Numerical Algorithms (2023) 93:27–66
https://doi.org/10.1007/s11075-022-01401-z

ORIGINAL PAPER

An analysis of approximation algorithms for iterated stochastic integrals and a Julia and MATLAB simulation toolbox

Felix Kastner[1] · **Andreas Rößler**[1]

Received: 26 January 2022 / Accepted: 18 August 2022 / Published online: 12 December 2022

Abstract

For the approximation and simulation of twofold iterated stochastic integrals and the corresponding Lévy areas w.r.t. a multi-dimensional Wiener process, we review four algorithms based on a Fourier series approach. Especially, the very efficient algorithm due to Wiktorsson and a newly proposed algorithm due to Mrongowius and Rößler are considered. To put recent advances into context, we analyse the four Fourier-based algorithms in a unified framework to highlight differences and similarities in their derivation. A comparison of theoretical properties is complemented by a numerical simulation that reveals the order of convergence for each algorithm. Further, concrete instructions for the choice of the optimal algorithm and parameters for the simulation of solutions for stochastic (partial) differential equations are given. Additionally, we provide advice for an efficient implementation of the considered algorithms and incorporated these insights into an open source toolbox that is freely available for both Julia and MATLAB programming languages. The performance of this toolbox is analysed by comparing it to some existing implementations, where we observe a significant speed-up.

Keywords Iterated stochastic integral · Lévy area · Stochastic simulation · Julia · MATLAB · Software toolbox · Stochastic differential equation · Stochastic partial differential equation

✉ Andreas Rößler
roessler@math.uni-luebeck.de

Felix Kastner
kastner@math.uni-luebeck.de

[1] Institute of Mathematics, Universität zu Lübeck, Ratzeburger Allee 160, 23562, Lübeck, Germany

1 Introduction

> In the future, we foresee that the use of area integrals when simulating strong solutions to SDEs will become as automatic as the use of random numbers from a normal distribution is today. After all, once a good routine has been developed and implemented in numerical libraries, the ordinary user will only need to call this routine from each program and will not need to be concerned with the details of how the routine works.
>
> — J. G. Gaines and T. J. Lyons (1994) [14]

Even though Wiktorsson published an efficient algorithm for the simulation of iterated integrals in 2001, this prediction by Gaines and Lyons has still, almost thirty years later, not come true yet. Among the twenty SDE simulation packages we looked at, only four have implemented Wiktorsson's algorithm. This suggests that either Wiktorsson's paper is not as well known as it should be or most authors of SDE solvers are not willing to dive into the intricacies of Lévy area simulation. This paper is an attempt to fix this.

In many applications where stochastic (ordinary) differential equations (SDEs) or stochastic partial differential equations (SPDEs) are used as a mathematical model the exact solution of these equations can not be calculated explicitly. Therefore, numerical approximations have to be computed in these cases. Often numerical integrators solely based on increments of the driving Wiener process or Q-Wiener process like Euler-type schemes are applied because they can be easily implemented. In general, these schemes attain only low orders of strong convergence, see, e.g., [29, 40]. This is because the best strong convergence rate we can expect to get is in general at most $1/2$ if only finitely many observations of the driving Wiener process are used by a numerical scheme, see, e.g., Clark and Cameron [7] and Dickinson [11]. One exception to this is the case when the diffusion satisfies a so-called commutativity condition [29, 33, 40, 46]. However, in many applications this commutativity condition is not satisfied.

In order to obtain higher order numerical methods for SDEs and SPDEs in general, one needs to incorporate iterated stochastic integrals as they naturally appear in stochastic Taylor expansions of the solution process. Since the distribution of these iterated stochastic integrals is not known explicitly in case of a multi-dimensional driving Wiener process, efficient algorithms for their approximation are needed to achieve some higher order of strong convergence compared to Euler-type schemes. Thus, in general, to approximate or to simulate solutions for SDEs or SPDEs efficiently the use of iterated stochastic integrals is essential.

Up to now, there exist only a few approaches for the approximation of iterated stochastic integrals and the corresponding Lévy areas in L^2-norm or in the strong sense. After Lévy defined the eponymous stochastic area integral in [35], the first approximation algorithms appeared in the 1980s [37, 40]. These are based on a Fourier series expansion of the Wiener process and have an error of order $O(c^{-1/2})$ with c denoting the computational cost, considered as the number of random variables used. The first higher order algorithm was presented by Rydén and Wiktorsson in [47]. Here, the method B' from that paper has an error of order $O(c^{-1})$ but is

only applicable to two-dimensional Wiener processes. In the same year, Wiktorsson published his groundbreaking algorithm of order $O(c^{-1})$ which also works for higher-dimensional Wiener processes [53] but has a worse error constant. Twenty years later, Mrongowius and Rößler [41] recently proposed a variation of Wiktorsson's algorithm that has an improved error constant and is easier to implement. It turns out that in the two-dimensional setting it even beats the method B', that is, the error constant is by a factor $1/\sqrt{2}$ smaller compared to method B' for the same amount of work.

These higher order algorithms allow to effectively use strong order 1 schemes for SDEs and SPDEs with non-commutative noise. Taking into account the computational effort of SDE and SPDE integrators, one obtains the so-called effective order of convergence by considering errors versus computational cost, see also [33, 46]. For example, combining a strong order 1 SDE integrator, like the Milstein scheme or a stochastic Runge–Kutta scheme proposed in [46], with a Lévy area algorithm of order $O(c^{-1/2})$ achieves an effective order of convergence that is only $1/2$. This is the same as the effective order of convergence for the classical Euler–Maruyama scheme. But, using instead a higher order Lévy area algorithm of order $O(c^{-1})$ results in a significantly higher effective order of convergence that is $2/3$. Similar improvements can be achieved for SPDEs, see also [19, 20, 46, 53] for a detailed discussion.

The algorithms that we focus on in the present paper are based on a trigonometric Fourier series expansion of the corresponding Brownian bridge process. To be precise, we restrict our considerations to the naive truncated Fourier series approximation, the improved truncated Fourier series algorithm proposed by Milstein [40], see also [29, 30], and the higher order variants by Wiktorsson [53] and Mrongowius and Rößler [41]. As we will see, the latter two incorporate an additional approximation of the truncated terms to achieve an error of order $O(c^{-1})$.

Note that there also exist further variants of Fourier series based algorithms using different bases [13, 31, 32, 37]. These generally achieve the same convergence order as the Milstein algorithm but with a slightly worse constant. This matches Dickinson's result, that the Milstein algorithm is asymptotically optimal in the class of algorithms that only use linear information about the underlying Wiener process [11]. There is also work extending some of the mentioned algorithms to the infinite dimensional Hilbert space setting important for SPDEs [34].

Next to these Fourier series based algorithms, there exist also different approaches. See, e.g., [14, 39, 47, 51] for simulation algorithms or [10, 12] for approximation in a Wasserstein metric. However, to the best of our knowledge, these algorithms either lack a L^2-error estimate or come with additional assumptions on the target SDE and are thus not suitable for the general strong approximation of SDEs. Moreover, for the approximation of Lévy areas driven by fractional Brownian motion, we refer to [42, 43].

The aims of this paper are twofold. On the one hand, we give an introduction to the different Fourier series based algorithms under consideration and emphasize their similarities and differences. A special focus lies on the analysis of the computational complexity for these algorithms that is important in order to finally justify their application for SDE or SPDE approximation and simulation problems. On the other hand, we provide an efficient implementation of these algorithms. This is essential

for a reasonable application in the first place. The result is a new Julia and MATLAB software toolbox for the simulation of twofold iterated stochastic integrals and their Lévy areas, see [27, 28]. These two packages make it feasible to use higher order approximation schemes, like Milstein or stochastic Runge–Kutta schemes, for non-commutative SDEs and SPDEs.

The paper is organized as follows: In Section 2 we give a brief introduction to the theoretical background of iterated stochastic integrals with the corresponding Lévy areas and the Fourier series expansion which builds the basis for the approximation algorithms under consideration. Based on this preliminary section, we detail the derivation of the four algorithms under consideration in Section 3. Besides the theoretical foundations, we focus on the efficient implementation of the four algorithms. In Section 4, error estimates for each algorithm are gathered for direct comparison. Here, it is worth noting that some slightly improved result has been found for Wiktorsson's algorithm. Furthermore, we review the application of these algorithms to the numerical simulation of SDEs. Continuing in Section 5, we give a detailed analysis of the computational complexity and determine the optimal algorithm for a range of parameters. In addition, a numerical simulation confirms the theoretical orders of convergence from the previous section. The paper closes in Section 6 with the description of the newly developed software toolbox and a runtime benchmark against currently available implementations.

2 Theoretical foundations for the simulation of iterated stochastic integrals

Here, we give a brief introduction to twice iterated stochastic integrals in terms of Wiener processes (also called Brownian motions) and their relationship to Lévy areas. In addition, the infinite dimensional case of a Q-Wiener process is briefly mentioned as well. Based on these fundamentals, the well known Fourier series expansion of the Brownian bridge process is presented, which builds the basis for all algorithms in this paper.

2.1 Iterated stochastic integrals and the Lévy area

Let $(\Omega, \mathcal{F}, \mathrm{P})$ be a complete probability space and let $(W_t)_{t\in[0,T]}$ be an m-dimensional Wiener process for some $0 < T < \infty$ with $(W_t)_{t\in[0,T]} = ((W_t^1, \dots, W_t^m)^\top)_{t\in[0,T]}$, i.e., the components $(W_t^i)_{t\in[0,T]}$, $i = 1, \dots, m$, are independent scalar Wiener processes. Further, let $\|\cdot\|$ denote the Euclidean norm if not stated otherwise and let $\|X\|_{L^2(\Omega)} = \mathrm{E}\big[\|X\|^2\big]^{1/2}$ for any $X \in L^2(\Omega)$ in the following. We are interested in simulating the increments $\Delta W_{t_0,t_0+h}^i = W_{t_0+h}^i - W_{t_0}^i$ and $\Delta W_{t_0,t_0+h}^j = W_{t_0+h}^j - W_{t_0}^j$ of the Wiener process together with the twice iterated stochastic integrals

$$\mathcal{I}_{(i,j)}(t_0, t_0+h) = \int_{t_0}^{t_0+h} \int_{t_0}^{s} \mathrm{d}W_r^i \, \mathrm{d}W_s^j \tag{1}$$

for some $0 \le t_0 < t_0 + h \le T$ and $1 \le i, j \le m$. Note that for the Wiener process $(\hat{W}_t)_{t\in[0,h]}$ with $\hat{W}_t = W_{t_0+t} - W_{t_0}$, see [26, Ch. 2, Lem. 9.4], it holds that

$$\int_0^h \int_0^s \mathrm{d}\hat{W}_r^i \, \mathrm{d}\hat{W}_s^j = \int_{t_0}^{t_0+h} \int_{t_0}^s \mathrm{d}W_r^i \, \mathrm{d}W_s^j \tag{2}$$

for $1 \le i, j \le m$. Moreover, due to the time-change formula for stochastic integrals [26, Ch. 3, Prop. 4.8] one can show that for the scaled Wiener process $(\tilde{W}_t)_{t\in[0,1]}$ with $\tilde{W}_t = \frac{1}{\sqrt{h}} \hat{W}_{ht}$, see [26, Ch. 2, Lem. 9.4], it holds

$$h \int_0^1 \int_0^s \mathrm{d}\tilde{W}_r^i \, \mathrm{d}\tilde{W}_s^j = \int_0^h \int_0^s \mathrm{d}\hat{W}_r^i \, \mathrm{d}\hat{W}_s^j \,. \tag{3}$$

As a result of this, without loss of generality we restrict our considerations to the case $t_0 = 0$ in the following and denote

$$\mathcal{I}_{(i,j)}(h) = \int_0^h \int_0^s \mathrm{d}W_r^i \, \mathrm{d}W_s^j \,. \tag{4}$$

Further, let $\mathcal{I}(h) = (\mathcal{I}_{(i,j)}(h))_{1\le i,j\le m}$ be the $m \times m$-matrix containing all iterated stochastic integrals.

In some special cases, one may circumvent the simulation of the iterated stochastic integrals by making use of the relationship

$$\frac{1}{2}\big(\mathcal{I}_{(i,j)}(h) + \mathcal{I}_{(j,i)}(h)\big) = \frac{1}{2} \, W_h^i \, W_h^j \tag{5}$$

for $1 \le i < j \le m$, see, e.g., [29], where the left-hand side represents the symmetric part of the iterated stochastic integrals that can be expressed by the corresponding increments of the Wiener process. The right-hand side of (5) can be easily simulated since the random variables $W_h^i \sim \mathcal{N}(0, h)$ are i.i.d. distributed for $1 \le i \le m$. In case $i = j$ we can calculate explicitly that

$$\mathcal{I}_{(i,i)}(h) = \frac{1}{2}\big((W_h^i)^2 - h\big) \tag{6}$$

which follows from the Itô formula, see, e.g., [26].

The problem to simulate realizations of iterated stochastic integrals is directly related to the simulation of the corresponding so-called Lévy area [35] $A_{(i,j)}(h)$ defined as the skew-symmetric part of the iterated integrals

$$A_{(i,j)}(h) = \frac{1}{2}\big(\mathcal{I}_{(i,j)}(h) - \mathcal{I}_{(j,i)}(h)\big) \tag{7}$$

for $1 \le i, j \le m$. We denote by $A(h) = (A_{(i,j)}(h))_{1\le i,j\le m}$ the $m \times m$ matrix of all Lévy areas. Thus it holds that $A(h) = -A(h)^\top$ and $A_{(i,i)} = 0$ for all $1 \le i \le m$. Due to (5), it follows that $\mathcal{I}_{(i,j)}(h) = \frac{1}{2} \, W_h^i \, W_h^j + A_{(i,j)}(h)$ for $i \ne j$. Now, the difficult part is to simulate $\mathcal{I}_{(i,j)}(h)$ for $i \ne j$ because the distribution of the corresponding Lévy area $A_{(i,j)}(h)$ is not known.

For the infinite dimensional setting as it is the case for SPDEs of evolutionary type, we follow the approach in [34]. Therefore, we consider a U-valued and in general infinite dimensional Q-Wiener process $(W_t^Q)_{t\in[0,T]}$ taking values in some separable real Hilbert space U. Let $(\eta_i)_{i\in\mathbb{N}}$ denote the eigenvalues of the trace class,

non-negative and symmetric covariance operator $Q \in L(U)$ w.r.t. an orthonormal basis (ONB) of eigenfunctions $(e_i)_{i\in\mathbb{N}}$ such that $Qe_i = \eta_i e_i$ for all $i \in \mathbb{N}$. We assume that $\eta_i > 0$ for $i = 1, \ldots, m$. Then, the orthogonal projection of W_t^Q to the m-dimensional subspace $U_m = \text{span}\{e_i : 1 \le i \le m\}$ of U is given by

$$W_t^{Q,m} = \sum_{i=1}^{m} \sqrt{\eta_i}\, e_i\, W_t^i, \quad t \in [0, T],$$

where $(W_t^i)_{t\in[0,T]}$, $1 \le i \le m$, are independent scalar Wiener processes, see, e.g., [8, 45]. The corresponding finite-dimensional covariance operator Q_m is then defined as $Q_m = \text{diag}(\eta_1, \ldots, \eta_m) \in \mathbb{R}^{m\times m}$ and the vector $(\langle W_h^{Q,m}, e_i\rangle)_{1\le i\le m}$ is multivariate $\mathcal{N}(0, hQ_m)$ distributed for $0 < h \le T$. We want to simulate the Hilbert space valued iterated stochastic integral

$$\int_0^h \Psi \int_0^s \Phi \, \mathrm{d}W_u^{Q,m} \, \mathrm{d}W_s^{Q,m} = \sum_{i,j=1}^{m} \mathcal{I}_{(i,j)}^{Q}(h)\, \Psi(\Phi e_i, e_j) \tag{8}$$

where $\mathcal{I}_{(i,j)}^{Q}(h) = \int_0^h \int_0^s \langle \mathrm{d}W_u^Q, e_i\rangle_U \, \langle \mathrm{d}W_s^Q, e_j\rangle_U$ and where Ψ and Φ are some suitable linear operators, see [34] for details. Let $\mathcal{I}^{Q_m}(h) = (\mathcal{I}_{(i,j)}^{Q}(h))_{1\le i,j\le m}$ denote the corresponding $m \times m$-matrix. Since $\mathcal{I}^{Q_m}(h) = Q_m^{1/2}\mathcal{I}(h)\,Q_m^{1/2}$, this random matrix can be expressed by a transformation of the $m\times m$-matrix $\mathcal{I}(h)$. Therefore, we mainly concentrate on the approximation of $\mathcal{I}_{(i,j)}(h)$ for $1 \le i, j \le m$ in the following and refer to [34] for more details and error estimates in the infinite dimensional setting.

2.2 The Fourier series approach

We start by focusing on the Fourier series expansion of the Brownian bridge process, see [29, 30, 40]. The integrated Fourier series expansion is the basis for all algorithms considered in Section 3. Given an m-dimensional Wiener process $(W_t)_{t\in[0,h]}$, we consider the corresponding tied down Wiener process $\left(W_t - \frac{t}{h}W_h\right)_{t\in[0,h]}$, which is also called a Brownian bridge process, whose components can be expanded into a Fourier series which results in

$$W_t^i = \frac{t}{h}W_h^i + \frac{1}{2}a_0^i + \sum_{r=1}^{\infty}\left(a_r^i \cos\left(\frac{2\pi r}{h}t\right) + b_r^i \sin\left(\frac{2\pi r}{h}t\right)\right) \quad \text{P-a.s.} \tag{9}$$

with random coefficients

$$a_r^i = \frac{2}{h}\int_0^h \left(W_s^i - \frac{s}{h}W_h^i\right)\cos\left(\frac{2\pi r}{h}s\right)\mathrm{d}s$$

and

$$b_r^i = \frac{2}{h}\int_0^h \left(W_s^i - \frac{s}{h}W_h^i\right)\sin\left(\frac{2\pi r}{h}s\right)\mathrm{d}s$$

for $1 \le i \le m$ and $t \in [0,h]$. Using the distributional properties of the Wiener integral it easily follows that the coefficients are Gaussian random variables with

$$a_0^i \sim \mathcal{N}\left(0, \frac{1}{3}h\right), \quad a_r^i \sim \mathcal{N}\left(0, \frac{1}{2\pi^2 r^2}h\right) \quad \text{and} \quad b_r^i \sim \mathcal{N}\left(0, \frac{1}{2\pi^2 r^2}h\right). \tag{10}$$

From the boundary conditions it follows that $a_0^i = -2\sum_{r=1}^{\infty} a_r^i$. One can easily prove that the random variables W_h^q, a_k^i and b_l^j for $i,j,q \in \{1,\dots,m\}$ and $k,l \in \mathbb{N}$ are all independent, while each a_0^i depends on a_r^i for all $r \in \mathbb{N}$. Using this representation Lévy was the first to derive the following series representation of what is now called Lévy area [35] denoted as $A_{(i,j)}(h) = \frac{1}{2}(\mathcal{I}_{(i,j)}(h) - \mathcal{I}_{(j,i)}(h))$. By integrating (9) with respect to the Wiener process $(W_t^j)_{t\in[0,h]}$ and following the representation in [29, 30], we get the representation

$$\mathcal{I}_{(i,j)}(h) = \frac{1}{2}W_h^i W_h^j - \frac{1}{2}h\,\delta_{i,j} + A_{(i,j)}(h) \tag{11}$$

with the Lévy area

$$A_{(i,j)}(h) = \pi \sum_{r=1}^{\infty} r\left(a_r^i\left(b_r^j - \frac{1}{\pi r}W_h^j\right) - \left(b_r^i - \frac{1}{\pi r}W_h^i\right)a_r^j\right) \tag{12}$$

for $i,j \in \{1,\dots,m\}$. This series converges in $L^2(\Omega)$, see, e.g., [29, 30, 40]. In the following, we consider the whole matrix of all iterated stochastic integrals

$$\mathcal{I}(h) = \frac{1}{2}\left(W_h W_h^\top - h\,I_m\right) + A(h) \tag{13}$$

with identity matrix I_m and with the Lévy area matrix $A(h)$ which can be written as

$$A(h) = \sum_{r=1}^{\infty}\left(W_h a_r^\top - a_r W_h^\top\right) + \pi\sum_{r=1}^{\infty} r\left(a_r b_r^\top - b_r a_r^\top\right) \tag{14}$$

where $a_r = (a_r^i)_{1\le i\le m}$ and $b_r = (b_r^i)_{1\le i\le m}$. The basic approach to the approximation of the Lévy area consists in truncating the series (14). Then, one may additionally approximate some or even all of the arising rest terms in order to improve the approximation. If both sums in (14) are truncated at a point $p \in \mathbb{N}$, we denote the truncated Lévy area matrix as $A^{(p)}(h)$ and the rest terms as $R_1^{(p)}(h)$ and $R_2^{(p)}(h)$ where

$$A^{(p)}(h) := \pi\sum_{r=1}^{p} r\left(a_r\left(b_r - \frac{1}{\pi r}W_h\right)^\top - \left(b_r - \frac{1}{\pi r}W_h\right)a_r^\top\right), \tag{15}$$

$$R_1^{(p)}(h) := \sum_{r=p+1}^{\infty} W_h a_r^\top - a_r W_h^\top, \tag{16}$$

$$R_2^{(p)}(h) := \pi\sum_{r=p+1}^{\infty} r\left(a_r b_r^\top - b_r a_r^\top\right), \tag{17}$$

such that $A(h) = A^{(p)}(h) + R_1^{(p)}(h) + R_2^{(p)}(h)$. These terms build the basis for the simulation algorithms of the iterated stochastic integrals that will be discussed in the following sections.

2.3 Approximation vs. simulation

In this article, we restrict our considerations to the simulation problem of iterated stochastic integrals, which has to be distinguished from the corresponding approximation problem. Although all algorithms under consideration can also be applied for the approximation of iterated stochastic integrals, we don't go into details here and refer to, e.g., [41, 47, 53]. It is worth noting that for the approximation problem where one is interested to approximate some fixed realization of the iterated stochastic integrals together with the realization of the increments of the Wiener process, one needs some information about the realization of the path of the involved Wiener process. What kind of information is needed depends on the approximation algorithm to be applied. E.g., for a truncated Fourier series algorithm one may assume to have access to the realizations of the first n Fourier coefficients a_0^i, a_r^i and b_r^i for $i \in \{1, \ldots, m\}$ and $1 \leq r \leq n$ together with the realizations of the increments of the Wiener process. Then, using this information the goal is to calculate an approximation for the iterated stochastic integrals such that, e.g., the strong or L^2-error is as small as possible. However, for many problems one does not have this information about realizations but rather tries to model uncertainty by, e.g., random processes. In such situations, one is often interested in the simulation of possible scenarios that may occur. Therefore, no prescribed information is given a priori and one has to generate realizations of all involved random variables. This can be done easily whenever one can sample from some well known distribution. However, if the distribution of some random variable such as an iterated stochastic integral is not known, one has to sample from an approximate distribution. In contrast to weak or distributional approximation, one still has to care about, e.g., the strong or L^2-error between the simulated approximate realization and a corresponding hypothetical exact realization. This means that for each simulated realization one can find at least one corresponding posteriori realization such that a strong or L^2-error estimate with some prescribed precision is always fulfilled. Since in many or even nearly all practical situations information about uncertainty is not explicitly available, stochastic simulation algorithms are frequently used and therefore of course most relevant.

2.4 Simulating with a prescribed precision

In the following, we consider the problem of simulating realizations of the iterated stochastic integrals $\mathcal{I}(h) = (\mathcal{I}_{(i,j)}(h))_{1\leq i,j\leq m}$ provided that the increments $W(h) = (W_h^i)_{1\leq i\leq m}$ are given. The simulation of realizations of the stochastically independent increments W_h^i is straightforward since their distribution $W_h^i \sim \mathcal{N}(0, h)$ is known and sampling from a Gaussian distribution can be done easily in principle. However, for the simulation of the iterated stochastic integrals the situation is different because the random variables $\mathcal{I}_{(i,j)}(h)$ are not independent and as yet we don't know their joint

(conditional) distribution explicitly. Therefore, we need some approximate sampling method.

There exist different approaches to do this, and we restrict our attention to sampling algorithms where the L^2-error can be controlled. This is important, especially if such a simulation algorithm is combined with a numerical SDE or SPDE solver for the simulation of root mean square approximations with some prescribed precision.

To be precise, given some precision $\varepsilon > 0$ assume that our simulation algorithm gives us by sampling a realization of the increments $\hat{W}_h(\omega)$ and a realization of our approximation $\hat{\mathcal{I}}^\varepsilon(h)(\omega)$ for the iterated stochastic integrals for some $\omega \in \Omega$. Then, for all algorithms under consideration we can assume that there exist copies $\tilde{W}_h$ and $\tilde{\mathcal{I}}(h)$ of the random variables W_h and $\mathcal{I}(h)$ on the probability space $(\Omega, \mathcal{F}, \mathrm{P})$ such that $\tilde{W}_h$ together with $\tilde{\mathcal{I}}(h)$ have the same joint distribution as W_h together with $\mathcal{I}(h)$ and such that $\hat{W}_h(\omega), \hat{\mathcal{I}}^\varepsilon(h)(\omega)$ are an approximation of $\tilde{W}_h(\omega), \tilde{\mathcal{I}}(h)(\omega)$ for P-almost all $\omega \in \Omega$ in the sense that $\hat{W}_h = \tilde{W}_h$ P-a.s. and

$$\big\|\tilde{\mathcal{I}}_{(i,j)}(h) - \hat{\mathcal{I}}^\varepsilon_{(i,j)}(h)\big\|_{L^2(\Omega)} \leq \varepsilon$$

for $1 \leq i, j \leq m$. This is also known as a kind of coupling, see, e.g., [47] for details. For simplicity of notation, we do not distinguish between the random variables W_h, $\mathcal{I}(h)$ and their copies $\tilde{W}_h, \tilde{\mathcal{I}}(h)$ in the following.

2.5 Relationship between different error criteria

Usually, we are interested in simulating the whole matrix $\mathcal{I}(h)$ with some prescribed precision. Therefore, it is often appropriate to consider the L^2-error with respect to some suitable matrix norm depending on the problem to be simulated. Here, we focus our attention to the error with respect to the norms

$$\|M\|_{\max,L^2} = \max_{i,j} \|M_{i,j}\|_{L^2(\Omega)} \qquad \|M\|_{L^2,\max} = \mathrm{E}\big[\max_{i,j} |M_{i,j}|^2\big]^{1/2}$$
$$\|M\|_{\mathrm{F},L^2} = \Big(\sum_{i,j} \mathrm{E}\big[|M_{i,j}|^2\big]\Big)^{1/2} \qquad \|M\|_{L^2,\mathrm{F}} = \mathrm{E}\big[\|M\|_{\mathrm{F}}^2\big]^{1/2}$$

for some random matrix $M = (M_{i,j}) \in L^2(\Omega, \mathbb{R}^{m\times n})$ where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm. Clearly, the F, L^2-norm and the L^2, F-norm coincide in the L^2-setting. In general, these norms are equivalent with

$$\|M\|_{\max,L^2} \leq \|M\|_{L^2,\max} \leq \|M\|_{L^2,\mathrm{F}} \leq \sqrt{mn}\|M\|_{\max,L^2}.$$

In practice we mostly need the max, L^2-norm and the L^2, F-norm as these are the natural expressions that show up if numerical schemes for the approximation of SDEs and SPDEs are applied, respectively, see [29, Cor. 10.6.5] and [34].

Since the symmetric part of the iterated stochastic integrals can be calculated exactly, we concentrate on the skew-symmetric Lévy area part. Let $\mathrm{Skew}_m \subset \mathbb{R}^{m\times m}$ denote the vector space of real, skew-symmetric matrices. Then, $\mathcal{I}(h) - \hat{\mathcal{I}}^\varepsilon(h) = A(h) - \hat{A}^\varepsilon(h)$ where $\hat{A}^\varepsilon(h)$ is the corresponding approximation of the Lévy area matrix $A(h)$. Note that the elements of the Lévy area matrix are identically distributed and that $A(h), \hat{A}^\varepsilon(h) \in \mathrm{Skew}_m$. Therefore, the following more precise relationship between the matrix norms under consideration is used in the following.

Lemma 1 *Let $A \in L^2(\Omega, \mathrm{Skew}_m)$, i.e., $A(\omega) = -A(\omega)^\top$ for* P*-a.a. $\omega \in \Omega$ and $A_{i,j} \in L^2(\Omega)$ for all $1 \le i, j \le m$. Then, it holds*

$$\|A\|_{L^2,\mathrm{F}} \le \sqrt{m^2 - m}\, \|A\|_{\max,L^2}\,, \tag{18}$$

$$\sqrt{2}\, \|A\|_{L^2,\max} \le \|A\|_{L^2,\mathrm{F}}\,. \tag{19}$$

Additionally, equality holds in (18) *if the elements $A_{i,j}$ for $1 \le i, j \le m$ with $i \ne j$ have identical second absolute moments.*

Proof Since $A \in L^2(\Omega, \mathrm{Skew}_m)$ it holds $A_{i,i} = 0$ P-a.s. for all $1 \le i \le m$. Then, it follows

$$\|A\|^2_{L^2,\mathrm{F}} = \sum_{i,j} \mathrm{E}\big[|A_{i,j}|^2\big] \le \sum_{\substack{i,j \\ i \ne j}} \max_{k,l} \mathrm{E}\big[|A_{k,l}|^2\big] = (m^2 - m)\, \|A\|^2_{\max,L^2}\,.$$

If $\mathrm{E}\big[|A_{i,j}|^2\big]$ is constant for all $i \ne j$, then we have $\mathrm{E}\big[|A_{i,j}|^2\big] = \max_{k,l} \mathrm{E}\big[|A_{k,l}|^2\big]$ for $i \ne j$ and equality holds.

Finally, the second inequality follows easily due to

$$\|A\|^2_{L^2,\max} = \mathrm{E}\Big[\max_{i,j} |A_{i,j}|^2\Big] = \mathrm{E}\Big[\max_{\substack{i,j \\ i<j}} |A_{i,j}|^2\Big] \le \mathrm{E}\Big[\sum_{\substack{i,j \\ i<j}} |A_{i,j}|^2\Big] = \frac{1}{2}\mathrm{E}\Big[\sum_{i,j} |A_{i,j}|^2\Big].$$

□

Taking into account the relationship between the described matrix norms, we concentrate on the max, L^2- error of the simulated Lévy area

$$\|A(h) - \hat{A}^\varepsilon(h)\|_{\max,L^2} \le \varepsilon$$

for $\varepsilon > 0$ because we can directly convert between the different norms according to Lemma 1.

3 The algorithms for the simulation of Lévy areas

For the approximation of the iterated stochastic integrals $\mathcal{I}(h)$ in (13), one has to approximate the corresponding Lévy areas $A(h)$ in (14). A first approach might be to truncate the expansion of the Lévy areas at some appropriate index p as in (15) such that the truncation error is sufficiently small. This is the main idea for the first algorithm called Fourier algorithm which is presented in Section 3.1. In order to get some better approximation, one may take into account either some parts or even the whole tail sum as well. Adding the exact simulation of $R_1^{(p)}(h)$ in (16), which belongs to the Fourier coefficient a_0^i, results in the second algorithm presented in Section 3.2 which is due to Milstein [40]. In contrast to Milstein, in the seminal paper by Wiktorsson [53] the whole tail sum $R_1^{(p)}(h) + R_2^{(p)}(h)$ is approximated. The algorithm proposed by Wiktorsson is described in Section 3.3 and has some improved order of convergence compared to the first two algorithms. Finally, the recently proposed algorithm by Mrongowius and Rößler [41] combines the ideas of the algorithms due

to Milstein and Wiktorsson by simulating $R_1^{(p)}(h)$ exactly and by approximating the tail sum $R_2^{(p)}(h)$ in a similar way as in the algorithm by Wiktorsson. The algorithm by Mrongowius and Rößler is described in Section 3.4. In order to efficiently implement the two algorithms proposed by Wiktorsson and by Mrongowius and Rößler, the central idea is to replace the involved matrices by the actions they encode. This also helps avoiding computationally intensive Kronecker products. In the following we discuss in detail mathematically equivalent reformulations of these algorithms that finally lead to efficient implementations with significantly reduced computational complexity.

Since the symmetric part of the iterated integrals is known explicitly, all algorithms under consideration can be interpreted as algorithms for the simulation of the skew-symmetric part which is exactly the Lévy area $A(h)$. Moreover, due to the scaling relationship (3) it is sufficient to focus our attention on the simulation of the Lévy area with step size $h = 1$ because the general case can be obtained by rescaling, i.e., by multiplying the simulated Lévy area with the desired step size. In the following discussions of the different algorithms, we always assume that the increments W_h^i for $1 \leq i \leq m$, which are independent and $\mathcal{N}(0, h)$ distributed Gaussian random variables, are given.

3.1 The Fourier algorithm

What we call here the Fourier algorithm can be considered the 'easy' approach — not taking into account any approximation of the rest terms. This variant has not been getting much attention in the literature, since with very little additional effort the error can be reduced by a factor of $1/\sqrt{3}$, see, e.g., [13] and Section 4. However, since this is the foundation for all considered algorithms every implementation detail automatically benefits all algorithms. Thus we discuss it as its own algorithm.

3.1.1 Derivation of the Fourier algorithm

Let $p \in \mathbb{N}$ be given. Then, the approximation using the Fourier algorithm is defined as

$$\hat{A}^{\mathrm{FS},(p)}(h) \coloneqq A^{(p)}(h), \tag{20}$$

i.e., just the truncated Lévy area without any further rest approximations. Defining

$$\alpha_r^i = \sqrt{\frac{2\pi^2 r^2}{h}} a_r^i \quad \text{and} \quad \beta_r^i = \sqrt{\frac{2\pi^2 r^2}{h}} b_r^i \tag{21}$$

for $1 \leq i \leq m$ and $1 \leq r \leq p$ it follows that these random variables are independent and standard Gaussian distributed. The approximation in terms of these standard Gaussian random variables results in

$$\hat{A}^{\mathrm{FS},(p)}(h) = \frac{h}{2\pi} \sum_{r=1}^{p} \frac{1}{r} \left(\alpha_r \left(\beta_r - \sqrt{\frac{2}{h}} W_h \right)^\top - \left(\beta_r - \sqrt{\frac{2}{h}} W_h \right) \alpha_r^\top \right), \tag{22}$$

where $\alpha_r = (\alpha_r^i)_{1 \leq i \leq m}$ and $\beta_r = (\beta_r^i)_{1 \leq i \leq m}$ are the corresponding random vectors.

3.1.2 Implementation of the Fourier algorithm

Looking at (22), it is easy to see that the dyadic products in the series are needed twice due to the relationship $\left(\beta_r - \sqrt{\frac{2}{h}} W_h\right)\alpha_r^\top = \left(\alpha_r \left(\beta_r - \sqrt{\frac{2}{h}} W_h\right)^\top\right)^\top$. This means that any efficient implementation of the above approximation should only compute them once. Next, notice that this can be applied to the whole sum: instead of evaluating each term in the series and then adding them together it is more efficient to split the sum and evaluate it only once. Introducing

$$S^{\mathrm{FS},(p)} = \sum_{r=1}^{p} \frac{1}{r} \left(\alpha_r \left(\beta_r - \sqrt{\tfrac{2}{h}} W_h\right)^\top\right) \tag{23}$$

we can rewrite the Fourier approximation as

$$\hat{A}^{\mathrm{FS},(p)}(h) = \frac{h}{2\pi}\left(S^{\mathrm{FS},(p)} - S^{\mathrm{FS},(p)\top}\right). \tag{24}$$

By first calculating and storing $S^{\mathrm{FS},(p)}$ and then computing $\hat{A}^{\mathrm{FS},(p)}(h)$ one can save $(p-1)m^2$ basic arithmetic operations if (24) is applied instead of naively implementing (22). The algorithm based on (24) is given as pseudocode by Algorithm 1.

Next, we have a closer look at the calculation of $S^{\mathrm{FS},(p)}$. For an implementation that results in a fast computation of the iterated stochastic integrals, we want to exploit fast matrix multiplication routines as provided by, e.g., specialized "BLAS"[1] (Basic Linear Algebra Subprograms) implementations like OpenBLAS[2] or the Intel® Math Kernel Library[3]. In fact, if we gather the coefficient column vectors α_r and β_r for $1 \le r \le p$ into the two $m \times p$-matrices

$$\alpha = (\alpha_1 \mid \ldots \mid \alpha_p) \quad \text{and} \quad \tilde{\beta} = \left(\frac{\beta_1 - \sqrt{\frac{2}{h}} W_h}{1} \middle| \ldots \middle| \frac{\beta_p - \sqrt{\frac{2}{h}} W_h}{p} \right)$$

we can compute $S^{\mathrm{FS},(p)}$ as a matrix product $S^{\mathrm{FS},(p)} = \alpha\tilde{\beta}^\top$ utilizing the aforementioned fast multiplication algorithms. Of course, the drawback is that we have to store all coefficients in memory at the same time whereas before we only needed to store α_r and β_r of the current iteration for the summation.

There is also a middle ground between these two extremes – for some $n \in \{1, \ldots, p\}$ generate the $m \times n$-matrices $\alpha^{(k)}$ and $\beta^{(k)}$ for $1 \le k \le \lceil \frac{p}{n} \rceil$ by partitioning the columns of the matrices α and $\tilde{\beta}$ in the same way into $\lceil \frac{p}{n} \rceil$ groups $\alpha^{(k)}$ and $\tilde{\beta}^{(k)}$ for $1 \le k \le \lceil \frac{p}{n} \rceil$, each of maximal n vectors, and adding together sequentially to get $S^{\mathrm{FS},(p)} = \sum_{k=1}^{\lceil p/n \rceil} \alpha^{(k)} \tilde{\beta}^{(k)\top}$. Here, the firstly discussed extremes correspond to $n = 1$ and $n = p$, respectively.

[1] http://www.netlib.org/blas/
[2] https://www.openblas.net/
[3] https://software.intel.com/en-us/mkl

Require: W ▷ m-dimensional Wiener increment for $h = 1$ [vector of length m]
Require: p ▷ truncation parameter for tail sum [natural number]

function LEVYAREA_FOURIER(W, p)
  $\alpha \leftarrow \text{RANDN}(m, p)$ ▷ generate $m \times p$ Gaussian random numbers
  $\beta \leftarrow \text{RANDN}(m, p)$ ▷ generate $m \times p$ Gaussian random numbers
  for all columns β_r of β **do**
    $\beta_r \leftarrow \frac{1}{r}(\beta_r - \sqrt{2}W)$
  end for
  $S \leftarrow \alpha\beta^\top$ ▷ call BLAS gemm function here
  $A \leftarrow \frac{1}{2\pi}(S - S^\top)$
  return A ▷ return approximation of Lévy area $A(1)$
end function

Algorithm 1 Lévy area — Fourier method.

3.2 The Milstein algorithm

Now we proceed with the algorithm that Milstein first investigated in his influential work [40]. He called it the Fourier method, but he already incorporated a simple rest approximation compared to what we call the Fourier algorithm in Section 3.1. This algorithm has been generalized to multiple iterated integrals by Kloeden, Platen and Wright [30], see also [29]. Note that Dickinson [11] showed that this method is asymptotically optimal in the setting where only a finite number of linear functionals of the Wiener process are allowed to be used.

3.2.1 Derivation of the Milstein algorithm

For some prescribed $p \in \mathbb{N}$, the rest term approximation employed by Milstein [40] is concerned with the exact simulation of the term $R_1^{(p)}(h)$ given by (16). Utilizing the independence of the coefficients a_r^i and their distributional properties (10), the random vector $\sum_{r=p+1}^{\infty} a_r$ possesses a multivariate Gaussian distribution with zero mean and covariance matrix $\frac{h}{2\pi^2}\sum_{r=p+1}^{\infty}\frac{1}{r^2} I_m = \frac{h}{2\pi^2}\psi_1(p+1) I_m$. Here, ψ_1 denotes the trigamma function defined by $\psi_1(z) = \frac{\mathrm{d}^2}{\mathrm{d}z^2}\ln(\Gamma(z))$ with respect to the gamma function Γ which can also be represented as the series $\psi_1(z) = \sum_{n=0}^{\infty}\frac{1}{(z+n)^2}$. Thus, the vector γ_1 defined by

$$\gamma_1 = \sqrt{\frac{2\pi^2}{h\,\psi_1(p+1)}}\sum_{r=p+1}^{\infty} a_r \tag{25}$$

is $\mathcal{N}(0_m, I_m)$ distributed. Note that γ_1 is independent from W_h, from all β_k as well as from α_r for $r = 1, \ldots, p$. Now, one can rewrite the rest term as

$$R_1^{(p)}(h) = \sqrt{\frac{h\,\psi_1(p+1)}{2\pi^2}}\left(W_h\gamma_1^\top - \gamma_1 W_h^\top\right) \tag{26}$$

and the approximation of the Lévy area $A(h) \approx \hat{A}^{\mathrm{Mil},(p)}(h)$ with the Milstein algorithm is defined by

$$\hat{A}^{\mathrm{Mil},(p)}(h) = \hat{A}^{\mathrm{FS},(p)}(h) + R_1^{(p)}(h) \tag{27}$$

with $R_1^{(p)}(h)$ given by (26). We emphasize that the exact simulation of $R_1^{(p)}(h)$ is possible because the distribution of γ_1 is explicitly known to be multivariate standard Gaussian. Provided W_h is given, we have to simulate (24) and we additionally have to add the term $R_1^{(p)}(h)$ given in (26) for the Milstein algorithm in order to reduce the error. Note that the exact simulation of $R_1^{(p)}(h)$ by (26) does not improve the order of convergence. However, it provides a by a factor of $1/\sqrt{3}$ smaller error constant and thus, in most cases, less random variables are needed compared to the Fourier algorithm in order to guarantee some prescribed error bound.

3.2.2 Implementation of the Milstein algorithm

For the implementation of the Milstein algorithm we again separate the Lévy area approximation into two appropriate parts $S^{\mathrm{Mil},(p)}$ and $-S^{\mathrm{Mil},(p)\top}$. Then, the Milstein approximation is computed as

$$\hat{A}^{\mathrm{Mil},(p)}(h) = \frac{h}{2\pi}\left(S^{\mathrm{Mil},(p)} - S^{\mathrm{Mil},(p)\top}\right) \tag{28}$$

where looking at (26) and taking into account (23), we see that $S^{\mathrm{Mil},(p)}$ can be expressed as

$$S^{\mathrm{Mil},(p)} = S^{\mathrm{FS},(p)} + \sqrt{2\psi_1(p+1)}\,\frac{W_h}{\sqrt{h}}\,\gamma_1^{\top} \tag{29}$$

and thus can be easily simulated. Finally, the Milstein algorithm is presented as pseudocode in Algorithm 2.

Require: W ▷ m-dimensional Wiener increment for $h = 1$ [vector of length m]
Require: p ▷ truncation parameter for tail sum [natural number]

function LEVYAREA_MILSTEIN(W, p)
  $\alpha \leftarrow \mathrm{RANDN}(m, p)$ ▷ generate $m \times p$ Gaussian random numbers
  $\beta \leftarrow \mathrm{RANDN}(m, p)$ ▷ generate $m \times p$ Gaussian random numbers
  $\gamma_1 \leftarrow \mathrm{RANDN}(m)$ ▷ generate $m \times 1$ Gaussian random numbers
  for all columns β_r of β **do**
    $\beta_r \leftarrow \frac{1}{r}(\beta_r - \sqrt{2}W)$
  end for
  $S \leftarrow \alpha\beta^{\top}$ ▷ call BLAS gemm function here
  $S \leftarrow S + \sqrt{2\,\mathrm{TRIGAMMA}(p+1)}\,W\gamma_1^{\top}$ ▷ simple tail approximation
  $A \leftarrow \frac{1}{2\pi}\left(S - S^{\top}\right)$
  return A ▷ return approximation of Lévy area $A(1)$
end function

Algorithm 2 Lévy area — Milstein method.

3.3 The Wiktorsson algorithm

In the Milstein algorithm, the tail sum $R^{(p)}(h)$ is only partially taken into account which reduces the error constant but has no influence on the order of convergence w.r.t. the number of necessary realizations of independent Gaussian random variables. An improvement of the order of convergence has been first achieved by Wiktorsson [53]. He did not incorporate $R_1^{(p)}(h)$ on its own but instead tried to find an approximation of the whole tail sum $R^{(p)}(h)$ by analysing its conditional distribution in a new way. In this section, we briefly explain the main idea Wiktorsson used to derive his sophisticated approximation of the truncation terms. Therefore, we first introduce a suitable notation together with some useful identities. Wiktorsson's idea is also the basis for the algorithm proposed by Mrongowius and Rößler [41].

3.3.1 Vectorization and the Kronecker product representation

In the following, let $\mathrm{vec}\colon \mathbb{R}^{m\times m} \to \mathbb{R}^{m^2}$ be the column-wise vectorization operator for quadratic matrices and let $\mathrm{mat}\colon \mathbb{R}^{m^2} \to \mathbb{R}^{m\times m}$ be its inverse mapping such that $\mathrm{mat}(\mathrm{vec}(A)) = A$ for all matrices $A \in \mathbb{R}^{m\times m}$. Then, for the Kronecker product of two vectors $u, v \in \mathbb{R}^m$ it holds $v \otimes u = \mathrm{vec}(uv^\top)$. Further, let $P_m \in \mathbb{R}^{m^2\times m^2}$ be the permutation matrix that satisfies $P_m(u \otimes v) = v \otimes u$ for all $u, v \in \mathbb{R}^m$. Note that it can be explicitly constructed as $P_m = \sum_{i,j=1}^m e_i e_j^\top \otimes e_j e_i^\top$ where e_i denotes the ith canonical basis vector in $\mathbb{R}^m$. Then, it follows that

$$P_m \mathrm{vec}(A) = \mathrm{vec}(A^\top) \tag{30}$$

and we have the identity

$$(A \otimes I_m)\mathrm{vec}(B) = \mathrm{vec}(BA^\top) \tag{31}$$

for $A, B \in \mathbb{R}^{m\times m}$, see, e.g., [21, 38].

Since we are dealing with skew-symmetric matrices, it often suffices to consider the $M = \frac{m(m-1)}{2}$ entries below the diagonal. Therefore, Wiktorsson [53] applies the matrix $K_m \in \mathbb{R}^{M\times m^2}$ that selects the entries below the diagonal of a vectorized $m \times m$-matrix. This matrix can be explicitly defined by

$$K_m = \sum_{1\le j<i\le m} \tilde{e}_{i-\frac{j(j+1)}{2}+(j-1)m}(e_j^\top \otimes e_i^\top)$$

where $\tilde{e}_k$ is the kth canonical basis vector in $\mathbb{R}^M$. Note that the matrix K_m is clearly not invertible when considered as acting from $\mathbb{R}^{m^2}$ to $\mathbb{R}^M$, but it has $K_m^\top$ as a right inverse. If we restrict the domain of K_m to the M-dimensional subspace $\{\mathrm{vec}(A) : A \text{ strictly lower triangular}\} \subset \mathbb{R}^{m^2}$, then $K_m^\top$ is also a left inverse to K_m. That means, we can reconstruct at least all skew-symmetric matrices in the sense that

$$(I_{m^2} - P_m)K_m^\top K_m \mathrm{vec}(A) = \mathrm{vec}(A) \tag{32}$$

for any skew-symmetric $A \in \mathbb{R}^{m\times m}$. As a result of this, it holds that

$$(I_{m^2} - P_m)K_m^\top K_m (I_{m^2} - P_m) = I_{m^2} - P_m\,. \tag{33}$$

On the other hand, if we are given a vector $a \in \mathbb{R}^M$ we get the corresponding skew-symmetric matrix $A \in \mathbb{R}^{m\times m}$ using the mat-operator as

$$A = \mathrm{mat}\big((I_{m^2} - P_m)K_m^\top a\big)\,. \tag{34}$$

3.3.2 Derivation of the Wiktorsson algorithm

Due to the skew-symmetry of the Lévy area matrix $A(h)$, we can restrict our considerations to the vectorization of the lower triangle. In this way, following the idea of Wiktorsson [53] it is possible to analyse the conditional covariance structure of the resulting vector and to reduce the Lévy area approximation problem to the problem of finding a good approximation of the covariance matrix. Applying this approach to the tail sum $R^{(p)}(h)$ of the truncated series (15) and using (21), it holds that

$$\mathrm{vec}(R^{(p)}(h)) = \frac{h}{2\pi}\sum_{r=p+1}^{\infty}(P_m - I_{m^2})\left(\alpha_r \otimes \frac{1}{r}\left(\beta_r - \sqrt{\frac{2}{h}}W_h\right)\right), \tag{35}$$

which is an $\mathbb{R}^{m^2}$-valued random vector. Since $R^{(p)}(h)$ is skew-symmetric as well, it suffices to consider the M entries below the diagonal. Let $r^{(p)}(h) = K_m\mathrm{vec}(R^{(p)}(h))$ denote the M-dimensional random vector of the elements below the diagonal of $R^{(p)}(h)$. Now, for each p the random vector $\frac{\sqrt{2}\pi}{h\sqrt{\psi_1(p+1)}}r^{(p)}(h)$ is conditionally Gaussian with conditional expectation 0_M and some conditional covariance matrix $\Sigma^{(p)} = \Sigma^{(p)}(h)$ which depends on the random vectors $\beta_r, r \geq p+1$ and W_h. We use a slightly different scaling than Wiktorsson, because we think it better highlights the structure of the covariance matrices. In this setting the covariance matrix is given by

$$\begin{aligned}\Sigma^{(p)} = \frac{1}{2}K_m(I_{m^2}-P_m)\cdot\Big(\frac{1}{\psi_1(p+1)}\sum_{r=p+1}^{\infty}\frac{1}{r^2}\big(\beta_r-\sqrt{\frac{2}{h}}W_h\big)\big(\beta_r-\sqrt{\frac{2}{h}}W_h\big)^\top\otimes I_m\Big)\\ \cdot(I_{m^2} - P_m)K_m^\top\,.\end{aligned} \tag{36}$$

Therefore, there exists a standard normally distributed M-dimensional random vector γ such that

$$\sqrt{\Sigma^{(p)}}\,\gamma = \frac{\sqrt{2}\pi}{h\sqrt{\psi_1(p+1)}}\,r^{(p)}(h) \quad \text{P-a.s.} \tag{37}$$

As $p \to \infty$, these covariance matrices converge in the L^2, F-norm to the matrix

$$\Sigma^{\infty} = I_M + \frac{1}{h}K_m(I_{m^2} - P_m)(W_hW_h^\top \otimes I_m)(I_{m^2} - P_m)K_m^\top \tag{38}$$

that is independent of the random vectors β_r, see [53, Thm. 4.2]. Now, we can take the underlying $\mathcal{N}(0_M, I_M)$ distributed random vector γ from (37) in order to approximate the tail sum as

$$r^{(p)}(h) \approx \frac{h\sqrt{\psi_1(p+1)}}{\sqrt{2}\pi}\sqrt{\Sigma^{\infty}}\,\gamma \tag{39}$$

where the exact conditional covariance matrix $\Sigma^{(p)}$ in (37) is replaced by Σ^{∞}, which can be computed explicitly. Finally, we can approximate the corresponding skew-symmetric matrix $R^{(p)}(h)$ by the reconstructed matrix $\hat{R}^{\mathrm{Wik},(p)}(h)$ defined as

$$\hat{R}^{\mathrm{Wik},(p)}(h) = \mathrm{mat}\Big((I_{m^2} - P_m)K_m^\top \frac{h}{\sqrt{2\pi}}\sqrt{\psi_1(p+1)}\sqrt{\Sigma^{\infty}}\gamma\Big) \tag{40}$$

due to (34). The matrix square root of Σ^{∞} in the above formula can be explicitly calculated as

$$\sqrt{\Sigma^{\infty}} = \frac{\Sigma^{\infty} + \sqrt{1 + \frac{\|W_h\|^2}{h}}I_M}{1 + \sqrt{1 + \frac{\|W_h\|^2}{h}}}, \tag{41}$$

see [53, Thm. 4.1]. Thus, the approximation of the Lévy area $A(h) \approx \hat{A}^{\mathrm{Wik},(p)}(h)$ using the tail sum approximation as proposed by Wiktorsson is defined as

$$\hat{A}^{\mathrm{Wik},(p)}(h) = \hat{A}^{\mathrm{FS},(p)}(h) + \hat{R}^{\mathrm{Wik},(p)}(h) \tag{42}$$

with $\hat{A}^{\mathrm{FS},(p)}(h)$ and $\hat{R}^{\mathrm{Wik},(p)}(h)$ given by (20) and (40), respectively.

3.3.3 Implementation of the Wiktorsson algorithm

Having a closer look at the tail approximation (40), it is easy to see that it is not directly suitable for an efficient implementation. Multiple matrix-vector products involving matrices of size $m^2 \times m^2$ result in an algorithmic complexity of roughly $\mathcal{O}(m^4)$. Additionally the Kronecker product with an identity matrix as it appears in the representation of Σ^{∞}, from an algorithmic point of view, only serves to duplicate values. Among other things, this leads to unnecessary high memory requirements. Therefore, we derive a better representation using the properties of the vec-operator mentioned in Section 3.3.1.

As a first step, let us insert expression (38) for Σ^{∞} into the representation (41) of $\sqrt{\Sigma^{\infty}}$. Then, we get

$$\begin{aligned}\sqrt{\Sigma^{\infty}} &= \frac{I_M + \frac{1}{h}K_m(I_{m^2} - P_m)(W_hW_h^\top \otimes I_m)(I_{m^2} - P_m)K_m^\top + \sqrt{1 + \frac{\|W_h\|^2}{h}}I_M}{1 + \sqrt{1 + \frac{\|W_h\|^2}{h}}} \\ &= \frac{K_m(I_{m^2} - P_m)(W_hW_h^\top \otimes I_m)(I_{m^2} - P_m)K_m^\top}{h\Big(1 + \sqrt{1 + \frac{\|W_h\|^2}{h}}\Big)} + I_M. \end{aligned} \tag{43}$$

Using (33) and (40), we obtain that

$$\begin{aligned}\hat{R}^{\mathrm{Wik},(p)}(h) = \frac{h}{2\pi}\mathrm{mat}\Bigg((I_{m^2} - P_m)\Bigg(&\frac{(W_hW_h^\top \otimes I_m)(I_{m^2} - P_m)\sqrt{2\psi_1(p+1)}K_m^\top\gamma}{h\Big(1 + \sqrt{1 + \frac{\|W_h\|^2}{h}}\Big)} \\ &+\sqrt{2\psi_1(p+1)}K_m^\top\gamma\Bigg)\Bigg).\end{aligned} \tag{44}$$

Next, observing that γ is always immediately reshaped by $K_m^\top$ we define the random $m \times m$-matrix $\Gamma = \mathrm{mat}\big(K_m^\top\gamma\big)$ to be the reshaped matrix. As a result of this, it holds

$K_m^\top \gamma = \mathrm{vec}(\Gamma)$ with $\Gamma_{i,j} \sim \mathcal{N}(0,1)$ for $i > j$ and $\Gamma_{i,j} = 0$ for $i \leq j$. Now, we apply (31) as well as (30) in order to arrive at the final representation

$$\hat{R}^{\mathrm{Wik},(p)}(h) = \frac{h}{2\pi}\mathrm{mat}\Bigg((I_{m^2} - P_m)\mathrm{vec}\Bigg(\frac{\sqrt{2\psi_1(p+1)}(\Gamma - \Gamma^\top)W_h W_h^\top}{h\Big(1+\sqrt{1+\frac{\|W_h\|^2}{h}}\Big)} + \sqrt{2\psi_1(p+1)}\Gamma\Bigg)\Bigg). \tag{45}$$

With (23) we define

$$S^{\mathrm{Wik},(p)} = S^{\mathrm{FS},(p)} + \frac{\sqrt{2\psi_1(p+1)}(\Gamma - \Gamma^\top)}{1+\sqrt{1+\frac{\|W_h\|^2}{h}}} \cdot \frac{W_h W_h^\top}{h} + \sqrt{2\psi_1(p+1)}\Gamma \tag{46}$$

which allows for an efficient implementation of Wiktorsson's algorithm using

$$\hat{A}^{\mathrm{Wik},(p)}(h) = \frac{h}{2\pi}\Big(S^{\mathrm{Wik},(p)} - S^{\mathrm{Wik},(p)\top}\Big). \tag{47}$$

This version of Wiktorsson's algorithm that is optimized for an efficient implementation can be found in Algorithm 3 presented as pseudocode.

Require: W ▷ m-dimensional Wiener increment for $h = 1$ [vector of length m]
Require: p ▷ truncation parameter for tail sum [natural number]

function LEVYAREA_WIKTORSSON(W, p)
  $\alpha \leftarrow$ RANDN(m, p) ▷ generate $m \times p$ Gaussian random numbers
  $\beta \leftarrow$ RANDN(m, p) ▷ generate $m \times p$ Gaussian random numbers
  $\Gamma \leftarrow$ RANDN_TRIL(m, m) ▷ gen. strictly lower triangular matrix of $\frac{m(m-1)}{2}$ Gaussian r.n.
  for all columns β_r of β **do**
    $\beta_r \leftarrow \frac{1}{r}(\beta_r - \sqrt{2}W)$
  end for
  $S \leftarrow \alpha\beta^\top$ ▷ call BLAS gemm function here
  $\Gamma \leftarrow \sqrt{2\,\mathrm{TRIGAMMA}(p+1)}\,\Gamma$
  $S \leftarrow S + \big(1+\sqrt{1+\|W\|^2}\big)^{-1}(\Gamma - \Gamma^\top)WW^\top + \Gamma$
  $A \leftarrow \frac{1}{2\pi}(S - S^\top)$
  return A ▷ return approximation of Lévy area $A(1)$
end function

Algorithm 3 Lévy area — Wiktorsson method.

3.4 The Mrongowius–Rößler algorithm

Compared to the Fourier algorithms that make use of linear functionals of the Wiener processes only, the algorithm proposed by Wiktorsson [53] reduces the computational cost significantly by including also a kind of non-linear information of the Wiener

processes for the additional approximation of the remainder terms. However, Wiktorsson's algorithm needs the calculation of the square root of a $M \times M$ covariance matrix, which can be expensive for high-dimensional problems. The following algorithm that has been recently proposed by Mrongowius and Rößler [41] improves the algorithm by Wiktorsson such that the error estimate allows for a smaller constant by a factor $1/\sqrt{5}$, see Section 4. In addition, the covariance matrix used in the algorithm by Mrongowius and Rößler is just the identity matrix and thus independent of W_h. Compared to Wiktorsson's algorithm, this makes the calculation of the square root of the full covariance matrix Σ^∞ depending on W_h in (43) obsolete. This allows for saving computational cost, especially in the context of the simulation of SDE and SPDE solutions where the value of W_h varies each time step. Thus, the algorithm by Mrongowius and Rößler saves computational cost and allows for an easier implementation that is discussed in the following.

3.4.1 Derivation of the Mrongowius–Rößler algorithm

Instead of approximating the whole rest term $R^{(p)}(h)$ at once as in Wiktorsson's algorithm, Mrongowius and Rößler combine the exact approximation of the rest term $R_1^{(p)}(h)$ in (26) with an approximation of the second rest term $R_2^{(p)}(h)$ that is related to Wiktorsson's approach. For the approximation of $R_2^{(p)}(h)$, we rewrite this term as

$$\operatorname{vec}(R_2^{(p)}(h)) = \frac{h}{2\pi} \sum_{r=p+1}^{\infty} \frac{1}{r} (P_m - I_{m^2})(\alpha_r \otimes \beta_r) \tag{48}$$

and we define the M-dimensional vector $r_2^{(p)}(h) = K_m \operatorname{vec}(R_2^{(p)}(h))$. Then, the random vector $\frac{\sqrt{2}\pi}{h\sqrt{\psi_1(p+1)}} r_2^{(p)}(h)$ is conditionally Gaussian with conditional expectation 0_M and conditional covariance matrix $\Sigma_2^{(p)}$ that depends on $\alpha_r, r \geq p+1$. Here, the covariance matrix is given by (cf. (36))

$$\Sigma_2^{(p)} = \frac{1}{2} K_m (I_{m^2} - P_m) \Big(I_m \otimes \frac{1}{\psi_1(p+1)} \sum_{r=p+1}^{\infty} \frac{1}{r^2} \alpha_r \alpha_r^\top \Big) (I_{m^2} - P_m) K_m^\top . \tag{49}$$

As a result of this, it holds

$$\frac{\sqrt{2}\pi}{h\sqrt{\psi_1(p+1)}} r_2^{(p)}(h) = \sqrt{\Sigma_2^{(p)}}\, \gamma_2$$

with some $\mathcal{N}(0_M, I_M)$ distributed random vector γ_2 that is given by

$$\gamma_2 = \frac{\sqrt{2}\pi}{h\sqrt{\psi_1(p+1)}} \Big(\Sigma_2^{(p)} \Big)^{-1/2} r_2^{(p)}(h) . \tag{50}$$

Mrongowius and Rößler showed that these covariance matrices converge as $p \to \infty$ in the L^2, F-norm to the very simple, constant identity matrix [41, Prop. 4.2]

$$\Sigma_2^\infty = I_M . \tag{51}$$

Next, the approximation of the remainder term $r_2^{(p)}(h)$ based on the approximate covariance matrix Σ_2^∞ is calculated as

$$\hat{r}_2^{(p)}(h) = \frac{h}{\sqrt{2}\pi}\sqrt{\psi_1(p+1)}\left(\Sigma_2^\infty\right)^{1/2}\gamma_2 = \frac{h}{\sqrt{2}\pi}\sqrt{\psi_1(p+1)}\gamma_2 \tag{52}$$

where γ_2 is the $\mathcal{N}(0_M, I_M)$ distributed random vector in (50). From this vector we can rebuild the full $m \times m$ matrix as shown in (34) by

$$\hat{R}_2^{\mathrm{MR},(p)}(h) = \mathrm{mat}\Big((I_{m^2} - P_m)K_m^\top \hat{r}_2^{(p)}(h)\Big). \tag{53}$$

Now, the approximation of the Lévy area by the Mrongowius–Rößler algorithm is defined by the standard Fourier approximation combined with two rest term approximations

$$\hat{A}^{\mathrm{MR},(p)}(h) = \hat{A}^{\mathrm{FS},(p)}(h) + R_1^{(p)}(h) + \hat{R}_2^{\mathrm{MR},(p)}(h) \tag{54}$$

with $\hat{A}^{\mathrm{FS},(p)}(h)$, $R_1^{(p)}(h)$ and $\hat{R}_2^{\mathrm{MR},(p)}(h)$ defined in (20), (26) and (53), respectively.

3.4.2 Implementation of the Mrongowius–Rößler algorithm

Similar to Section 3.3.3, we simplify the expression (53) by eliminating the matrices P_m and K_m as well as the costly reshaping operation. This can be done in a much easier way due to the simple structure of the covariance matrix for this algorithm. First, we define $\Gamma_2 = \mathrm{mat}\big(K_m^\top \gamma_2\big)$ to be the lower triangular matrix corresponding to γ_2. Then, it follows that

$$\hat{R}_2^{\mathrm{MR},(p)}(h) = \frac{h}{2\pi}\sqrt{2\psi_1(p+1)}\left(\Gamma_2 - \Gamma_2^\top\right). \tag{55}$$

Now, with $S^{\mathrm{FS},(p)}$ in (23) we can define

$$S^{\mathrm{MR},(p)} = S^{\mathrm{FS},(p)} + \sqrt{2\psi_1(p+1)}\,\frac{W_h}{\sqrt{h}}\,\gamma_1^\top + \sqrt{2\psi_1(p+1)}\,\Gamma_2 \tag{56}$$

and then we finally arrive at

$$\hat{A}^{\mathrm{MR},(p)}(h) = \frac{h}{2\pi}\left(S^{\mathrm{MR},(p)} - S^{\mathrm{MR},(p)\top}\right) \tag{57}$$

that allows for an efficient implementation. This formulation of the algorithm due to Mrongowius and Rößler is also given in Algorithm 4 as pseudocode.

4 Error estimates and simulation of iterated stochastic integrals

The considered algorithms differ significantly with respect to their accuracy and computational cost. Therefore, we first compare the corresponding approximation error estimates in the max, L^2-norm. For $h > 0$ and $p \in \mathbb{N}$, let

$$\hat{\mathcal{I}}^{\mathrm{Alg},(p)}(h) = \frac{W_h W_h^\top - hI_m}{2} + \hat{A}^{\mathrm{Alg},(p)}(h) \tag{58}$$

Require: W ▷ m-dimensional Wiener increment for $h = 1$ [vector of length m]
Require: p ▷ truncation parameter for tail sum [natural number]

function LEVYAREA_MRONGOWIUS_ROESSLER(W, p)
  $\alpha \leftarrow \text{RANDN}(m, p)$ ▷ generate $m \times p$ Gaussian random numbers
  $\beta \leftarrow \text{RANDN}(m, p)$ ▷ generate $m \times p$ Gaussian random numbers
  $\gamma_1 \leftarrow \text{RANDN}(m, 1)$ ▷ generate $m \times 1$ Gaussian random numbers
  $\Gamma_2 \leftarrow \text{RANDN\_TRIL}(m, m)$ ▷ gen. strictly lower triangular matrix of $\frac{m(m-1)}{2}$ Gaussian r.n.
  for all columns β_r of β **do**
    $\beta_r \leftarrow \frac{1}{r}(\beta_r - \sqrt{2}\,W)$
  end for
  $S \leftarrow \alpha\beta^\top$ ▷ call BLAS gemm function here
  $S \leftarrow S + \sqrt{2\,\text{TRIGAMMA}(p+1)}(W\gamma_1^\top + \Gamma_2)$
  $A \leftarrow \frac{1}{2\pi}(S - S^\top)$
  return A ▷ return approximation of Lévy area $A(1)$
end function

Algorithm 4 Lévy area — Mrongowius–Rößler method.

where 'Alg' denotes the corresponding algorithm that is applied for the approximation of the Lévy areas.

Theorem 2 *Let $h > 0$, $p \in \mathbb{N}$, let $\mathcal{I}$ denote the matrix of all iterated stochastic integrals as in (13) and let $\hat{\mathcal{I}}$ denote the corresponding approximations defined in (58).*

(i) *For the Fourier algorithm with $\hat{A}^{\mathrm{FS},(p)}(h)$ as in (22), it holds*

$$\|\mathcal{I}(h) - \hat{\mathcal{I}}^{\mathrm{FS},(p)}(h)\|_{\max,L^2} \le \sqrt{\frac{3}{2\pi^2}} \cdot \frac{h}{\sqrt{p}}\,. \tag{59}$$

(ii) *For the Milstein algorithm with $\hat{A}^{\mathrm{Mil},(p)}(h)$ as in (27), it holds*

$$\|\mathcal{I}(h) - \hat{\mathcal{I}}^{\mathrm{Mil},(p)}(h)\|_{\max,L^2} \le \sqrt{\frac{1}{2\pi^2}} \cdot \frac{h}{\sqrt{p}}\,. \tag{60}$$

(iii) *For the Wiktorsson algorithm with $\hat{A}^{\mathrm{Wik},(p)}(h)$ as in (42), it holds*

$$\|\mathcal{I}(h) - \hat{\mathcal{I}}^{\mathrm{Wik},(p)}(h)\|_{\max,L^2} \le \sqrt{\frac{5m}{12\pi^2}} \cdot \frac{h}{p}\,. \tag{61}$$

(iv) *For the Mrongowius–Rößler algorithm with $\hat{A}^{\mathrm{MR},(p)}(h)$ as in (54), it holds*

$$\|\mathcal{I}(h) - \hat{\mathcal{I}}^{\mathrm{MR},(p)}(h)\|_{\max,L^2} \le \sqrt{\frac{m}{12\pi^2}} \cdot \frac{h}{p}\,. \tag{62}$$

To the best of our knowledge, there is no reference for (59) yet. Therefore, we present the proof for completeness. The proof for error estimate (60) can be found in [29, 40]. The presented error estimate (61) follows from the original result in [53] and

is an improvement that we also prove in the following. Finally, for the error estimate (62) we refer to [41].

Proof (i) For the proof of (59), we first observe that $\mathrm{E}\big[\big|\mathcal{I}_{(i,i)}(h)-\hat{\mathcal{I}}^{\mathrm{FS},(p)}_{(i,i)}(h)\big|^2\big]=0$ for $i=1,\dots,m$. Further, noting that all a_k^i and b_l^j for $k,l\in\mathbb{N}$ and $i,j=1,\dots,m$ are i.i.d. Gaussian random variables with distribution given in (10) that are also independent from W_h^q for $q=1,\dots,m$, we calculate with (12) for $i\neq j$ that

$$\begin{aligned}\mathrm{E}\Big[\big|\mathcal{I}_{(i,j)}(h)-\hat{\mathcal{I}}^{\mathrm{FS},(p)}_{(i,j)}(h)\big|^2\Big]&=\mathrm{E}\Big[\Big|\pi\sum_{r=p+1}^{\infty}r\Big(a_r^i\Big(b_r^j-\frac{1}{\pi r}W_h^j\Big)-\Big(b_r^i-\frac{1}{\pi r}W_h^i\Big)a_r^j\Big)\Big|^2\Big]\\&=2\pi^2\sum_{r=p+1}^{\infty}r^2\,\mathrm{E}\Big[\big(a_r^i\big)^2\Big]\mathrm{E}\Big[\Big(b_r^j-\frac{1}{\pi r}W_h^j\Big)^2\Big]\\&=\frac{3h^2}{2\pi^2}\sum_{r=p+1}^{\infty}\frac{1}{r^2}\,.\end{aligned}\tag{63}$$

Then, the error estimate (59) follows with the estimate

$$\sum_{r=p+1}^{\infty}\frac{1}{r^2}\le\int_p^{\infty}\frac{1}{r^2}\,\mathrm{d}r=\frac{1}{p}\,.$$

(iii) In order to prove (61), we proceed analogously to [41]. Instead of [53, Lemma 4.1] we employ the variant [41, Lemma 4.3] and note that one can prove the following stronger version of [53, Theorem 4.2] similar to [41, Proposition 4.2]

$$\begin{aligned}&\sum_{q=1}^{M}\mathrm{E}\Big[\big|(\Sigma^{(p)}-\Sigma^{\infty})_{r,q}\big|^2\,\Big|\,W_h\Big]\\&\quad=\frac{\sum_{k=p+1}^{\infty}1/k^4}{(2\sum_{k=p+1}^{\infty}1/k^2)^2}\cdot 2\Big(m+m\frac{(W_h^i)^2}{h}+m\frac{(W_h^j)^2}{h}+2\frac{\|W_h\|^2}{h}\Big)\end{aligned}$$

for all $1\le r\le M$ where $1\le j<i\le m$ with $(j-1)(m-\frac{j}{2})+i-j=r$. Recognizing that $\mathrm{E}\Big[m+m\frac{(W_h^i)^2}{h}+m\frac{(W_h^j)^2}{h}+2\frac{\|W_h\|^2}{h}\Big]=5m$ and using that $\frac{\sum_{k=p+1}^{\infty}1/k^4}{(\sum_{k=p+1}^{\infty}1/k^2)^2}\le\frac{1}{3p}$ (see also the proof of [53, Theorem 4.2]) completes the proof.

□

Here, it has to be pointed out that the Milstein algorithm is asymptotically optimal in the class of algorithms that only make use of linear functionals of the Wiener processes like, e.g., the Fourier algorithm, see [11]. Algorithms that belong to this class have also been considered in the seminal paper by Clark and Cameron, see [7], where as a consequence it is shown that the same order of convergence as for the Milstein algorithm can be obtained if the Euler–Maruyama scheme is applied for approximating the iterated stochastic integrals. On the other hand, the algorithms by Wiktorsson

and by Mrongowius and Rößler do not belong to this class as they make also use of non-linear information about the Wiener process. That is why these two algorithms allow for a higher order of convergence w.r.t. the parameter p. While the algorithm by Wiktorsson is based on the Fourier algorithm approach, the improved algorithm by Mrongowius and Rößler is build on the asymptotically optimal approach of the Milstein algorithm. As a result of this, the algorithm by Mrongowius and Rößler allows for a smaller error constant by a factor $1/\sqrt{5}$ compared to the Wiktorsson algorithm.

Next to the error estimates in $L^2(\Omega)$-norm presented in Theorem 2, it is worth mentioning that there also exist error estimates in general $L^q(\Omega)$-norm with $q > 2$ for the Milstein algorithm and for the Mrongowius–Rößler algorithm, see [41, Prop. 4.6, Thm. 4.8].

Require: W_h ▷ m-dimensional Wiener increment [vector of length m]
Require: h ▷ step size of the increment [scalar]
Require: p ▷ truncation parameter for tail sum [natural number]

function ITERATED_INTEGRALS(W_h, h, p) ▷ based on the error estimates
  $W_{\text{std}} \leftarrow \frac{1}{\sqrt{h}} W_h$ ▷ standardisation
  $A \leftarrow$ LEVY_AREA(W_{std}, p) ▷ call algorithm for Lévy area approximation
  $\mathcal{I}_{\text{std}} \leftarrow \frac{1}{2} W_{\text{std}} W_{\text{std}}^\top - \frac{1}{2} I_m + A$
  $\mathcal{I} \leftarrow h \cdot \mathcal{I}_{\text{std}}$
  return $\mathcal{I}$ ▷ return matrix with iterated stoch. integrals
end function

Algorithm 5 Calculation of the iterated integrals.

For the simulation of the iterated stochastic integrals $\hat{\mathcal{I}}^{\text{Alg},(p)}$ by making use of (58) and by applying one of the presented algorithms (Alg) in Sections 3.1–3.4 for the approximate simulation of the Lévy areas we refer to the pseudocode of Algorithm 5. That is, given a realization of an increment of the m-dimensional Wiener process W_h w.r.t. step size $h > 0$ and a value of the truncation parameter $p \in \mathbb{N}$ for the tail sum of the Lévy area approximation as input parameters, Algorithm 5 calculates and returns the matrix $\hat{\mathcal{I}}^{\text{Alg},(p)}(h)$ based on the algorithm LEVY_AREA() for the Lévy area approximation that has to be replaced by the choice of any of the algorithms described in Sections 3.1–3.4.

If iterated stochastic integrals are needed for numerical approximations of, e.g., solutions of SDEs in the root mean square sense, then one has to simulate these iterated stochastic integrals with sufficient accuracy in order to preserve the overall convergence rate of the approximation scheme. For SDEs, let $\gamma > 0$ denote the order of convergence in the L^2-norm of the numerical scheme under consideration, e.g., $\gamma = 1$ for the well known Milstein scheme [29, 40] or a corresponding stochastic Runge–Kutta scheme [46]. Then, the required order of convergence for the approximation of some random variable like, e.g., an iterated stochastic integral, that occurs in a given numerical scheme for SDEs such that its order γ is preserved has been established in [40, Lem. 6.2] and [29, Cor. 10.6.5]. Since it holds

$\mathrm{E}\big[\mathcal{I}_{(i,j)}(h) - \hat{\mathcal{I}}_{(i,j)}^{\mathrm{Alg},(p)}(h)\big] = 0$ for $1 \le i, j \le m$ for all algorithms under consideration in Section 3, the iterated stochastic integral approximations need to fulfil

$$\big\|\mathcal{I}_{(i,j)}(h) - \hat{\mathcal{I}}_{(i,j)}^{\mathrm{Alg},(p)}(h)\big\|_{L^2(\Omega)} \le c_1\, h^{\gamma+\frac{1}{2}} \tag{64}$$

for all $1 \le i, j \le m$, all sufficiently small $h > 0$ and some constant $c_1 > 0$ independent of h. This condition can equivalently be expressed using the max, L^2-norm as

$$\big\|\mathcal{I}(h) - \hat{\mathcal{I}}^{\mathrm{Alg},(p)}(h)\big\|_{\max,L^2} \le c_1\, h^{\gamma+\frac{1}{2}}\,. \tag{65}$$

Together with Theorem 2 this condition specifies how to choose the truncation parameter p.

For the Fourier algorithm (Algorithm 1) or the Milstein algorithm (Algorithm 2) we have to choose p such that

$$p \ge \frac{c_2}{c_1^2} \cdot h^{1-2\gamma} \in \mathcal{O}\big(h^{1-2\gamma}\big) \tag{66}$$

with $c_2 = \frac{3}{2\pi^2}$ for the Fourier algorithm and $c_2 = \frac{1}{2\pi^2}$ for the Milstein algorithm.

On the other hand, for the Wiktorsson algorithm (Algorithm 3) or the Mrongowius–Rößler algorithm (Algorithm 4) we can choose p such that

$$p \ge \frac{\sqrt{c_3}}{c_1} \cdot h^{\frac{1}{2}-\gamma} \in \mathcal{O}\big(h^{\frac{1}{2}-\gamma}\big) \tag{67}$$

where $c_3 = \frac{5m}{12\pi^2}$ for the Wiktorsson algorithm and $c_3 = \frac{m}{12\pi^2}$ for the Mrongowius–Rößler algorithm.

As a result of this, we have to choose $p \in \mathcal{O}\big(h^{-1}\big)$ if Algorithm 1 or Algorithm 2 is applied and $p \in \mathcal{O}\big(h^{-\frac{1}{2}}\big)$ if Algorithm 3 or Algorithm 4 is applied in the Milstein scheme or any other numerical scheme for SDEs which has an order of convergence $\gamma = 1$ in the L^2-norm and thus also strong order of convergence $\gamma = 1$. Hence, the algorithm by Wiktorsson as well as the algorithm by Mrongowius and Rößler typically need a significantly lower value of the truncation parameter p compared to the Fourier algorithm and the Milstein algorithm. Additionally, for the algorithm by Mrongowius and Rößler the parameter p can be chosen to be by a factor $1/\sqrt{5}$ smaller than for the algorithm by Wiktorsson.

5 Comparison of the algorithms and their performance

In this section we will look at the costs of each algorithm to achieve a given precision. In particular we will see how to choose the optimal parameters for each algorithm in a given setting. After that we show the results of a simulation study to confirm the theoretical order of convergence for each algorithm.

Table 1 Number of random numbers that need to be generated in terms of the dimension m of the Wiener process and the truncation parameter p

Algorithm	# $\mathcal{N}(0,1)$ random numbers
Fourier	$2pm$
Milstein	$2pm + m$
Wiktorsson	$2pm + \frac{m^2-m}{2}$
Mrongowius–Rößler	$2pm + \frac{m^2-m}{2} + m$

5.1 Comparison of computational cost

We will measure the computational cost in terms of the number of random numbers drawn from a standard Gaussian distribution that have to be generated for the simulation of all twice iterated stochastic integrals corresponding to one given increment of an underlying m-dimensional Wiener process. For a fixed truncation parameter p these can be easily counted in the derivation or in the pseudocode of the algorithms presented above. E.g., Algorithm 1 needs $2pm$ random numbers corresponding to the Fourier coefficients α_r^i and β_r^i for $1 \leq r \leq p$ and $1 \leq i \leq m$. For each algorithm these costs are summarized in Table 1.

In practice however, a fixed precision $\bar{\varepsilon}$ is usually given with respect to some norm for the error and we are interested in the minimal cost to simulate the iterated stochastic integrals to this precision. For this we need to choose the truncation parameter $p = p(m, h, \bar{\varepsilon})$ that may depend on the dimension m, the step size h and the desired precision $\bar{\varepsilon}$, as discussed in Section 4, and insert this expression for p into the cost in Table 1. Again, considering for example the Fourier algorithm (Algorithm 1), we calculate from (59) that the truncation parameter has to be chosen as $p \geq \frac{3h^2}{2\pi^2\bar{\varepsilon}^2}$ if we measure the error in the max, L^2-norm. This allows us to calculate the actual cost to be $c(m, h, \bar{\varepsilon}) = 2pm = \frac{3h^2 m}{\pi^2\bar{\varepsilon}^2}$. If instead we want to bound the error in the L^2, F-norm, we have to choose $p \geq \frac{3h^2(m^2-m)}{2\pi^2\bar{\varepsilon}^2}$ with the cost of $c(m, h, \bar{\varepsilon}) = \frac{3h^2(m^3-m^2)}{\pi^2\bar{\varepsilon}^2}$.

On the other hand, sometimes we are willing to invest a specific amount of computation time or cost and are interested in the minimal error we can achieve without exceeding this given cost budget. Rearranging the expression from above for the cost, we find that given some fixed cost budget $\bar{c} > 0$ we can get an upper bound on the max, L^2-error for the Fourier algorithm of $\varepsilon(m, h, \bar{c}) = \frac{\sqrt{3}h\sqrt{m}}{\pi\sqrt{\bar{c}}}$. In Table 2, we list the lower bound for the truncation parameter $p(m, h, \bar{\varepsilon})$, the resulting cost $c(m, h, \bar{\varepsilon})$ and the minimal achievable error $\varepsilon(m, h, \bar{c})$ for a prescribed error $\bar{\varepsilon}$ or a given cost budget $\bar{c}$ for all four algorithms and two different error criteria introduced in Section 2.5.

Now we can calculate for concrete situations the cost of each algorithm and determine the cheapest one. Consider again the example of a numerical scheme for SDEs with L^2-convergence of order $\gamma = 1$. In this situation we have seen before that we have to couple the error and the step size as $\bar{\varepsilon} = \mathcal{O}(h^{3/2})$ (cf. (65)). Using this coupling the cost only depends on the dimension, the step size and the chosen error norm. In Fig. 1 we choose $\bar{\varepsilon} = h^{3/2}$ and we visualize for two of the random matrix norms

Table 2 Comparison of theoretical properties of the Algorithms 1–4 in Section 3. The cutoff $p(m, h, \bar{\varepsilon})$ describes how to choose the truncation parameter p in order to guarantee that the error in the corresponding norm is less than $\bar{\varepsilon}$. The cost $c(m, h, \bar{\varepsilon})$ is the number of random numbers that need to be generated to achieve this error. Finally, the error $\varepsilon(m, h, \bar{c})$ is the minimal achievable error given a fixed cost budget $\bar{c}$. Note that m and h are the dimension of the Wiener process and the step size, respectively

Algorithm	Objective	Applied error criterion	
		$\|\cdot\|_{\max,L^2}$	$\|\cdot\|_{L^2,\mathrm{F}}$
Fourier			
	cutoff $p(m,h,\bar{\varepsilon})$	$\frac{3h^2}{2\pi^2\bar{\varepsilon}^2}$	$\frac{3h^2(m^2-m)}{2\pi^2\bar{\varepsilon}^2}$
	cost $c(m,h,\bar{\varepsilon})$	$\frac{3h^2m}{\pi^2\bar{\varepsilon}^2}$	$\frac{3h^2(m^3-m^2)}{\pi^2\bar{\varepsilon}^2}$
	error $\varepsilon(m,h,\bar{c})$	$\frac{\sqrt{3}}{\pi}\cdot\frac{h\cdot\sqrt{m}}{\sqrt{\bar{c}}}$	$\frac{\sqrt{3}}{\pi}\cdot\frac{h\cdot m\sqrt{m-1}}{\sqrt{\bar{c}}}$
Milstein			
	cutoff $p(m,h,\bar{\varepsilon})$	$\frac{h^2}{2\pi^2\bar{\varepsilon}^2}$	$\frac{h^2(m^2-m)}{2\pi^2\bar{\varepsilon}^2}$
	cost $c(m,h,\bar{\varepsilon})$	$\frac{h^2m}{\pi^2\bar{\varepsilon}^2}+m$	$\frac{h^2(m^3-m^2)}{\pi^2\bar{\varepsilon}^2}+m$
	error $\varepsilon(m,h,\bar{c})$	$\frac{1}{\pi}\cdot\frac{h\cdot\sqrt{m}}{\sqrt{\bar{c}-m}}$	$\frac{1}{\pi}\cdot\frac{h\cdot m\sqrt{m-1}}{\sqrt{\bar{c}-m}}$
Wiktorsson			
	cutoff $p(m,h,\bar{\varepsilon})$	$\frac{\sqrt{5}h\sqrt{m}}{\sqrt{12}\pi\bar{\varepsilon}}$	$\frac{\sqrt{5}hm\sqrt{m-1}}{\sqrt{12}\pi\bar{\varepsilon}}$
	cost $c(m,h,\bar{\varepsilon})$	$\frac{\sqrt{5}hm^{3/2}}{\sqrt{3}\pi\bar{\varepsilon}}+\frac{m^2-m}{2}$	$\frac{\sqrt{5}hm^2\sqrt{m-1}}{\sqrt{3}\pi\bar{\varepsilon}}+\frac{m^2-m}{2}$
	error $\varepsilon(m,h,\bar{c})$	$\frac{2\sqrt{5}}{\sqrt{3}\pi}\cdot\frac{h\cdot m^{3/2}}{2\bar{c}-m^2+m}$	$\frac{2\sqrt{5}}{\sqrt{3}\pi}\cdot\frac{h\cdot m^2\sqrt{m-1}}{2\bar{c}-m^2+m}$
Mrongowius–Rößler			
	cutoff $p(m,h,\bar{\varepsilon})$	$\frac{h\sqrt{m}}{\sqrt{12}\pi\bar{\varepsilon}}$	$\frac{hm\sqrt{m-1}}{\sqrt{12}\pi\bar{\varepsilon}}$
	cost $c(m,h,\bar{\varepsilon})$	$\frac{hm^{3/2}}{\sqrt{3}\pi\bar{\varepsilon}}+\frac{m^2+m}{2}$	$\frac{hm^2\sqrt{m-1}}{\sqrt{3}\pi\bar{\varepsilon}}+\frac{m^2+m}{2}$
	error $\varepsilon(m,h,\bar{c})$	$\frac{2}{\sqrt{3}\pi}\cdot\frac{h\cdot m^{3/2}}{2\bar{c}-m^2-m}$	$\frac{2}{\sqrt{3}\pi}\cdot\frac{h\cdot m^2\sqrt{m-1}}{2\bar{c}-m^2-m}$

introduced in Section 2.5 the algorithm with minimal cost subject to dimension and step size.

The first thing we see is that the Wiktorsson algorithm is never the best choice in these scenarios. This is because for the Mrongowius–Rößler algorithm the required value of the truncation parameter can be reduced by a factor $\sqrt{5}$ while only requiring m additional random numbers. Figure 1 confirms that this trade-off is always worth it, i.e., the Mrongowius–Rößler algorithm always outperforms the Wiktorsson algorithm.

Moreover, for the max, L^2-error we see in the left plot of Fig. 1 that if we fix some step size then there always exists some sufficiently high dimension such that it will be more efficient to apply the Milstein algorithm. This is a consequence of the higher order dependency on the dimension of the Mrongowius–Rößler algorithm (and also for the Wiktorsson algorithm) compared to the Milstein algorithm which dominates the error for large enough dimension if the step size is fixed. This means the Milstein algorithm is the optimal choice in this setting.

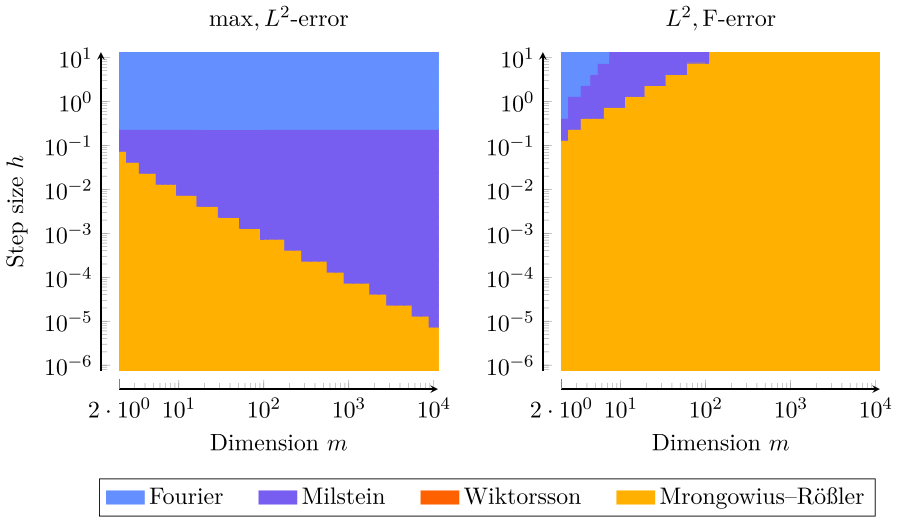

Fig. 1 Optimal algorithm given the dimension m and the step size h in the respective norm for $\bar{\varepsilon} = h^{3/2}$

However, for any fixed dimension of the Wiener process (as it is the case for, e.g., SDEs) there always exists some threshold step size such that for all step sizes smaller than this threshold the Mrongowius–Rößler algorithm is more efficient than the other algorithms. This is due to the higher order of convergence in time of the Mrongowius–Rößler algorithm. Thus, as the step size tends to zero or if the step size is sufficiently small, the Mrongowius–Rößler algorithm is the optimal choice.

The situation in the right plot of Fig. 1 for the L^2, F-error is different, because the simple scaling factor due to Lemma 1 does not translate into a simple scaling of the associated cost. Indeed it is no simple scaling at all if the cost contains additive terms independent of the error. Instead, only the error-dependent part of the cost is scaled by $m^2 - m$ for the Fourier and Milstein algorithms and by $\sqrt{m^2 - m}$ for the Wiktorsson and Mrongowius–Rößler algorithms (cf. Table 2). This is due to the different exponents of the truncation parameter in the corresponding error estimates. As a result of this, for the L^2, F-error the Mrongowius–Rößler algorithm is the optimal choice except for the particular case of a rather big step size and a low dimension of the Wiener process.

5.2 A study on the order of convergence

Convergence plots are an appropriate tool to complement theoretical convergence results with numerical simulations. Unfortunately, studying the convergence of random variables can be difficult, especially if the target (limit) random variable that has to be approximated can not be simulated exactly because its distribution is unknown. One way out of this problem is to substitute the target random variable by a highly accurate approximate random variable that allows to draw realizations from its known distribution.

In the present simulation study, we substitute the target random variable $\mathcal{I}(h)$ by the approximate random variable $\mathcal{I}^{\mathrm{ref},(p_{\mathrm{ref}})}(h) = \hat{\mathcal{I}}^{\mathrm{FS},(p_{\mathrm{ref}})}(h)$ based on the fundamental and accepted Fourier algorithm with high accuracy due to some large value for the parameter p_{ref}. Therefore, we refer to $\mathcal{I}^{\mathrm{ref},(p_{\mathrm{ref}})}(h)$ as the reference random variable and let $\hat{\mathcal{I}}^{\mathrm{Alg},(p)}(h)$ denote the approximation based on one of the algorithms described in Section 3. We focus on the max, L^2-error and the L^2, F-error of the approximation algorithms that we want to compare. We proceed as follows: Let $p_{\mathrm{ref}} \in \mathbb{N}$ be sufficiently large. First, some realizations of W_h and the reference random variable $\mathcal{I}^{\mathrm{ref},(p_{\mathrm{ref}})}(h)$ are simulated. Then, the corresponding realizations of the Fourier coefficients α_r and β_r for $1 \leq r \leq p_{\mathrm{ref}}$ are stored together with W_h. Next, the approximations $\hat{\mathcal{I}}^{\mathrm{Alg},(p)}(h)$ are calculated based on the same realization, which has to be done carefully. The stored Fourier coefficients and the stored increment of the Wiener process are used to compute approximate solutions using each of the algorithms with a truncation parameter $p \ll p_{\mathrm{ref}}$. Especially, we are able to exactly extract from the stored Fourier coefficients and the stored increment of the Wiener process the corresponding realizations of the underlying standard Gaussian random variables that are used in Wiktorsson's algorithm as well as in the Mrongowius–Rößler algorithm to approximate the tail.

To be precise, set $h = 1$ and choose a large value p_{ref} for the reference random variable, for example $p_{\mathrm{ref}} = 10^6$. Now, simulate and store the Wiener increment W_h as well as the Fourier coefficients α_r^i and β_r^i for $i = 1, \dots, m$ and $r = 1, \dots, p_{\mathrm{ref}}$. Then, calculate the reference random variable $\mathcal{I}^{\mathrm{ref},(p_{\mathrm{ref}})}(h)$ using the Fourier algorithm. Since every realization of the reference random variable $\mathcal{I}^{\mathrm{ref}}(h)$ is also an approximation to some realization of the random variable $\mathcal{I}(h)$, we can control the precision in max, L^2-norm of the reference random variable exactly due to (63), i.e., it holds

$$\left\| \mathcal{I}(h) - \mathcal{I}^{\mathrm{ref},(p_{\mathrm{ref}})}(h) \right\|_{\mathrm{max},L^2} = \left(\frac{3h^2}{2\pi^2} \left(\frac{\pi^2}{6} - \sum_{r=1}^{p_{\mathrm{ref}}} \frac{1}{r^2} \right) \right)^{\frac{1}{2}}.$$

For a large enough value of p_{ref} this error will be sufficiently small such that it can be neglected in our considerations. This justifies to work with the reference random variable.

Next, choose a truncation value $p \ll p_{\mathrm{ref}}$ for the approximation under consideration. First, calculate $\hat{\mathcal{I}}^{\mathrm{FS},(p)}(h)$ by the Fourier algorithm using only the first p stored Fourier coefficients. Then, extract the necessary standard Gaussian vectors from the remaining stored Fourier coefficients using (25), (35), (37) and (50), i.e., calculate

$$\gamma_1 = \frac{1}{\sqrt{\psi_1(p+1)}} \sum_{r=p+1}^{p_{\mathrm{ref}}} \frac{1}{r} \alpha_r \,,$$

$$\gamma = \frac{1}{\sqrt{2\psi_1(p+1)}} \left(\Sigma^{(p)} \right)^{-1/2} K_m (P_m - I_{m^2}) \sum_{r=p+1}^{p_{\mathrm{ref}}} \frac{1}{r} \left(\alpha_r \otimes \left(\beta_r - \sqrt{2} W_h \right) \right),$$

$$\gamma_2 = \frac{1}{\sqrt{2\psi_1(p+1)}} \left(\Sigma_2^p \right)^{-1/2} K_m (P_m - I_{m^2}) \sum_{r=p+1}^{p_{\mathrm{ref}}} \frac{1}{r} (\alpha_r \otimes \beta_r) \,.$$

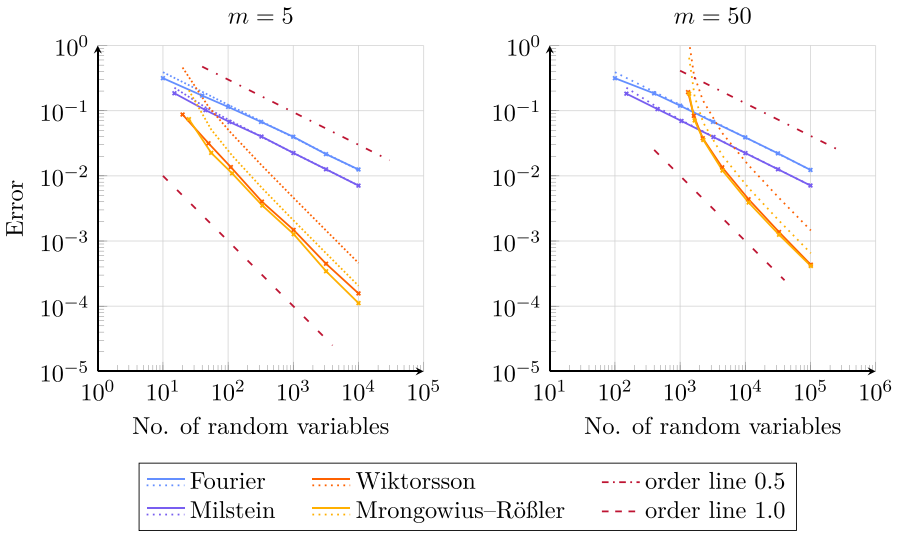

Fig. 2 max, L^2-error versus the number of random variables (cost) used for different dimensions of the Wiener process based on 100 realizations. The error is computed w.r.t. the reference random variable with $p_{\text{ref}} = 10^6$ and for $h = 1$. The dotted lines show the corresponding theoretical error bounds according to Theorem 2

Calculating the vectors γ_1, γ and γ_2 based on the already simulated and stored Fourier coefficients guarantees that they fit correctly together with the realization of the reference random variable. Note that for $r > p_{\text{ref}}$ the Fourier coefficients α_r^i and β_r^i for $i = 1, \ldots, m$ of the reference random variable are all zero. Now, we calculate $\hat{\mathcal{I}}^{\text{Mil},(p)}(h)$ by the Milstein, $\hat{\mathcal{I}}^{\text{Wik},(p)}(h)$ by the Wiktorsson and $\hat{\mathcal{I}}^{\text{MR},(p)}(h)$ by the Mrongowius–Rößler algorithm using only the first p Fourier coefficients as well as γ_1, γ and γ_2, respectively.

In Fig. 2 the computed max, L^2-errors are plotted versus the costs. Here, the cost is determined as the number of random numbers necessary to calculate the corresponding approximation for the iterated stochastic integral for each algorithm. To generate the random numbers we use the Mersenne Twister generator from Julia version 1.6. The steeper slope of the Wiktorsson and Mrongowius–Rößler algorithms can be clearly seen. This confirms the higher order of convergence of these algorithms in contrast to the Fourier and Milstein algorithms. Furthermore, it is interesting to observe that the Wiktorsson as well as the Mrongowius–Rößler algorithm perform better than their theoretical upper error bound given in Theorem 2. This gives reason to suspect that the error estimates for both algorithms given in Theorem 2 are not sharp.

6 A simulation toolbox for Julia and MATLAB

There exist several software packages to simulate SDEs in various programming languages. A short search brought up the following 20 toolboxes: for the C++ programming language [4, 25], for the Julia programming language [2, 48–50], for

Mathematica [54], for MATLAB [17, 22, 44, 52], for the Python programming language [1, 3, 15, 16, 36] and for the R programming language [6, 18, 23, 24]. However, only four of these toolboxes seem to contain an implementation of an approximation for the iterated stochastic integrals using Wiktorsson's algorithm and none of them provide an implementation of the recently proposed Mrongowius–Rößler algorithm. To be precise, Wiktorsson's algorithm is contained in the software packages `SDELab` [17] in Matlab as well as in its Julia continuation `SDELab2.jl` [50], in the Python package `sdeint` [1] and more recently also in the Julia package `StochasticDiffEq.jl` [49].

Note that among these four software packages only three have the source code readily available on GitHub:

- The software package `SDELab2.jl` does not seem to be maintained any more. It is worth mentioning that its implementation avoids the explicit use of Kronecker products. However, it is not as efficient as the implementations discussed in Section 3.3.3. Moreover, some simple test revealed that the implementation seems to suffer from some typos that may cause inexact results. Additionally, the current version of this package does not run with the current Julia version 1.6.
- The software package `sdeint` is purely written in Python and as such it is inherently slower than comparable implementations in a compiled language. Furthermore, it is based on a rather naïve implementation, e.g., explicitly constructing the Kronecker products and the permutation and selection matrices as described in Section 3.3.2. However, this is computational costly and thus rather inefficient, see also the discussion in Section 3.3.3.
- The software package `StochasticDiffEq.jl` offers the latest implementation of Wiktorsson's approximation in the high-performance language Julia. Up to now, it is actively maintained and has a strong focus on good performance. It is basically a slightly optimized version of the `SDELab2.jl` implementation and thus it also does not achieve the maximal possible efficiency, especially compared to the discussion in Section 3.3.3.

In summary, there is currently no package providing an efficient algorithm for the simulation of iterated integrals. That is the main reason we provide a new simulation toolbox for Julia and MATLAB that, among others, features Wiktorsson's algorithm as well as the Mrongowius–Rößler algorithm. In Section 6.1 we present some more features of the toolbox. After giving usage examples in Section 6.2, we analyse the performance of our toolbox as compared to some existing implementations in Section 6.3.

6.1 Features of the new Julia and MATLAB simulation toolbox

We introduce a new simulation toolbox for the simulation of twofold iterated stochastic integrals and the corresponding Lévy areas for the Julia, see also [5], and MATLAB programming languages. The aim of the toolbox is to provide both high performance and ease of use. On the one hand, experts can directly control each part of the algorithms. On the other hand, our software can automatically choose omitted parameters.

Thus, a non-expert user only has to provide the Wiener increment and the associated step size and all other parameters will be chosen in an optimal way.

The toolbox is available as the software packages `LevyArea.jl` and `LevyArea.m` for Julia and MATLAB, respectively. Both software packages are freely available from Netlib[4] as the `na57` package and from GitHub [27, 28]. Additionally, the Julia package is registered in the 'General' registry of Julia and can be installed using the built-in package manager, while the MATLAB package is listed on the 'MATLAB Central - File Exchange' and can be installed using the Add-On Explorer. This allows for an easy integration of these packages into other software projects.

Both packages provide the Fourier algorithm, the Milstein algorithm, the Wiktorsson algorithm and, to the best of our knowledge, for the first time the Mrongowius–Rößler algorithm. As the most important feature, all four algorithms are implemented following the insights from Section 3 to provide fast and highly efficient implementations. In Section 6.3 the performance of the Julia package is analysed in comparison to existing software.

Additional features include the ability to automatically choose the optimal algorithm based on the costs listed in Table 2. That is, given the increment of the driving Wiener process, the step size and an error bound in some norm, both software packages will automatically determine the optimal algorithm and the associated optimal value for the truncation parameter. Recall that optimal always means in terms of the number of random numbers that have to be generated in order to obtain minimal computing time. Thus, using the option for an optimal choice of the algorithm results in the best possible performance for each setting. Especially, for the simulation of SDE solutions with some strong order 1 numerical scheme, both software packages can determine all necessary parameters by passing only the Wiener increment and the step size to the toolbox. Everything else is determined automatically such that the global order of convergence of the numerical integrator is preserved.

Furthermore, it is possible to directly simulate iterated stochastic integrals based on Q-Wiener processes on finite-dimensional spaces as they typically appear for the approximate simulation of solutions to SPDEs. In that case, the eigenvalues of the covariance operator Q need to be passed to the software toolbox in order to compute the correctly scaled iterated stochastic integrals. This scaling is briefly described at the end of Section 2.1, see also [34] for a detailed discussion.

6.2 Usage of the software package

Next, we give some basic examples how to make use of the two software packages `LevyArea.jl` for Julia and `LevyArea.m` for MATLAB. The aim is not to give a full documentation but rather to show some example invocations of the main functions. For more information we refer to the documentation that comes with the software and can be easily accessed in Julia and MATLAB.

[4] http://www.netlib.org/numeralgo/

6.2.1 The Julia package `LevyArea.jl`

The following code works with Julia version 1.6. First of all, the software package `LevyArea.jl` [27] needs to be installed to make it available. Therefore, start Julia and enter the package manager by typing `]` (a closing square bracket). Then execute

```
pkg> add LevyArea
```

which downloads the package and adds it to the current project. After the installation of the package, one can load the package and initialize some variables that we use in the following:

```
julia> using LevyArea
julia> m = 5; # dimension of Wiener process
julia> h = 0.01; # step size or length of time interval
julia> err = 0.05; # error bound
julia> W = sqrt(h) * randn(m); # increment of Wiener process
```

Here, W is the m-dimensional vector of increments of the driving Wiener process on some time interval of length h.

For the simulation of the corresponding twofold iterated stochastic integrals, one can use the following default call of the function `iterated_integrals` where only the increment and the step size are mandatory:

```
julia> II = iterated_integrals(W,h)
```

In this example, the error $\bar{\varepsilon}$ is not explicitly specified. Therefore, the function assumes the desired precision to be $\bar{\varepsilon} = h^{3/2}$ as it has to be chosen for the numerical solution of SDEs, see end of Section 4, and automatically chooses the optimal algorithm according to the logic in Section 5.1, see also Fig. 1. If not stated otherwise, the default error criterion is the max, L^2-error and the function returns the $m \times m$ matrix `II` containing a realization of the approximate iterated stochastic integrals that correspond to the given increment W.

The desired precision $\bar{\varepsilon}$ can be optionally provided using a third positional argument:

```
julia> II = iterated_integrals(W,h,err)
```

Again, the software package automatically chooses the optimal algorithm as analysed in Section 5.1.

To determine which algorithm is chosen by the package without simulating any iterated stochastic integrals yet, the function `optimal_algorithm` can be used. The arguments to this function are the dimension of the Wiener process, the step size and the desired precision:

```
julia> alg = optimal_algorithm(m,h,err); # output: Fourier()
```

It is also possible to choose the algorithm directly using the keyword argument `alg`. The value can be one of `Fourier()`, `Milstein()`, `Wiktorsson()` and `MronRoe()`:

```
julia> II = iterated_integrals(W,h; alg=Milstein())
```

As the norm for the considered error, e.g., the max, L^2- and L^2, F-norm can be selected using a keyword argument. The accepted values are `MaxL2()` and `FrobeniusL2()`:

```
julia> II = iterated_integrals(W,h,err; error_norm=FrobeniusL2())
```

If iterated stochastic integrals for some Q-Wiener process need to be simulated, like for the numerical simulation of solutions to SPDEs, then the increment of the Q-Wiener process together with the square roots of the eigenvalues of the associated covariance operator have to be provided, see Section 2.1:

```
julia> q = [1/k^2 for k=1:m]; # eigenvalues of cov. operator
julia> QW = sqrt(h) * sqrt.(q) .* randn(m); # Q-Wiener increment
julia> IIQ = iterated_integrals(QW,sqrt.(q),h,err)
```

In this case, the function `iterated_integrals` utilizes a scaling of the iterated stochastic integrals as explained in Section 2.1 and also adjusts the error estimates appropriately such that the error bound holds w.r.t. the iterated stochastic integrals $\mathcal{I}^Q(h)$ based on the Q-Wiener process. Here the error norm defaults to the L^2, F-error.

Note that all discussed keyword arguments are optional and can be combined as favoured. Additional information can be found using the Julia help mode:

```
julia> ?iterated_integrals
julia> ?optimal_algorithm
```

6.2.2 The MATLAB package `LevyArea.m`

The following code works with MATLAB version 2020a. The installation of the software package `LevyArea.m` [28] in MATLAB is done either by copying the package folder `+levyarea` into the current working directory or by installing the MATLAB toolbox file `LevyArea.mltbx`. This can also be done through the Add-On Explorer.

The main function of the toolbox is the function `iterated_integrals`. It can be called by prepending the package name `levyarea.iterated_integrals`. However, since this may be cumbersome one can import the used functions once by

```
>> import levyarea.iterated_integrals
>> import levyarea.optimal_algorithm
```

and then one can omit the package name by simply calling `iterated_integrals` or `optimal_algorithm`, respectively. In the following, we assume that the two functions are imported, so that we can always call them directly without the package name.

For the examples considered next we initialize some auxiliary variables:

```
>> m = 5;    % dimension of Wiener process
>> h = 0.01;     % step size or length of time interval
>> err = 0.05;   % error bound
>> W = sqrt(h) * randn(m,1);     % increment of Wiener process
```

Note that W denotes the m-dimensional vector of increments of the Wiener process on some time interval of length h.

For directly simulating the twofold iterated stochastic integrals given the increment of the Wiener process, one can call the main function `iterated_integrals` passing the increment and the step size:

```
>> II = iterated_integrals(W,h)
```

These two parameters are mandatory. In this case, the precision is set to the default $\bar{\varepsilon} = h^{3/2}$ and the optimal algorithm is applied automatically according to Section 5.1 (see also Fig. 1). The default norm for the error is set to be the max, L^2-norm.

The desired precision $\bar{\varepsilon}$ can be optionally provided using a third positional argument:

```
>> II = iterated_integrals(W,h,err)
```

Here, again the software package automatically chooses the optimal algorithm as analysed in Section 5.1.

In order to determine which algorithm is optimal for some given parameters without simulating the iterated stochastic integrals yet, the function `optimal_algorithm` can be used:

```
>> alg = optimal_algorithm(m,h,err); % output: 'Fourier'
```

The arguments to this function are the dimension of the Wiener process, the step size and the desired precision.

On the other hand, it is also possible to choose the used algorithm directly using a key-value pair. The value can be one of '`Fourier`', '`Milstein`', '`Wiktorsson`' and '`MronRoe`'. E.g., to use the Milstein algorithm call:

```
>> II = iterated_integrals(W,h,'Algorithm','Milstein')
```

The desired norm for the prescribed error bound can also be selected using a key-value pair. The accepted values are `'MaxL2'` and `'FrobeniusL2'` for the max, L^2- and L^2, F-norm, respectively. E.g., in order to use the L^2, F-norm call:

```
>> II = iterated_integrals(W,h,err,'ErrorNorm','FrobeniusL2')
```

The simulation of numerical solutions to SPDEs often requires iterated stochastic integrals based on Q-Wiener processes. In that case, the square roots of the eigenvalues of the associated covariance operator need to be provided. Therefore, first define all necessary variables and then call the function `iterated_integrals` using the key '`QWiener`' as follows:

```
>> q = 1./(1:m)'.^2; % eigenvalues of covariance operator
>> QW = sqrt(h) * sqrt(q) .* randn(m,1); % Q-Wiener increment
>> IIQ = iterated_integrals(QW,h,err,'QWiener',sqrt(q))
```

In this case, the function utilizes the scaling of the iterated stochastic integrals as explained in Section 2.1 and it also adjusts the error estimates w.r.t. $\mathcal{I}^Q(h)$ appropriately.

Note that all discussed keyword arguments (key-value pairs) are optional and can be combined as desired. Additional information can be found using the `help` function:

```
>> help iterated_integrals
>> help levyarea.optimal_algorithm
```

6.3 Benchmark comparison

To assess the performance of the new software package, we compare it with some existing implementations. Therefore, we consider the software packages `SDELab2.jl` [50], `sdeint` [1] and `StochasticDiffEq.jl` [49]. First, the software package `sdeint` is completely written in Python and a quick simulation example with $m = 50$, $h = 0.01$, $\varepsilon = 0.001$ and thus $p = 15$ takes 9.9 seconds to generate the matrix of iterated integrals with Wiktorsson's algorithm. For the same parameters, our implementation of Wiktorsson's algorithm in the package `LevyArea.jl` completed in only $5.5 \cdot 10^{-6}$ seconds. This is a speed-up by a factor 1.8×10^6 and therefore we exclude the Python package `sdeint` from our further comparison. Moreover, since the package `SDELab2.jl` seems to be unmaintained and does not run unmodified on current Julia versions, it is excluded from our benchmark as well. Thus, we compare the new package `LevyArea.jl` with the implementation in the package `StochasticDiffEq.jl` in the following.

For the benchmark we measure the time it takes to generate the full matrix of iterated stochastic integrals over a range of step sizes and for two different dimensions of the driving Wiener process. To deal with measurement noise, we average the computing times over 100 runs. Further, in order to guarantee a fair comparison, we calculate

the value of the truncation parameter according to Table 2 for all algorithms under consideration. We consider the setting where a strong order 1 numerical scheme is applied for the simulation of solutions to SDEs. This setting is of high importance for many applications and a typical situation where iterated stochastic integrals need to be efficiently simulated. In order to retain the strong order 1 if the twofold iterated stochastic integrals are replaced by their approximations, we need to choose the precision for the simulated iterated stochastic integrals following the discussion at the end of Section 4. Therefore, the error is always chosen as $\bar{\varepsilon} = h^{3/2}$ in the max, L^2-norm and we consider step sizes $10^0, 10^{-1}, \ldots, 10^{-8}$ for the case $m = 100$. For the case of $m = 1000$, we initially use the same step sizes, however we refrain from applying the smaller step sizes for the Fourier and Milstein algorithms whenever the computing time or the needed memory becomes too large for reasonable computations.

The Julia package `StochasticDiffEq.jl` is used at version 6.37.1 and we run the in-place function `StochasticDiffEq.get_iterated_I!(h,W,nothing,iip,p)` where `iip` is a preallocated buffer. Note that the creation of this buffer is not included in our computing time measurements. The newly proposed Julia package `LevyArea.jl` is used at its current version 1.0.0. The benchmark is performed on a computer with an Intel Xeon E3-1245 v5 CPU at 3.50 GHz and 32 GB of memory using Julia version 1.6. Furthermore, the Julia package `DrWatson.jl` [9] is employed. The simulation results for both settings with $m = 100$ and $m = 1000$ are shown in Fig. 3.

Considering the benchmark results in Fig. 3, we can see that Wiktorsson's algorithm as well as the Mrongowius–Rößler algorithm attain a higher order of convergence compared to the Fourier and the Milstein algorithms. This confirms the

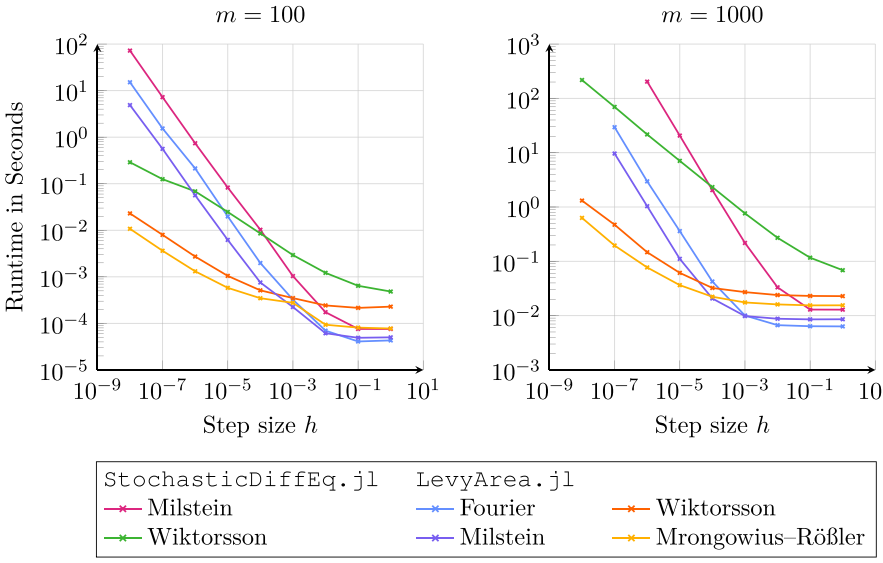

Fig. 3 Runtime of the algorithms for variable step size and two different dimensions of the underlying Wiener process. The value of the truncation parameter is always chosen to guarantee that $\|\mathcal{I} - \hat{\mathcal{I}}\|_{\max,L^2} \leq h^{3/2}$ according to the error estimates in Theorem 2. The timings are averaged over 100 repetitions

Table 3 Minimal and maximal relative speed-ups of the new implementations in `LevyArea.jl` with respect to the implementations in `StochasticDiffEq.jl` for two different dimensions of the underlying Wiener process

Algorithm `A` from	Algorithm `B` from	Relative speed-up ($\frac{t_A}{t_B}$)	
`StochasticDiffEq.jl`	`LevyArea.jl`	$m = 100$	$m = 1000$
Milstein	Milstein	1.5×–14.8×	1.5×–196.8×
Wiktorsson	Wiktorsson	2.1×–25.1×	3.0×–165.8×
Wiktorsson	Mrongowius–Rößler	6.2×–52.1×	4.4×–356.1×

See also Fig. 3

theoretical results in Theorem 2. Moreover, the simulation results show that for the Milstein algorithm as well as for Wiktorsson's algorithm the implementation in the newly proposed package `LevyArea.jl` (blue, purple, orange and yellow colours) clearly outperforms the implementations in the package `StochasticDiffEq.jl` (red and green colours). This is due to the ideas for an efficient implementation presented in Section 3 that are incorporated in the package `LevyArea.jl`. This allows for a speed-up by factors up to 165.8 for Wiktorsson's algorithm in the case of $m = 1000$ for the range of parameters we tested.

Comparing the cases $m = 100$ and $m = 1000$, it seems that the implementations in the package `StochasticDiffEq.jl` have a much higher overhead for high-dimensional Wiener processes.

Moreover, it can be seen that for both settings the Mrongowius–Rößler algorithm is the best algorithm for sufficiently small step sizes as they typically arise for SDE and SPDE approximation problems. This confirms the theoretical results presented in Fig. 1.

The relative speed-up of the implementations in package `LevyArea.jl` compared to package `StochasticDiffEq.jl` are specified in Table 3. There is a serious speed-up for each algorithm in the package `LevyArea.jl` that becomes even greater when step sizes are getting smaller or if the dimension of the Wiener process is rather high. Thus, the proposed software package `LevyArea.jl` allows for very efficient simulations of iterated stochastic integrals with very good performance also for small step sizes and small error bounds as well as for high-dimensional Wiener processes. This makes the package valuable especially for SDE and SPDE simulations based on higher order approximation schemes.

Funding Open Access funding enabled and organized by Projekt DEAL. This work received funding and support from the Graduate School for Computing in Medicine and Life Sciences funded by Germany's Excellence Initiative (DFG GSC 235/2).

Data availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Aburn, M.: sdeint. Version 0.2.2. https://github.com/mattja/sdeint (2021)
2. Åkerlindh, C.: SDEModels.jl. Version 0.2.0. https://github.com/Godisemo/SDEModels.jl (2019)
3. Ansmann, G.: Efficiently and easily integrating differential equations with JiTCODE, JiTCDDE, and JiTCSDE. In: Chaos: An interdisciplinary journal of nonlinear science vol. 28.4, pp. 043116. https://doi.org/10.1063/1.5019320 (2018)
4. Avramidis, E., Lalik, M., Akman. O.E.: SODECL: an open-source library for calculating multiple orbits of a system of stochastic differential equations in parallel. In: ACM transactions on mathematical software vol. 46.3, pp. 1–21. https://doi.org/10.1145/3385076 (2020)
5. Bezanson, J., et al: Julia: A fresh approach to numerical computing. In: SIAM Review vol. 59.1, pp. 65–98. https://doi.org/10.1137/141000671 (2017)
6. Brouste, A., et al.: The YUIMA Project: a computational framework for simulation and inference of stochastic differential equations. In: Journal of statistical software vol. 57.4. https://doi.org/10.18637/jss.v057.i04 (2014)
7. Clark, J.M.C., Cameron, R.J.: The maximum rate of convergence of discrete approximations for stochastic differential equations. In: Stochastic differential systems filtering and control. Lecture notes in control and information sciences. Springer, pp. 162–171. https://doi.org/10.1007/BFb0004007 (1980)
8. Da Prato, G., Zabczyk, J.: Stochastic Equations in Infinite Dimensions. Second edition. Vol. 152. Encyclopedia of Mathematics and its Applications, pp. xviii+493. Cambridge University Press, Cambridge (2014). https://doi.org/10.1017/CBO9781107295513
9. Datseris, G., et al.: DrWatson: the perfect sidekick for your scientific inquiries. In: Journal of open source software vol 5.54, pp. 2673. https://doi.org/10.21105/joss.02673 (2020)
10. Davie, A.: KMT theory applied to approximations of SDE. In: Stochastic analysis and applications 2014. In honour of Terry Lyons. Selected articles based on the presentations at the conference, Oxford, UK, September 23–27, 2013. Cham: Springer, pp. 185–201. https://doi.org/10.1007/978-3-319-11292-3_7 (2014)
11. Dickinson, A.S.: Optimal approximation of the second iterated integral of Brownian motion. In: Stoch. Anal. Appl. vol. 25.5, pp. 1109–1128. https://doi.org/10.1080/07362990701540592 (2007)
12. Flint, G., Lyons, T.: Pathwise approximation of SDEs by coupling piecewise abelian rough paths. In: arXiv:1505.01298 (2015)
13. Foster, J., Habermann, K.: Brownian bridge expansions for Lévy area approximations and particular values of the Riemann zeta function. In: Combinatorics, Probability and Computing (2022). https://doi.org/10.1017/S096354832200030X
14. Gaines, J.G., Lyons, T.J.: Random generation of stochastic area integrals. In: SIAM Journal on applied mathematics vol. 54.4, pp. 1132–1146. https://doi.org/10.1137/S0036139992235706 (1994)
15. Gevorkyan, M.N., et al.: Stochastic Runge-Kutta software package for stochastic differential equations. In: Dependability engineering and complex systems. Springer pp. 169–179. https://doi.org/10.1007/978-3-319-39639-2_15 (2016)
16. Gevorkyan, M.N., et al.: Issues in the software implementation of stochastic numerical Runge-Kutta. In: Developments in language theory. Springer pp. 532–546. https://doi.org/10.1007/978-3-319-99447-5_46 (2018)

17. Gilsing, H., Shardlow, T.: SDELab: A package for solving stochastic differential equations in MATLAB. In: Journal of computational and applied mathematics vol. 205.2, pp. 1002–1018. https://doi.org/10.1016/j.cam.2006.05.037 (2007)
18. Guidoum, A.C., Boukhetala, K.: Performing parallel Monte Carlo and moment equations methods for Itô and Stratonovich stochastic differential systems: R Package Sim.DiffProc. In: Journal of statistical software vol. 96.2. https://doi.org/10.18637/jss.v096.i02 (2020)
19. von Hallern, C., Rößler, A.: A derivative-free Milstein type approximation method for SPDEs covering the Non-Commutative Noise case. To appear in: Stoch. PDE: Anal. Comp., https://doi.org/10.1007/s40072-022-00274-6 (2022)
20. von Hallern, C., Rößler, A.: An analysis of the Milstein scheme for SPDEs without a commutative noise condition. In: Monte Carlo and quasi-Monte Carlo methods. MCQMC 2018. Proceedings of the 13th international conference on Monte Carlo and quasi-Monte Carlo methods in scientific computing, Rennes, France, July 1–6, 2018. Cham: Springer, pp. 503–521. https://doi.org/10.1007/978-3-030-43465-6_25 (2020)
21. Henderson, H.V., Searle, S.R.: The vec-permutation matrix, the vec operator and Kronecker products: a review. In: Linear and Multilinear Algebra vol. 9.4, pp. 271–288. https://doi.org/10.1080/03081088108817379 (1981)
22. Horchler, A.D.: SDETools. Version 1.2. https://github.com/horchler/SDETools (2013)
23. Iacus, S.M.: Simulation and Inference For Stochastic Differential Equations. Springer, New York (2008). https://doi.org/10.1007/978-0-387-75839-8
24. Iacus, S.M., Yoshida, N.: Simulation and Inference for Stochastic Processes with YUIMA, Springer International Publishing. https://doi.org/10.1007/978-3-319-55569-0 (2018)
25. Janicki, A., Izydorczyk, A., Gradalski. P.: Computer simulation of stochastic models with SDE-Solver software package. In: Computational science – ICCS 2003. international conference, Melbourne, Australia and St. Petersburg, Russia, June 2–4, 2003. Proceedings, Part I. Berlin: Springer, pp. 361–370. https://doi.org/10.1007/3-540-44860-8_37 (2003)
26. Karatzas, I., Shreve, S.E.: Brownian motion and stochastic calculus, Springer-Verlag. https://doi.org/10.1007/978-1-4612-0949-2 (1991)
27. Kastner, F., Rößler, A.: LevyArea.jl. https://doi.org/10.5281/zenodo.5883748. https://github.com/stochastics-uni-luebeck/LevyArea.jl (2022)
28. Kastner, F., Rößler, A.: LevyArea.m. 2022. https://doi.org/10.5281/zenodo.5883929. https://github.com/stochastics-uni-luebeck/LevyArea.m
29. Kloeden, P.E., Platen, E.: Numerical solution of stochastic differential equations. Second corrected printing. vol. 23. Applications of Mathematics. Springer, Berlin, pp. xxxvi+632. https://doi.org/10.1007/978-3-662-12616-5 (1995)
30. Kloeden, P.E., Platen, E., Wright, I.W.: The approximation of multiple stochastic integrals. In: Stoch. Anal. Appl. vol. 10.4, pp. 431–441. https://doi.org/10.1080/07362999208809281 (1992)
31. Kuznetsov, D.F.: A comparative analysis of efficiency of using the Legendre polynomials and trigonometric functions for the numerical solution of Ito stochastic differential equations. In: Computational mathematics and mathematical physics vol. 59.8, pp. 1236–1250. https://doi.org/10.1134/s0965542519080116 (2019)
32. Kuznetsov, D.F.: Development and application of the fourier method for the numerical solution of Ito stochastic differential equations. In: Computational Mathematics and Mathematical Physics vol. 58.7, pp. 1058–1070. https://doi.org/10.1134/s0965542518070096 (2018)
33. Leonhard, C., Rößler, A.: Enhancing the order of the Milstein scheme for stochastic partial differential equations with commutative noise. In: SIAM J. Numer. Anal. vol. 56.4, pp. 2585–2622. https://doi.org/10.1137/16M1094087 (2018)
34. Leonhard, C., Rößler, A.: Iterated stochastic integrals in infinite dimensions: approximation and error estimates. In: Stoch. PDE: Anal. Comp., vol. 7.2, pp. 209–239. https://doi.org/10.1007/s40072-018-0126-9 (2019)
35. Lévy, P.: Wieners random function, and other Laplacian random functions. In: Proceedings of the Second Berkeley symposium on mathematical statistics and probability, 1950. University of California Press, Berkeley and Los Angeles, pp. 171–187 (1951)
36. Li, X.: torchsde. Version 0.2.4. https://github.com/google-research/torchsde (2021)
37. Liske, H., Platen, E., Wagner, W.: About mixed multiple Wiener integrals. Prepr., Akad. Wiss. DDR, Inst. Math. P-MATH-23/82, vol. 17 (1982)

38. Magnus, J.R., Neudecker, H.: The commutation matrix: some properties and applications. In: The Annals of Statistics, vol. 7.2, pp. 381–394. https://doi.org/10.1214/aos/1176344621 (1979)
39. Malham, S.J.A., Wiese, A.: Efficient almost-exact Lévy area sampling. In: Statistics & Probability Letters vol. 88, pp. 50–55. https://doi.org/10.1016/j.spl.2014.01.022 (2014)
40. Milstein, G.N.: Numerical Integration of Stochastic Differential Equations. vol. 313. Mathematics and its Applications. Translated and Revised from the Russian 1988 Original, pp. viii+169. Kluwer Academic Publishers, Dordrecht (1995). https://doi.org/10.1007/978-94-015-8455-5
41. Mrongowius, J., Rößler, A.: On the approximation and simulation of iterated stochastic integrals and the corresponding Lévy areas in terms of a multidimensional Brownian motion. In: Stoch. Anal. Appl., vol. 40.3, p. 397–425. https://doi.org/10.1080/07362994.2021.1922291 (2022)
42. Neuenkirch, A., Tindel, S., Unterberger, J.: Discretizing the fractional Lévy area. In: Stochastic Processes and their Applications vol. 120.2, pp. 223–254. https://doi.org/10.1016/j.spa.2009.10.007 (2010)
43. Neuenkirch, A., Shalaiko, T.: The maximum rate of convergence for the approximation of the fractional Lévy area at a single point. In: Journal of Complexity vol. 33, pp. 107–117. https://doi.org/10.1016/j.jco.2015.09.008 (2016)
44. Picchini, U.: SDE Toolbox. Version 1.4.1. http://sdetoolbox.sourceforge.net (2017)
45. Prévôt, C., Röckner, M.: A Concise Course on Stochastic Partial Differential Equations. vol. 1905 Lecture Notes in Mathematics, pp. vi+144. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-70781-3
46. Rößler, A.: Runge-Kutta methods for the strong approximation of solutions of stochastic differential equations. In: SIAM J. Numer. Anal. vol. 48.3, pp. 922–952. https://doi.org/10.1137/09076636X (2010)
47. Rydén, T., Wiktorsson, M.: On the simulation of iterated Itô integrals. In: Stochastic Process. Appl. vol. 91.1, pp. 151–168. https://doi.org/10.1016/S0304-4149(00)00053-3 (2001)
48. Schauer, M.: Bridge.jl. Version 0.11.6. https://github.com/mschauer/Bridge.jl (2021)
49. SciML: StochasticDiffEq.jl. Version 6.37.1. https://github.com/SciML/StochasticDiffEq.jl (2021)
50. Shardlow, T.: SDELab2. Version 1.0. https://github.com/tonyshardlow/SDELAB2 (2016)
51. Stump, D.M., Hill, J.M.: On an infinite integral arising in the numerical integration of stochastic differential equations. In: Proceedings of the Royal society of London. Series A. mathematical, physical and engineering sciences vol. 461.2054, pp. 397–413. https://doi.org/10.1098/rspa.2004.1379 (2005)
52. The MathWorks Inc.: Financial toolbox. Natick, Massachusetts, United States. https://www.mathworks.com/help/finance/ (2021)
53. Wiktorsson, M.: Joint characteristic function and simultaneous simulation of iterated Itô integrals for multiple independent Brownian motions. In: Ann. Appl. Probab. vol. 11.2, pp. 470–487. https://doi.org/10.1214/aoap/1015345301 (2001)
54. Wolfram Research: ItoProcess. Version 12.2.0. https://reference.wolfram.com/language/ref/ItoProcess.html (2016)