-discrete approximation of a continuous-time process X , with δ the maximum time increment of the step size.

1. An approximation method is *strong order* γ of convergence, if at fixed time T we have:

$$\mathbb{E}|X_T - X_T^\delta| \leq C\delta^\gamma, \quad \forall \delta < \delta_0.$$

2. An approximation method is said to converge *weakly of order* β , if for any fixed time T and any $2(\beta + 1)$ continuous differentiable function g of polynomial growth, we have:

$$|\mathbb{E}g(X_T) - \mathbb{E}g(X_T^\delta)| \leq C\delta^\beta, \quad \forall \delta < \delta_0.$$

In both cases $\delta_0 > 0$ and C do not depend on δ .

Notice that all numerical methods implemented in this package are of strong order γ of convergence.

To obtain:

- A formal display of SDEs, with all inputs.

Function	Main arguments	Description
<code>snssdekd()</code>	<code>drift</code> and <code>diffusion</code> coefficients as R expression, <code>corr</code> correlation structure, <code>type</code> , <code>method</code> , <code>M</code> number of trajectories, <code>N</code> number of simulation steps, <code>x0</code> , <code>t0</code> , <code>T</code> and <code>Dt</code> .	The (S3) generic function of simulation of numerical solutions to k -dimensional Itô and Stratonovich SDEs, with different stochastic numerical schemes.
<code>rsdekd()</code>	' <code>snssdekd</code> ' output, at any time between <code>t0</code> and <code>T</code> .	Random generation for M realizations of ' <code>snssdekd</code> ' at any fixed time.
<code>dsdekd()</code>	' <code>snssdekd</code> ' output, at any time between <code>t0</code> and <code>T</code> , <code>pdf</code> for joint or marginal density.	Kernel approximation of the transitional density function (uni-, bi- and tri-variate) of ' <code>snssdekd</code> '.
<code>MCM.sde()</code>	' <code>snssdekd</code> ' output, <code>statistic</code> , <code>R</code> number of MC replicates, <code>time</code> (fixed), <code>parallel</code> , <code>ncpus</code> , <code>cl</code> .	Parallel Monte Carlo method: simulation and approximation of a quantity of interest of SDEs at fixed time.
<code>MEM.sde()</code>	<code>drift</code> , <code>diffusion</code> and <code>corr</code> coefficients as R expression, <code>type</code> , <code>solve</code> , <code>time</code> (vector).	Symbolic and numerical computations of moment equations of SDEs (means, variances and covariance).
<code>TEX.sde()</code>	' <code>MCM.sde</code> ' and ' <code>MEM.sde</code> ' outputs.	Creating $\text{\LaTeX}$ tables and mathematical expressions (reproducible research).

Table 1: Brief description of important functions in package **Sim.DiffProc**.

- Numerical solutions of SDEs.
- Summary descriptive statistics for the solution process based on the simulated M -trajectories at any fixed time $t \in [t_0, T]$, i.e., mean, variance, median, mode, quantiles, minimum, maximum, coefficient of variation (relative variability), skewness, kurtosis, and the k th central moment.
- Statistical sampling based on the simulation of M -trajectories for SDEs at any fixed time $t \in [t_0, T]$, i.e., random generation of $\{X_t^\Delta(\omega_1), X_t^\Delta(\omega_2), \dots, X_t^\Delta(\omega_M)\}$.
- Kernel approximation of the transitional density (uni-, bi- and tri-variate) of SDEs, based on the simulation of M -trajectories at any fixed time $t \in [t_0, T]$.
- Monte Carlo simulation-approximation of any interesting quantity at any fixed time $t \in [t_0, T]$, with parallel computing.
- Symbolic and numerical computations by an analytical approximation of the first and central second-order moments (means, variances and covariances) of SDEs.
- Plot(s) routines for various classes of objects in the **Sim.DiffProc** package.

In Table 1 we give a summary of the key functions introduced in this package (where $k \in \{1, 2, 3\}$).

2.3. Parallel implementation of Monte Carlo method: ‘MCM.sde’ details

Monte Carlo methods are based on the analogy between probability and volume. They involve massive computation due to the replications in simulations (some problems are: computing column means, bootstrapping, etc.). These structures allow one to develop highly sophisticated Monte Carlo simulation based on the strong law of large numbers. In many cases, these perform extremely well in circumstances where other methods fail. Often, the purpose of the Monte Carlo simulation is to approximate an expectation:

$$\mu_t = \mathbb{E}(h(X_t)), \quad (6)$$

where X_t is the solution of the SDE (2) (or (3)) at fixed time t , for some given function h by the expectation:

$$\mu_{t,\Delta} = \mathbb{E}\left(h\left(X_t^\Delta\right)\right), \quad (7)$$

where X_t^Δ is the corresponding iterate of the numerical scheme with a constant step size Δ . In practice, (7) can only be approximated by the arithmetic mean:

$$\hat{\mu}_{t,\Delta,M} := \frac{1}{M} \sum_{j=1}^M h\left(X_t^\Delta(\omega_j)\right) \quad (8)$$

with M independent simulated realizations $\{X_t^\Delta(\omega_1), X_t^\Delta(\omega_2), \dots, X_t^\Delta(\omega_M)\}$ of a discrete time approximation X_t^Δ at fixed time t .

The approximation (8) depends on which M realizations of the numerical scheme are used to calculate it. To obtain a statistically reliable result, in the following, we apply the form with R batches with M realizations each. Then their mean is determined, which is the total sample mean of realizations $\{X_t^\Delta(\omega_{1,1}), \dots, X_t^\Delta(\omega_{i,j}), \dots, X_t^\Delta(\omega_{R,M})\}$:

$$\hat{\mu}_{t,\Delta,M,R} := \frac{1}{MR} \sum_{i=1}^R \sum_{j=1}^M h\left(X_t^\Delta(\omega_{i,j})\right). \quad (9)$$

The mean (9) is itself only an approximation of the desired value (6), since $\hat{\mu}_{t,\Delta,M,R}$ converges almost surely (i.e., for almost every generated sequence) to $\mathbb{E}(h(X_t))$ by the strong law of large numbers whenever $\mathbb{E}|h(X_t)| < \infty$. The confidence interval about (9) is then given by:

$$\hat{\mu}_{t,\Delta,M,R} \pm \sqrt{\frac{\hat{\sigma}_R^2}{R}} \text{St}_{R-1}(1 - \alpha/2), \quad (10)$$

where $\hat{\sigma}_R^2$ is the unbiased sample variance of the batch means, and $\text{St}_{R-1}(1 - \alpha/2)$ is the inverse of Student’s *St* cumulative distribution function with $R - 1$ degrees of freedom and confidence parameter $\alpha \in]0, 1[$, if R is small. For large R , we can use the normal distribution instead of the *St*-distribution (due to the central limit theorem). The accuracy of the simulation is therefore inversely proportional to the square root of the number of trials. This means that to double the accuracy the number of trials has to be quadrupled. It is clear that how good the approximation of (6) is will depend on the accuracy of the stochastic numerical scheme and step size Δ . A study of Komori, Saito, and Mitsui (1994) indicates that as Δ decreases, the lack of independence in the samples from a random number generator (RNG)

typically degrades the computation before rounding errors become significant. We will discuss this important feature in the following.

Generally, the Monte Carlo method on SDEs is not easy to implement, particularly in a multidimensional situation. Furthermore, the computational times are very significant. Hence computational performance can be increased by the use of parallel computing. Parallel computing is the simultaneous execution of the source code of one or more programs, to improve capacity and speed up the computations (see, e.g., Schmidberger and Mansmann 2010; Lim and Tjhi 2015). R offers a unified way of doing parallelization with the base package **parallel**, it can be easily run in parallel on several cores. There are several different approaches available to achieve parallelization and not all approaches are available for all platforms (see McCallum and Weston 2011). We will use two main methods to parallelize the Monte Carlo method: implicit parallelism (multiple cores on a single machine, i.e., **multicore**) with function `mclapply()`, and explicit parallelism (cluster with a multiple machine, i.e., **snow**) with function `parLapply()`. The conceptual differences between these two methods are illustrated in the following paragraphs.

The first method uses the *forking*² techniques (Urbanek 2014) on machines running with POSIX operating systems with multiple cores, and would simply call `lapply()` on Microsoft Windows operating systems, so that the code works but sees no speed gain. In contrast, on MacOS and Unix operating systems, the most straightforward way to enable parallel processing is by switching from using `lapply()` to `mclapply()`. This method will use all cores available to it. If you do not want to (either because you are on a shared system or you just want to save processing power for other purposes) you can set this to a value lower than the number of cores you have. Setting it to 1 disables parallel processing, and setting it higher than the number of available cores has no effect. We can easily check the number of cores we have access to with the command `detectCores()` available in package **parallel**.

The second method to parallel processing is more complicated, but works on any system including Windows systems. The computations in parallel can be extended to all cores on a single computer or to networked computers (see Appendix E), it uses the PVM, MPI, NWS standards as well as direct SOCKETS communication technologies for distributed memory computing, for more details on this technique see the original articles of Rossini, Tierney, and Li (2007) and Tierney, Rossini, and Li (2009). The general process we will follow is:

- i) Start a cluster with n nodes.
- ii) Execute any preprocessing code necessary in each node (e.g., loading a package).
- iii) Use `parLapply()` as a replacement for `lapply()`.
- iv) Destroy the cluster.

Results based on Monte Carlo simulations are not reproducible unless we set the random seed in their R code using function `set.seed()`. The **parallel** package contains support for multiple RNG streams based on the algorithms discussed in L'Ecuyer (1999) and L'Ecuyer, Simard, Chen, and Kelton (2002), with support for both **multicore** and **snow** clusters, and is the only RNG in R that supports multiple streams to generate random numbers in parallel environments. In addition there are also several packages on CRAN, see, e.g., **rlecuyer** (Sevcikova,

²The name *fork* is derived from the POSIX standard for operating systems. In this context, a *fork* is a process that generates a copy of itself. This works on any POSIX systems and not Windows.

Rossini, and L'Ecuyer 2019), **rstream** (Leydold 2020) and **doRNG** (Gaujoux 2020). In R we selected the RNG by calling `RNGkind("L'Ecuyer-CMRG")` and the `clusterSetRNGStream()` function (see `help("RNGkind")` and `help("clusterSetRNGStream")`). The idea of L'Ecuyer-CMRG RNG is that each separate worker generates random numbers independently from the others, and its period is around 2^{191} , that will provide one stream of random numbers for about 264 workers. The seed of this generator can be easily advanced a given number of steps, making it very useful for true parallel RNG support.

Roughly, the best and easiest way to analyze the performance of an application is to measure the execution time. Consequently, an application can be compared with an improved version through the execution times:

$$\text{Speed-up} \approx \frac{\text{execution time for a program without enhancements (sequential)}}{\text{execution time for a program using the enhancements (parallel)}}.$$

To summarize, the Monte Carlo simulation-approximation in this paper consists of:

- 1) Exhibiting a probabilistic representation of μ_t (any interesting quantity at a fixed time t) of the type (6) such that the numerical solution of the stochastic system (2) is efficiently simulated.
- 2) Applying the strong law of large numbers in order to approximate (6).
- 3) To get the system status at fixed time T (i.e., the state of SDEs at this set time T); we automatically use linear interpolation with the R function `approxfun()` available in base package **stats** (i.e., the state of the SDEs is linearly interpolated if $T \neq n\Delta$).

Remark 1. *The whole idea might work in general when the simulation step size Δ is very small (high frequency “assumption 2”). To use the numerical methods mentioned previously (Milstein scheme 1 and 2, Taylor scheme of strong order 1.5) the assumptions on the step size Δ of simulation can be relaxed, but more regularity on the coefficients of drift and diffusion is needed (i.e., existence of partial derivatives up to order 2).*

Parallel computing with **multicore** (shared memory) can be summarized into the following steps:

- 1) Detect the number of CPUs cores (`ncpus`) using `detectCores()`.
- 2) Set parallel RNG using `RNGkind("L'Ecuyer-CMRG")`.
- 3) Parallel computation of (9) with `mclapply()`, and leave `mc.set.seed` to `TRUE` (the `mclapply()` function can set the random number seeds for each worker for us, when the `mc.set.seed` argument is set to `TRUE`; we do not need to call `clusterSetRNGStream()`).

For parallel computing with **snow** (distributed memory) the steps are:

- 1) Start the cluster (`cl`); we use the function `makePSOCKcluster()`.
- 2) Send a list of objects to all workers using the function `clusterExport()`.
- 3) Load packages on all workers using the function `clusterEvalQ()`.

- 4) Set parallel RNG on the cluster using the function `clusterSetRNGStream()`.
- 5) Parallel computation of (9) with `parLapply()`.
- 6) Stop the cluster with `stopCluster()`.

The R function call consists of (see usage in the package help files):

```
MCM.sde(model, statistic, R = 100, time, exact = NULL, names = NULL,
        level = 0.95, parallel = c("no", "multicore", "snow"),
        ncpus = getOption("ncpus", 1L), cl = NULL, ...)
```

The input arguments for `MCM.sde()` are:

model: an object of class ‘`snssde1d`’, ‘`snssde2d`’ and ‘`snssde3d`’.

statistic: a function that when applied to the **model** (SDEs) returns a vector containing the statistic(s) of interest.

R: number of Monte Carlo replicates (R batches > 1), this will be a single positive integer.

time: fixed time at which the quantity of interest **statistic** is approximated.

exact: a named list giving the exact statistic(s), if it exists the bias calculation will be performed.

names: names for the statistic(s) of interest. Default **names** = `c("mu1", "mu2", ...)`.

level: confidence level of the required interval(s).

parallel: the type of parallel operation to be used. "multicore" does not work on Microsoft Windows operating systems, but on Unix it uses parallel operations. Default **parallel** = "no".

ncpus: an integer value specifying the number of cores to be used in the parallelized procedure. Default is 1 core of the machine.

cl: an optional parallel cluster for use if **parallel** = "snow". Default **cl** = `makePSOCKcluster(rep("localhost", ncpus))`.

...: other named arguments for **statistic** which are passed unchanged each time it is called through the **...** argument.

```
plot(x, index = 1, type = c("all", "hist", "qqplot", "boxplot", "CI"), ...)
```

This takes a ‘`MCM.sde`’ S3 object and produces plot(s) using the R Monte Carlo replicates of the interesting quantity **statistic**.

x: an object of class ‘`MCM.sde`’.

index: the index of the variable of interest within the output of class ‘`MCM.sde`’.

type: type of plots: "hist" computes a histogram, "qqplot" produces a normal QQ plot, "boxplot" produces a box-and-whisker plot, and "CI" plots confidence intervals. Default **type** = "all".

...: arguments to be passed to `par()` function to set graphical parameters.

2.4. Moment equations method: ‘MEM.sde’ details

Rodriguez and Tuckwell (1996, 2000) introduced an elegant and systematic analytical method which consists of replacing the system of stochastic differential equations with a system of deterministic equations representing the dynamics of the means, variances, and covariances of the components (2) (or (3)). This method requires much less computation since it involves only the solution of a system of coupled deterministic ordinary differential equations. Also, it leads directly to solutions for the moments of the state variables, such as their means and variances which are the fundamental properties that we seek for in a model. We report a slight improvement of their method by adding the case of Stratonovich interpretation; hence the approach will be adopted in both cases, Itô and Stratonovich SDEs.

Let us assume that the distribution function of $\mathbf{X}(t)$ is concentrated near the mean point $\bar{\mathbf{X}}(t) = \{\bar{X}_1(t), \bar{X}_2(t), \dots, \bar{X}_d(t)\}$ (that is, $P(|\mathbf{X}(t) - \bar{\mathbf{X}}(t)| < \epsilon)$, for some (usually small) positive ϵ , is close to 1) and is symmetric around this point (see Rodriguez and Tuckwell 1996). For relatively small noise amplitudes, we take advantage of the fact that the third and higher-order odd central moments are close to zero and that the fourth and higher-order even moments are small relative to the second moment. Therefore, the exact means $\mu_j(t) = E(X_j(t))$, the variances $\text{VAR}_j(t) = E[(X_j(t) - \mu_j(t))^2]$ and the covariances $K_{ij}(t) = E[(X_i(t) - \mu_i(t))(X_j(t) - \mu_j(t))]$ can be approximated by $\mathbf{m}(t) = \{m_1(t), m_2(t), \dots, m_d(t)\}$, $\mathbf{S}(t) = \{S_1(t), S_2(t), \dots, S_d(t)\}$ and $C_{ij}(t)$ where $i, j = 1, 2, \dots, d$ ($i < j$), respectively, which obey a system of $(1/2)d(d+3)$ ordinary differential equations.

The vector of d means $\mathbf{m}(t)$ for the various components X_j , at time t is found to satisfy the following systems of ODEs:

$$\frac{d}{dt} m_j(t) = \{A_j\}_{(t, \mathbf{m}(t))} + \frac{1}{2} \sum_{l=1}^d \sum_{p=1}^d \left\{ \frac{\partial^2}{\partial x_l \partial x_p} A_j \right\}_{(t, \mathbf{m}(t))} C_{lp}(t), \quad (11)$$

whereas the vector of d variances $\mathbf{S}(t)$ at time t is determined by:

$$\begin{aligned} \frac{d}{dt} S_j(t) = & 2 \left\{ \frac{\partial}{\partial x_j} A_j \right\}_{(t, \mathbf{m}(t))} S_j(t) + 2 \sum_{\substack{l=1 \\ l \neq j}}^d \left\{ \frac{\partial}{\partial x_l} A_j \right\}_{(t, \mathbf{m}(t))} C_{lj}(t) + \sum_{k=1}^m \{g_{jk}^2\}_{(t, \mathbf{m}(t))} \\ & + \sum_{k=1}^m \sum_{l=1}^d \left\{ \left(\frac{\partial g_{jk}}{\partial x_l} \right)^2 + g_{jk} \frac{\partial^2 g_{jk}}{\partial x_l^2} \right\}_{(t, \mathbf{m}(t))} S_l(t) \\ & + \sum_{k=1}^m \sum_{l=1}^d \sum_{p=1}^d \left\{ \frac{\partial g_{jk}}{\partial x_p} \frac{\partial g_{jk}}{\partial x_l} + g_{jk} \frac{\partial^2 g_{jk}}{\partial x_l \partial x_p} \right\}_{(t, \mathbf{m}(t))} C_{lp}(t) \end{aligned} \quad (12)$$

and the $(1/2)d(d-1)$ covariance $\mathbf{C}(t)$ at time t is determined by:

$$\begin{aligned} \frac{d}{dt} C_{ij}(t) = & \frac{1}{2} \sum_{k=1}^m \sum_{l=1}^d \sum_{p=1}^d \left\{ g_{jk} \frac{\partial^2 g_{ik}}{\partial x_l \partial x_p} + \frac{\partial g_{ik}}{\partial x_l} \frac{\partial g_{jk}}{\partial x_p} + \frac{\partial g_{ik}}{\partial x_p} \frac{\partial g_{jk}}{\partial x_l} + g_{ik} \frac{\partial^2 g_{jk}}{\partial x_l \partial x_p} \right\}_{(t, \mathbf{m}(t))} C_{lp}(t) \\ & + \sum_{l=1}^d \left\{ \frac{\partial}{\partial x_l} A_i \right\}_{(t, \mathbf{m}(t))} C_{lj}(t) + \sum_{l=1}^d \left\{ \frac{\partial}{\partial x_l} A_j \right\}_{(t, \mathbf{m}(t))} C_{il}(t) \\ & + \sum_{k=1}^m \{g_{ik} g_{jk}\}_{(t, \mathbf{m}(t))} \end{aligned} \quad (13)$$

with:

$$A_i = f_i + \frac{\nu - 1}{2} \sum_{j=1}^d \sum_{k=1}^m g_{jk} \frac{\partial}{\partial x_j} g_{ik} \quad \text{and} \quad A_j = f_j + \frac{\nu - 1}{2} \sum_{i=1}^d \sum_{k=1}^m g_{ik} \frac{\partial}{\partial x_i} g_{jk},$$

where $\nu = 1$ and 2 stand, respectively, for the case of Itô and Stratonovich SDEs.

For the general linear system the solutions of these equations give explicit results (as illustrated in Appendix B). Otherwise, the moment equations can be solved numerically. R offers several tools for solving a system of ODEs, many of these tools can be found in the CRAN Task View on *Differential Equations* at <https://CRAN.R-project.org/view=DifferentialEquations> (Soetaert and Petzoldt 2020). For an easy transition from a package such as **deSolve** (Soetaert, Cash, and Mazzia 2012), we include a function containing the symbolic computations of the system (11–13). For this we use the package **Deriv** (Clausen and Sokol 2020) and the `D()` function available in the base package **stats**, and a list of parameters and their values in function `MEM.sde()`, and we will use function `ode()` (general solver for ODEs) from package **deSolve**, to solve numerically the obtained system.

The R function call consists of (see usage in the package help files):

```
MEM.sde(drift, diffusion, corr = NULL, type = c("ito", "str"),
        solve = FALSE, parms = NULL, init = NULL, time = NULL, ...)
```

The input arguments for `MEM.sde()` are:

- drift:** an R vector of expressions which contains the drift specification (1D, 2D and 3D).
- diffusion:** an R vector of expressions which contains the diffusion specification (1D, 2D and 3D).
- corr:** an R vector of expressions which contains the correlation structure between the Wiener processes (for any 2D and 3D). The default is `corr = NULL`.
- type:** type of SDEs to be used; "ito" for Itô form and "str" for Stratonovich form. The default is `type = "ito"`.
- solve:** if `solve = FALSE` only the symbolic computation of system (11–13) will be made; and if `solve = TRUE` a numerical approximation of the obtained system will be performed.
- parms:** parameters passed to `drift` and `diffusion` expressions.
- init:** initial (state) values of system (11–13).
- time:** time sequence (vector) for which output is sought; the first value of time must be the initial time.
- ...:** arguments to be passed to function `ode()` available in package **deSolve**, if `solve = TRUE`.

The returned object can be summarized using:

```
summary(object, at, ...)
```

which provides the summary results at a fixed time, defined by the argument `at`.

- object:** an object of the class 'MEM.sde'.
- at:** fixed time between minimum and maximum of time sequence (vector). The default is `at = T`.

For dynamic plot(s) of outputs, we will use the functions available in the **deSolve** package, e.g., `matplot.OD()`.

3. Model specification and functionalities

In the following, the functionality of the package is illustrated by examples. We present the main functions and their applications in univariate and multivariate cases (Itô and Stratonovich SDEs) to demonstrate their flexibility, the potential for parallelization, and customization for stochastic analysis. For each example, we start by setting the seed and modifying the R options using

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> options(scipen = 10, digits = 5)
```

so that readers can replicate the results.

3.1. One-dimensional SDE

The Black-Scholes model

Assume that we want to describe the following SDE type:

$$\text{Itô form:} \quad dX_t = \frac{1}{2}\theta^2 X_t dt + \theta X_t dW_t, \quad X_0 = x_0 > 0 \quad (14)$$

$$\text{Stratonovich form:} \quad dX_t = \frac{1}{2}\theta^2 X_t dt + \theta X_t \circ dW_t, \quad X_0 = x_0 > 0 \quad (15)$$

In the above $f(t, x) = \frac{1}{2}\theta^2 x$ and $g(t, x) = \theta x$ ($\theta > 0$), W_t is a standard Wiener process. To simulate both models (14) and (15) using the `snsde1d()` function, we need to specify:

- **drift** and **diffusion** coefficients as R expressions that depend on the state variable **x** and time variable **t**;
- number of simulation steps **N** (default: **N** = 1000);
- number of trajectories to be simulated **M** = 5000 (default: **M** = 1);
- initial conditions **t0** = 0, **x0** = 1 and ending time **T** = 1 (default: **t0** = 0, **x0** = 0 and **T** = 1);
- integration step size **Dt** = 0.001 (default: **Dt** = (**T** - **t0**) / **N**);
- the form of the process using argument **type** = "ito" for Itô and **type** = "str" for Stratonovich (default: **type** = "ito");
- numerical method to be used (default: **method** = "euler").

It is extremely easy to simulate 5000 trajectories of models (14) and (15) by the Euler-Maruyama scheme, with $\theta = 0.75$ and $x_0 = 1$, as follows:


```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> theta <- 0.75
R> x0 <- 1
R> f <- expression(0.5 * theta^2 * x)
R> g <- expression(theta * x)
```

Itô form (14):

```
R> mod1 <- snssde1d(drift = f, diffusion = g, x0 = x0, M = 5000,
+   type = "ito")
R> mod1
```

Itô Sde 1D:

```
| dX(t) = 0.5 * theta^2 * X(t) * dt + theta * X(t) * dW(t)
```

Method:

```
| Euler scheme with order 0.5
```

Summary:

```
| Size of process      | N = 1001.
| Number of simulation | M = 5000.
| Initial value       | x0 = 1.
| Time of process     | t in [0,1].
| Discretization      | Dt = 0.001.
```

Stratonovich form (15):

```
R> mod2 <- snssde1d(drift = f, diffusion = g, x0 = x0, M = 5000,
+   type = "str")
R> mod2
```

Stratonovich Sde 1D:

```
| dX(t) = 0.5 * theta^2 * X(t) * dt + theta * X(t) o dW(t)
```

Method:

```
| Euler scheme with order 0.5
```

Summary:

```
| Size of process      | N = 1001.
| Number of simulation | M = 5000.
| Initial value       | x0 = 1.
| Time of process     | t in [0,1].
| Discretization      | Dt = 0.001.
```

The `snssde1d()` function outputs an object of type time series ‘ts’ which can be handled in R as such, e.g., it can be plotted for both models via `plot(mod1)` and `plot(mod2)`. The statistical measures can be approximated for the X_t solution of the SDE at any fixed time $t \in [t_0, T]$, i.e., descriptive statistics of M realizations $\{X_t^\Delta(\omega_1), X_t^\Delta(\omega_2), \dots, X_t^\Delta(\omega_M)\}$ at fixed time t . For example the summary statistics of simulation outputs of `mod1` and `mod2` at ending time T is given by `summary(mod1, at = 1)` and `summary(mod2, at = 1)`.

For $t \geq 0$, the models (14) and (15) have log-normal transition density $p_\theta(t, X_t | X_{t_0} = x_0)$; i.e., the density of the distribution of X_t given $X_{t_0} = x_0$, with the mean and variance of its logarithm transform (i.e., the log-mean and log-variance) for each SDE type, are given by:

$$\begin{aligned} \mu_{\text{mod1}} &= \log(x_0) & \text{and} & \quad \sigma_{\text{mod1}}^2 = \theta^2 t \\ \mu_{\text{mod2}} &= \log(x_0) + \frac{1}{2}\theta^2 t & \text{and} & \quad \sigma_{\text{mod2}}^2 = \theta^2 t \end{aligned}$$

with mean and variance:

$$\begin{aligned} E(X_t^{\text{mod1}}) &= x_0 e^{\frac{1}{2}\theta^2 t} & \text{and} & \quad \text{VAR}(X_t^{\text{mod1}}) = x_0^2 e^{\theta^2 t} (e^{\theta^2 t} - 1) \\ E(X_t^{\text{mod2}}) &= x_0 e^{\theta^2 t} & \text{and} & \quad \text{VAR}(X_t^{\text{mod2}}) = x_0^2 e^{2\theta^2 t} (e^{\theta^2 t} - 1) \end{aligned}$$

Before talking about the approximation of transition density $p_\theta(t, X_t | X_{t_0} = x_0)$ for each type of model, we will illustrate in a one-dimensional linear case that the mean and variance for (14) and (15), coincide exactly with results obtained by MEM. We obtain the symbolic computation of a system of two ordinary differential equations for the mean and variance for each model using the `MEM.sde()` function (procedure outlined in Section 2.4). We can do this by simply running the following commands.

Moment equations of (14):

```
R> MEM.sde(drift = f, diffusion = g, type = "ito")
```

Itô Sde 1D:

```
| dX(t) = (0.5 * theta^2 * X(t)) * dt + theta * X(t) * dW(t)
| t in [t0,T].
```

Moment equations:

```
| dm(t) = 0.5 * (theta^2 * m(t))
| dS(t) = theta^2 * (2 * S(t) + m(t)^2)
```

Moment equations of (15):

```
R> MEM.sde(drift = f, diffusion = g, type = "str")
```

Stratonovich Sde 1D:

```
| dX(t) = 0.5 * theta^2 * X(t) * dt + theta * X(t) o dW(t)
| t in [t0,T].
```

Moment equations:

```
| dm(t) = theta^2 * m(t)
| dS(t) = theta^2 * (3 * S(t) + m(t)^2)
```

The exact solution of two symbolic systems of ODEs obtained by the function `MEM.sde()`, gives exactly the results previously known for mean and variance of both models, with initial conditions $m(0) = x_0$ and $S(0) = 0$. We obtain the numerical results by assigning the value `TRUE` to the argument `solve`, the initial value by `c(m = x0, S = 0)` and a vector containing the time points at which the solution is determined by `seq(0, 1, by = 0.001)`. In this case, we have assigned the outputs for each model to objects called `mem.mod1` and `mem.mod2` for further use in the R session.

Figure 1: Time dependence of the mean (left) and variance (right) for both models (14) and (15) calculated using function `MEM.sde()`, which exactly corresponds to the known values.

```
R> start <- c(m = x0, S = 0)
R> mem.mod1 <- MEM.sde(drift = f, diffusion = g, type = "ito", solve = TRUE,
+   parms = c(theta = 0.75), init = start, time = seq(0, 1, by = 0.001))
R> summary(mem.mod1, at = 1)
```

Approximation of moment at time 1

```
| m(1) = 1.3248
| S(1) = 1.3252
```

```
R> mem.mod2 <- MEM.sde(drift = f, diffusion = g, type = "str", solve = TRUE,
+   parms = c(theta = 0.75), init = start, time = seq(0, 1, by = 0.001))
R> summary(mem.mod2, at = 1)
```

Approximation of moment at time 1

```
| m(1) = 1.755
| S(1) = 2.3257
```

The dynamic plots of the outputs (mean $m(t)$ and variance $S(t)$) for each model are given in Figure 1. This can be done in R using the code:

```
R> plot(mem.mod1$sol.ode, mem.mod2$sol.ode, ylab = "m(t)", select = "m",
+   xlab = "Time", main = "", col = 2:3, lty = 1, las = 1, lwd = 2)
R> legend("topleft", c(expression(m[mod1](t), m[mod2](t))), inset = 0.01,
+   col = 2:3, lty = 1, lwd = 2, cex = 1.4)
R> plot(mem.mod1$sol.ode, mem.mod2$sol.ode, ylab = "S(t)", select = "S",
+   xlab = "Time", main = "", col = 2:3, lty = 1, las = 1, lwd = 2)
R> legend("topleft", c(expression(S[mod1](t), S[mod2](t))), inset = 0.01,
+   col = 2:3, lty = 1, lwd = 2, cex = 1.4)
```

Now, we use the methodology of the kernel approach (see, e.g., [Silverman 1986](#); [Wand and Jones 1995](#); [Scott 2015](#)) to approximate the transition density $p_\theta(t, X_t | X_{t_0} = x_0)$, based on

Figure 2: Time-evolution of the approximate transitional density for (14) and (15) at ending time $T = 1$ (left), and at fixed time points $t = \{0.2, 0.4, 0.6, 0.8, 1\}$ made using package **ggridges** (right), with initial condition $t_0 = 0$ and $X_0 = 1$.

the simulated trajectories of SDEs (i.e., output of function `snsde1d()`). The `dsde1d()` function (based on function `density()` available in base package **stats**) can be used to show the kernel approximation for $p_\theta(t, X_t | X_{t_0} = x_0)$. This consists of giving the initial condition of the SDE, i.e., the starting value `x0`, and the starting time `t0`. These two elements can be defined by assigning values to the arguments of function `snsde1d()`. The argument `at = 1` in function `dsde1d()` represents the fixed time t where the transition density is to be evaluated. For each type of SDE, the kernel estimation of transition density at ending time $T = 1$ are reported, see the left panel of Figure 2. Since the density approximation is evaluated over time, it can best be visualized with arranged multiple density plots in a staggered fashion, also known as ridgeline plots. Motivated by this, we can use the **ggridges** package (Wilke 2020), which provides a convenient way of visualizing changes in distributions over time, and we are reshaping the outputs with function `melt()` available in the **reshape** (or **reshape2**) package (Wickham 2007); see the right panel of Figure 2 (for the R code see Appendix A).

With the transition densities from (14) and (15) at $t = 1$:

```
R> dens.mod1 <- dsde1d(mod1, at = 1)
R> dens.mod2 <- dsde1d(mod2, at = 1)
R> plot(dens.mod1, xlim = c(0, 8))
R> plot(dens.mod2, add = TRUE)
R> legend("topright", c("mod1", "mod2"), inset = 0.01, col = c("#FF00004B",
+   "#0000FF4B"), pch = 15, cex = 1.6)
```

The argument `hist = TRUE` in function `dsde1d()` produces a histogram plot, with an optional curve by argument `dens` for the true transition density, if known, see Figure 3.

Log mean and log variance of (14) and (15) at ending time $t = 1$:

Figure 3: Histograms of simulated 5000-trajectories for (14) and (15) and true transitional density at ending time $T = 1$, with initial condition $t_0 = 0$ and $X_0 = 1$. Left: $X_{t=1}^{\text{mod1}} | X_0 = 1$ and right: $X_{t=1}^{\text{mod2}} | X_0 = 1$.

```
R> mu.mod1 <- log(x0)
R> sigma.mod1 <- sqrt(theta^2)
R> mu.mod2 <- log(x0) + 0.5 * theta^2
R> sigma.mod2 <- sqrt(theta^2)
R> plot(dens.mod1, dens = function(x) dlnorm(x, meanlog = mu.mod1,
+      sdlog = sigma.mod1), hist = TRUE, xlim = c(0, 8))
R> plot(dens.mod2, dens = function(x) dlnorm(x, meanlog = mu.mod2,
+      sdlog = sigma.mod2), hist = TRUE, xlim = c(0, 8))
```

Now we give an illustration of the approximation of a quantity of interest (e.g., mean and variance for an SDE at a fixed time t) using simulated data by the parallelized Monte Carlo method (procedure outlined in Section 2.3), and we compare the results obtained with the exact values. The exact values of mean and variance for each model are declared in R using:

```
R> E.mod1 <- function(t) x0 * exp(0.5 * theta^2 * t)
R> V.mod1 <- function(t) x0^2 * exp(theta^2 * t) * (exp(theta^2 * t) - 1)
R> E.mod2 <- function(t) x0 * exp(theta^2 * t)
R> V.mod2 <- function(t) x0^2 * exp(2 * theta^2 * t) * (exp(theta^2 * t) - 1)
```

We define the function of a quantity of interest by `stat.fun1d` in the current R session:

```
R> stat.fun1d <- function(data, i) {
+   d <- data[i, ]
+   return(c(mean(d), var(d)))
+ }
```

We apply the Monte Carlo method at a fixed time $T = 1$, using function `MCM.sde()`, with `parallel = "snow"` and `ncpus = 4` on Windows on a single machine (on a Unix platform, we can use `parallel = "multicore"` and for computer clusters see Appendix E).

Figure 4: QQ plots of the quantiles of the estimator of mean and variance for (14) and (15) from parallel Monte Carlo versus the theoretical quantile values from a normal distribution.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> mcm.mod1 <- MCM.sde(model = mod1, statistic = stat.fun1d, R = 100,
+   time = 1, exact = list(m = E.mod1(1), S = V.mod1(1)),
+   parallel = "snow", ncpus = 4)
R> mcm.mod1
```

Itô Sde 1D:

```
| dX(t) = 0.5 * theta^2 * X(t) * dt + theta * X(t) * dW(t)
| t in [0,1] with mesh equal to 0.001
```

PMCM Based on 100 Batches with 5000-Realisations at time 1:

	Exact Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
m	1.3248	1.3231	-0.00170	0.00163	0.01633 (1.31989 , 1.32627)
S	1.3252	1.3166	-0.00859	0.00824	0.08244 (1.30042 , 1.33272)

```
R> mcm.mod2 <- MCM.sde(model = mod2, statistic = stat.fun1d, R = 100,
+   time = 1, exact = list(m = E.mod2(1), S = V.mod2(1)),
+   parallel = "snow", ncpus = 4)
R> mcm.mod2
```

Stratonovich Sde 1D:

```
| dX(t) = 0.5 * theta^2 * X(t) * dt + theta * X(t) o dW(t)
| t in [0,1] with mesh equal to 0.001
```

PMCM Based on 100 Batches with 5000-Realisations at time 1:

	Exact Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
m	1.7550	1.7526	-0.00246	0.00216	0.02165 (1.74836 , 1.75682)
S	2.3257	2.3084	-0.01731	0.01444	0.14474 (2.28012 , 2.33672)

The QQ plot (see Figure 4) indicates that the circles are all very close to the red line (theoretical quantile values from a normal distribution), close enough to say these estimators come from a normal distribution. There is a little random wriggle around the red line, which does not disqualify these estimators from being normal.

		Parallel Monte Carlo					
		R = 100		R = 500		R = 1000	
		mod1	mod2	mod1	mod2	mod1	mod2
$E(X_1)$	Exact	1.3248	1.7550	1.3248	1.7550	1.3248	1.7550
	Estimate	1.3231	1.7526	1.3252	1.7554	1.3251	1.7552
	Bias	-0.0017	-0.0025	0.0004	0.0003	0.0003	0.0002
	Std.Error	0.0016	0.0021	0.0007	0.0009	0.0005	0.0006
	RMSE	0.0163	0.0216	0.0163	0.0216	0.0159	0.0211
	CI 2.5%	1.3199	1.7484	1.3238	1.7535	1.3241	1.7540
	CI 97.5%	1.3263	1.7569	1.3266	1.7573	1.3261	1.7566
$VAR(X_1)$	Exact	1.3252	2.3257	1.3252	2.3257	1.3252	2.3257
	Estimate	1.3166	2.3084	1.3230	2.3197	1.3244	2.3220
	Bias	-0.0086	-0.0174	-0.0022	-0.0059	-0.0008	-0.0036
	Std.Error	0.0083	0.0144	0.0040	0.0070	0.0032	0.0055
	RMSE	0.0825	0.1448	0.0894	0.1567	0.0999	0.1750
	CI 2.5%	1.3005	2.2801	1.3152	2.3060	1.3182	2.3112
	CI 97.5%	1.3328	2.3368	1.3309	2.3335	1.3306	2.3330

Table 2: Simulation results for mean and variance of `mod1` and `mod2` approximated by the parallel Monte Carlo method, with various batches (R batches of 5000 trajectories simulated by the Euler-Maruyama scheme) at a fixed time $T = 1$.

```
R> plot(mcm.mod1, index = 1, type = "qqplot")
R> plot(mcm.mod1, index = 2, type = "qqplot")
R> plot(mcm.mod2, index = 1, type = "qqplot")
R> plot(mcm.mod2, index = 2, type = "qqplot")
```

In Table 2, we give for each type an approximation of the mean and variance obtained by function `MCM.sde()` with various batches at a fixed time $T = 1$. Through the results obtained, we find that the parallelized Monte Carlo method is very robust and offers excellent performance. Moreover, the central limit theorem guarantees convergence and asymptotic normality of estimators. Due to the parallel random number generation support, this avoids the problem of lack of independence in the samples from multiple RNG. Hence a replication number of few independent realizations is sufficient to obtain a suitable approximation for the first and central second-order moments to provide information on the shape of the distribution.

The standard error (`Std.Error`) and root mean square error (`RMSE`) are chosen as criteria for comparison between the Itô and Stratonovich interpretations. In the case of the Stratonovich form, note that the two criteria are higher than those found in the Itô form (see Figure 5). This is explained by the fact that stochastic calculus with the Stratonovich form increases the variability of a statistic, due to the drift correction formula (4).

We can also confirm these results by the bootstrap method (Davison and Hinkley 1997). In R, we use function `boot()` available in package `boot` (Canty and Ripley 2020) on the same function of statistics of interest `stat.fun1d`, and using function `rsde1d()` to get the initial sample. More formally:

```
R> library("boot")
R> data.mod1 <- data.frame(rsde1d(mod1, at = 1))
```

Figure 5: Standard error (left) and root mean square error (right) for means of Itô mod1 and Stratonovich mod2, R-batches with M -trajectories simulated at a fixed time $T = 1$.

```
R> boot.mod1 <- boot(data.mod1, statistic = stat.fun1d, R = 1000,
+   sim = "ordinary", parallel = "snow", ncpus = 4)
R> boot.mod1
```

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:

```
boot(data = data.mod1, statistic = stat.fun1d, R = 1000, sim = "ordinary",
     parallel = "snow", ncpus = 4)
```

Bootstrap Statistics :

	original	bias	std. error
t1*	1.3398	0.0014313	0.017048
t2*	1.3917	0.0050765	0.083983

```
R> data.mod2 <- data.frame(rsde1d(mod2, at = 1))
R> boot.mod2 <- boot(data.mod2, statistic = stat.fun1d, R = 1000,
+   sim = "ordinary", parallel = "snow", ncpus = 4)
R> boot.mod2
```

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:

```
boot(data = data.mod2, statistic = stat.fun1d, R = 1000, sim = "ordinary",
     parallel = "snow", ncpus = 4)
```

Bootstrap Statistics :

	original	bias	std. error
t1*	1.7195	-0.00026092	0.020519
t2*	2.1407	0.00142817	0.132146

		Number of processors (ncpus)						
		1		2		4		
		mod1	mod2	mod1	mod2	mod1	mod2	
Speed-up and execution time	R = 100	multicore	1	1	1.529	1.588	1.844	1.888
			57.346	62.086	37.514	39.101	31.096	32.880
		snow	1	1	1.507	1.503	1.828	1.855
			57.346	62.086	38.166	41.201	31.376	33.462
	R = 500	muticore	1	1	1.725	1.752	2.064	2.158
			370.100	393.950	214.509	224.810	179.310	182.553
		snow	1	1	1.702	1.737	2.040	2.111
			370.100	393.950	217.389	226.798	181.434	186.581
	R = 1000	multicore	1	1	2.012	2.089	2.508	2.569
			749.061	798.307	372.292	382.169	298.635	310.723
		snow	1	1	2.003	2.064	2.501	2.532
			749.061	798.307	374.041	386.847	299.497	315.310

Table 3: Speed-up and execution time (in seconds) for simulation results obtained in Table 2, with various numbers of processors.

The tie between bootstrap and Monte Carlo simulation of a statistic is obvious, both are based on repetitive sampling and then a direct examination of the results. A big difference between the methods, however, is that bootstrapping uses the original, initial sample as the population from which to re-sample, whereas Monte Carlo simulation is based on setting up a data generating process (with known values of the parameters). This strategy could be used to approximate some quantities, like in the bootstrap, but also to theoretically investigate some general characteristic of an estimator which is hard to derive analytically.

Table 3 shows speed-up and computing time for mean and variance of (14) and (15) approximated by function `MCM.sde()` at a fixed time $T = 1$, with different number of processors. This timing test was done on a 2.20 GHz Intel Core i5-5200U processor running Unix platform (Ubuntu 17.04 operating system), and 8 GB 1600 MHz DDR3 of memory. We notice that the execution time with **multicore** (shared memory), which is suitable for Unix systems, is faster than **snw** (distributed memory).

The nonlinear mean reversion model

Aït-Sahalia (1996) proposed a nonlinear model including polynomial terms for modeling interest rates. In the following, we study the simplest version of this model which satisfies the nonlinear SDE:

$$dX_t = \left(\beta_0 + \beta_1 X_t + \beta_2 X_t^2 + \beta_3 X_t^{-1} \right) dt + \sigma X_t^\rho dW_t. \quad (16)$$

Some natural restrictions have to be imposed on the parameter values $\theta = (\beta_0, \beta_1, \beta_2, \beta_3, \sigma, \rho)$ to have a meaningful specification of the model; for more details we refer the reader to the Appendix in Aït-Sahalia (1996). In general, there are no exact distributional results. This model has been estimated empirically using various techniques by many authors including Aït-Sahalia (1996), Conley, Hansen, Luttmer, and Scheinkman (1997) and Gallant and Tauchen (1998) just to mention a few references. Later Aït-Sahalia (1999, 2002) proposed an approx-

imation of transition densities via the Hermite polynomial expansion³. We shall report the results of the simulation analysis under assumption 2 (high frequency: $T = n\Delta$ fixed, $\Delta \rightarrow 0$ as $n \rightarrow \infty$) only for the case: $\beta_0 > 0$, $\beta_1 < 0$, $\beta_2 \leq 0$, $\beta_3 > 0$, $\sigma > 0$ and $\rho > 1$.

We can simulate the numerical solution of this model quite easily with parameter values: $(\beta_0 = 2, \beta_1 = -1, \beta_2 = -0.5, \beta_3 = 4, \sigma = 0.25, \rho = 1.5)$ and initial conditions: $x_0 = 5$, $t_0 = 0$, step size $\Delta = 0.005$, as follows:

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> beta0 <- 2
R> beta1 <- -1
R> beta2 <- -0.5
R> beta3 <- 4
R> sigma <- 0.25
R> f <- expression((beta0 + beta1 * x + beta2 * x^2 + beta3 * x^-1))
R> g <- expression(sigma * x^1.5)
R> mod <- snssde1d(drift = f, diffusion = g, x0 = 5, M = 5000, Dt = 0.005)
```

We apply the parallelized Monte Carlo method for the model (16) at ending time $T = 5$, to the same previous function of interest `stat.fun1d` (i.e., mean and variance of (16) at ending time $T = 5$); we get the following result:

```
R> mcm.mod <- MCM.sde(model = mod, statistic = stat.fun1d, R = 100,
+   time = 5, ncpus = 4, names = c("m", "S"), parallel = "snow")
R> mcm.mod$MC
```

```
      Estimate Std.Error  CI( 2.5 % , 97.5 % )
m  2.00055    0.00039 ( 1.99979 , 2.00131 )
S  0.06551    0.00016 ( 0.0652 , 0.06582 )
```

To create the code for the L^AT_EX table of the output results `mcm.mod`, see Section 5.1.

From the Markovian property of the diffusion, it is also possible to define the stochastic transient of the model (16) by the time evolution of the probability density function $P(x; t)$, which satisfies a Fokker-Planck equation also called Kolmogorov forward equation (e.g., see Risken 1996):

$$\partial_t P(x; t) = \frac{1}{2} \sigma^2 \frac{\partial^2}{\partial x^2} \left[x^{2\rho} P(x; t) \right] - \frac{\partial}{\partial x} \left[\left(\frac{\beta_2 x^3 + \beta_1 x^2 + \beta_0 x + \beta_3}{x} \right) P(x; t) \right]. \quad (17)$$

An analytic solution of the partial differential equation (17) is unlikely due to the nonlinearity of the model (16). By using moment equations method (11–12), we obtain a set of two equations of moments (i.e., $m(t)$ mean and $S(t)$ variance) in closed form through function `MEM.sde()`. Furthermore, if the user is interested in exporting the equations displayed by printing ‘`MEM.sde`’, to be used in a L^AT_EX document, see Section 5.2. In R:

³The `sde` package (Iacus 2008) contains the function `HPloglik()`: Hermite polynomial expansion approximation of the log-likelihood of a process solution of an SDE. This function is useful to approximate maximum likelihood estimators when the transition density of the process is not known.

```
R> MEM.sde(drift = f, diffusion = g, type = "ito")
```

$$\begin{cases} \frac{d}{dt}m(t) = \beta_0 + (\beta_1 + \beta_2 m(t)) m(t) + \frac{\beta_3}{m(t)} + \left(\frac{\beta_3}{m(t)^3} + \beta_2 \right) S(t) \\ \frac{d}{dt}S(t) = \sigma^2 m(t)^3 + \left(2\beta_1 + (4\beta_2 + 3\sigma^2) m(t) - \frac{2\beta_3}{m(t)^2} \right) S(t) \end{cases} \quad (18)$$

The stochastic transient with initial conditions ($m(0) = x_0, S(0) = 0$) is numerically solved from the system (18). We do this by just running the following command:

```
R> mem.mod <- MEM.sde(drift = f, diffusion = g, type = "ito", solve = TRUE,
+   init = c(m = 5, S = 0), time = seq(0, 5, by = 0.001))
R> summary(mem.mod, at = 5)
```

Approximation of moment at time 5

```
| m(5) = 2
| S(5) = 0.06557
```

In Figure 6 we show the results obtained by the Monte Carlo simulation of model (16) and the system of moment equations (18), where there is an excellent agreement between the two approaches over time $t \in [0, 5]$. The following code has been used to produce Figure 6.

```
R> plot(mem.mod$sol.ode, lty = 2, lwd = 4, ylab = "m(t)", select = "m",
+   las = 1, main = "")
R> lines(time(mod), apply(mod$X, 1, mean), col = 2, lwd = 2)
R> legend("topright", c(expression("MEM", "MCM")), col = 1:2, lty = c(2, 1))
R> plot(mem.mod$sol.ode, lty = 2, lwd = 4, ylab = "S(t)", select = "S",
+   main = "")
R> lines(time(mod), apply(mod$X, 1, var), col = 2, lwd = 2)
R> legend("topright", c(expression("MEM", "MCM")), col = 1:2, lty = c(2, 1))
```

In order to approximate the transitional density of the model (16) using the methodology proposed by Pedersen (1995), we used function `dcSim()`⁴ from package `sde` (see Iacus 2008, p. 134), and we compare the results obtained with that of function `dsde1d()`. The following code approximates the transition density at ending time $T = 5$ by both functions, with initial conditions $t_0 = 0$ and $x_0 = 5$.

With `dcSim()`:

```
R> parm <- c(beta0, beta1, beta2, beta3, sigma)
R> drift <- function(t, x, parm) parm[1] + parm[2] * x + parm[3] * x^2 +
+   parm[4] * x^-1
R> diffu <- function(t, x, parm) parm[5] * x^1.5
R> x <- seq(1, 3.5, length = 100)
R> dens <- NULL
R> for (to in x) {
+   dens <- c(dens, sde::dcSim(x0 = 5, x = to, t = 5, d = drift,
+     s = diffu, theta = parm, N = 100))
+ }
```

⁴`dcSim()`: simulated transition density of a diffusion process based on Pedersen's method.

Figure 6: Approximation of mean and variance as functions of time, where there is good agreement between results for Monte Carlo simulation based on one batch of 5000 realizations of (16) and moment equations (18).

With `dsde1d()`:

```
R> R <- 100
R> dens_R <- lapply(1:R, function(i) dsde1d(eval(mod$call), at = 5))
R> y <- apply(sapply(1:R, function(i) dens_R[[i]]$res$y), 1, mean)
R> xx <- apply(sapply(1:R, function(i) dens_R[[i]]$res$x), 1, mean)
R> plot(xx, dens, type = "l", xlim = c(1, 3.5), ylab = "transition density")
R> polygon(xx, y, col = "#0000FF4B", border = "#0000FF82")
R> legend("topright", c("dcSim()", "dsde1d()"), col = c(1, "#0000FF4B"),
+       lty = c(1, NA), pch = c(NA, 15), inset = 0.01)
```

Indeed, the approximate solutions of the transition density with functions `dcSim()` and `dsde1d()` are nearly identical, as we see in the left panel of Figure 7.

Now, we consider in particular the cumulative distribution function (CDF) $F_X(x)$ of a random variable X with the function defined at any point x as:

$$F_X(x) = P(X \leq x). \quad (19)$$

There are two well-known estimators for the CDF. The first estimator is the empirical cumulative distribution function (ECDF), and the second is the kernel estimator of the distribution function (KCDF). In fact, we can quickly estimate (19) for model (16) in a specific time T at any point x by both approaches using the formulation (9):

$$\hat{F}_{X_T^\Delta}(x) = \begin{cases} \frac{1}{MR} \sum_{i=1}^R \sum_{j=1}^M \mathbb{1}_{\{X_T^\Delta(\omega_{i,j}) \leq x\}}, & \text{ECDF} \\ \frac{1}{MR} \sum_{i=1}^R \sum_{j=1}^M H\left(\frac{x - X_T^\Delta(\omega_{i,j})}{h}\right), & \text{KCDF} \end{cases} \quad (20a) \quad (20b)$$

Figure 7: Left: Approximate transitional density for model (16), using the **sde** package (**dcSim()**: black line) and the **Sim.DiffProc** package (**dsde1d()**: light blue). Right: Empirical and kernel estimator of CDF for model (16). Initial conditions $t_0 = 0$, $x_0 = 5$ and ending time $T = 5$.

where $\mathbb{1}_{\{\cdot\}}$ is the indicator function and $H(x) = \int_{-\infty}^x K(t)dt$, with $K(\cdot)$ the kernel function and h the bandwidth parameter. R has a function **ecdf()** to compute the ECDF of an individual data set. The function returns a ‘function’ object to compute the value of $F_n(x)$ at any point x . Let us write an R function **ecdf.fun** to compute (20a) as follows:

```
R> ecdf.fun <- function(data, i) {
+   d <- data[i, ]
+   Fn <- stats::ecdf(d)
+   return(c(Fn(seq(1.5, 3.75, by = 0.25))))
+ }
```

To calculate the second estimator (20b), we need to use function **kcdf()** available in package **ks** (Duong 2007). Furthermore, we calculate the bandwidth h using the 1- and 2-stage plug-in selectors of Polansky and Baker (2000), with function **hpi.kcde()** (to go further into the theory see the recent article of Duong 2016). In R we define the function below:

```
R> kcdf.fun <- function(data, i, stage) {
+   d <- data[i, ]
+   Fhat <- ks::kcde(d, h = ks::hpi.kcde(d, nstage = stage))
+   return(predict(Fhat, x = seq(1.5, 3.75, by = 0.25)))
+ }
```

Finally, the use of **MCM.sde()** is quite simple, i.e., as in the previous examples. The numerical results of both methods are reported in Table 4, and shown in the right panel of Figure 7.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> mcm.ecdf <- MCM.sde(model = mod, statistic = ecdf.fun, R = 100, time = 5,
+   names = paste0("P(X <= ", seq(1.5, 3.75, by = 0.25), ")"),
+   parallel = "snow", ncpus = 4)
```

$\hat{F}_{X_T}(x)$	x	ECDF (20a)	KCDF (20b)	
			PB plug-in	
			1-stage	2-stage
	1.50	0.00514 ± 0.00010 (0.00494, 0.00534)	0.00591 ± 0.00009 (0.00573, 0.00609)	0.00590 ± 0.00009 (0.00572, 0.00608)
	1.75	0.15344 ± 0.00050 (0.15246, 0.15442)	0.15847 ± 0.00048 (0.15753, 0.15941)	0.15845 ± 0.00048 (0.15751, 0.15939)
	2.00	0.54974 ± 0.00068 (0.54841, 0.55107)	0.55369 ± 0.00062 (0.55247, 0.55491)	0.55370 ± 0.00062 (0.55248, 0.55492)
	2.25	0.84682 ± 0.00047 (0.84590, 0.84774)	0.84834 ± 0.00047 (0.84742, 0.84926)	0.84835 ± 0.00047 (0.84743, 0.84927)
	2.50	0.95960 ± 0.00030 (0.95901, 0.96019)	0.95996 ± 0.00029 (0.95939, 0.96053)	0.95996 ± 0.00029 (0.95939, 0.96053)
	2.75	0.99045 ± 0.00013 (0.99020, 0.99070)	0.99051 ± 0.00013 (0.99026, 0.99076)	0.99051 ± 0.00013 (0.99026, 0.99076)
	3.00	0.99785 ± 0.00006 (0.99773, 0.99797)	0.99786 ± 0.00006 (0.99774, 0.99798)	0.99786 ± 0.00006 (0.99774, 0.99798)
	3.25	0.99953 ± 0.00003 (0.99947, 0.99959)	0.99953 ± 0.00003 (0.99947, 0.99959)	0.99953 ± 0.00003 (0.99947, 0.99959)
	3.50	0.99988 ± 0.00002 (0.99984, 0.99992)	0.99988 ± 0.00002 (0.99984, 0.99992)	0.99988 ± 0.00002 (0.99984, 0.99992)
	3.75	0.99997 ± 0.00001 (0.99995, 0.99999)	0.99997 ± 0.00001 (0.99995, 0.99999)	0.99997 ± 0.00001 (0.99995, 0.99999)

Table 4: Results obtained from estimators (20a) and (20b) of the CDF for model (16) at ending time $T = 5$. (estimate ± standard error) and (CI 2.5%, CI 97.5%).

```
R> mcm.kcdf1 <- MCM.sde(model = mod, statistic = kcdf.fun, R = 100, time = 5,
+   names = paste0("P(X <= ", seq(1.5, 3.75, by = 0.25), " ")),
+   stage = 1, parallel = "snow", ncpus = 4)
R> mcm.kcdf2 <- MCM.sde(model = mod, statistic = kcdf.fun, R = 100, time = 5,
+   names = paste0("P(X <= ", seq(1.5, 3.75, by = 0.25), " ")),
+   stage = 2, parallel = "snow", ncpus = 4)
```

3.2. Two-dimensional SDEs

Ornstein-Uhlenbeck process and its integral

The Ornstein-Uhlenbeck (OU) process has a long history in physics. Introduced in essence by Langevin (Lemons and Gythiel 1997) in his famous 1908 paper on Brownian motion, the process received a more thorough mathematical examination several decades later by Uhlenbeck and Ornstein (1930) and Wang and Uhlenbeck (1945). The OU process is understood here to be the univariate continuous Markov process X_t . In mathematical terms, the equation is written as an Itô equation:

$$dX_t = \frac{1}{\mu}(\theta - X_t)dt + \sqrt{\sigma}dW_t, \quad X_0 = x_0. \quad (21)$$

In this equation, μ and σ are positive constants called, respectively, the relaxation time and the diffusion constant. $\theta \in \mathbb{R}$ is the asymptotic mean of X_t . The time integral of the OU process X_t (or indeed of any process X_t) is defined to be the process Y_t that satisfies:

$$Y_t = Y_0 + \int X_t dt \Leftrightarrow dY_t = X_t dt, \quad Y_0 = y_0. \quad (22)$$

Y_t is not itself a Markov process; however, X_t and Y_t together comprise a bivariate continuous Markov process (see Gillespie 1996). We wish to find the solutions X_t and Y_t to the coupled time-evolution (21) and (22):

$$\begin{cases} dX_t = \frac{1}{\mu}(\theta - X_t)dt + \sqrt{\sigma}dW_t \\ dY_t = X_t dt \end{cases} \quad (23)$$

We first take note of the following well-known result, which can be derived from the system (23). For any $t > 0$ the OU process X_t and its integral Y_t will have a bivariate normal distribution with means and variances given by (where the approximations of the first and central second-order moments with MEM coincide exactly with the known values, see Appendix B for demonstration):

$$\begin{aligned} E(X_t) &= \theta + (x_0 - \theta)e^{-t/\mu} \quad \text{and} \quad E(Y_t) = y_0 + \theta t + (x_0 - \theta)\mu(1 - e^{-t/\mu}) \\ \text{VAR}(X_t) &= \frac{\sigma\mu}{2}(1 - e^{-2t/\mu}) \quad \text{and} \quad \text{VAR}(Y_t) = \sigma\mu^3\left(\frac{t}{\mu} - 2(1 - e^{-t/\mu}) + \frac{1}{2}(1 - e^{-2t/\mu})\right) \end{aligned}$$

and the covariance is:

$$\text{COV}(X_t, Y_t) = \frac{\sigma\mu^2}{2}(1 - 2e^{-t/\mu} + e^{-2t/\mu}).$$

We now have a complete and exact solution to the problem of the time evolution of X_t and Y_t . Next, we will use this information to compare with parallelized Monte Carlo simulation. For (23), we simulate 5000-trajectories using function `snssde2d()`, with $\mu = 1$, $\theta = 2$, $\sigma = 0.5$, $t \in [0, 20]$ and initial value $(x_0, y_0) = (-5, 0)$. The simulation output of (23) is assigned to an object called `OUI`. We can plot the system via `plot(OUI)`, and use `summary(OUI, at = 20)` to obtain the descriptive statistics of $\{(X_T^\Delta(\omega_1), Y_T^\Delta(\omega_1)), (X_T^\Delta(\omega_2), Y_T^\Delta(\omega_2)), \dots, (X_T^\Delta(\omega_M), Y_T^\Delta(\omega_M))\}$ at ending time $T = 20$.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> mu <- 1
R> theta <- 2
R> sigma <- 0.5
R> x0 <- -5
R> y0 <- 0
R> f <- expression(1/mu * (theta - x), x)
R> g <- expression(sqrt(sigma), 0)
R> OUI <- snssde2d(drift = f, diffusion = g, M = 5000, T = 20,
+   x0 = c(x0, y0))
```

First, we declare the exact values of the first and central second-order moments of (23), for comparison with the estimated results.

```
R> Ex <- function(t) theta + (x0 - theta) * exp(-t/mu)
R> Vx <- function(t) 0.5 * sigma * mu * (1 - exp(-2 * (t/mu)))
R> Ey <- function(t) y0 + theta * t + (x0 - theta) * mu * (1 - exp(-t/mu))
R> Vy <- function(t) sigma * mu^3 * ((t/mu) - 2 * (1 - exp(-t/mu))) +
+   0.5 * (1 - exp(-2 * (t/mu)))
R> Cxy <- function(t) 0.5 * sigma*mu^2 * (1 - 2 * exp(-t/mu) +
+   exp(-2 * (t/mu)))
```

Secondly, we define the function for the statistics of interest by `stat.fun2d` (i.e., means, variances and covariance of (23)).

```
R> stat.fun2d <- function(data, i) {
+   d <- data[i, ]
+   return(c(mean(d$x), mean(d$y), var(d$x), var(d$y), cov(d$x, d$y)))
+ }
```

Next, we apply the parallelized Monte Carlo method for model (23) at ending time $T = 20$:

```
R> mcm.oui <- MCM.sde(model = OUI, statistic = stat.fun2d, R = 100,
+   time = 20, parallel = "snow", ncpus = 4, exact = list(m1 = Ex(20),
+   m2 = Ey(20), S1 = Vx(20), S2 = Vy(20), C12 = Cxy(20)))
R> mcm.oui
```

Itô Sde 2D:

```
| dX(t) = 1/mu * (theta - X(t)) * dt + sqrt(sigma) * dW1(t)
| dY(t) = X(t) * dt + 0 * dW2(t)
| t in [0,20] with mesh equal to 0.02
```

PMCM Based on 100 Batches with 5000-Realisations at time 20:

	Exact	Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
m1	2.00	1.99966	-0.00034	0.00073	0.00725	(1.99823 , 2.00109)
m2	33.00	32.99981	-0.00019	0.00411	0.04085	(32.99175 , 33.00787)
S1	0.25	0.25296	0.00296	0.00050	0.00580	(0.25198 , 0.25394)
S2	9.25	9.27241	0.02241	0.02009	0.20115	(9.23303 , 9.31179)
C12	0.25	0.24562	-0.00438	0.00215	0.02182	(0.24141 , 0.24983)

We notice that a number of 100 independent replications of parallelized Monte Carlo is usually enough to get a good approximation at a fixed time $T = 20$. The resulting plots of simulation over time $t \in [0, 20]$ are given in Figure 8, where there is good agreement between results of Monte Carlo and exact values of the first and central second-order moments of system (23).

We have several packages for creating a 3D plot in R, which can be very useful for visualizing the features of a more complex density. For static plotting, we use the `scatterplot3d` package (Ligges and Mächler 2003) and interactive plotting is provided by `rgl` (Adler and Murdoch

Figure 8: Results of Monte Carlo simulation based on one batch of 5000 realizations of (23) and exact values of the first and central second-order moments (R code: supplementary materials).

Figure 9: Left: A marginal transition density of x and y over the 5000 realizations of (23) at $t = \{0, 2, 4, \dots, 20\}$. Right: An example of a contour plot of the transitional density of (23), superimposed on the simulated mean trajectory $(m_1(t), m_2(t))$ (R code: supplementary materials).

2020). The arguments `pdf = "Joint"` and `at = 20` produce a kernel approximation of the bivariate density function (based on the function `kde2d()` available in the package **MASS**; Venables and Ripley 2002) of the couple (X_t, Y_t) at any fixed time using function `dsde2d()`, with several forms of display (`"contour"`, `"image"`, `"persp"`, `"rgl"`), e.g., see the right panel of Figure 9. The argument `pdf = "Marginal"` produces an approximation of a marginal density function of the process. We use the **gggridges** package to show the evolution of the marginal density of X_t and Y_t at $t = \{0, 2, 4, \dots, 20\}$, see the left panel of Figure 9.

Figure 10: Left: The animation of contour plots of evolution the transition density of (23) at $t = \{0, 1, \dots, 20\}$, superimposed on the simulated mean trajectory. Right: Interactive 3D plot of bivariate density obtained from a 5000 realizations of (23) at ending time $T = 20$. Using the function `dsde2d()` with `pdf = "Joint"` and `at = 20`.

We may generate contour plots of the transition density at fixed time points $t = \{0, 1, \dots, 20\}$ with `display = "contour"`, the animation plots are made using the **animation** package (Xie 2013); see the left panel of Figure 10.

```
R> mycol <- colorRampPalette(c("gray96", "blue", "green", "yellow", "red"))
R> for (i in seq(0, 20, by = 1)) {
+   plot(dsde2d(OUI, pdf = "Joint", at = i, lims = c(-6, 4, -5, 42), h = 2),
+       display = "contour", color = mycol,
+       main = paste0("Transition Density at t = ", i),
+       plot.axes = {
+         lines(my[seq(1, which(time(OUI) == i), by = 1)] ~
+             mx[seq(1, which(time(OUI) == i), by = 1)], lty = 2)
+         axis(1, seq(-10, 10, by = 1))
+         axis(2, seq(-50, 50, by = 5))
+         legend("topleft", lty = 2, legend = "Simulated Expectation",
+             inset = 0.01)}
+ }
```

To create an interactive 3D plot for the transition density at ending time $T = 20$, we use `display = "rgl"`, see the right panel of Figure 10. Users can also export the matrix of transition density values and create 3D plots in other programs (for example, see Section 3.3, where we used package **ggplot2**; Wickham 2009); and to move the graph plotted in 3D using the mouse see the `writeASY()` function available in the **rgl** package.

```
R> denJ <- dsde2d(OUI, pdf = "Joint", at = 20, h = 2,
+   lims = c(-1, 5, 23, 44))
```

```
R> plot(denJ, display = "rgl", col2d = "lightblue")
```

The bivariate cumulative distribution function (BCDF) $F_{(X,Y)}(x, y)$ of two random variables X and Y for a couple of fixed level (x, y) , are given by:

$$F_{(X,Y)}(x, y) = P(X \leq x, Y \leq y) \quad (24)$$

We can easily approximate (24) at any fixed time T by the empirical BCDF. As in a one-dimensional case, using formulation (9) for the EBCDF as:

$$\hat{F}_{(X_T^\Delta, Y_T^\Delta)}(x, y) = \frac{1}{MR} \sum_{i=1}^R \sum_{j=1}^M \mathbb{1}_{\{X_T^\Delta(\omega_{i,j}) \leq x, Y_T^\Delta(\omega_{i,j}) \leq y\}} \quad (25)$$

and we compare it with the exact bivariate cumulative distribution of system (23), i.e.,

$$(X_T, Y_T) \sim \Phi(\mu, \Sigma), \quad (26)$$

where Φ corresponds to the bivariate normal cumulative distribution function, with mean vector μ and variance-covariance matrix Σ at fixed time T . In R, we begin by declaring the exact values of (26) at ending time $T = 20$ in coupled points $(x \in (1, 2, 3), y \in (28, 33, 38))$. For that we need to use `pmvnorm()` from the **mvtnorm** package (Genz and Bretz 2009; Genz et al. 2020).

```
R> T <- 20
R> Mu <- c(Ex(T), Ey(T))
R> Sigma <- matrix(c(Vx(T), Cxy(T), Cxy(T), Vy(T)), ncol = 2)
R> Fxy1 <- mvtnorm::pmvnorm(upper = c(1, 28), mean = Mu, sigma = Sigma)[1]
R> Fxy2 <- mvtnorm::pmvnorm(upper = c(2, 28), mean = Mu, sigma = Sigma)[1]
R> Fxy3 <- mvtnorm::pmvnorm(upper = c(3, 28), mean = Mu, sigma = Sigma)[1]
R> Fxy4 <- mvtnorm::pmvnorm(upper = c(1, 33), mean = Mu, sigma = Sigma)[1]
R> Fxy5 <- mvtnorm::pmvnorm(upper = c(2, 33), mean = Mu, sigma = Sigma)[1]
R> Fxy6 <- mvtnorm::pmvnorm(upper = c(3, 33), mean = Mu, sigma = Sigma)[1]
R> Fxy7 <- mvtnorm::pmvnorm(upper = c(1, 38), mean = Mu, sigma = Sigma)[1]
R> Fxy8 <- mvtnorm::pmvnorm(upper = c(2, 38), mean = Mu, sigma = Sigma)[1]
R> Fxy9 <- mvtnorm::pmvnorm(upper = c(3, 38), mean = Mu, sigma = Sigma)[1]
R> Ext.val <- as.list(c(Fxy1, Fxy2, Fxy3, Fxy4, Fxy5, Fxy6, Fxy7, Fxy8,
+   Fxy9))
```

Then we used function `ebvcdf()`⁵ (empirical bivariate cumulative distribution function) from package **bivariate** (Spurdle 2020) to compute the EBCDF (25), as follows:

```
R> ecdf2d.fun <- function(data, i) {
+   d <- data[i, ]
+   xy <- as.matrix(expand.grid(x = c(1, 2, 3), y = c(28, 33, 38)))
+   Fnxy <- bivariate::ebvcdf(x = d$x, y = d$y)
+   return(Fnxy(x = xy[, "x"], y = xy[, "y"]))
+ }
```

⁵Users can also use function `kcde()` (multivariate kernel CDF) from package **ks** (see Duong 2016).

At last, we apply the parallelized Monte Carlo at ending time $T = 20$. We obtain the following results.

```
R> name.out <- c(paste0("F(", 1:3, ", 28)"), paste0("F(", 1:3, ", 33)"),
+   paste0("F(", 1:3, ", 38)"))
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> mcm.ecdf2d <- MCM.sde(model = OUI, statistic = ecdf2d.fun, R = 100,
+   time = T, ncpus = 4, parallel = "snow", exact = Ext.val,
+   names = name.out)
R> mcm.ecdf2d$MC
```

	Exact	Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
F(1, 28)	0.00233	0.00236	0.00003	0.00032	0.00064	(0.00173 , 0.00299)
F(2, 28)	0.03177	0.03224	0.00047	0.00097	0.00199	(0.03034 , 0.03414)
F(3, 28)	0.04964	0.04916	-0.00048	0.00101	0.00209	(0.04718 , 0.05114)
F(1, 33)	0.01487	0.01436	-0.00051	0.00065	0.00140	(0.01309 , 0.01563)
F(2, 33)	0.27628	0.27284	-0.00344	0.00153	0.00461	(0.26984 , 0.27584)
F(3, 33)	0.49212	0.49120	-0.00092	0.00126	0.00268	(0.48873 , 0.49367)
F(1, 38)	0.02230	0.02140	-0.00090	0.00104	0.00226	(0.01936 , 0.02344)
F(2, 38)	0.48168	0.48268	0.00100	0.00346	0.00699	(0.4759 , 0.48946)
F(3, 38)	0.92949	0.92956	0.00007	0.00059	0.00118	(0.9284 , 0.93072)

Bivariate nonlinear dynamic system: The general stochastic Haken-Zwanzig equations

The stochastic version⁶ of a bivariate Haken-Zwanzig model (Friedrich and Haken 1988; Leung and Lai 1993) is used to demonstrate the slaving principle in Synergetics (Haken 2004) and it is also related to the more complex *ABCDE* model (see, e.g., Friedrich and Haken 1992). We propose the general stochastic model as Stratonovich equations:

$$\begin{cases} dX_t = (\mu X_t - X_t Y_t) dt + \sigma_1 \left(\mathbb{1}_{\{k=0\}} + X_t \mathbb{1}_{\{k=1\}} + X_t(1 - Y_t) \mathbb{1}_{\{k=2\}} \right) \circ dW_{1,t} \\ dY_t = -(\alpha Y_t - X_t^2) dt + \sigma_2 \left(\mathbb{1}_{\{k=0\}} + Y_t \mathbb{1}_{\{k=1\}} + Y_t(1 - X_t) \mathbb{1}_{\{k=2\}} \right) \circ dW_{2,t} \end{cases} \quad (27)$$

where rate parameters (μ, α) are non-negative, $(\sigma_1, \sigma_2) \geq 0$ describes the amplitude of the fluctuating quantity, $k = 0, 1$ and 2 respectively for additive noise, linear and nonlinear multiplicative noises. Obtaining the analytical solution of (27) is not possible, numerical methods are required. A complete description of the solution implies assignments of values to the two rate parameters (μ, α) and to the two initial conditions (x_0, y_0) . This model is stable and has three fixed points: the first point is $e_u = (0, 0)$ describing the extinction of the dynamic process, it is an unstable source; and the second (two) points are $e_s = (\pm\sqrt{\mu\alpha}, \mu)$ representing the coexistence of two-modes (x, y) , and they are stable since the eigenvalues of stability are given by:

$$\begin{cases} -0.5\alpha \pm 0.5\sqrt{\alpha^2 - 8\mu\alpha} & , \text{ if } \alpha > 8\mu \\ -0.5\alpha \pm 0.5\sqrt{8\mu\alpha - \alpha^2} & , \text{ if } \alpha \leq 8\mu \end{cases}$$

⁶For a deterministic version of a bivariate Haken-Zwanzig model see Fuchs (2013a, p. 148).

Figure 11: The flow field, fixed points and nullclines (red and blue lines are x - and y -nullclines, respectively) of the deterministic Haken-Zwanzig model, with $\mu = 1$ and $\alpha = 3$.

where i denotes the imaginary number $\sqrt{-1}$. The flow field, fixed points e_u and e_s , together with the nullclines are shown in Figure 11, which is made using package **phaseR** (Grayling 2014); for the R code see Appendix C.1.

The stochastic transient of system (27) is now described by a probability density function $P(x, y; t)$, which is found to satisfy a Fokker-Planck equation (forward), and is expressed in the Stratonovich representation by:

$$\begin{aligned} \partial_t P = & -\partial_x \left[\left((\mu x - xy) + \frac{k}{2^k} \sigma_1^2 x (1-y)^{2^k-2} \right) P \right] + \frac{1}{2} \sigma_1^2 \partial_{xx} \left[x^{\frac{4k}{2^k}} (1-y)^{k(k-1)} P \right] \\ & -\partial_y \left[\left((-\alpha y + x^2) + \frac{k}{2^k} \sigma_2^2 y (1-x)^{2^k-2} \right) P \right] + \frac{1}{2} \sigma_2^2 \partial_{yy} \left[y^{\frac{4k}{2^k}} (1-x)^{k(k-1)} P \right] \end{aligned} \quad (28)$$

Should we be able to solve $P(x, y; t)$ analytically, then statistical moments can be calculated explicitly to provide relevant information (for example, the peak location and width of $P(x, y; t)$ are determined by the first two orders moments). However, since an explicit solution of (28) is unlikely; one can quickly obtain the final results of the moment equations for each type of noise with function **MEM.sde()**, yet numerical treatment is required to solve these coupled equations with initial conditions: $m_1(0) = x_0, m_2(0) = y_0, S_1(0) = S_2(0) = C_{12}(0) = 0$, and for parameters $\mu = 1$ and $\alpha = 3$. The system (27) allows us to treat and study the effect of the three types of noise: if $k = 0$ we have an additive noise, which signifies that the two representation Itô and Stratonovich are equivalent; and if $k = 1, 2$ we have linear and nonlinear multiplicative noise, respectively. We were assigning appropriate values as $\sigma_1 = 0.1$ and $\sigma_2 = 0.05$; for a concise presentation, we shall report results only for $\sigma_1 > \sigma_2$. In the case $k = 0$, we choose the initial state $x_0 = y_0 = 0$, and in the other two cases $x_0 = y_0 = 1$.

In Table 5, we give for each type of noise an approximation of the first and central second-order moments at ending time $T = 100$ using functions **MEM.sde()** and **MCM.sde()**, where

			MEM	MCM	
				R = 100	R = 500
$x_0 = 0, y_0 = 0$	$k = 0$	$m_1(t)$	-0.00083	-0.00296 ± 0.00235	0.00053 ± 0.00108
		$m_2(t)$	1.00166	0.99801 ± 0.00011	0.99816 ± 0.00004
		$S_1(t)$	3.00499	2.99382 ± 0.00042	2.99430 ± 0.00016
		$S_2(t)$	0.00042	0.00485 ± 0.00001	0.00485 ± 0.00000
		$C_{12}(t)$	0	-0.00021 ± 0.00015	0.00008 ± 0.00007
$x_0 = 1, y_0 = 1$	$k = 1$	$m_1(t)$	1.72870	1.72672 ± 0.00017	1.72689 ± 0.00007
		$m_2(t)$	0.99997	0.99902 ± 0.00015	0.99937 ± 0.00007
		$S_1(t)$	0.01025	0.01487 ± 0.00003	0.01485 ± 0.00001
		$S_2(t)$	0.01046	0.01335 ± 0.00003	0.01335 ± 0.00001
		$C_{12}(t)$	0.00870	0.00979 ± 0.00003	0.00979 ± 0.00001
	$k = 2$	$m_1(t)$	1.73180	1.73181 ± 0.00002	1.73180 ± 0.00001
		$m_2(t)$	1.00000	0.99999 ± 0.00003	0.99998 ± 0.00001
		$S_1(t)$	0.00011	0.00015 ± 0.00000	0.00015 ± 0.00000
		$S_2(t)$	0.00023	0.00033 ± 0.00000	0.00033 ± 0.00000
		$C_{12}(t)$	0	0.00003 ± 0.00000	0.00003 ± 0.00000

Table 5: Summary of results for moment equations and parallel Monte Carlo simulation (estimate $\pm$ standard error) at ending time $T = 100$ (R code is listed in Appendix C.2).

there is a good agreement between these two methods. Generally, an agreement between the results of these methods was reasonably close (see the left panel of Figure 12), which was obtained when the net mean driving force was large and the variances small. This is to be expected from the assumptions under which the moment equations are derived.

The right panel of Figure 12 shows the effect of noise on system dynamics. We clearly see the acceleration of the transition to stable points $e_s = (\pm\sqrt{\mu\alpha}, \mu)$ of (27), which also appears in Figure 13 where the transition of the limit cycle to a fixed point e_s was predicted in numerical studies of the steady state probability. We found that in cases $k = 1$ and 2 influence of noise becomes state dependent since the structures of $g(x, y)$ function (diffusion) which are no longer constant (linear and nonlinear function). Usually, additive noise causes smaller effects than multiplicative noises, and also could slow down or speed up the transient process of a dynamic system. From the simulation of system (27), we find that linear multiplicative noise (i.e., the case $k = 1$) causes significant effects (that is clearly shown in the right panel of Figure 12), due to the fact that the response of this system to noise is nonlinear. For a qualitative comparison of the effects of noise, we evaluate the relative fluctuation $R_j(t)$ for the three cases, which is given by:

$$R_j(t) = 100 \times \frac{\sqrt{S_j(t)}}{m_j(t)} \%, \text{ for } j = 1, 2.$$

As previously, we give an illustration of the approximation of $R_j(t)$ at ending time $T = 100$ with function `MCM.sde()`, for case $k = 1$ only (same procedure for the cases $k = 0$ and 2).

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> mu <- 1
R> alpha <- 3
```

Figure 12: Left: The realization of system (27) created using `plot2d()` plotted in (x, y) -space. Right: Approximation of the first and central second-order moments as functions of time, where there is good agreement between results for simulation based on one batch of 5000 realizations of system (27) and moment equations, for each type of noise (R code: Supplementary materials).

Figure 13: Left: An evolution of the marginal transition density of x and y . Center and right: contour and 3D plots of the transitional density at ending time $T = 100$, obtained from a 5000 realizations of system (27) for the three types of noise (R code: Supplementary materials).

```
R> sigma1 <- 0.1
R> sigma2 <- 0.05
R> I1 <- c(x = 1, y = 1)
R> f <- expression((mu * x - x * y), (-alpha * y + x^2))
R> g1 <- expression(sigma1 * x, sigma2 * y)
R> modk1 <- snssde2d(drift = f, diffusion = g1, M = 5000, x0 = I1, Dt = 0.1,
+   type = "str")
R> Rf.fun <- function(data, i) {
+   d <- data[i, ]
+   return(c(100 * (sd(d$x)/mean(d$x)), 100 * (sd(d$y)/mean(d$y))))
+ }
```

			MEM		MCM	
			Str	Itô	Str	Itô
$x_0 = 1, y_0 = 1$	$k = 0$	$R_1(t)$	3.13%	3.13%	4.14%	4.14%
		$R_2(t)$	6.13%	6.13%	6.95%	6.95%
	$k = 1$	$R_1(t)$	5.85%	5.86%	7.04%	7.04%
		$R_2(t)$	10.22%	10.25%	11.52%	11.54%
	$k = 2$	$R_1(t)$	0.61%	0.61%	0.70%	0.70%
		$R_2(t)$	1.50%	1.50%	1.80%	1.80%

Table 6: Relative fluctuations approximated by functions `MCM.sde()` and `MEM.sde()` at ending time $T = 100$, for both representations Itô and Stratonovich with the three types of noises (R code: Supplementary materials).

Figure 14: Time dependence of relative fluctuations $R(t)$ due to additive, linear and nonlinear multiplicative noises of (27), with $(x_0 = 1, y_0 = 1)$. Left: $R_1(t)$ and right: $R_2(t)$ (R code: Supplementary materials).

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> Rf_xy <- MCM.sde(modk1, statistic = Rf.fun, R = 100, time = 100,
+   parallel = "snow", ncpus = 4)
```

We can also use the outputs of function `MEM.sde()` (i.e., `sol.ode`: numerical solution of

moment equations), to calculate the quantity $R_j(t)$, through the analytical approximation of $m_j(t)$ and $S_j(t)$.

```
R> t <- seq(0, 100, by = 0.001)
R> S <- c(m1 = 1, m2 = 1, S1 = 0, S2 = 0, C12 = 0)
R> MEMSk1 <- MEM.sde(drift = f, diffusion = g1, solve = TRUE, init = S,
+   time = t, type = "str")
R> Rf_x <- 100 * (sqrt(MEMSk1$sol.ode[, "S1"]) /
+   MEMSk1$sol.ode[, "m1"])[length(t)]
R> Rf_y <- 100 * (sqrt(MEMSk1$sol.ode[, "S2"]) /
+   MEMSk1$sol.ode[, "m2"])[length(t)]
```

In Table 6, we summarize the results obtained for the two representations Itô and Stratonovich of the stochastic Haken-Zwanzig model. Firstly, we find that the approximation of $R(t)$ by function `MEM.sde()` (deterministic method) is slightly smaller than the values found by function `MCM.sde()` (statistical method). This is often the case because $R(t)$ is sensitive to minor changes in the mean $m(t)$ over time t (see the top of Figure 14). On the other hand, the relative fluctuations of the system with the two representations Itô and Stratonovich are indistinguishable in the three cases of noise (see the bottom of Figure 14). This means there is no effect of the type of model (Itô or Stratonovich) on the relative fluctuation, i.e., $R^{\text{Itô}}(t) = R^{\text{Str}}(t)$. Secondly, we notice that the relative fluctuation in case of linear multiplicative noise is greater than in case of additive and nonlinear multiplicative noises, i.e., $R_j^{k=1}(t) > R_j^{k=0}(t) > R_j^{k=2}(t)$ for $j = 1, 2$.

3.3. Three-dimensional SDEs

The radio-astronomy model

In radio-space science, signals from a star are analyzed to get evaluations of certain characteristic parameters of the star. According to [Le Gland \(1981\)](#), such a signal can be represented as a complex-valued stochastic process:

$$\frac{d\eta_t}{dt} = \mu_1 \exp((Z_t + \mu_2) i) + \sqrt{\sigma} \xi_t, \quad \forall t \in [t_0, T] \quad (29)$$

where $i = \sqrt{-1}$, μ_1 and μ_2 represent, respectively, the signal amplitude and the mean phase of η_t . Z_t is a real valued zero-mean process representing the effects of atmospheric turbulence on the signal η_t , ξ_t is a complex-valued Gaussian White noise process and σ is a positive constant. The stochastic process (29) can be written as a complex-valued SDE:

$$d\eta_t = \mu_1 \exp((Z_t + \mu_2) i) dt + \sqrt{\sigma} dW_t, \quad (30)$$

where W_t is a complex-valued Wiener process, that is $W_t = W_{1,t} + i W_{2,t}$ for independent Wiener processes $W_{1,t}$ and $W_{2,t}$, with $\eta_t = X_t + i Y_t$. The previous equation can be written as two real-valued equations:

$$\begin{cases} dX_t = \mu_1 \cos(Z_t + \mu_2) dt + \sqrt{\sigma} dW_{1,t} \\ dY_t = \mu_1 \sin(Z_t + \mu_2) dt + \sqrt{\sigma} dW_{2,t} \end{cases} \quad (31)$$

Le Gland (1981) used an Ornstein-Uhlenbeck process Z_t satisfying the model (21) with $\theta = 0$, to model the atmospheric turbulence. To simplify the complexity of this model, let us take the drift of the model (21) as ODE with $\theta = 0$ and for some positive parameter ν to model the effect of atmospheric turbulence; and let us assume that the diffusion coefficients are different in each equation. Hence the observed signal η_t consists of the following equations:

$$\begin{cases} dX_t = \mu_1 \cos(Z_t + \mu_2) dt + \sqrt{\sigma_1} dW_{1,t} \\ dY_t = \mu_1 \sin(Z_t + \mu_2) dt + \sqrt{\sigma_2} dW_{2,t} \\ \frac{dZ_t}{dt} = -\frac{Z_t}{\nu} \end{cases} \quad \forall x_0, y_0 \in \mathbb{R} \text{ and } z_0 \in \mathbb{R}_+^* \quad (32)$$

It is easy to simulate the numerical solution of system (32) by function `snssde3d()` (under the assumption 2: $T = n\Delta$ fixed and $\Delta \rightarrow 0$ as $n \rightarrow \infty$), with parameter values: $\mu_1 = 25$, $\mu_2 = \pi/4$, $\sigma_1 = 2$, $\sigma_2 = 1$ and $\nu = 12$; initial states: $(x_0 = 0, y_0 = 0, z_0 = 100)$, $t \in [0, 50]$ with step size $\Delta = 0.05$. In R:

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> mu1 <- 25
R> mu2 <- pi/4
R> sigma1 <- 2
R> sigma2 <- 1
R> nu <- 12
R> x0 <- 0
R> y0 <- 0
R> z0 <- 100
R> f <- expression(mu1 * cos(z+mu2), mu1 * sin(z+mu2), -z/nu)
R> g <- expression(sqrt(sigma1), sqrt(sigma2), 0)
R> mod3d <- snssde3d(drift = f, diffusion = g, x0 = c(x0, y0, z0),
+   M = 5000, Dt = 0.05)
```

For plotting (in time) we use simply the function `plot(mod3d, union = TRUE)`, and in the space (O, X, Y, Z) use the `plot3D()` function, with two display types "rgl" or "persp".

```
R> plot(mod3d, union = TRUE)
R> plot3D(mod3d, display = "persp", box = FALSE, col = "blue")
```

Now, we apply function `MEM.sde()` for the stochastic system (32). We obtain the symbolic system of ODEs (if we want to export the equations displayed by 'MEM.sde' to be used in a L^AT_EX document, see the system (43) in Section 5.2). In R this can be done quite easily:

```
R> mem.mod3d <- MEM.sde(drift = f, diffusion = g)
R> mem.mod3d
```

Itô Sde 3D:

```
| dX(t) = mu1 * cos(Z(t) + mu2) * dt + sqrt(sigma1) * dW1(t)
| dY(t) = mu1 * sin(Z(t) + mu2) * dt + sqrt(sigma2) * dW2(t)
| dZ(t) = -Z(t)/nu * dt + 0 * dW3(t)
| t in [t0, T].
```

Figure 15: A trajectory of the stochastic model (32) in time (left) and in xyz -space (right).

Moment equations:

$$\begin{aligned}
 | \quad dm_1(t) &= \mu_1 * (1 - 0.5 * S_3(t)) * \cos(\mu_2 + m_3(t)) \\
 | \quad dm_2(t) &= \mu_1 * (1 - 0.5 * S_3(t)) * \sin(\mu_2 + m_3(t)) \\
 | \quad dm_3(t) &= -(m_3(t)/\nu) \\
 | \quad dS_1(t) &= \sigma_1 - 2 * (C_{13}(t) * \mu_1 * \sin(\mu_2 + m_3(t))) \\
 | \quad dS_2(t) &= 2 * (C_{23}(t) * \mu_1 * \cos(\mu_2 + m_3(t))) + \sigma_2 \\
 | \quad dS_3(t) &= -(2 * (S_3(t)/\nu)) \\
 | \quad dC_{12}(t) &= \mu_1 * (C_{13}(t) * \cos(\mu_2 + m_3(t)) - C_{23}(t) * \sin(\mu_2 + m_3(t))) + \\
 &\quad \sqrt{\sigma_1} * \sqrt{\sigma_2} \\
 | \quad dC_{13}(t) &= -(C_{13}(t)/\nu + \mu_1 * S_3(t) * \sin(\mu_2 + m_3(t))) \\
 | \quad dC_{23}(t) &= \mu_1 * S_3(t) * \cos(\mu_2 + m_3(t)) - C_{23}(t)/\nu
 \end{aligned}$$

The exact solutions of the obtained system (43) are given explicitly. We find, the means:

$$m_1(t) = x_0 + \nu \mu_1 \left[\cos(\mu_2) (-\text{Ci}(m_3(t)) + \text{Ci}(z_0)) + \sin(\mu_2) (\text{Si}(m_3(t)) - \text{Si}(z_0)) \right] \quad (33a)$$

$$m_2(t) = y_0 + \nu \mu_1 \left[\cos(\mu_2) (-\text{Si}(m_3(t)) + \text{Si}(z_0)) - \sin(\mu_2) (\text{Ci}(m_3(t)) - \text{Ci}(z_0)) \right] \quad (33b)$$

$$m_3(t) = z_0 \exp(-t/\nu) \quad (33c)$$

and the variance-covariance matrix:

$$\Sigma = \begin{pmatrix} \sigma_1 t & \sqrt{\sigma_1 \sigma_2} t & 0 \\ \sqrt{\sigma_1 \sigma_2} t & \sigma_2 t & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad (33d)$$

with the cosine and sine integrals:

$$\begin{aligned}
 \text{Ci}(x) &= - \int_x^\infty \frac{\cos(s)}{s} ds = \gamma + \ln(x) + \int_0^x \frac{\cos(s) - 1}{s} ds \\
 \text{Si}(x) &= \int_0^x \frac{\sin(s)}{s} ds
 \end{aligned}$$

Figure 16: The means and variances-covariances of system (32) approximated by MEM and MCM (one batch of 5000 realizations). Left: $m_j(t)$, middle: $S_j(t)$ and right: $C_{ij}(t)$. (R code: Supplementary materials).

where γ is the Euler-Mascheroni constant, defined by:

$$\gamma = -\frac{d}{dx} \ln(\Gamma(1)) = -\int_0^\infty e^{-x} \ln(x) dx \approx 0.57721 \dots$$

Many special mathematical functions are present in R, see for example the CRAN Task View on *Numerical Mathematics* at <https://CRAN.R-project.org/view=NumericalMathematics> (Borchers 2020). To compute the cosine and sine integrals we can use the `Ci()` and `Si()` functions available in package **pracma** (Borchers 2019). For the Euler-Mascheroni constant γ , we can use simply the base function `digamma()`, which gives the first derivative of the logarithm of the gamma function $\Gamma(x)$, with $\gamma = -\text{digamma}(1)$.

```
R> library("pracma")
R> m1 <- function(t) x0 + nu * mu1 * (cos(mu2) * (-Ci(z0 * exp(-t/nu)) +
+   Ci(z0)) + sin(mu2) * (Si(z0 * exp(-t/nu)) - Si(z0)))
R> m2 <- function(t) y0 + nu * mu1 * (cos(mu2) * (-Si(z0 * exp(-t/nu)) +
+   Si(z0)) - sin(mu2) * (Ci(z0 * exp(-t/nu)) - Ci(z0)))
R> c(m1(50), m2(50))
```

```
[1] -144.58013 -57.80049
```

```
R> -digamma(1)
```

```
[1] 0.57722
```

In Figure 16 we compare the results obtained from MEM with those of MCM. We notice that the means and variances for the two methods are practically indistinguishable, but there is a bad agreement between the covariance curves ($C_{12}^{\text{MEM}}(t) \neq C_{12}^{\text{MCM}}(t)$). This is due to the assumptions for the validity of the MEM which are not satisfied, implying that agreement with simulation is poor, i.e., the distribution function of system (32) is not concentrated near the mean point.

We conclude this example by estimating the parameters for the system (32). Since we have explicit theoretical moments (33a–33d), we want to estimate the vector of the parameters

$(\mu_1, \mu_2, \sigma_1, \sigma_2)$ by the method of moments; this consists of using the empirical moments. For this, the empirical moments $\overline{x}_t$ and $\overline{y}_t$, which we respectively equal to the theoretical moments (33a) and (33b) as follows:

$$\overline{x}_t = \frac{1}{M} \sum_{j=1}^M x_t(\omega_j) = m_1(t), \quad \overline{y}_t = \frac{1}{M} \sum_{j=1}^M y_t(\omega_j) = m_2(t), \quad (t \text{ fixed})$$

and the same for second-order theoretical moments (33d), that we equal to the sample variance:

$$\text{VAR}(x_t) = S_1(t), \quad \text{VAR}(y_t) = S_2(t), \quad (t \text{ fixed})$$

The estimators for $\mu_1, \mu_2, \sigma_1, \sigma_2$ denoted by $\hat{\mu}_1, \hat{\mu}_2, \hat{\sigma}_1, \hat{\sigma}_2$ are described as the solution to the following system:

$$\begin{cases} \overline{x}_t - x_0 - \hat{\nu} \hat{\mu}_1 \left[\cos(\hat{\mu}_2) (-\text{Ci}(z_t) + \text{Ci}(z_0)) + \sin(\hat{\mu}_2) (\text{Si}(z_t) - \text{Si}(z_0)) \right] & = 0 & (34a) \\ \overline{y}_t - y_0 - \hat{\nu} \hat{\mu}_1 \left[\cos(\hat{\mu}_2) (-\text{Si}(z_t) + \text{Si}(z_0)) - \sin(\hat{\mu}_2) (\text{Ci}(z_t) - \text{Ci}(z_0)) \right] & = 0 & (34b) \\ \text{VAR}(x_t) - \hat{\sigma}_1 t & = 0 & (34c) \\ \text{VAR}(y_t) - \hat{\sigma}_2 t & = 0 & (34d) \end{cases}$$

from (33c), we directly get the estimator $\hat{\nu}$:

$$\hat{\nu} = \frac{t}{\ln(z_0) - \ln(z_t)} \quad (35a)$$

where z_t is the numerical solution of the ODE Z_t (since $\overline{z}_t = z_t(\omega_j)$ and $\text{VAR}(z_t) = 0$, $\forall j = 1 \dots M$ for all t fixed).

Then, to obtain the estimator $\hat{\mu}_1$, we apply the operation (34a) $\times \sin(\hat{\mu}_2)$ - (34b) $\times \cos(\hat{\mu}_2)$, we get:

$$\hat{\mu}_1 = \frac{-(\overline{x}_t - x_0) \sin(\hat{\mu}_2) + (\overline{y}_t - y_0) \cos(\hat{\mu}_2)}{\hat{\nu} (\text{Si}(z_0) - \text{Si}(z_t))} \quad (35b)$$

we substitute (35b) in (34a) and (34b); then to get the estimator $\hat{\mu}_2$, we apply the following operation (34a) $\times \cos(\hat{\mu}_2)$ - (34b) $\times \sin(\hat{\mu}_2)$, we find:

$$\hat{\mu}_2 = -\arctan \left[\frac{-(\overline{x}_t - x_0) (\text{Si}(z_0) - \text{Si}(z_t)) + (\overline{y}_t - y_0) (\text{Ci}(z_0) - \text{Ci}(z_t))}{-(\overline{x}_t - x_0) (\text{Ci}(z_0) - \text{Ci}(z_t)) - (\overline{y}_t - y_0) (\text{Si}(z_0) - \text{Si}(z_t))} \right] \quad (35c)$$

Finally, for the estimation of the diffusion coefficients, from (34c) and (34d) it is easy to find for any t fixed:

$$\hat{\sigma}_1 = \frac{1}{t(M-1)} \sum_{j=1}^M (x_t(\omega_j) - \overline{x}_t)^2 \quad (35d)$$

$$\hat{\sigma}_2 = \frac{1}{t(M-1)} \sum_{j=1}^M (y_t(\omega_j) - \overline{y}_t)^2 \quad (35e)$$

Generally, it is very difficult to find explicit results, especially in a nonlinear case, so we always require some numerical treatment. We propose two solutions to computing numerically the

estimators if we have explicit moments like in our case. First, we will use the `multiroot()` function available in package `rootSolve` (Soetaert 2020) to solve a nonlinear system (34a–34d); this function consists of root-finding of n nonlinear equations, using a Newton-Raphson method (an extension of R function `uniroot()`). The downside of this function is that the result will depend on the initial guess of the roots and it is not guaranteed that the roots will actually be found (that is, the method may fail). The second solution is more efficient; we are using function `gmm()`⁷ from package `gmm` (Chaussé 2010) to estimate the parameters of the nonlinear system (34a–34d). This function uses the generalized method of moments of Hansen (1982), and the result has associated methods like `coef`, `vcov` and `confint`. Inference can therefore be performed on estimators. In R, we use these two procedures and compare them with explicit results (35b–35e).

```
R> root.eq <- function(x, t) {
+   A <- x0 + nu_hat * x[1] * (cos(x[2]) * (-Ci(zt) + Ci(z0)) +
+     sin(x[2]) * (Si(zt) - Si(z0)))
+   B <- y0 + nu_hat * x[1] * (cos(x[2]) * (-Si(zt) + Si(z0)) -
+     sin(x[2]) * (Ci(zt) - Ci(z0)))
+   c(xbar - A, ybar - B, varx - x[3] * t, vary - x[4] * t)
+ }
R> gmm.eq <- function(p, x, t = T) {
+   A <- x0 + nu_hat * p[1] * (cos(p[2]) * (-Ci(zt) + Ci(z0)) +
+     sin(p[2]) * (Si(zt) - Si(z0)))
+   B <- y0 + nu_hat * p[1] * (cos(p[2]) * (-Si(zt) + Si(z0)) -
+     sin(p[2]) * (Ci(zt) - Ci(z0)))
+   m1 <- x[, 1] - A
+   m2 <- x[, 2] - B
+   S1 <- (x[, 1] - mean(x[, 1]))^2 - p[3] * t
+   S2 <- (x[, 2] - mean(x[, 2]))^2 - p[4] * t
+   f <- cbind(m1, m2, S1, S2)
+   return(f)
+ }
R> expl.eq <- function(t) {
+   A <- -(xbar - x0) * (Si(z0) - Si(zt)) + (ybar - y0) * (Ci(z0) - Ci(zt))
+   B <- -(xbar - x0) * (Ci(z0) - Ci(zt)) - (ybar - y0) * (Si(z0) - Si(zt))
+   C <- -(xbar - x0) * sin(-atan(A/B)) + (ybar - y0) * cos(-atan(A/B))
+   mu1_hat <- C/(nu_hat * (Si(z0) - Si(zt)))
+   mu2_hat <- -atan(A/B)
+   sig1_hat <- varx/t
+   sig2_hat <- vary/t
+   return(c(mu1 = mu1_hat, mu2 = mu2_hat, sig1 = sig1_hat,
+     sig2 = sig2_hat))
+ }
```

Now, we compute $\hat{\nu}$ and empirical moments at any fixed time, for example at ending time $T = 50$:

⁷We can also use function `gel()` “generalized empirical likelihood method” from package `gmm`, but it is very unstable numerically, for more details see Chaussé (2010).

```

R> T <- 50
R> data <- rsde3d(mod3d, at = T)
R> Means <- apply(data, 2, mean)
R> Var <- apply(data, 2, var)
R> xbar <- Means[["x"]]
R> ybar <- Means[["y"]]
R> zt <- data$z[1]
R> varx <- Var[["x"]]
R> vary <- Var[["y"]]
R> nu_hat <- T/(log(z0) - log(zt))
R> nu_hat

```

```
[1] 11.975
```

We assign values to the initial guesses $c(\mu_1 = 5, \mu_2 = 1, \text{sig1} = 0.1, \text{sig2} = 0.1)$:

```

R> library("rootSolve")
R> library("gmm")
R> res1 <- c(mu1, mu2, sigma1, sigma2)
R> res2 <- multiroot(root.eq, start = c(mu1 = 5, mu2 = 1, sig1 = 0.1,
+   sig2 = 0.1), parms = c(t = T))$root
R> res3 <- gmm(g = gmm.eq, x = data, t0 = c(mu1 = 5, mu2 = 1, sig1 = 0.1,
+   sig2 = 0.1), method = "Nelder-Mead", control = list(maxit = 1000,
+   reltol = 1e-25))
R> res4 <- expl.eq(T)
R> data.frame(rbind(True.value = res1, Multi.root = res2, GMM = coef(res3),
+   Explicit = res4))

```

	mu1	mu2	sig1	sig2
True.value	25.00000	0.78540	2.00000	1.00000
Multi.root	24.86331	0.79045	2.02504	1.00175
GMM	24.86331	0.79045	2.02464	1.00155
Explicit	24.86331	0.79045	2.02504	1.00175

As we see the outputs are consistent, the approaches `multiroot()` and `gmm()` are entirely in line with the results which are explicitly found. The explicit estimators of model (32) as a function of t can be inspected graphically in Figure 17. The `gmm()` function is worth using because, as said before, one can obtain the variance-covariance matrix and confidence intervals numerically. In this case, we highly recommend it.

```
R> res3
```

```
Method twoStep
```

```
Objective function value: 4.7687e-21
```

	mu1	mu2	sig1	sig2
	24.86331	0.79045	2.02464	1.00155

```
Convergence code = 0
```

Figure 17: Explicit estimators of model (32) in $t \in [0, 50]$ (R code: Supplementary materials).

```
R> vcov(res3)
```

	mu1	mu2	sig1	sig2
mu1	4.7069e-04	-3.5478e-06	-2.2289e-05	-2.2747e-07
mu2	-3.5478e-06	4.9271e-07	-4.4379e-07	9.2461e-07
sig1	-2.2289e-05	-4.4379e-07	1.5620e-03	5.6923e-06
sig2	-2.2747e-07	9.2461e-07	5.6923e-06	3.7738e-04

```
R> confint(res3, level = 0.95)
```

Wald type confidence interval

#####

	0.025	0.975
mu1	24.82079	24.90583
mu2	0.78907	0.79183
sig1	1.94718	2.10210
sig2	0.96347	1.03962

Transformation of a scalar SDE into three dimensions

Next we consider a scalar SDE driven by three independent⁸ Wiener processes $(W_{1,t}, W_{2,t}, W_{3,t})$:

$$dX_t = \mu W_{1,t} dt + \sigma W_{2,t} dW_{3,t}, \quad X_0 = x_0, \quad \forall t \in [t_0, T] \quad (35)$$

To simulate the numerical solution of (35) with package **Sim.DiffProc**, we make a transformation to a system of three equations as follow:

$$(35) \Leftrightarrow \begin{cases} dX_t = \mu Y_t dt + \sigma Z_t dW_{1,t} \\ dY_t = dW_{2,t} \\ dZ_t = dW_{3,t} \end{cases} \quad \forall t \in [t_0, T] \quad (36)$$

⁸To simulate stochastic models (3D) driven by correlated Wiener processes, see Appendix D.2.

We simply use function `snssde3d()`, with $\mu = 0.75$, $\sigma = 0.5$, initial states $(x_0, y_0, z_0) = (10, 0, 0)$ and $t \in [0, 5]$.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> x0 <- 10
R> mu <- 0.75
R> sigma <- 0.5
R> f <- expression(mu*y, 0, 0)
R> g <- expression(sigma*z, 1, 1)
R> mod <- snssde3d(drift = f, diffusion = g, M = 5000, x0 = c(x0, 0, 0),
+   T = 5)
```

For any $t \in [0, T]$ the process X_t will have a normal distribution with mean and variance given by (function `MEM.sde()` gives exactly the known values):

$$E(X_t) = x_0 \quad \text{and} \quad \text{VAR}(X_t) = \frac{1}{3}\mu^2 t^3 + \frac{1}{2}\sigma^2 t^2$$

and the auto-covariance function:

$$\text{COV}(X_s, X_t) = \left[\frac{1}{2} \left((s+t)\mu^2 + \sigma^2 \right) - \frac{2}{3}\mu^2 \min(s, t) \right] \min(s, t)^2, \quad \forall s, t \in [0, T]. \quad (37)$$

In order to approximate the density of model (35), based on simulated trajectories, i.e., $X_t^\Delta = \{X_t^\Delta(\omega_1), X_t^\Delta(\omega_2), \dots, X_t^\Delta(\omega_M)\}$ at time $T = 5$, we will use function `density()`. In R (see left panel of Figure 18):

```
R> T <- 5
R> Ex <- function(t) x0 + 0 * t
R> Vx <- function(t) (1/3) * mu^2 * t^3 + (1/2) * sigma^2 * t^2
R> XT <- rsde3d(mod, at = T)$x
R> plot(density(XT))
R> curve(dnorm(x, mean = Ex(T), sd = sqrt(Vx(T))), add = TRUE, col = 2,
+   lty = 2)
R> legend("topleft", c("Estimate", "True"), inset = 0.01, col = 1:2,
+   lty = 1:2)
```

One can also estimate the auto-covariance function of the simulated trajectories of (35) with empirical function `COV()`, and we will graphically compare them with the exact values (37). For that one can use graphics functions for 2D available in R (`image`, `contour` or `filled.contour`, for more details see `demo(image)`). For example, we use the package **ggplot2** (Wickham 2009), which creates nice graphics with many options to choose from (for more detail see <https://ggplot2-book.org/>); see the right panel of Figure 18.

Moreover, if we want to test the normality of the distribution, i.e., if a sample is normally distributed or not, we use the Shapiro and Wilk (1965) normality test. The null hypothesis of this test follows a normal distribution, so if `p-value` < 0.1 , one concludes that the sample does not follow a normal distribution (Royston 1995). This then allows using some statistical tests if the null hypothesis is retained. In R we use function `shapiro.test()` available in base package **stats** (we can also use the Kolmogorov-Smirnov test with function `ks.test()`,

Figure 18: Left: Kernel density obtained from a 5000 realizations of (35) and true density at time $T = 5$, with initial condition $t_0 = 0$ and $x_0 = 10$. Right: Empirical variance-covariance of model (35), black lines represent exact values (R code: Supplementary materials). $\mu = 0.75$ and $\sigma = 0.5$.

it allows testing if a sample follows any given distribution), and we define the function of statistics of interest (i.e., approximation of W-statistic and p value of the test) by `test.fun`. Finally, we apply the parallelized MC at $T = 5$ to check the normality test.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> test.fun <- function(data, i) {
+   d <- data[i, ]
+   res <- stats::shapiro.test(d$x)
+   return(c(res$statistic[[1]], res$p.value))
+ }
R> mc.test <- MCM.sde(mod, statistic = test.fun, R = 100, time = 5,
+   parallel = "snow", ncpus = 4, names = c("W-statistic", "p-value"))
R> mc.test$MC
```

	Estimate	Std.Error	CI(2.5 % , 97.5 %)
W-statistic	0.99955	0.00002	(0.99951 , 0.99959)
p-value	0.42369	0.02906	(0.36673 , 0.48065)

The MC results return a non-significant $p\text{-value} = 0.42369 \pm 0.02906$, so we cannot reject the null hypothesis, i.e., the model (35) is normally distributed at a fixed time $T = 5$.

Now we want to approximate the parameters of the distribution of (35) at fixed time $T = 5$; we will use the `fitdistr()` function available in package **MASS** (maximum likelihood fitting of univariate distributions, see `help("fitdistr")`). As previously, we define the function of statistics of interest, i.e., the estimate of parameters of the normal distribution by `estimate.fun` (in fact of any distribution, see `help("Distributions")`), and we use easily function `MCM.sde()`.

```

R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> estimate.fun <- function(data, i) {
+   d <- data[i, ]
+   est <- MASS::fitdistr(d$x, densfun = "normal")
+   return(c(est$estimate[[1]], est$estimate[[2]]))
+ }
R> mc.estimate <- MCM.sde(mod, statistic = estimate.fun, R = 100, time = 5,
+   ncpus = 4, parallel = "snow", exact = list(mean = Ex(5),
+   sd = sqrt(Vx(5))))
R> mc.estimate$MC

```

	Exact	Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
mean	10.0000	10.0045	0.00445	0.00746	0.07437	(9.98983 , 10.01907)
sd	5.1539	5.1433	-0.01055	0.00506	0.05145	(5.13341 , 5.15325)

4. Application to European options pricing

In order to illustrate how the R package **Sim.DiffProc** can be used in practical situations, we give an overview of using the package to approximate the prices of Black-Scholes (1D) and Heston models (2D) European call and put options, with application to real data sets.

4.1. Black-Scholes European options pricing

The price of a European option $C(s, x)$ at time s is the discounted value at the risk-free rate of interest r with maturity T , that is (e.g., see [Iacus 2011](#), Chapter 6):

$$C(s, x) = e^{-r(T-s)} \mathbb{E} f(X_T^{s,x}), \quad (38)$$

where $X_T^{s,x}$ is the price of the underlying at maturity T . The payoff function $f(x)$ for European call and put options with strike price K , is given by:

$$f(x) = \begin{cases} \max(0, x - K), & \text{call option,} \\ \max(0, K - x), & \text{put option.} \end{cases} \quad (39)$$

We can easily approximate (38) using the formulation (9).

$$\hat{C}(s, x) = \frac{1}{MR} e^{-r(T-s)} \sum_{i=1}^R \sum_{j=1}^M \begin{cases} \max(0, X_T^\Delta(\omega_{i,j}) - K), & \text{call option,} \\ \max(0, K - X_T^\Delta(\omega_{i,j})), & \text{put option.} \end{cases} \quad (40)$$

We will use the function `GBSOption()` available in package **fOptions** ([Wuertz, Setz, and Chalabi 2017](#)). This function is handy to calculate the exact formulas for the call and put options of the generalized Black-Scholes model. So, for example, we calculate the price of a contract with the following parameters: the price of the underlying $S_0 = 100$, the strike price $K = 150$, the time to maturity $T = 1$, the interest rate $r = 0.1$, the cost of carrying term $\mu = 0.75$, and the volatility $\sigma = 0.25$.

```

R> library("fOptions")
R> mu <- 0.75
R> sigma <- 0.25
R> S0 <- 100
R> K <- 150
R> r <- 0.1
R> T <- 1
R> call_true <- GBSOption(TypeFlag = "c", S = S0, X = K, Time = T, r = r,
+   b = mu, sigma = sigma)@price
R> put_true <- GBSOption(TypeFlag = "p", S = S0, X = K, Time = T, r = r,
+   b = mu, sigma = sigma)@price
R> c(call_true, put_true)

```

```
[1] 57.3707  1.5422
```

In order to evaluate the option price with the Monte Carlo method, firstly we need to be able to simulate M paths of the Black-Scholes model. For that we use the `snssde1d()` function, which is extremely simple.

```

R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> fx <- expression(mu * x)
R> gx <- expression(sigma * x)
R> BS <- snssde1d(drift = fx, diffusion = gx, x0 = S0, M = 5000)

```

Secondly, it is quite easy to implement the quantity of interest (40) in R. Let us write a function `EuroOpt.fun` to compute the call and put option price of the Black-Scholes model (or indeed of any process X_t).

```

R> EuroOpt.fun <- function(data, i, r, K, T, s) {
+   d <- data[i, ]
+   call <- mean(pmax(0, d - K)) * exp(-r * (T - s))
+   put <- mean(pmax(0, K - d)) * exp(-r * (T - s))
+   return(c(call, put))
+ }

```

Now we compare the Monte Carlo approximation of the price and the exact price of a European call/put options already obtained by function `GBSOption()`.

```

R> mcm.EuroOpt <- MCM.sde(model = BS, statistic = EuroOpt.fun, R = 100,
+   parallel = "snow", ncpus = 4, time = T, exact = list(Call = call_true,
+   Put = put_true), r = r, K = K, T = T, s = 0)
R> mcm.EuroOpt$MC

```

	Exact Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
Call	57.3707	57.3784	0.00770	0.06683	0.66503 (57.24741 , 57.50937)
Put	1.5422	1.5249	-0.01737	0.00712	0.07296 (1.5109 , 1.5388)

Figure 19: Black-Scholes European options pricing of call and put in $t \in [0, 1]$ (R code: Supplementary materials).

As we see, the parallelized Monte Carlo method gives a reasonable approximation of the price of a European call/put option to maturity $T = 1$. The resulting plots of simulation in time $t \in [0, 1]$ are given in Figure 19, where there is good agreement between the results of Monte Carlo and exact values. So, when the formula of the price is not available in explicit form, for example, the underlying process is not a Black-Scholes model (for example model (16)), the `MCM.sde()` function is useful to evaluate the European options pricing of any model.

4.2. European options pricing under stochastic volatility: Heston model

Under the Heston (1993) model, the stock price X_t follows a Black-Scholes type stochastic process, and its stochastic variance Y_t follows a Cox, Ingersoll, and Ross (1985) process. Hence, the Heston model is described by the bivariate system of SDEs:

$$\begin{cases} dX_t = \mu X_t dt + \sqrt{Y_t} X_t dB_{1,t} \\ dY_t = \nu (\theta - Y_t) dt + \sigma \sqrt{Y_t} dB_{2,t} \end{cases} \quad (41)$$

where the two Wiener processes are correlated, that is $E(dB_{1,t}dB_{2,t}) = \rho dt$. In the second equation, volatility is mean-reverting to its long-run level θ with speed ν ; σ is the volatility of volatility. If the condition $2\nu\theta > \sigma^2$ holds, then the drift is sufficiently large for the variance process Y_t to be guaranteed positive and not reach zero (known as the Feller condition). The correlation is an important part of the model (41). If the correlation is negative, volatility increases as the underlying asset falls and volatility decreases as the underlying rises. In particular, for the Monte Carlo simulation, it is more convenient to first generate independent Wiener processes; we can easily translate these correlated Wiener processes into uncorrelated ones by Cholesky decomposition which gives the decomposition of $(B_{i,t})_{i=1,2}$ onto the orthogonal basis $(W_{i,t})_{i=1,2}$:

$$\begin{cases} dB_{1,t} = dW_{1,t} \\ dB_{2,t} = \rho dW_{1,t} + \sqrt{1 - \rho^2} dW_{2,t} \end{cases}$$

Note that $E(dB_{1,t}dB_{2,t}) = \rho dt$. Moreover, since $(W_{i,t})_{i=1,2}$ are two independent Wiener processes: $E(dW_{1,t}dW_{2,t}) = 0$ and $E(dW_{i,t}^2)_{i=1,2} = dt$. We can rewrite the Heston model (41) in the orthogonal basis as:

$$(41) \Leftrightarrow \begin{cases} dX_t = \mu X_t dt + \sqrt{Y_t} X_t dW_{1,t} \\ dY_t = \nu (\theta - Y_t) dt + \sigma \sqrt{Y_t} \left(\rho dW_{1,t} + \sqrt{1 - \rho^2} dW_{2,t} \right) \end{cases} \quad (42)$$

Now let us use the functions `callHestoncf()` and `putCallParity()` available in R package **NMOF** (Gilli, Maringer, and Schumann 2019). These functions are useful for calculating the price of a European call and put with the characteristic function of the log price in the Heston model. In R we calculate the value of the call and put under the following parameters: the price of the underlying $x_0 = 100$, initial variance $y_0 = 0.3$, the strike price $K = 120$, the time to maturity $T = 1$, risk-free rate $\mu = 0.15$, dividend yield $q = 0$, long run variance $\theta = 0.25$, speed of mean reversion $\nu = 2$, volatility of volatility $\sigma = 0.2$ and finally the correlation $\rho = 0.75$.

```
R> library("NMOF")
R> x0 <- 100
R> y0 <- 0.3
R> rho <- 0.75
R> K <- 120
R> T <- 1
R> mu <- 0.15
R> q <- 0
R> theta <- 0.25
R> nu <- 2
R> sigma <- 0.2
R> callHMcF <- callHestoncf(S = x0, X = K, tau = T, r = mu, q = q, v0 = y0,
+   vT = theta, rho = rho, k = nu, sigma = sigma)
R> putHMcF <- putCallParity(what = "put", call = callHMcF, S = x0, X = K,
+   tau = T, r = mu, q = q)
R> c(callHMcF, putHMcF)
```

```
[1] 19.524 22.809
```

Like before, to evaluate the price of a European call and put with the Monte Carlo method, firstly we need to be able to simulate M paths of the Heston model (41). For that we use function `snssde2d()`⁹, where we define a new argument `corr`¹⁰ using a symmetric positive-definite square matrix (two dimensions), which contains the correlation structure between the Wiener processes $B_{1,t}$ and $B_{2,t}$; i.e., $E(dB_{1,t}dB_{2,t}) = \rho dt$.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> f <- expression(mu * x, nu * (theta - y))
```

⁹In the correlation case, currently, we can use only the Euler-Maruyama and Milstein scheme.

¹⁰We can define the correlation structure by a variance-covariance matrix since we use function `cov2cor()` from the base package **stats**, which transforms a covariance matrix into the corresponding correlation matrix efficiently. For the moments equations of the Heston model obtained by function `MEM.sde()`, see Appendix D.1.

```
R> g <- expression(sqrt(y) * x, sigma * sqrt(y))
R> Sigma <- matrix(c(1, 0.75, 0.75, 1), nrow = 2, ncol = 2)
R> HM <- snssde2d(drift = f, diffusion = g, corr = Sigma, x0 = c(x0, y0),
+   M = 5000)
```

Let us write a new function `EuroOpt.fun2d` to compute the call and put options price of the Heston model (or indeed of any bivariate process).

```
R> EuroOpt.fun2d <- function(data, i, r, K, T) {
+   d <- data[i, ]
+   call <- mean(pmax(0, d$x - K)) * exp(-r * T)
+   put <- mean(pmax(0, K - d$x)) * exp(-r * T)
+   return(c(call, put))
+ }
```

Now we compare the parallel Monte Carlo approximation of the price of a European call and put with that already obtained by the characteristic function,

```
R> mc.HM <- MCM.sde(model = HM, statistic = EuroOpt.fun2d, R = 100, exact =
+   list("CallHM" = callHMcf, "PutHM" = putHMcf), parallel = "snow",
+   ncpus = 4, r = mu, K = K, T = T)
R> mc.HM
```

Itô Sde 2D:

```
| dX(t) = mu * X(t) * dt + sqrt(Y(t)) * X(t) * dB1(t)
| dY(t) = nu * (theta - Y(t)) * dt + sigma * sqrt(Y(t)) * dB2(t)
| t in [0,1] with mesh equal to 0.001
| Correlation structure:
      1.00 0.75
      0.75 1.00
```

PMCM Based on 100 Batches with 5000-Realisations at time 1:

	Exact	Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
CallHM	19.524	19.501	-0.02339	0.06447	0.64189	(19.37461 , 19.62733)
PutHM	22.809	22.811	0.00137	0.03422	0.34052	(22.74362 , 22.87776)

As we can see, the price of the call (put) provided by the characteristic function of the Heston model and the parallel Monte Carlo approach are quite close (the relative difference is 0.117% for call and 0.008% for put). In Figure 20, we can see the importance of the correlation ρ , where there is a dependence between the call (put) price and volatility σ . Indeed if the correlation is negative then the call (put) price increases with the fall in volatility.

4.3. European options pricing approximation to real data sets

Now, if we look at the market price of, e.g., a call or put option at a given fixed time, we can compare the price predicted from real data by the exact Black and Scholes formula with (40) (i.e., compare the results of `GBSOption()` vs. `MCM.sde()` using real data). We are

Figure 20: The dependence of call (put) options price on the correlation and volatility. Heston model with: $X_0 = 100$, $Y_0 = 0.3$, $K = 120$, $T = 1$, $\mu = 0.15$, $\nu = 2$ and $\theta = 0.25$ (R code: Supplementary materials).

therefore interested in the period ending in February 2019, and we get the 1000 closing levels of S&P500¹¹ prior to that date. For that purpose, we use function `get.hist.quote()` from the `tseries` package (Trapletti and Hornik 2019).

```
R> library("tseries")
R> S <- get.hist.quote(start = "2015-03-05", end = "2019-02-24",
+   instrument = "^gspc", provider = "yahoo", quote = "Close")
R> tail(S)
```

```
      Close
2019-02-14 2745.73
2019-02-15 2775.60
2019-02-19 2779.76
2019-02-20 2784.70
2019-02-21 2774.88
2019-02-22 2792.67
```

We then plot the data with function `chartSeries()` from the `quantmod` package (Ryan, Ulrich, Thielen, Teetor, and Bronder 2020), see Figure 21.

```
R> library("quantmod")
R> chartSeries(S, theme = chartTheme("white"), TA = NULL, name = "S&P 500")
```

Now we calculate the annual interest rate r and historical volatility σ of 1000 closing levels of S&P500 by two techniques. We consider $\Delta t = 1/252$ (because we use daily data) and the

¹¹The Standard and Poor's 500 is an index of the 500 most dominant companies listed on the US stock market. There are also about twenty foreign companies. This is the benchmark par excellence.

Figure 21: Close values of the S&P 500.

expiry date was 22 February 2019 which corresponds to 1000 days, thus we set $T = 1000\Delta t$. We create the log-returns using `diff(log(S))` and then we estimate the parameters of the Black-Scholes model (see [Iacus 2011](#), Chapter 5). In R we obtain immediately the following estimator:

```
R> Dt <- 1/252
R> X <- diff(log(S))
R> sigma.hat <- sqrt(var(X, na.rm = TRUE)/Dt)[1, 1]
R> r.hat <- (mean(X, na.rm = TRUE)/Dt) + 0.5 * sigma.hat^2
R> c(r.hat, sigma.hat)
```

```
[1] 0.08125 0.13762
```

As second approach¹², we use maximum likelihood estimation based on the true likelihood of the Black-Scholes model. This is accomplished in R using the `mle()` function from the base package `stats4`, and the `dcBS()` function (the true density function of the Black-Scholes model) available in the package `sde` ([Iacus 2016](#)). In this case, the R code is quite easy to implement:

```
R> library("stats4")
R> X <- ts(sapply(1:nrow(S), function(i) S[, "Close"][[i]]), start = 0,
+       deltat = Dt)
R> BS.lik <- function(r.hat, sigma.hat) {
+   n <- length(X)
```

¹²We can use function `qmle()` “quasi-maximum likelihood estimation for the SDEs” from the `yuima` package (see [Iacus and Yoshida 2018](#), Chapter 2).

```

+   Dt <- deltat(X)
+   -sum(sde::dcBS(x = X[2:n], Dt, x0 = X[1:(n-1)],
+     theta = c(r.hat, sigma.hat), log = TRUE))
+ }
R> fit.BS <- stats4::mle(BS.lik, start = list(r.hat = 1, sigma.hat = 1),
+   method = "L-BFGS-B", lower = c(-Inf, 0.01))
R> summary(fit.BS)

```

Maximum likelihood estimation

Call:

```
stats4::mle(minuslogl = BS.lik, start = list(r.hat = 1, sigma.hat = 1),
method = "L-BFGS-B", lower = c(-Inf, 0.01))
```

Coefficients:

	Estimate	Std. Error
r.hat	0.08125	0.06908
sigma.hat	0.13756	0.00307

-2 log L: 8850

It is reasonable at this point to extract the model coefficients with function `coef()` and get the confidence intervals with function `confint()` as follows:

```

R> coef(fit.BS)

   r.hat  sigma.hat 
0.08125  0.13756 

R> confint(fit.BS, level = 0.95)

```

Profiling...

	2.5 %	97.5 %
r.hat	-0.05425	0.21681
sigma.hat	0.13173	0.14381

We can now calculate the exact price of call and put options of 1000 closing levels of S&P500 by identifying all quantities, the strike price $K = 2600$, $S_0 = 2101.04$, the expiry date $T = 1000\Delta t$ which corresponds to 22 February 2019. The estimated annual interest rate was $\hat{r} = 0.08125$ and historical volatility $\hat{\sigma} = 0.13756$. In order to obtain the standard formulas, one has to put `b = r.hat` in function `GBSOption()`.

```

R> S0 <- S[[1]]
R> K <- 2600
R> T <- nrow(S) * (1/252)
R> s <- 0
R> r.hat <- coef(fit.BS)[[1]]

```

```
R> sigma.hat <- coef(fit.BS)[[2]]
R> call_BS <- GBSOption(TypeFlag = "c", S = S0, X = K, Time = T, r = r.hat,
+   b = r.hat, sigma = sigma.hat)@price
R> put_BS <- GBSOption(TypeFlag = "p", S = S0, X = K, Time = T, r = r.hat,
+   b = r.hat, sigma = sigma.hat)@price
R> c(call_BS, put_BS)

[1] 342.83 125.24
```

As in the previous example we simulate M paths of the Black-Scholes model, with the two estimated parameters of the model $\hat{r}$ and $\hat{\sigma}$, and $X_0 = S_0$, $\Delta t = 1/252$.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> fx <- expression(r.hat * x)
R> gx <- expression(sigma.hat * x)
R> BS <- snssde1d(drift = fx, diffusion = gx, x0 = S0, Dt = 1/252, M = 5000)
```

We apply the parallelized Monte Carlo method on the previous function `EuroOpt.fun`. Finally, we find the price of the call and put options of 1000 closing levels of S&P500.

```
R> mcm.EuroOpt <- MCM.sde(BS, statistic = EuroOpt.fun, R = 500,
+   parallel = "snow", ncpus = 4, time = T, exact = list(Call = call_BS,
+   Put = put_BS), r = r.hat, K = K, T = T, s = s)
R> mcm.EuroOpt$MC
```

	Exact	Estimate	Bias	Std.Error	RMSE	CI(2.5 % , 97.5 %)
Call	342.83	342.88	0.04737	0.28666	6.4036	(342.31841 , 343.44209)
Put	125.24	125.26	0.02052	0.12779	2.8548	(125.00936 , 125.51028)

We obtain an excellent agreement between the exact values and the result obtained by the parallelized Monte Carlo method (see the left panel of Figure 22). Thus, in practice when the explicit formulas for calculating the price of the options do not exist, it is always possible to rely on the formula (40).

As a demonstration, if we want to approximate the time τ (if it exists before the time to maturity T) at which the price of call equals the price of put with a strike price $K = 2600$, i.e., $\text{Call}(K, \tau) = \text{Put}(K, \tau)$. The necessary R code to perform this approximation is then:

```
R> call_MC <- sapply(time(BS), function(i)
+   mean(pmax(0, rsde1d(BS, at = i) - K)) * exp(-r.hat * (i - 0)))
R> put_MC <- sapply(time(BS), function(i)
+   mean(pmax(0, K - rsde1d(BS, at = i))) * exp(-r.hat * (i - 0)))
R> call_fun <- stats::approxfun(time(BS), call_MC)
R> put_fun <- stats::approxfun(time(BS), put_MC)
R> tau <- stats::uniroot(f = function(x) (call_fun(x) - put_fun(x)),
+   lower = 0, upper = TRUE, tol = .Machine$double.eps)$root
R> tau
```

```
[1] 2.608
```

Figure 22: Left: The price of call and put options of 1000 closing levels of S&P500 with strike price $K = 2600$. The strike price K as a function of the time τ at which the price of call equals the price of put (between 05 March 2015 and 22 February 2019).

K	τ	Price of Call/Put	Corresponds to the date
2200	0.57495	87.93	2015-09-30
2300	1.11792	122.96	2016-04-18
2400	1.65086	148.75	2016-10-27
2500	2.13039	169.11	2017-04-21
2600	2.60786	187.23	2017-10-12
2700	3.06512	202.19	2018-03-29
2800	3.53046	217.44	2018-09-14
2900	3.94199	230.74	2019-02-12

Table 7: The sensitivity of the strike price K with respect to the time τ at which the price of call equals the price of put, i.e., $\text{Call}(K, \tau) = \text{Put}(K, \tau)$ (R code: Supplementary materials).

Indeed, $\tau = 2.608$ then corresponds to 12 October 2017. Moreover, this is shown in the left panel of Figure 22. The right panel of Figure 22 shows that the strike price K is an increasing function of the time τ at which the price of call equals the price of put. We summarize the numerical results obtained in Table 7.

```
R> time(S)[which.min(time(BS) <= tau)]
```

```
[1] "2017-10-12"
```

5. Miscellaneous tools for the Sim.DiffProc environment

A simple utility writing available in function `TEX.sde()` produces the related L^AT_EX codes (tables and math expressions) using the **Sim.DiffProc** package, which can be copied and pasted in a scientific article. Note that there are already several tools available for the R environment for

generating L^AT_EX code. Many of these tools can be found in the CRAN Task View on *Reproducible Research* at <https://CRAN.R-project.org/view=ReproducibleResearch> (Blischak and Hill 2020). In the following we present the new tools for constructing L^AT_EX tables and mathematical expressions with the **Sim.DiffProc** package. The R function call consists of:

```
TEX.sde(object, ...)
```

The input parameters for `TEX.sde()` are:

- `object`: an object of class ‘`MCM.sde`’ or ‘`MEM.sde`’; or an R vector of expressions of SDEs, i.e., drift and diffusion coefficients.
- `...`: if `object` of class ‘`MCM.sde`’, arguments to be passed to `kable()` function available in package **knitr** (Xie 2015).

5.1. L^AT_EX table for objects of class ‘`MCM.sde`’

The Monte Carlo results of the ‘`MCM.sde`’ class can be presented in terms of L^AT_EX tables. Our goal is to make the `TEX.sde()` method modular and flexible. We thus leave tasks, such as formatting of the table, possible to be modified by the user. For this, we are using function `kable()` (for more details see `help("kable")`) available in package **knitr** (Xie 2015), which converts an output of ‘`MCM.sde`’ class into the corresponding L^AT_EX code for a table. In Section 3.1 we have assigned the outputs of the Monte Carlo method to an object called `mcm.mod`. In R, we create the L^AT_EX table for this object using the following code:

```
R> TEX.sde(object = mcm.mod, booktabs = TRUE, align = "r", caption =
+   paste("\LaTeX~table for output results of class \class{MCM.sde} ",
+   "generated by \code{TEX.sde()} method.")
```

```
%% LaTeX table generated in R 4.0.2 by TEX.sde() method
%% Copy and paste the following output in your LaTeX file
```

```
\begin{table}
\caption{\label{tab:}
  \LaTeX~table for output results of class \class{MCM.sde}
  generated by \code{TEX.sde()} method.}
\centering
\begin{tabular}[t]{l}{lrrrr}
\toprule
& Estimate & Std.Error & CI( 2.5 \% , 97.5 \% )\\
\midrule
$m(t)$ & 2.00055 & 0.00039 & ( 1.99979 , 2.00131 )\\
$S(t)$ & 0.06551 & 0.00016 & ( 0.0652 , 0.06582 )\\
\bottomrule
\end{tabular}
\end{table}
```

This L^AT_EX code can be included in L^AT_EX documents. If we use `booktabs = TRUE` in the previous function, the L^AT_EX add-on package **booktabs** must be loaded in the `.tex` document.

	Estimate	Std.Error	CI(2.5 % , 97.5 %)
$m(t)$	2.00055	0.00039	(1.99979 , 2.00131)
$S(t)$	0.06551	0.00016	(0.0652 , 0.06582)

Table 8: $\text{\LaTeX}$ table for output results of class ‘MCM.sde’ generated by `TEX.sde()` method.

Table 8 shows the results of applying our `TEX.sde()` method to an object of class ‘MCM.sde’. If the results of the simulation become complicated enough, creating $\text{\LaTeX}$ tables from R significantly reduces the work, especially if the simulation study needs to be repeated due to bug fixes, enhancements, or modified implementation.

5.2. $\text{\LaTeX}$ mathematics for objects of class ‘MEM.sde’

Since many reports are produced using R and $\text{\LaTeX}$, it is very useful to be able to automatically convert mathematical expressions from one language to another. In the following, our goal is to automatically convert an R expression for an object of class ‘MEM.sde’ into its appropriate $\text{\LaTeX}$ representation. We will use the R codes available in the book of Wickham (2015, p. 320–328), with some modifications, improvements and adaptation with the **Sim.DiffProc** package. To give some concrete examples, we want to automatically generate the $\text{\LaTeX}$ code appropriate for the moment equations obtained from the model (23) using method `TEX.sde()`.

```
R> f <- expression(1/mu * (theta - x), x)
R> g <- expression(sqrt(sigma), 0)
R> mem.oui <- MEM.sde(drift = f, diffusion = g)
R> TEX.sde(object = mem.oui)
```

```
%% LaTeX equation generated in R 4.0.2 by TEX.sde() method
%% Copy and paste the following output in your LaTeX file
```

```
\begin{equation}\label{eq:}
\begin{cases}
\begin{split}
\frac{d}{dt} m_{\{1\}}(t) \sim &= \frac{\left(\theta - m_{\{1\}}(t)\right)}{\mu} \backslash \\
\frac{d}{dt} m_{\{2\}}(t) \sim &= m_{\{1\}}(t) \backslash \backslash \\
\frac{d}{dt} S_{\{1\}}(t) \sim &= \sigma - 2 \backslash, \left(\frac{S_{\{1\}}(t)}{\mu}\right) \backslash \backslash \\
\frac{d}{dt} S_{\{2\}}(t) \sim &= 2 \backslash, C_{\{12\}}(t) \backslash \backslash \\
\frac{d}{dt} C_{\{12\}}(t) &= S_{\{1\}}(t) - \frac{C_{\{12\}}(t)}{\mu}
\end{split}
\end{cases}
\end{equation}
```

This can be typed with $\text{\LaTeX}$ to produce a system (44–45). For another example of a more complex system such as (32), we have assigned the output of the moment equations to an object called `mem.mod3d` in Section 3.3, and we easily create the $\text{\LaTeX}$ code for this object by simply running the following command:

```
R> TEX.sde(object = mem.mod3d)
```

The output of the previous code can be copied and pasted into a `.tex` document to produce the system (43). Note that it is obvious that the $\text{\LaTeX}$ package **amsmath** must also be loaded in the `.tex` document.

$$\left\{ \begin{array}{l} \frac{d}{dt}m_1(t) = \mu_1 (1 - 0.5 S_3(t)) \cos(\mu_2 + m_3(t)) \\ \frac{d}{dt}m_2(t) = \mu_1 (1 - 0.5 S_3(t)) \sin(\mu_2 + m_3(t)) \\ \frac{d}{dt}m_3(t) = -\frac{m_3(t)}{\nu} \\ \frac{d}{dt}S_1(t) = \sigma_1 - 2 C_{13}(t) \mu_1 \sin(\mu_2 + m_3(t)) \\ \frac{d}{dt}S_2(t) = 2 C_{23}(t) \mu_1 \cos(\mu_2 + m_3(t)) + \sigma_2 \\ \frac{d}{dt}S_3(t) = -2 \frac{S_3(t)}{\nu} \\ \frac{d}{dt}C_{12}(t) = \mu_1 (C_{13}(t) \cos(\mu_2 + m_3(t)) - C_{23}(t) \sin(\mu_2 + m_3(t))) + \sqrt{\sigma_1} \sqrt{\sigma_2} \\ \frac{d}{dt}C_{13}(t) = -\mu_1 S_3(t) \sin(\mu_2 + m_3(t)) - \frac{C_{13}(t)}{\nu} \\ \frac{d}{dt}C_{23}(t) = \mu_1 S_3(t) \cos(\mu_2 + m_3(t)) - \frac{C_{23}(t)}{\nu} \end{array} \right. \quad (43)$$

5.3. $\text{\LaTeX}$ mathematics for an R expression of SDEs

In this section, we will convert the R expressions of an SDE, i.e., drift and diffusion coefficients into their $\text{\LaTeX}$ mathematical equivalents with the same procedures as previous. For instance for the drift and diffusion of (32):

```
R> f <- expression(mu1 * cos(z+mu2), mu1 * sin(z + mu2), -z/nu)
R> g <- expression(sqrt(sigma1), sqrt(sigma2), 0)
R> TEX.sde(object = c(drift = f, diffusion = g))
```

```
%% LaTeX equation generated in R 4.0.2 by TEX.sde() method
%% Copy and paste the following output in your LaTeX file
```

```
\begin{equation}\label{eq:}
\begin{cases}
\begin{aligned}
dX_{\{t\}} &= \mu_{\{1\}} \backslash, \backslash\cos(Z_{\{t\}}+\mu_{\{2\}})\backslash:dt + \sqrt{\sigma_{\{1\}}}\backslash:dW_{\{1, t\}}\backslash\backslash \\
dY_{\{t\}} &= \mu_{\{1\}} \backslash, \backslash\sin(Z_{\{t\}}+\mu_{\{2\}})\backslash:dt + \sqrt{\sigma_{\{2\}}}\backslash:dW_{\{2, t\}}\backslash\backslash \\
dZ_{\{t\}} &= \frac{-Z_{\{t\}}}{\nu} \backslash:dt + 0 \backslash:dW_{\{3, t\}}
\end{aligned}
\end{cases}
\end{equation}
```

This in $\text{\LaTeX}$ will create the system (32). Another much more sophisticated example is given by:

```

R> f <- expression((alpha * x * (1 - x/beta) - delta * x^2 *
+   y/(kappa + x^2)), (gamma * x^2 * y / (kappa + x^2) - mu * y^2))
R> g <- expression(sqrt(sigma1) * x * (1 - y), abs(sigma2) * y * (1 - x))
R> TEX.sde(object = c(drift = f, diffusion = g))

```

$$\begin{cases} dX_t = \left(\alpha X_t \left(1 - \frac{X_t}{\beta} \right) - \frac{\delta X_t^2 Y_t}{(\kappa + X_t^2)} \right) dt + \sqrt{\sigma_1} X_t (1 - Y_t) dW_{1,t} \\ dY_t = \left(\frac{\gamma X_t^2 Y_t}{(\kappa + X_t^2)} - \mu Y_t^2 \right) dt + |\sigma_2| Y_t (1 - X_t) dW_{2,t} \end{cases}$$

6. Discussion

In this work we have presented the most used methods in stochastic dynamics systems, analyzing in detail the moment equation method and Monte Carlo simulation (version parallel). The MCS approach is very general and works in almost all circumstances especially when the simulation mesh is very small (under assumption 2 “high frequency”). For high-dimensional functionals, it is sometimes the only method of obtaining a solution for real engineering problems, and it is a powerful tool to approximate an interesting quantity in an empirical manner, especially when we do not have an analytic form of a solution. Besides being more efficient than analytical-based approaches, it has the advantages that the tools of deterministic analysis can be fully exploited. As a drawback, the computational costs required are often very high. The computational efforts increase with the dimension and the complexity of the nonlinear structural models. The cost of this generality is the large sample sizes (dependence between simulated samples) required to obtain reasonably accurate estimates. The parallel RNG support is perhaps the most interesting and important feature of parallel computing, which allows us to avoid the problem of lack of independence in the samples from multiple RNG.

The moment equation method is advantageous for analyzing the effects of noise in stochastic systems. These moments implicitly include information on relative fluctuations $R(t)$. Sometimes, it provides explicit results (generally in the linear case); otherwise, we can quickly solve these equations by standard numerical methods. This avoids greedy calculations of time associated with the need to average over many batches of MCS. The moments obtained compared favorably with those obtained by MCS when the noise was small. Stochastic analysis always requires some approximation treatment. The moment equation method provides here a concise picture of a stochastic transient. Generally in the linear case we find explicit results, but in the nonlinear case it is very difficult, as we have seen in Section 3.3. It furthermore enables an analytical study of the transient characteristics so that noise effects could be explicitly and systematically compared. Results derived from Section 3.2 demonstrate that there is no effect of the type of stochastic model (Itô or Stratonovich) on the relative fluctuation. We have confirmed these results analytically in Section 3.1, where we easily find $R^{\text{mod1}}(t) = R^{\text{mod2}}(t)$. Since the response of a dynamic system to noise is nonlinear, noisy systems could reveal more complex and exciting phenomena than deterministic systems. Further study of the effects of noise on nonlinear systems is needed.

We have demonstrated how the **Sim.DiffProc** package may be used to simulate, estimate and analyze the dynamics of scalar and multivariate systems of Itô and Stratonovich SDEs driven

by uncorrelated and correlated Wiener processes. Moreover, we highlighted the performance of symbolic and numerical computation of the package, through several examples and applications. Two central components of the package are the approximation of a quantity of interest for a stochastic model with parallelized MC, and an analytical approach for the determination of the first and central second-order moments, which are highly useful for studying stochastic dynamics. The R language is also a powerful high-level programming language that is well suited for expressing complex mathematical computations, as stochastic analysis of SDEs. There can be a tension between language features that support interactive use, and features that support the development of reliable, high performance code. $\text{\LaTeX}$ mathematics are complex. Fortunately, they are well documented. That said, they have a fairly simple structure, which could be used as a quick starting point for the more complex editing of a scientific article using R and $\text{\LaTeX}$. We hope that the package presented here and the updated survey on the subject might be of help for practitioners and researchers in the field who might want to implement new methods and ideas using R package **Sim.DiffProc**. One of the most significant limitations of the latest version of the package (version 4.8), is that jump-type stochastic models are not developed in this version, making modeling limited to a class of SDEs. Future efforts will work toward extending the package to more complex systems, driven by fractional Brownian motion and Lévy processes, with jumps specified.

Acknowledgments

We thank the editors and the anonymous referees for their thorough review and highly appreciate the comments and suggestions which significantly contributed to increasing the quality of our article.

The work described in this paper was supported by the project entitled by: “Modeling, Actuarial Risk, Insurance, Financial, Premium Principle, Simulation” under the number “C00L03UN160420150004”, within the framework National Commission of Evaluation and Futurology of the University Research, University of Science and Technology Houari Boumediene, Ministry for Higher Education and Scientific Research, Algeria.

References

- Adler D, Murdoch D (2020). **rgl: 3D Visualization Using OpenGL**. R package version 0.100.54, URL <https://CRAN.R-project.org/package=rgl>.
- Aït-Sahalia Y (1996). “Testing Continuous-Time Models of the Spot Interest Rate.” *The Review of Financial Studies*, **9**(2), 385–426.
- Aït-Sahalia Y (1999). “Transition Densities for Interest Rate and Other Nonlinear Diffusions.” *The Journal of Finance*, **54**(4), 1361–1395. doi:10.1111/0022-1082.00149.
- Aït-Sahalia Y (2002). “Maximum Likelihood Estimation of Discretely Sampled Diffusions: A Closed-Form Approximation Approach.” *Econometrica*, **70**(1), 223–262. doi:10.1111/1468-0262.00274.

- Alibrandi U, Ricciardi G (2012). “Stochastic Methods in Nonlinear Structural Dynamics.” In *Nondeterministic Mechanics*, pp. 3–60. Springer-Verlag. doi:10.1007/978-3-7091-1306-6_1.
- Allen E (2007). *Modeling with Itô Stochastic Differential Equations*, volume 22 of *Mathematical Modelling: Theory and Applications*. Springer-Verlag. doi:10.1007/978-1-4020-5953-7.
- Bezanson J, Edelman A, Karpinski S, Shah VB (2017). “Julia: A Fresh Approach to Numerical Computing.” *SIAM Review*, **59**(1), 65–98. doi:10.1137/141000671.
- Blischak J, Hill A (2020). *CRAN Task View: Reproducible Research*. Version 2020-09-07, URL <https://CRAN.R-project.org/view=ReproducibleResearch>.
- Borchers HW (2019). **pracma**: *Practical Numerical Math Functions*. R package version 2.2.9, URL <https://CRAN.R-project.org/package=pracma>.
- Borchers HW (2020). *CRAN Task View: Numerical Mathematics*. Version 2020-05-25, URL <https://CRAN.R-project.org/view=NumericalMathematics>.
- Bover DCC (1978). “Moment Equation Methods for Nonlinear Stochastic Systems.” *Journal of Mathematical Analysis and Applications*, **65**(2), 306–320. doi:10.1016/0022-247x(78)90182-8.
- Brouste A, Fukasawa M, Hino H, Iacus SM, Kamatani K, Koike Y, Masuda H, Nomura R, Ogihara T, Shimuzu Y, Uchida M, Yoshida N (2014). “The YUIMA Project: A Computational Framework for Simulation and Inference of Stochastic Differential Equations.” *Journal of Statistical Software*, **57**(4), 1–51. doi:10.18637/jss.v057.i04.
- Burrage K, Burrage P, Higham DJ, Kloeden PE, Platen E (2006). “Comment on “Numerical Methods for Stochastic Differential Equations”.” *Physical Review E*, **74**, 068701. doi:10.1103/physreve.74.068701.
- Canty A, Ripley BD (2020). **boot**: *Bootstrap R (S-PLUS) Functions*. R package version 1.3-25, URL <https://CRAN.R-project.org/package=boot>.
- Chaussé P (2010). “Computing Generalized Method of Moments and Generalized Empirical Likelihood with R.” *Journal of Statistical Software*, **34**(11), 1–35. doi:10.18637/jss.v034.i11.
- Chen L (1996). “A Three-Factor Model of the Term Structure of Interest Rates.” In *Interest Rate Dynamics, Derivatives Pricing, and Risk Management*, pp. 1–36. Springer-Verlag. doi:10.1007/978-3-642-46825-4_1.
- Clausen A, Sokol S (2020). **Deriv**: *R-Based Symbolic Differentiation*. R package version 4.0.1, URL <https://CRAN.R-project.org/package=Deriv>.
- Conley TG, Hansen LP, Luttmer EGJ, Scheinkman JA (1997). “Short-Term Interest Rates as Subordinated Diffusions.” *The Review of Financial Studies*, **10**(3), 525–577.
- Cox JC, Ingersoll JE, Ross SA (1985). “A Theory of the Term Structure of Interest Rates.” *Econometrica*, **53**(2), 385–407. doi:10.2307/1911242.

- Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press. doi:[10.1017/cbo9780511802843](https://doi.org/10.1017/cbo9780511802843).
- Duong T (2007). “**ks**: Kernel Density Estimation and Kernel Discriminant Analysis for Multivariate Data in R.” *Journal of Statistical Software*, **21**(7), 1–16. doi:[10.18637/jss.v021.i07](https://doi.org/10.18637/jss.v021.i07).
- Duong T (2016). “Non-Parametric Smoothed Estimation of Multivariate Cumulative Distribution and Survival Functions, and Receiver Operating Characteristic Curves.” *Journal of the Korean Statistical Society*, **45**(1), 33–50. doi:[10.1016/j.jkss.2015.06.002](https://doi.org/10.1016/j.jkss.2015.06.002).
- Eddelbuettel D (2020). *CRAN Task View: High-Performance and Parallel Computing with R*. Version 2020-08-10, URL <https://CRAN.R-project.org/view=HighPerformanceComputing>.
- Eugster MJA, Knaus J, Porzelius C, Schmidberger M, Vicedo E (2011). “Hands-on Tutorial for Parallel Computing with R.” *Computational Statistics*, **26**(2), 219–239. doi:[10.1007/s00180-010-0206-4](https://doi.org/10.1007/s00180-010-0206-4).
- Fishman GS (1996). *Monte Carlo: Concepts, Algorithms, and Applications*. Springer-Verlag. doi:[10.1007/978-1-4757-2553-7](https://doi.org/10.1007/978-1-4757-2553-7).
- Freidlin MI, Wentzell AD (2012). *Random Perturbations of Dynamical Systems*, volume 260 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag. doi:[10.1007/978-3-642-25847-3](https://doi.org/10.1007/978-3-642-25847-3).
- Friedrich R, Haken H (1988). “Exact Stationary Probability Distribution for a Simple Model of a Nonequilibrium Phase Transition.” *Zeitschrift für Physik B Condensed Matter*, **71**(4), 515–517. doi:[10.1007/bf01313938](https://doi.org/10.1007/bf01313938).
- Friedrich R, Haken H (1992). “Nonequilibrium Phase Transition in a System with Chaotic Dynamics. The ABCDE Model.” *Physics Letters A*, **164**(3–4), 299–304. doi:[10.1016/0375-9601\(92\)91108-4](https://doi.org/10.1016/0375-9601(92)91108-4).
- Fuchs A (2013a). *Nonlinear Dynamics in Complex Systems: Theory and Applications for the Life-, Neuro- And Natural Sciences*. Springer-Verlag. doi:[10.1007/978-3-642-33552-5](https://doi.org/10.1007/978-3-642-33552-5).
- Fuchs C (2013b). *Inference for Diffusion Processes: With Applications in Life Sciences*. Springer-Verlag. doi:[10.1007/978-3-642-25969-2](https://doi.org/10.1007/978-3-642-25969-2).
- Gallant AR, Tauchen G (1998). “Reprojecting Partially Observed Systems with Application to Interest Rate Diffusions.” *Journal of the American Statistical Association*, **93**(441), 10–24. doi:[10.2307/2669598](https://doi.org/10.2307/2669598).
- Gaujoux R (2020). **doRNG**: Generic Reproducible Parallel Backend for foreach Loops. R package version 1.8.2, URL <https://CRAN.R-project.org/package=doRNG>.
- Genz A, Bretz F (2009). *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag. doi:[10.1007/978-3-642-01689-9](https://doi.org/10.1007/978-3-642-01689-9).

- Genz A, Bretz F, Miwa T, Mi X, Leisch F, Scheipl F, Hothorn T (2020). **mvtnorm**: *Multivariate Normal and tt Distributions*. R package version 1.1-1, URL <https://CRAN.R-project.org/package=mvtnorm>.
- Gikhman II, Skorokhod AV (1972). *Stochastic Differential Equations*, volume 72 of *Ergebnisse der Mathematik und ihrer Grenzgebiete. 2. Folge*. Springer-Verlag.
- Gikhman II, Skorokhod AV (2007). “Martingales and Stochastic Integrals.” In *The Theory of Stochastic Processes III*, Classics in Mathematics, pp. 1–112. Springer-Verlag. doi:10.1007/978-3-540-49941-1_1. Reprint of the 1974 Edition (Grundlehren der mathematischen Wissenschaften, Vol. 232).
- Gillespie DT (1996). “Exact Numerical Simulation of the Ornstein-Uhlenbeck Process and Its Integral.” *Physical Review E*, **54**, 2084–2091. doi:10.1103/physreve.54.2084.
- Gilli M, Maringer D, Schumann E (2019). *Numerical Methods and Optimization in Finance*. 2nd edition. Elsevier/Academic Press. URL <http://www.enricoschumann.net/NMOF>.
- Glasserman P (2004). *Monte Carlo Methods in Financial Engineering*, volume 53 of *Stochastic Modelling and Applied Probability*. Springer-Verlag. doi:10.1007/978-0-387-21617-1.
- Graham C, Talay D (2013). *Stochastic Simulation and Monte Carlo Methods: Mathematical Foundations of Stochastic Simulation*, volume 68 of *Stochastic Modelling and Applied Probability*. Springer-Verlag. doi:10.1007/978-3-642-39363-1.
- Grayling MJ (2014). “**phaseR**: An R Package for Phase Plane Analysis of Autonomous ODE Systems.” *The R Journal*, **6**(2), 43–51. doi:10.32614/rj-2014-023.
- Guidoum AC, Boukhetala K (2020). **Sim.DiffProc**: *Simulation of Diffusion Processes*. R package version 4.8, URL <https://CRAN.R-project.org/package=Sim.DiffProc>.
- Gushchin AA (2015). *Stochastic Calculus for Quantitative Finance*. Elsevier.
- Hadeler KP, de Mottoni P, Schumacher K (1980). “Dynamic Models for Animal Orientation.” *The Journal of Mathematical Biology*, **10**, 307–332. doi:10.1007/bf00276092.
- Haken H (2004). *Synergetics: Introduction and Advanced Topics*. Springer-Verlag. doi:10.1007/978-3-662-10184-1.
- Han X, Kloeden PE (2017). *Random Ordinary Differential Equations and Their Numerical Solution*, volume 85 of *Probability Theory and Stochastic Modelling*. Springer-Verlag. doi:10.1007/978-981-10-6265-0.
- Hansen LP (1982). “Large Sample Properties of Generalized Method of Moments Estimators.” *Econometrica*, **50**(4), 1029–1054. doi:10.2307/1912775.
- Helland IS (1983). “Diffusion Models for the Dispersal of Insects near an Attractive Center.” *The Journal of Mathematical Biology*, **18**, 103–122. doi:10.1007/bf00280660.
- Henderson D, Plaschko P (2006). *Stochastic Differential Equations in Science and Engineering*. World Scientific. doi:10.1142/5806.

- Heston SL (1993). “A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options.” *The Review of Financial Studies*, **6**(2), 327–343.
- Iacus SM (2008). *Simulation and Inference for Stochastic Differential Equations: With R Examples*. Springer-Verlag. doi:[10.1007/978-0-387-75839-8](https://doi.org/10.1007/978-0-387-75839-8).
- Iacus SM (2011). *Option Pricing and Estimation of Financial Models with R*. John Wiley & Sons. doi:[10.1002/9781119990079](https://doi.org/10.1002/9781119990079).
- Iacus SM (2016). *sde: Simulation and Inference for Stochastic Differential Equations*. R package version 2.0.15, URL <https://CRAN.R-project.org/package=sde>.
- Iacus SM, Yoshida N (2018). *Simulation and Inference for Stochastic Processes with **yuima**: A Comprehensive R Framework for SDEs and Other Stochastic Processes*. Springer-Verlag. doi:[10.1007/978-3-319-55569-0](https://doi.org/10.1007/978-3-319-55569-0).
- Itô K (1944). “Stochastic Integral.” *Proceedings of the Imperial Academy*, **20**, 519–529.
- Itô K (1946). “On a Stochastic Integral Equation.” *Proceedings of Japan Academy*, **22**, 32–35.
- Itô K (1987). *Selected Papers, Edited and with an Introduction by SRS Varadhan and Daniel W. Stroock*. Springer-Verlag.
- Itô K, McKean, Jr HP (1996). *Diffusion Processes and Their Sample Paths*. Classics in Mathematics. Springer-Verlag. doi:[10.1007/978-3-642-62025-6](https://doi.org/10.1007/978-3-642-62025-6). Reprint of the 1974 Edition (Grundlehren der mathematischen Wissenschaften, Vol. 125).
- Jedrzejewski F (2009). *Modèles Aléatoires et Physique Probabiliste*. Springer-Verlag. doi:[10.1007/978-2-287-99308-4](https://doi.org/10.1007/978-2-287-99308-4).
- Karatzas I, Shreve SE (1988). *Brownian Motion and Stochastic Calculus*, volume 113 of *Graduate Texts in Mathematics*. Springer-Verlag. doi:[10.1007/978-1-4684-0302-2](https://doi.org/10.1007/978-1-4684-0302-2).
- Kasdin NJ (1995). “Runge-Kutta Algorithm for the Numerical Integration of Stochastic Differential Equations.” *Journal of Guidance, Control, and Dynamics*, **18**(1), 114–120. doi:[10.2514/3.56665](https://doi.org/10.2514/3.56665).
- Klebaner FC (2005). *Introduction to Stochastic Calculus with Applications*. 2nd edition. Imperial College Press. doi:[10.1142/p386](https://doi.org/10.1142/p386).
- Kloeden PE, Platen E (1989). “A Survey of Numerical Methods for Stochastic Differential Equations.” *Stochastic Hydrology and Hydraulics*, **3**(3), 155–178. doi:[10.1007/bf01543857](https://doi.org/10.1007/bf01543857).
- Kloeden PE, Platen E (1991a). “Relations between Multiple Itô and Stratonovich Integrals.” *Stochastic Analysis and Applications*, **9**(3), 311–321. doi:[10.1080/07362999108809243](https://doi.org/10.1080/07362999108809243).
- Kloeden PE, Platen E (1991b). “Stratonovich and Itô Stochastic Taylor Expansions.” *Mathematische Nachrichten*, **151**(1), 33–50. doi:[10.1002/mana.19911510103](https://doi.org/10.1002/mana.19911510103).

- Kloeden PE, Platen E (1999). *Numerical Solution of Stochastic Differential Equations*, volume 23 of *Stochastic Modelling and Applied Probability*. 3rd edition. Springer-Verlag. doi:10.1007/978-3-662-12616-5.
- Kloeden PE, Platen E, Schurz H (2003). *Numerical Solution of SDE Through Computer Experiments*. Universitext, 3rd edition. Springer-Verlag. doi:10.1007/978-3-642-57913-4.
- Komori Y, Saito Y, Mitsui T (1994). “Some Issues in Discrete Approximate Solution for Stochastic Differential Equations.” *Computers & Mathematics with Applications*, **28**(10–12), 269–278. doi:10.1016/0898-1221(94)00197-9.
- Lange K (2002). *Mathematical and Statistical Methods for Genetic Analysis*. Statistics for Biology and Health, 2nd edition. Springer-Verlag. doi:10.1007/978-0-387-21750-5.
- Le Gland F (1981). *Estimation de Paramètres Dans les Processus Stochastiques en Observation Incomplète: Application à un Problème de Radio-Astronomie*. Ph.D. thesis, Université Paris IX (Dauphine). URL <https://portail.bu.dauphine.fr/fileviewer/index.php?doc=1981PA090069>.
- L’Ecuyer P (1999). “Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators.” *Operations Research*, **47**(1), 159–164. doi:10.1287/opre.47.1.159.
- L’Ecuyer P, Simard R, Chen EJ, Kelton WD (2002). “An Object-Oriented Random-Number Package with Many Long Streams and Substreams.” *Operations Research*, **50**(6), 1073–1075. doi:10.1287/opre.50.6.1073.358.
- Lemons DS, Gythiel A (1997). “Paul Langevin’s 1908 Paper “On the Theory of Brownian Motion” [“Sur la théorie du mouvement brownien,” C. R. Acad. Sci. (Paris) 146, 530–533 (1908)].” *American Journal of Physics*, **65**(11), 1079–1081. doi:10.1119/1.18725.
- Leung HK, Lai BC (1993). “Stochastic Transient of a Noise-Perturbed Haken-Zwanzig Model.” *Physical Review E*, **47**, 3043–3047. doi:10.1103/physreve.47.3043.
- Leydold J (2020). **rstream**: Streams of Random Numbers. R package version 1.3.6, URL <https://CRAN.R-project.org/package=rstream>.
- Ligges U, Mächler M (2003). “scatterplot3d – An R Package for Visualizing Multivariate Data.” *Journal of Statistical Software*, **8**(11), 1–20. doi:10.18637/jss.v008.i11.
- Lim A, Tjhi W (2015). *R High Performance Programming*. Packt Publishing.
- Liu JS (2004). *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag. doi:10.1007/978-0-387-76371-2.
- Mackevicius V (2016). *Stochastic Models of Financial Mathematics*. Elsevier. doi:10.1016/c2016-0-01246-9.
- Maruyama G (1955). “Continuous Markov Processes and Stochastic Equations.” *Rendiconti Del Circolo Matematico Di Palermo*, **4**(1), 48–90. doi:10.1007/bf02846028.
- McCallum QE, Weston S (2011). *Parallel R: Data Analysis in the Distributed World*. O’Reilly Media.

- McKean, Jr HP (1969). *Stochastic Integrals*. Probability and Mathematical Statistics: A Series of Monographs and Textbooks. Academic Press. doi:[10.1016/c2013-0-12301-x](https://doi.org/10.1016/c2013-0-12301-x).
- McShane EJ (1974). “V - Stochastic Differential Equations.” In EJ McShane (ed.), *Stochastic Calculus and Stochastic Models*, Probability and Mathematical Statistics: A Series of Monographs and Textbooks, pp. 152–179. Academic Press. doi:[10.1016/b978-0-12-486250-0.50010-3](https://doi.org/10.1016/b978-0-12-486250-0.50010-3).
- Milstein GN (1975). “Approximate Integration of Stochastic Differential Equations.” *Theory of Probability & Its Applications*, **19**(3), 557–562. doi:[10.1137/1119062](https://doi.org/10.1137/1119062).
- Milstein GN (1979). “A Method of Second-Order Accuracy Integration of Stochastic Differential Equations.” *Theory of Probability & Its Applications*, **23**(2), 396–401. doi:[10.1137/1123045](https://doi.org/10.1137/1123045).
- Mishra BSP, Dehuri S, Kim E, Wang GN (2016). *Techniques and Environments for Big Data Analysis: Parallel, Cloud, and Grid Computing*, volume 17 of *Studies in Big Data*. Springer-Verlag. doi:[10.1007/978-3-319-27520-8](https://doi.org/10.1007/978-3-319-27520-8).
- Moore CM, Stieha CR, Nolting BC, Cameron MK, Abbott KC (2016). “**QPot**: An R Package for Stochastic Differential Equation Quasi-Potential Analysis.” *The R Journal*, **8**(2), 19–38. doi:[10.32614/rj-2016-031](https://doi.org/10.32614/rj-2016-031).
- Øksendal B (2003). *Stochastic Differential Equations: An Introduction with Applications*. Universitext, 6th edition. Springer-Verlag. doi:[10.1007/978-3-642-14394-6](https://doi.org/10.1007/978-3-642-14394-6).
- Panik MJ (2017). *Stochastic Differential Equations: An Introduction with Applications in Population Dynamics Modeling*. John Wiley & Sons. doi:[10.1002/9781119377399](https://doi.org/10.1002/9781119377399).
- Pedersen AR (1995). “A New Approach to Maximum Likelihood Estimation for Stochastic Differential Equations Based on Discrete Observations.” *Scandinavian Journal of Statistics*, **22**(1), 55–71.
- Pienaar EAD, Varughese MM (2016a). **DiffusionRgqd**: *An R Package for Performing Inference and Analysis on Time-Inhomogeneous Quadratic Diffusion Processes*. R package version 0.1.3, URL <https://CRAN.R-project.org/package=DiffusionRgqd>.
- Pienaar EAD, Varughese MM (2016b). **DiffusionRimp**: *Inference and Analysis for Diffusion Processes via Data Imputation and Method of Lines*. R package version 0.1.2, URL <https://CRAN.R-project.org/package=DiffusionRimp>.
- Pienaar EAD, Varughese MM (2016c). **DiffusionRjgqd**: *An R Package for Performing Inference and Analysis on Time-Inhomogeneous Quadratic Jump Diffusions with State-Dependent Intensity*. R package version 0.1.1, URL <https://CRAN.R-project.org/package=DiffusionRjgqd>.
- Platen E (1982). “A Generalized Taylor Formula for Solutions of Stochastic Equations.” *Sankhya: The Indian Journal of Statistics A*, **44**(2), 163–172.
- Platen E (1995). “On Weak Implicit and Predictor-Corrector Methods.” *Mathematics and Computers in Simulation*, **38**(1), 69–76. doi:[10.1016/0378-4754\(93\)e0068-g](https://doi.org/10.1016/0378-4754(93)e0068-g).

- Platen E (1999). “An Introduction to Numerical Methods for Stochastic Differential Equations.” *Acta Numerica*, **8**, 197–246. doi:[10.1017/s0962492900002920](https://doi.org/10.1017/s0962492900002920).
- Platen E, Bruti-Liberati N (2010). *Numerical Solution of Stochastic Differential Equations with Jumps in Finance*, volume 64 of *Stochastic Modelling and Applied Probability*. Springer-Verlag. doi:[10.1007/978-3-642-13694-8](https://doi.org/10.1007/978-3-642-13694-8).
- Polansky AM, Baker ER (2000). “Multistage Plug-In Bandwidth Selection for Kernel Distribution Function Estimates.” *Journal of Statistical Computation and Simulation*, **65**(1–4), 63–80. doi:[10.1080/00949650008811990](https://doi.org/10.1080/00949650008811990).
- Primak S, Kontorovich V, Lyandres V (2004). *Stochastic Methods and Their Applications to Communications, Stochastic Differential Equations Approach*. John Wiley & Sons. doi:[10.1002/0470021187](https://doi.org/10.1002/0470021187).
- Racicot FE, Théoret R (2006). *Finance Computationnelle et Gestion Des Risques: Ingénierie Financière Avec Applications Excel (Visual Basic) et MATLAB*. Presses de l’Université du Québec.
- Rackauckas C, Nie Q (2017). “**DifferentialEquations.jl** – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia.” *The Journal of Open Source Software*, **5**(1). doi:[10.5334/jors.151](https://doi.org/10.5334/jors.151).
- R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Risken H (1996). *The Fokker-Planck Equation: Methods of Solution and Applications*, volume 18 of *Springer Series in Synergetics*. 2nd edition. Springer-Verlag. doi:[10.1007/978-3-642-61544-3](https://doi.org/10.1007/978-3-642-61544-3).
- Robert C, Casella G (2010). *Introducing Monte Carlo Methods with R*. Use R! Springer-Verlag. doi:[10.1007/978-1-4419-1576-4](https://doi.org/10.1007/978-1-4419-1576-4).
- Rodriguez R, Tuckwell HC (1996). “Statistical Properties of Stochastic Nonlinear Dynamical Models of Single Spiking Neurons and Neural Networks.” *Physical Review E*, **54**, 5585–5590. doi:[10.1103/physreve.54.5585](https://doi.org/10.1103/physreve.54.5585).
- Rodriguez R, Tuckwell HC (2000). “A Dynamical System for the Approximate Moments of Nonlinear Stochastic Models of Spiking Neurons and Networks.” *Mathematical and Computer Modelling*, **31**(4–5), 175–180. doi:[10.1016/s0895-7177\(00\)00036-4](https://doi.org/10.1016/s0895-7177(00)00036-4).
- Rogers LCG, Williams D (1994). *Diffusions, Markov Processes and Martingales*, volume 1. 2nd edition. Cambridge University Press. doi:[10.1017/cbo9781107590120](https://doi.org/10.1017/cbo9781107590120).
- Rogers LCG, Williams D (2000). *Diffusions, Markov Processes and Martingales*, volume 2. 2nd edition. Cambridge University Press. doi:[10.1017/cbo9780511805141](https://doi.org/10.1017/cbo9780511805141).
- Rolski T, Schmidli H, Schmidt V, Teugels J (2008). *Stochastic Processes for Insurance and Finance*. John Wiley & Sons. doi:[10.1002/9780470317044](https://doi.org/10.1002/9780470317044).
- Rossini AJ, Tierney L, Li N (2007). “Simple Parallel Statistical Computing in R.” *Journal of Computational and Graphical Statistics*, **16**(2), 399–420. doi:[10.1198/106186007x178979](https://doi.org/10.1198/106186007x178979).

- Rössler A (2003). “Coefficients of Runge-Kutta Schemes for Itô Stochastic Differential Equations.” *Proceedings in Applied Mathematics and Mechanics*, **3**(1), 571–572. doi:[10.1002/pamm.200310553](https://doi.org/10.1002/pamm.200310553).
- Rössler A (2004). “Runge-Kutta Methods for Stratonovich Stochastic Differential Equation Systems with Commutative Noise.” *Journal of Computational and Applied Mathematics*, **164–165**, 613–627. doi:[10.1016/j.cam.2003.09.009](https://doi.org/10.1016/j.cam.2003.09.009).
- Rössler A (2007). “Second Order Runge-Kutta Methods for Stratonovich Stochastic Differential Equations.” *BIT Numerical Mathematics*, **47**(3), 657–680. doi:[10.1007/s10543-007-0130-3](https://doi.org/10.1007/s10543-007-0130-3).
- Rössler A (2010). “Runge-Kutta Methods for the Strong Approximation of Solutions of Stochastic Differential Equations.” *SIAM Journal on Numerical Analysis*, **48**(3), 922–952. doi:[10.1137/09076636x](https://doi.org/10.1137/09076636x).
- Royston P (1995). “Remark AS R94: A Remark on Algorithm AS 181: The W-Test for Normality.” *Journal of the Royal Statistical Society C*, **44**(4), 547–551. doi:[10.2307/2986146](https://doi.org/10.2307/2986146).
- Ryan JA, Ulrich JM, Thielen W, Teetor P, Bronder S (2020). **quantmod**: *Quantitative Financial Modelling Framework*. R package version 0.4.17, URL <https://CRAN.R-project.org/package=quantmod>.
- Saito Y, Mitsui T (1993). “Simulation of Stochastic Differential Equations.” *The Annals of the Institute of Statistical Mathematics*, **45**(3), 419–432. doi:[10.1007/bf00773344](https://doi.org/10.1007/bf00773344).
- Schmidberger M, Mansmann U (2010). “Parallel Computing with the R Language in a Supercomputing Environment.” In S Wagner, M Steinmetz, A Bode, MM Müller (eds.), *High Performance Computing in Science and Engineering*, pp. 769–780. Springer-Verlag. doi:[10.1007/978-3-642-13872-0_64](https://doi.org/10.1007/978-3-642-13872-0_64).
- Schmidberger M, Morgan M, Eddelbuettel D, Yu H, Tierney L, Mansmann U (2009). “State of the Art in Parallel Computing with R.” *Journal of Statistical Software*, **31**(1), 1–27. doi:[10.18637/jss.v031.i01](https://doi.org/10.18637/jss.v031.i01).
- Scott DW (2015). *Multivariate Density Estimation: Theory, Practice, and Visualization*. 2nd edition. John Wiley & Sons. doi:[10.1002/9781118575574](https://doi.org/10.1002/9781118575574).
- Sevcikova H, Rossini T, L’Ecuyer P (2019). **rlecuyer**: *R Interface to RNG with Multiple Streams*. R package version 0.3-5, URL <https://CRAN.R-project.org/package=rlecuyer>.
- Shapiro SS, Wilk MB (1965). “An Analysis of Variance Test for Normality (Complete Samples).” *Biometrika*, **52**(3/4), 591–611. doi:[10.2307/2333709](https://doi.org/10.2307/2333709).
- Silverman BW (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC. doi:[10.1201/9781315140919](https://doi.org/10.1201/9781315140919).
- Sobczyk K (1991). *Stochastic Differential Equations: With Applications to Physics and Engineering*, volume 40 of *Mathematics and Its Applications*. Springer-Verlag. doi:[10.1007/978-94-011-3712-6](https://doi.org/10.1007/978-94-011-3712-6).

- Socha L (2008). *Linearization Methods for Stochastic Dynamic Systems*, volume 730 of *Lecture Notes in Physics*. Springer-Verlag. doi:10.1007/978-3-540-72997-6.
- Soetaert K (2020). **rootSolve**: *Nonlinear Root Finding, Equilibrium and Steady-State Analysis of Ordinary Differential Equations*. R package version 1.8.2.1, URL <https://CRAN.R-project.org/package=rootSolve>.
- Soetaert K, Cash J, Mazzia F (2012). *Solving Differential Equations in R*. Use R! Springer-Verlag. doi:10.1007/978-3-642-28070-2.
- Soetaert K, Petzoldt T (2020). *CRAN Task View: Differential Equations*. Version 2020-01-12, URL <https://CRAN.R-project.org/view=DifferentialEquations>.
- Soong TT (1973). *Random Differential Equations in Science and Engineering*, volume 103 of *Mathematics in Science and Engineering*. Academic Press.
- Spurdle A (2020). **bivariate**: *Bivariate Probability Distributions*. R package version 0.6.0, URL <https://CRAN.R-project.org/package=bivariate>.
- Stratonovich RL (1966). “A New Representation for Stochastic Integrals and Equations.” *SIAM Journal on Control*, 4(2), 362–371. doi:10.1137/0304028.
- Thomopoulos NT (2013). *Essentials of Monte Carlo Simulation: Statistical Methods for Building Simulation Models*. Springer-Verlag. doi:10.1007/978-1-4614-6022-0.
- Tierney L, Rossini AJ, Li N (2009). “**snow**: A Parallel Computing Framework for the R System.” *International Journal of Parallel Programming*, 37(1), 78–90. doi:10.1007/s10766-008-0077-2.
- Tierney L, Rossini AJ, Li N, Sevcikova H (2018). **snow**: *Simple Network of Workstations*. R package version 0.4-3, URL <https://CRAN.R-project.org/package=snow>.
- Trapletti A, Hornik K (2019). **tseries**: *Time Series Analysis and Computational Finance*. R package version 0.10-47, URL <https://CRAN.R-project.org/package=tseries>.
- Uhlenbeck GE, Ornstein LS (1930). “On the Theory of the Brownian Motion.” *Physical Review*, 36, 823–841. doi:10.1103/physrev.36.823.
- Urbanek S (2014). **multicore**: *Parallel Processing of R Code on Machines with Multiple Cores or CPUs*. R package version 0.1-8, URL <http://www.rforge.net/multicore/>.
- Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag. URL <http://www.stats.ox.ac.uk/pub/MASS4>.
- Wand MP, Jones MC (1995). *Kernel Smoothing*. Chapman & Hall/CRC. doi:10.1201/b14876.
- Wang MC, Uhlenbeck GE (1945). “On the Theory of the Brownian Motion II.” *Reviews of Modern Physics*, 17, 323–342. doi:10.1103/revmodphys.17.323.
- Weberruß VA (1993). *Universality in Statistical Physics and Synergetics: A Comprehensive Approach to Modern Theoretical Physics*. Vieweg+Teubner Verlag. doi:10.1007/978-3-663-13894-5.

- Wickham H (2007). “Reshaping Data with the **reshape** Package.” *Journal of Statistical Software*, **21**(12), 1–20. doi:[10.18637/jss.v021.i12](https://doi.org/10.18637/jss.v021.i12).
- Wickham H (2009). **ggplot2: Elegant Graphics for Data Analysis**. Use R! Springer-Verlag. doi:[10.1007/978-0-387-98141-3](https://doi.org/10.1007/978-0-387-98141-3).
- Wickham H (2015). *Advanced R*. The R Series. Chapman & Hall/CRC.
- Wilke CO (2020). **ggridges: Ridgeline Plots in ggplot2**. R package version 0.5.2, URL <https://CRAN.R-project.org/package=gggridges>.
- Wuertz D, Setz T, Chalabi Y (2017). **fOptions: Rmetrics – Pricing and Evaluating Basic Options**. R package version 3042.86, URL <https://CRAN.R-project.org/package=fOptions>.
- Xie Y (2013). “**animation**: An R Package for Creating Animations and Demonstrating Statistical Methods.” *Journal of Statistical Software*, **53**(1), 1–27. doi:[10.18637/jss.v053.i01](https://doi.org/10.18637/jss.v053.i01).
- Xie Y (2015). *Dynamic Documents with R and knitr*. 2nd edition. Chapman & Hall/CRC. doi:[10.1201/9781315382487](https://doi.org/10.1201/9781315382487).

A. Ridgeline plots

```

R> library("ggribes")
R> library("ggplot2")
R> library("reshape")
R> Seq_time <- seq(0.2, 1, by = 0.2)
R> Model <- c("mod1", "mod2")
R> Res1 <- do.call("cbind", lapply(Seq_time, function(i) rsde1d(mod1,
+   at = i)))
R> Res2 <- do.call("cbind", lapply(Seq_time, function(i) rsde1d(mod2,
+   at = i)))
R> Data <- data.frame(expand.grid(x = sort(rep(Seq_time, mod1$M)),
+   y = Model), reshape::melt(c(Res1, Res2)))
R> names(Data) <- c("time", "Model", "value")
R> ggplot(Data, aes(x = value, y = as.factor(time), fill = Model)) +
+   geom_density_ridges2(scale = 1.5, rel_min_height = 0.001, alpha = 0.2,
+   from = 0, to = 6) +
+   scale_fill_cyclical(values = c("#FF00004B", "#0000FF4B"),
+   guide = "legend", labels = c("mod1", "mod2"), name = "") +
+   scale_y_discrete(expand = c(0, 0)) +
+   scale_x_continuous(expand = c(0, 0)) +
+   xlab("Value") + ylab("Time") + theme_ridges()

```

B. Moment equations of OU process and its integral

We will illustrate in a two-dimensional linear case that the first and central second-order moments for system (23), found by the approximation procedure coincide exactly with the known values. Letting the means, variances and covariance of the coupled (X_t, Y_t) be $m(t) = (m_1(t), m_2(t))$, $S(t) = (S_1(t), S_2(t))$ and $C_{12}(t)$, respectively. Application of the above formulation (11) gives the following differential equations for the two means $m_1(t)$ and $m_2(t)$:

$$\begin{cases} \frac{d}{dt}m_1(t) = \frac{(\theta - m_1(t))}{\mu} \\ \frac{d}{dt}m_2(t) = m_1(t) \end{cases} \quad (44)$$

The exact solutions of (44) with $m_1(0) = x_0$ and $m_2(0) = y_0$ are:

$$\begin{aligned} m_1(t) &= \theta + (x_0 - \theta) e^{-t/\mu} \\ m_2(t) &= y_0 + \theta t + (x_0 - \theta) \mu (1 - e^{-t/\mu}) \end{aligned}$$

and with the application of the two formulas (12) and (13), then we have the following three

coupled differential equations for the two variances $S_1(t)$, $S_2(t)$ and the covariance $C_{12}(t)$:

$$\begin{cases} \frac{d}{dt}S_1(t) = \sigma - 2 \frac{S_1(t)}{\mu} \\ \frac{d}{dt}S_2(t) = 2 C_{12}(t) \\ \frac{d}{dt}C_{12}(t) = S_1(t) - \frac{C_{12}(t)}{\mu} \end{cases} \quad (45)$$

The exact solutions of (45) with initial conditions $S_1(0) = S_2(0) = C_{12}(0) = 0$ are:

$$\begin{aligned} S_1(t) &= \mu\sigma \left(1 - e^{-2t/\mu}\right) / 2 \\ S_2(t) &= \mu^3\sigma \left[\frac{t}{\mu} - 2 \left(1 - e^{-t/\mu}\right) + \frac{1}{2} \left(1 - e^{-2t/\mu}\right) \right] \\ C_{12}(t) &= \frac{\mu^2\sigma}{2} \left(1 - 2e^{-t/\mu} + e^{-2t/\mu}\right) \end{aligned}$$

which are exactly the known mean, variance and covariance for system (23).

C. Stochastic Haken-Zwanzig model

C.1. R code for Figure 11

```
R> library("phaseR")
R> HZ.model <- function(t, y, parameters) {
+   mu <- parameters[1]
+   alpha <- parameters[2]
+   x <- y[1]
+   y <- y[2]
+   dy <- numeric(2)
+   dy[1] <- mu * x - x * y
+   dy[2] <- -alpha * y + x^2
+   list(dy)
+ }
R> mu <- 1
R> alpha <- 3
R> parms <- c(mu, alpha)
R> HZ.flowField <- flowField(HZ.model, xlim = c(-4, 4), ylim = c(-1, 2.5),
+   parameters = parms, add = FALSE, points = 16, las = 1, lwd = 2)
R> HZ.nullclines <- nullclines(HZ.model, xlim = c(-4, 4), ylim = c(-1, 2.5),
+   parameters = parms, lwd = 2, col = c("red", "blue"), add.legend = FALSE)
R> points(0, 0, pch = 21, cex = 1.5, col = 1, bg = "white")
R> points(sqrt(mu * alpha), mu, pch = 19, cex = 1.5, col = 1)
R> points(-sqrt(mu * alpha), mu, pch = 19, cex = 1.5, col = 1)
R> text(0.15, -0.13, expression(e[u]), cex = 1.5)
R> text(-sqrt(mu * alpha), mu+0.15, expression(e[s[1]]), cex = 1.5)
R> text(+sqrt(mu * alpha), mu+0.15, expression(e[s[2]]), cex = 1.5)
```

C.2. R code for Table 5

```

R> mu <- 1
R> alpha <- 3
R> sigma1 <- 0.1
R> sigma2 <- 0.05
R> f <- expression((mu * x - x * y), (-alpha * y + x^2))
R> g0 <- expression(sigma1, sigma2)
R> g1 <- expression(sigma1 * x, sigma2 * y)
R> g2 <- expression(sigma1 * (1 - y) * x, sigma2 * (1 - x) * y)

```

Moment equations method for the system (27):

```

R> t <- seq(0, 100, by = 0.001)
R> S0 <- c(m1 = 0, m2 = 0, S1 = 0, S2 = 0, C12 = 0)
R> S1 <- c(m1 = 1, m2 = 1, S1 = 0, S2 = 0, C12 = 0)
R> MEMk0 <- MEM.sde(drift = f, diffusion = g0, solve = TRUE, init = S0,
+   time = t, type = "str")
R> MEMk1 <- MEM.sde(drift = f, diffusion = g1, solve = TRUE, init = S1,
+   time = t, type = "str")
R> MEMk2 <- MEM.sde(drift = f, diffusion = g2, solve = TRUE, init = S1,
+   time = t, type = "str")
R> summary(MEMk0, at = 100)
R> summary(MEMk1, at = 100)
R> summary(MEMk2, at = 100)

```

Parallel Monte Carlo method for the system (27):

```

R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> I0 <- c(x = 0, y = 0)
R> I1 <- c(x = 1, y = 1)
R> modk0 <- snssde2d(drift = f, diffusion = g0, M = 5000, x0 = I0, Dt = 0.1,
+   type = "str")
R> modk1 <- snssde2d(drift = f, diffusion = g1, M = 5000, x0 = I1, Dt = 0.1,
+   type = "str")
R> modk2 <- snssde2d(drift = f, diffusion = g2, M = 5000, x0 = I1, Dt = 0.1,
+   type = "str")
R> sde.fun2d <- function(data, i) {
+   d <- data[i, ]
+   return(c(mean(d$x), mean(d$y), var(d$x), var(d$y), cov(d$x, d$y)))
+ }
R> MCMk0 <- MCM.sde(modk0, statistic = sde.fun2d, R = 100, parallel = "snow",
+   ncpus = 4)
R> MCMk1 <- MCM.sde(modk1, statistic = sde.fun2d, R = 100, parallel = "snow",
+   ncpus = 4)
R> MCMk2 <- MCM.sde(modk2, statistic = sde.fun2d, R = 100, parallel = "snow",
+   ncpus = 4)

```

```

R> MCMk0$MC
R> MCMk1$MC
R> MCMk2$MC
R> MCMk0 <- MCM.sde(modk0, statistic = sde.fun2d, R = 500, parallel = "snow",
+   ncpus = 4)
R> MCMk1 <- MCM.sde(modk1, statistic = sde.fun2d, R = 500, parallel = "snow",
+   ncpus = 4)
R> MCMk2 <- MCM.sde(modk2, statistic = sde.fun2d, R = 500, parallel = "snow",
+   ncpus = 4)
R> MCMk0$MC
R> MCMk1$MC
R> MCMk2$MC

```

D. Stochastic correlation models

D.1. Stochastic volatility: Heston model

The first and central second-order moments for system (41), found by function `MEM.sde()` coincide with the `MCM.sde()` values. Using `MEM.sde()`, we get the symbolic computation, we just define the argument `corr` by an expression which contains the correlation structure between the Wiener processes $B_{1,t}$ and $B_{2,t}$; i.e., $E(dB_{1,t}dB_{2,t}) = \rho dt$.

```

R> f <- expression(mu * x, nu * (theta - y))
R> g <- expression(sqrt(y) * x, sigma * sqrt(y))
R> RHO <- expression(rho)
R> mem.HM <- MEM.sde(drift = f, diffusion = g, corr = RHO)
R> TEX.sde(object = mem.HM)

```

$$\left\{ \begin{array}{l} \frac{d}{dt}m_1(t) = \mu m_1(t) \\ \frac{d}{dt}m_2(t) = \nu (\theta - m_2(t)) \\ \frac{d}{dt}S_1(t) = S_1(t) (2\mu + m_2(t)) + m_1(t) (2C_{12}(t) + m_1(t)m_2(t)) \\ \frac{d}{dt}S_2(t) = \sigma^2 m_2(t) - 2(\nu S_2(t)) \\ \frac{d}{dt}C_{12}(t) = C_{12}(t) (\mu + \rho\sigma - \nu) + \rho\sigma m_1(t)m_2(t) \end{array} \right. \quad (46)$$

D.2. Stochastic mean and stochastic volatility: The Chen model

The system of SDEs that determines the instantaneous interest rate dynamics is given by [Chen \(1996\)](#) model type of three-factor:

$$\left\{ \begin{array}{l} dX_t = \mu (Y_t - X_t) dt + \sqrt{X_t Z_t} dB_{1,t} \\ dY_t = \nu (\theta - Y_t) dt + \sigma \sqrt{Y_t} dB_{2,t} \\ dZ_t = \phi (\tau - Z_t) dt + \xi \sqrt{Z_t} dB_{3,t} \end{array} \right. \quad (47)$$

where X_t are the dynamics of the short rate, its short term mean Y_t , and its volatility Z_t ; and the three Wiener processes are assumed to be correlated with Σ as follows:

$$\Sigma = \begin{pmatrix} 1 & \rho_1 & \rho_2 \\ \rho_1 & 1 & \rho_3 \\ \rho_2 & \rho_3 & 1 \end{pmatrix}$$

To simulate the numerical solution of (47) with function `snssde3d()`, we need to define the correlation (`corr`) by a variance-covariance or correlation matrix.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> x0 <- 4
R> y0 <- 0.1
R> z0 <- 2
R> mu <- 1.5
R> theta <- 2
R> nu <- 2.5
R> phi <- 1
R> sigma <- 0.2
R> xi <- 0.1
R> tau <- 0.2
R> f <- expression(mu * (y - x), nu * (theta - y), phi * (tau - z))
R> g <- expression(sqrt(x * z), sigma * sqrt(y), xi * sqrt(z))
R> Sigma <- matrix(c(1, -0.5, -0.25, -0.5, 1, 0.95, -0.25, 0.95, 1),
+   nrow = 3, ncol = 3)
R> CM <- snssde3d(drift = f, diffusion = g, corr = Sigma, x0 = c(x0, y0, z0),
+   T = 10, M = 5000)
R> CM
```

Itô Sde 3D:

```
| dX(t) = mu * (Y(t) - X(t)) * dt + sqrt(X(t) * Z(t)) * dB1(t)
| dY(t) = nu * (theta - Y(t)) * dt + sigma * sqrt(Y(t)) * dB2(t)
| dZ(t) = phi * (tau - Z(t)) * dt + xi * sqrt(Z(t)) * dB3(t)
| Correlation structure:
|   1.00 -0.50 -0.25
|  -0.50  1.00  0.95
|  -0.25  0.95  1.00
```

Method:

```
| Euler scheme with order 0.5
```

Summary:

```
| Size of process      | N = 1001.
| Number of simulation | M = 5000.
| Initial values      | (x0, y0, z0) = (4, 0.1, 2).
| Time of process     | t in [0, 10].
| Discretization      | Dt = 0.01.
```

Now, in `MEM.sde()` we define the the correlation (`corr`) by an expression (length equal to three) which contains the correlation coefficients (ρ_1, ρ_2, ρ_3) . The R code needed to do this is:

```
R> RHO <- expression(-0.5, -0.25, 0.95)
R> para <- c(mu = mu, theta = theta, nu = nu, phi = phi, sigma = sigma,
+   xi = xi, tau = tau)
R> init <- c(m1 = x0, m2 = y0, m3 = z0, S1 = 0, S2 = 0, S3 = 0, C12 = 0,
+   C13 = 0, C23 = 0)
R> MEM.CM <- MEM.sde(drift = f, diffusion = g, corr = RHO, solve = TRUE,
+   init = init, parms = para, time = seq(0, 10, by = 0.01))
R> summary(MEM.CM, at = 10)
```

Approximation of moment at time 10

```
| m1(10) = 2.00000 | S1(10) = 0.11694 | C12(10) = -0.01622
| m2(10) = 2.00000 | S2(10) = 0.01600 | C13(10) = -0.00076
| m3(10) = 0.20008 | S3(10) = 0.00100 | C23(10) = 0.00343
```

E. Running ‘MCM.sde’ quickly on many machines

The **snow** technique is essentially a very coarse grained remote strategy call. It relies upon delivery copies of code and information to remote procedures and then returning results. It is ill-suited for little assignments but very well suited for a reasonable number of moderate to large tasks. This is the technique utilized by R’s **parallel** package. This strategy may appear to be less productive and less advanced than shared memory strategies, but relying on the transmission of objects, it is in principle very easy to extend the technique of a single machine to several machines. This is the strategy considered in the following. Do the following: Collect a list of machine addresses that you can use. This is the hard part; it depends on your operating system, and something you should get help with if you have not already tried it. This assumes that you already understand how to configure secure shell (SSH) and for a Windows box use PuTTY (see the vignette of package **parallel**), for more details see the website <https://www.stat.uiowa.edu/~luke/R/cluster/cluster.html>. In this case, we use IPV4 addresses and hostnames. In our case the list is:

- *Master machine*: 2.20 GHz Intel Core i5-5200U processor, 8 GB 1600 MHz DDR3 of memory. IPV4: "192.168.1.105" and user: "acguidoum".
- *Second machine*: 2.50 GHz Intel Core i7-6500U processor, 8 GB 1600 MHz DDR3 of memory. IPV4: "192.168.1.108" and user: "TM161".

Note that we do not collect passwords because we assume that we have configured the appropriate `authorized_keys` and key pairs in the `.ssh` configurations of all these machines. We call the machine we use to issue the “master” global calculation. It is essential to try all these addresses with `ssh` in a terminal shell before trying them with R. Also the machine address you choose for the “master” must be an address that the machines can use to return to the main machine. Now, with the system things resolved, the R part is as follows. Start your cluster with:

```
R> machineAddresses <- list(list(host = "192.168.1.105", user = "acguidoum",
+   ncore = 4), list(host = "192.168.1.108", user = "TM161", ncore = 4))
```

```
R> spec <- lapply(machineAddresses, function(machine)
+   rep(list(list(host = machine$host, user = machine$user)),
+   machine$ncore))
R> spec <- unlist(spec, recursive = FALSE)
R> cl2m <- parallel::makeCluster(master = "192.168.1.105", type = "PSOCK",
+   spec = spec)
R> print(cl2m)

socket cluster with 8 nodes on hosts
'192.168.1.105', '192.168.1.108'
```

You can now run function `MCM.sde()` (e.g., for model (14)) using the two machines with 8 cores (nodes)). For the right tasks, this represents a substantial speed-up.

```
R> set.seed(1234, kind = "L'Ecuyer-CMRG")
R> f <- expression(0.5*theta^2*x)
R> g <- expression(theta*x)
R> mod1 <- snssde1d(drift = f, diffusion = g, x0 = x0, M = 5000,
+   type = "ito")
R> MCM.sde(model = mod1, statistic = stat.fun1d, R = 100, parallel = "snow",
+   ncpus = 8, cl = cl2m)
```

Affiliation:

Arsalane Chouaib Guidoum
 Department of Probability and Statistics
 Faculty of Mathematics
 University of Science and Technology Houari Boumediene
 BP 32 El-Alia, 16111 – U.S.T.H.B, Algeria
Current address:
 Department of Mathematics
 Institute of Science and Technology
 University Center of Tamanghasset, Algeria
 E-mail: acguidoum@usthb.dz

Kamal Boukhetala
 Faculty of Mathematics
 University of Science and Technology Houari Boumediene
 BP 32 El-Alia, 16111 – U.S.T.H.B, Algeria
 E-mail: kboukhetala@usthb.dz

Journal of Statistical Software

published by the Foundation for Open Access Statistics

November 2020, Volume 96, Issue 2

[doi:10.18637/jss.v096.i02](https://doi.org/10.18637/jss.v096.i02)

<http://www.jstatsoft.org/>

<http://www.foastat.org/>

Submitted: 2016-06-22

Accepted: 2019-10-20