b
UNIVERSITÄT
BERN

Strong Schemes for Numerical Solutions of Stochastic Differential Equations

Bachelor Thesis

Amr Umeri

December 1, 2017

Advisor: Prof. Dr. R. Gatto

Institute of Mathematical Statistics and Actuarial Science, University of Bern

Abstract

We will present the theory of stochastic integration due to Itô and discuss stochastic differential equations. We will present examples of such equations which can be solved analytically using the Itô lemma. We will use the same lemma to construct numerical schemes for strong approximate solutions. Then we will determine the convergence order of these schemes using simulations in the programming environment Python. Secondly, we will determine the convergence order analytically and give error bounds for our schemes. We will end the thesis with the conclusion that our estimated convergence orders of our schemes match the theoretical convergence orders for the considered stochastic differential equations.

Contents

Chapter 1

Introduction

Given the simplest first-order ordinary differential equation, where a: $\mathbb{R} \to \mathbb{R}$ is continuous, we may want to find the solution $X(t)$, if it exists:

$$\frac{\mathrm{d}}{\mathrm{d}t} X(t) = a(X(t), t), \quad X(0) = x_0, \quad t \in [0, T],$$

we then have the equivalent integral representation:

$$X(t) = x_0 + \int_0^t a(X(s), s)\, \mathrm{d}s.$$

In many theoretical and practical problems, we may want to include "randomness" (or "white noise") into the above equation. However the noise-term may be nowhere differentiable in the usual definition. Therefore it is reasonable to consider only the integral representation of such equations and to include the noise-term as *stochastic integral*. We will follow the Itô stochastic integration theory in order to define so-called *stochastic differential equations*. Consider the following stochastic integral equation:

$$X_t = x_0 + \int_0^t a(X_s, s)\, \mathrm{d}s + \int_0^t b(X_s, s)\, \mathrm{d}W_s, \quad t \in [0, T],$$

where we have given a probability space $(\Omega, \mathcal{F}, P)$. Equality is P-a.s, x_0 is a possible random initial value and the noise-term is included through the second integral, where the integrator is the *Wiener process*. Now the deterministic (Riemann-) integral can be understood as drift coefficient and the stochastic integral as diffusion (or volatility) coefficient. We want to find a stochastic process X_t which satisfies the equation and call it the *solution process*. We will say that the above equation is the mathematical interpretation of the stochastic differential equation which is given in symbolical (differential) form:

$$\mathrm{d}X_t = a(X_t, t)\mathrm{d}t + b(X_t, t)\mathrm{d}W_t.$$

In chapter 2 we will define the Wiener process (also called *Brownian motion*) and its discretization. And in chapter 3 we will discuss the classical theory of stochastic integration w.r.t the Wiener process. In section 3.2 we will present the fundamental theorem of stochastic calculus, the *lemma of Itô*, and give some examples. We will then use this important result in 3.3 in order to construct so-called *stochastic Taylor expansions* which can be seen as a generalization of the classical result to certain stochastic processes (which we will call *Itô-processes*).

In chapter 4 we will finally define stochastic differential equations and discuss conditions under which the solution exists and whether it is unique (almost surely). We will discuss 2 basic examples of stochastic differential equations for which closed-form solutions are known and give the calculations. The *geometric Brownian motion* and the *Ornstein-Uhlenbeck process*.

Stochastic differential equations with a known explicit solution are the exception from the rule. Thus it is important (similar to deterministic differential equations) to have numerical algorithms for approximative solutions. In chapter 5 we will truncate our stochastic Taylor expansions, in order to construct approximative schemes for stochastic differential equations. These are also called *time-discrete schemes* since we give the approximative values on some discrete points of the time interval [0,T]. We will consider strongly converging schemes. These are numerical algorithms which give path-wise approximations of the true solution. We will discuss the so-called *Euler-Maruyama* and the *Milstein* scheme and study their convergence properties. We will prove that the Euler scheme is converging (in $L_2[\Omega]$) to the true solution with order of convergence 0.5, and the Milstein scheme with order 1.0 respectively under some global Lipschitz-conditions on the coefficients.

In section 5.3 we will illustrate our numerical schemes on the considered stochastic differential equations and give Monte-Carlo estimates for the convergence orders. We will then end the thesis with a comparison of the analytical and the empirical results of the convergence orders for both schemes.

Chapter 2

Foundations

In this section we will present the *Wiener process*. Then we show a way on how to discretize such a process in order to display it on a computer and to allow sampling from this stochastic process. We will show an algorithm which allows us to increase the number of time steps given a discretized sample path. This algorithm will be usefull for our numerical simulations. In the last part of the chapter we will evaluate the *stochastic integral* of a basic example using two distinct definitions. We will notice that the stochastic integral can be defined on many ways. In the next chapters we will focus on one particular defintion of the stochastic integral, namely the *Itô stochastic integral*.

2.1 Stochastic processes

Definition 2.1 (Stochastic process)
A stochastic process is a collection of random variables on a probability space $(\Omega, \mathcal{F}, P)$ *indexed by time:*

$$\{X(t,\omega) \mid t \geq 0\}, \quad X(t,\omega)\colon [0,T] \times \Omega \to \mathbb{R} \text{ for T given.}$$

Measurability is meant to be w.r.t the product-sigma-algebra on $[0,T] \times \Omega$*. For each* $\omega \in \Omega$ *we can define the mapping:*

$$t \mapsto X(t,\omega)$$

which we will call the sample path*, which is a realisation for a given* ω*. If this mapping is continuous,* X_t *is called a continuous stochastic process.*

To simplify notation we will write X_t to denote a stochastic process on [0,T]. However notice that X_t is also a random variable for each t fixed.

Definition 2.2 (Wiener process)
A Wiener process W_t *(also called Brownian motion) is a continuous stochastic process satisfying:*

1. $W_0 = 0$ *P-a.s*
2. $W_t - W_s \sim \mathcal{N}(0, t-s)$
3. $W_{t_1} - W_{s_1}, \ldots, W_{t_n} - W_{s_n}$ *are independent for all non-overlapping intervals* $[s_i, t_i]$, $i = 1, \ldots, n$.

Definition 2.3 (Adapted processes)
Let $\mathcal{F}_t$ be the σ-algebra generated by $\{W_s \mid 0 \leq s \leq t\}$ for $t \in [0,T]$. $\mathcal{F}_t$ is called the filtration *induced by the Wiener process W_t. It holds that $\mathcal{F}_s \subset \mathcal{F}_t \subset \mathcal{F}$ for $s < t$, i.e. $\{\mathcal{F}_t\}_{t\in[0,T]}$ is an increasing sequence of σ-algebras. We will call a stochastic process X_t to be $\mathcal{F}_t$-adapted (or just adapted) if it is $\mathcal{F}_t$-measurable $\forall\ t \in [0,T]$.*
Take as example $X_t := W_{t+1}$, then X_t is not adapted.

2.2 Discretization of the Wiener process

Now We will consider the discretization of the Wiener process on [0,T] and we will denote it as $\overline{W}_t$. We will use constant time steps only. Let $\{t_0 = 0, t_1 = \delta, \dots, t_n = n\delta\}$ be a time-grid with constant stepsize $\delta := \frac{T}{n}$.

Algorithm 2.4 (Discretization of the Wiener process)

```
1. Initialize $\overline{W}_0 = 0$
2. For j = 0 to n-1
        a) Generate z$\sim \mathcal{N}(0,\delta)$
        b) Set $\overline{W}_{(j+1)\delta} = z + \overline{W}_{j\delta}$
3. Linear interpolation to get a continious version.
```

Then $(\overline{W}_0, \dots, \overline{W}_{n\delta}) \overset{d}{\sim} (W_0, \dots, W_{n\delta})$.

We can indeed show that a process constructed this way is converging to the Wiener process as n goes to infinity.[1] However the discretization itself is not a Wiener process since we lose the independence of the increments by using linear interpolation.

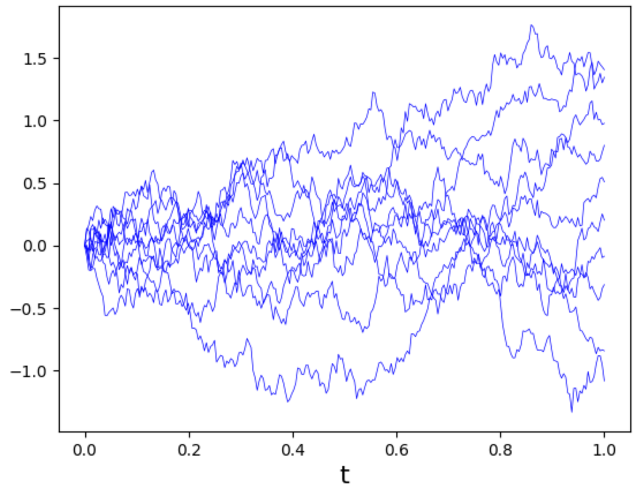

Figure 2.1: 10 realisations of discretizations of the Wiener process with n = 256 on [0,1].

[1]This fact is known as Donsker's invariance principle or as functional central limit theorem. See for example [1].

To get a refinement for a given discrete sample path, we can use the following procedure. As an input we give a realisation of a discrete sample path $(\overline{W}_{t_0}, \dots, \overline{W}_{t_n})$, n steps with step size δ. The output is a sample from the same process but with 2n steps: $(\overline{W}_{R,t_0}, \dots, \overline{W}_{R,t_n})$, which makes a step size of $\frac{\delta}{2}$. We can use this procedure recursively to get finer and finer sample paths and imitate the convergence stated above.[2]

Algorithm 2.5 (Refinement of a given discretized Wiener sample path)

```
1. For $j = 0$ to $n-1$
    a) Generate $z \sim \mathcal{N}(\frac{\overline{W}_{(j+1)\delta}+\overline{W}_{j\delta}}{2}, \frac{\delta}{4})$
    b) Set $\overline{W}_{R,j\delta} = \overline{W}_{j\delta}$
    c) Set $\overline{W}_{R,(j+\frac{1}{2})\delta} = z$
2. Set $\overline{W}_{R,n\delta} = \overline{W}_{n\delta}$
3. Linear interpolation.
```

This algorithm follows from the fact that $W_{(j+\frac{1}{2})\delta} |_{W_{j\delta}=a, W_{(j+1)\delta}=b} \sim \mathcal{N}(\frac{a+b}{2}, \frac{\delta}{4})$.

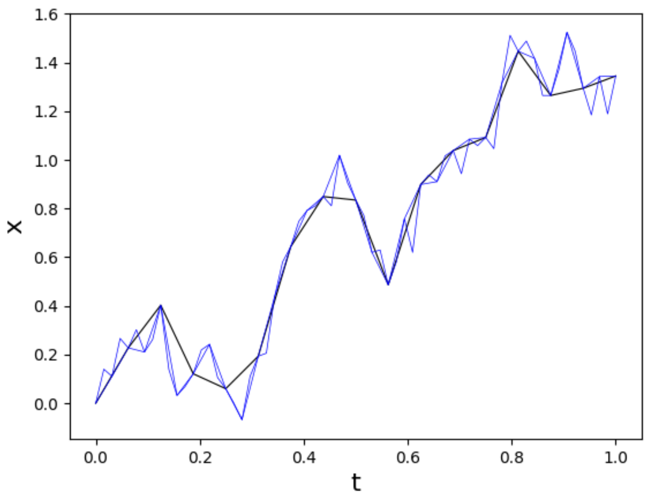

Figure 2.2: Finer versions of a given discretized Wiener process (black).

2.3 Stochastic integration: Itô and Stratonovich

In this section we will present 2 (of many other possible) definitions of the stochastic integral for a basic example. We will show that both definitions yield different values. Both definitions lead to 2 different stochastic calculii, the first one being called *Itô* - the second one *Stratonovich stochastic calculus*. In the next chapter (and in the rest of this thesis) we will focus on the definition due to Itô and it will become clear, why this definition is more practical.

[2]See also Paul Lévy's construction of the Wiener process in [9] which follows the same idea.

Our first attempt on defining the stochastic integral will be for the following example:

$$\int_0^T 2W_t \, \mathrm{d}W_t.$$

The idea is to construct a partition of [0,T] and to evaluate the function only on these countable amount of time intervals, similar to Riemann(-Stieltjes)-sum approximation in the discrete case. Then we take the limit of this sum and check for convergence in the $L^2[\Omega]$-norm. Thus the stochastic integral will be defined as a limit of a sum of random variables and is thus also a random variable on the same probability space, if it exists[3]. The limiting value will depend on which point on the intervals we evaluate the function. This is a counter-intuitive fact which does not hold in the deterministic case. If we evaluate the function at the start of each time interval, e.g. at t_j for $[t_j, t_{j+1}]$, we get the *Itô stochastic integral* and if we evaluate it at $t_{j+\frac{1}{2}}$, at the middle of each time interval, we get the *Stratonovich stochastic integral*.

Definition 2.6 (Riemann-sum approximation)
Let P_n be a partition of [0,T], where n denotes the amount of steps. For our purposes we will consider only constant step sizes δ, e.g. $P_n := \{t_0 = 0, \dots, t_n = n\delta = T\}$ where T is fixed and δ depends on n.

$$R_{1,n} := R_1(P_n) := \sum_{k=0}^{n-1} 2W_{t_k} \Delta W_{t_k} \qquad \text{(Itô)}$$

$$R_{2,n} := R_2(P_n) := \sum_{k=0}^{n-1} 2W_{t_{k+\frac{1}{2}}} \Delta W_{t_k} \qquad \text{(Stratonovich)}$$

where $\Delta W_{t_k} := W_{t_{k+1}} - W_{t_k}$ and $W_{t_{k+\frac{1}{2}}}$ is the value of the process taken in the center of the interval $[t_k, t_{k+1}]$.

The key question which arises is whether $R_{1,n}$ and $R_{2,n}$ converge (in $L^2[\Omega]$) as n goes to infinity (e.g. δ goes to 0). Indeed, using the proposition on quadratic variation which we will present later, we can show that the limits exist, but are not identical:

Lemma 2.7 (Stochastic integration of $\int_0^T 2W_t \, \mathrm{d}W_t$)
Let P_n be a partition of [0,T]. Then

$$R_{1,n} \to W_T^2 - T \quad (\textit{in } L^2[\Omega]) \textit{ as } n \to \infty \qquad \text{(Itô)}$$

$$R_{2,n} \to W_T^2 \quad (\textit{in } L^2[\Omega]) \textit{ as } n \to \infty \qquad \text{(Stratonovich)}$$

Notice also that $W_T^2 - T$ is a well-known martingale and W_T^2 is what we would get in the deterministic case.

In order to calculate this, we need the following proposition:

[3] The limit of a sequence of measurable functions remains measurable, if it exists.

Proposition 2.8 (Quadratic variation)
Let P_n be a partition of [0,T]. Note that the step size $\delta \to 0$ as $n \to \infty$. Then

$$\sum_{k=0}^{n-1}(W_{t_{k+1}} - W_{t_k})^2 \to T \text{ as } n \to \infty \qquad (\text{in } L^2[\Omega]).$$

This implies the heuristic $(dW_t)^2 = dt$ which is often seen in literature on stochastic calculus.

Proof We will use the independence property of the Wiener process.

$$\mathbb{E}[(\sum_{k=0}^{n-1}(W_{t_{k+1}} - W_{t_k})^2 - T)^2] = \mathbb{E}[(\sum_{k=0}^{n-1}((W_{t_{k+1}} - W_{t_k})^2 - \frac{T}{n}))^2]$$
$$= \mathbb{E}[(\sum_{k=0}^{n-1}(x_k^2 - \frac{T}{n}))^2] = \sum_{k,j=0}^{n-1}\mathbb{E}[(x_k^2 - \frac{T}{n})(x_j^2 - \frac{T}{n})]$$
$$(\text{where } W_{t_{k+1}} - W_{t_k} := x_k \sim \mathcal{N}(0, \frac{T}{n}))$$
$$= \sum_{l=0}^{n-1}\mathbb{E}[(x_k^2 - \frac{T}{n})^2] = \sum_{l=0}^{n-1}\text{Var}[x_k^2 - \frac{T}{n}] + \underbrace{\mathbb{E}[x_k^2 - \frac{T}{n}]^2}_{=0}$$
$$= \sum_{l=0}^{n-1}\text{Var}[x_k^2] = 2n(\frac{T}{n})^2 \to 0 \text{ as } n \to \infty. \qquad \square$$

Now we are able to show lemma 2.7.

Proof We will show the lemma only for the Itô-case:

$$\mathbb{E}[(\sum_{k=0}^{n-1} 2W_{t_k}(W_{t_{k+1}} - W_{t_k}) - (W_T^2 - T))^2]$$
$$= \mathbb{E}[(\sum_{k=0}^{n-1}(W_{t_{k+1}}^2 - W_{t_k}^2) - (W_{t_{k+1}} - W_{t_k})^2 - (W_T^2 - T))^2]$$
(Where we used that $2X(Y - X) = (Y^2 - X^2) - (X - Y)^2, \quad X := W_{t_k}, Y := W_{t_{k+1}}$)
$$= \mathbb{E}[(\sum_{k=0}^{n-1} -(W_{t_{k+1}} - W_{t_k})^2 + T)^2]$$
$$= \mathbb{E}[(\sum_{k=0}^{n-1}(W_{t_{k+1}} - W_{t_k})^2 - T)^2] \to 0, \text{ as } n \to \infty$$
using quadratic variation. $\square$

Is it possible to take other integrands than just $2W_t$? In the next chapter we will give the idea behind the construction of the Itô stochastic integral for a class of functions $\mathcal{C}$ and we will give an argument on why we choose the definition of Itô over the definition of Stratonovich.

Chapter 3

Itô Stochastic Calculus

In this section we will give the idea behind the construction of the stochastic integral in the Itô-sense for a certain class of functions $\mathcal{C}$. We will also present the fundamental theorem of stochastic calculus, namely *Itô's lemma*, and give examples. Later we will discuss an important application of this lemma, the so-called *stochastic Taylor expansion* (or *Wagner-Platen expansion*). We will use it to construct our numerical schemes which we will discuss in chapter 5.

3.1 Construction of the Itô stochastic integral

We want to define:

$$\int_0^T X_t \, \mathrm{d}W_t$$

for a large class of functions $X_t \in \mathcal{C}(0,T)$, $X_t \colon [0,T] \times \Omega \to \mathbb{R}$.

First let us define the space of square-integrable stochastic processes:

$$L^2[0,T] := \{X_t \colon [0,T] \times \Omega \to \mathbb{R} \mid X_t \text{ measurable}, \ \mathbb{E}[\int_0^T X_t^2 \mathrm{d}t] < \infty\}.$$

If we interpret the second condition as norm, we get the *complete normed space* $(L^2[0,T], \|\cdot\|_{L^2[0,T]})$. However we want to exclude functions from this class which are not $\mathcal{F}_t$-adapted. Let us consider now the subset $\mathcal{C} := \mathcal{C}[0,T]$ of all functions X_t in $L^2[0,T]$ which are adapted. A limit of a sequence of adapted processes remains adapted[1]. Thus $\mathcal{C}$ is a closed subset of $L^2[0,T]$ since it includes all its limits, which implies that $\mathcal{C}$ is also complete under the same norm. Thus we have the complete normed space $(\mathcal{C}, \|\cdot\|_{L^2[0,T]})$. Recall that we want to define the Itô stochastic integral for functions in $\mathcal{C}$. First we will define the stochastic integral for a simpler class of functions $\mathcal{E}$, s.t $\mathcal{E} \subset \mathcal{C}$, and then expand the definition to the space $\mathcal{C}$.

[1]See [6] for the proof.

Define the space of *random step functions* $\mathcal{E}$ which can be expressed in the following form:

$$\xi_t^n = \sum_{k=0}^{n-1} Z_k \cdot \mathbb{1}_{[t_k, t_{k+1})}$$

for a partition P_n of [0,T] and some random variables Z_k $\mathcal{F}_{t_k}$-measurable and square-integrable (this means Z_k is in $L_2[\Omega]$) for $k = 0, \ldots, n-1$. Obviously ξ_t^n is $\mathcal{F}_t$-adapted and $\mathcal{E} \subset \mathcal{C}$.

Definition 3.1 (Itô stochastic integral for random step functions)
Let $\xi_t^n \in \mathcal{E}$ be a random step function with the above representation. Then

$$\int_0^T \xi_t^n \, \mathrm{d}W_t := \sum_{k=0}^{n-1} Z_k \cdot (W_{t_{k+1}} - W_{t_k})$$

is the Itô stochastic integral of ξ_t^n. Note that the integral is a random variable and $\mathcal{F}_T$-measurable. If we allow the upper limit of the integral to be in the intervall [0,T], then the Itô stochastic integral becomes an adapted stochastic process. This would not be true in the Stratonovich calculus or if X_t were not adapted.

For $X_t \in \mathcal{C}$ we have the construction $\xi_t^n = \sum_{k=0}^{n-1} X_{t_k} \cdot \mathbb{1}_{[t_k, t_{k+1})}$ which is also adapted.

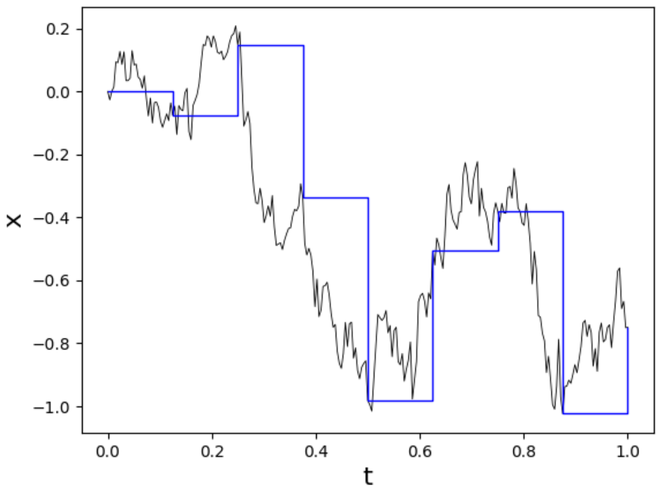

Figure 3.1: A sample of the Wiener process and its corresponding step function for n = 8.

The following result will be important for our construction of Itô stochastic integrals for functions in $\mathcal{C}$:

Proposition 3.2 (Itô-isometry for random step functions)
Let $\xi_t \in \mathcal{E}$. Then

$$\mathbb{E}[(\int_0^T \xi_t \, \mathrm{d}W_t)^2] = \mathbb{E}[\int_0^T (\xi_t)^2 \, \mathrm{d}t]$$

$$\| \int_0^T \xi_t \, \mathrm{d}W_t \|_{L^2[\Omega]} = \|\xi_t\|_{L^2[0,T]}.$$

We conclude that for all $\xi_t \in \mathcal{E}$ there exists $\int_0^T \xi_t \, \mathrm{d}W_t$ as an element in $L^2[\Omega]$.

Proof

$$\begin{aligned}
&\mathbb{E}[(\int_0^T \xi_t \, \mathrm{d}W_t)^2] \\
&= \mathbb{E}[(\sum_{k=0}^{n-1} Z_k \cdot (W_{t_{k+1}} - W_{t_k}))(\sum_{l=0}^{n-1} Z_l \cdot (W_{t_{l+1}} - W_{t_l}))] \\
&= \mathbb{E}[\sum_{k=0}^{n-1} Z_k^2 \cdot (W_{t_{k+1}} - W_{t_k})^2] + 2\mathbb{E}[\sum_{k>l=0}^{n-1} Z_k \cdot Z_l \cdot (W_{t_{k+1}} - W_{t_k})(W_{t_{l+1}} - W_{t_l})] \\
&= \sum_{k=0}^{n-1} \mathbb{E}[Z_k^2] \cdot \mathbb{E}[(W_{t_{k+1}} - W_{t_k})^2]
\end{aligned}$$

Where we used that Z_k^2, $(W_{t_{k+1}} - W_{t_k})^2$ and $Z_l \cdot (W_{t_{k+1}} - W_{t_k})(W_{t_{l+1}} - W_{t_l})$, Z_k are independent for $k > l$.

$$\sum_{k=0}^{n-1} \mathbb{E}[Z_k^2] \cdot (t_{k+1} - t_k) = \mathbb{E}[\int_0^T (\xi_t)^2 \, \mathrm{d}t] \qquad \square$$

The proof would not be possible in case of the Stratonovich calculus or if X_t were not adapted. For example if $X_t = W_{t+\frac{1}{2}}$.

Finally, using the following lemma, we can expand our definition of the stochastic integral from $\mathcal{E}$ to $\mathcal{C}$:

Lemma 3.3 (Itô stochastic integral)
Let $X_t \in \mathcal{C}$. Then there exists a sequence $(\xi_t^n)_{n \in \mathbb{N}} \in \mathcal{E}$ s.t. $\mathbb{E}[\int_0^T (X_t - \xi_t^n)^2 \mathrm{d}t] = \|X_t - \xi_t^n\|^2_{L^2[0,T]} \to 0$ as $n \to \infty$ and

$$\int_0^T \xi_t^n \mathrm{d}W_t \to \int_0^T X_t \mathrm{d}W_t \quad (\textit{in } L^2[\Omega]) \textit{ as } n \to \infty.$$

This means $\| \int_0^T X_t \mathrm{d}W_t - \int_0^T \xi_t^n \mathrm{d}W_t \|_{L^2[\Omega]} \to 0$ as $n \to \infty$. And we can define:

$$\int_0^T X_t \, \mathrm{d}W_t := \lim_{n \to \infty} \int_0^T \xi_t^n \, \mathrm{d}W_t \ (\textit{in } L^2[\Omega]).$$

Additionaly, the Itô-isometry holds also in $\mathcal{C}$.

Sketch of the construction:
The idea is to show that $\mathcal{E}$ is dense in $\mathcal{C}$ w.r.t the norm $\|\cdot\|_{L^2[0,T]}$ using an approximation procedure. This is technical, see [2] for the details. This means:
$\exists$ sequence $(\xi_t^n)_{n\in\mathbb{N}} \in \mathcal{E}$ s.t. $\|X_t - \xi_t^n\|_{L^2[0,T]} \to 0$ as $n \to \infty$ $\forall X_t \in \mathcal{C}$.
Since $L^2[0,T]$ is complete, $(\xi_t^n)_{n\in\mathbb{N}}$ is a Cauchy-sequence in this space. Using the Itô-isometry, we then have:
$\|\xi_t^n - \xi_t^m\|_{L^2[0,T]} = \| \int_0^T \xi_t^n \mathrm{d}W_t - \int_0^T \xi_t^m \mathrm{d}W_t\|_{L^2[\Omega]} \to 0$ as $n,m \to \infty$.
Hence $(\int_0^T \xi_t^n \mathrm{d}W_t)_{n\in\mathbb{N}}$ is a Cauchy-sequence in $L^2[\Omega]$. It is well known that this space is complete[2]. Thus, it makes sense to define:
$\int_0^T X_t\,\mathrm{d}W_t := \lim_{n\to\infty} \int_0^T \xi_t^n\,\mathrm{d}W_t \quad (\text{in } L^2[\Omega])$ as an element in $L^2[\Omega]$.

We still need to clarify why we choose Itô over Stratonovich. As can be seen, the definition due to Itô simplifies our calculations and enables us to define the stochastic integral in a straightforward way through the isometry. The isometry does not hold in the Stratonovich calculus. Additionaly we are able to identify the Itô stochastic integral as a martingale[3]. This is the main reason why we choose Itô over Stratonovich. However martingales will not be discussed in this thesis.

3.2 Lemma of Itô

The lemma of Itô is an important result. It enables us to evaluate stochastic integrals without using the definition as in section 2.7. The idea of the lemma is the following: Given a stochastic process X_t and its representation as stochastic integral equation, we may ask, what is the representation of $u(X_t)$ where u:$\mathbb{R} \to \mathbb{R}$ is a smooth map? We will present the lemma of Itô in 3 cases: In the first case, $X_t = W_t$ and we are interested in $u(W_t)$. Then we will consider more general stochastic processes X_t. Finally we will allow the mapping u:$\mathbb{R} \times [0,T] \to \mathbb{R}$ to depend on the paramter t.

Definition 3.4 (Itô process)
We will call a stochastic process X_t an Itô process if it has a known representation as stochastic integral equation of the following form:

$$X_t - X_0 = \int_0^t a(X_s, s)\,\mathrm{d}s + \int_0^t b(X_s, s)\,\mathrm{d}W_s$$

where $b(X_t,t) \in \mathcal{C}$, $a(X_t,t)$ $\mathcal{F}_t$-adapted and $\mathbb{E}[\int_0^T (a(X_t,t))^2 \mathrm{d}t] < \infty$.
Then $X_t \in L_2[\Omega]$ $\forall$ $t \in [0,T]$ fixed and we have the (symbolical) differential form:

$$\mathrm{d}X_t = a(X_t,t)\mathrm{d}t + b(X_t,t)\mathrm{d}W_t.$$

We will only consider the autonomous case where the functions $a(x,t) \equiv a(x)$ and $b(x,t) \equiv b(x)$ do not depend on the additional parameter t.

[2] $L^2[\Omega]$ is the usual space of square-integrable random variables on $(\Omega, \mathcal{F}, P)$.

[3] See [2] for the interpretation of Itô stochastic integrals as martingales. The expected value of the Itô stochastic integral is always 0 and the Itô stochastic integral can be defined as adapted stochastic process.

Lemma 3.5 (Itô lemma for the Wiener process)
Let W_t be the Wiener process. Let u:$\mathbb{R} \to \mathbb{R}$ be twice continuously differentiable with derivatives $u_x(x), u_{xx}(x)$. Define $Y_t := u(W_t)$. Then

$$Y_t - Y_0 = \int_0^t \frac{1}{2} u_{xx}(W_s)\,\mathrm{d}s + \int_0^t u_x(W_s)\,\mathrm{d}W_s$$

where equality is P-a.s. for $t \in [0, T]$.

Proof Proof follows from the general case, lemma 3.7, and by using that $W_t = \int_0^t \mathrm{d}W_s$. □

Lemma 3.6 (Itô lemma, non time-dependent case)
Let X_t be an Itô process. Let u:$\mathbb{R} \to \mathbb{R}$ be twice continuously differentiable with derivatives $u_x(x), u_{xx}(x)$. Define $Y_t := u(X_t)$. Then

$$Y_t - Y_0 = \int_0^t u_x(X_s)a(X_s) + \frac{1}{2} u_{xx}(X_s)b(X_s)^2\,\mathrm{d}s + \int_0^t u_x(X_s)b(X_s)\,\mathrm{d}W_s$$

where equality is P-a.s. for $t \in [0, T]$.

Proof See general case. □

Lemma 3.7 (Itô lemma, general case)
Let X_t be an Itô process. Let u:$\mathbb{R} \times [0, T] \to \mathbb{R}$ be twice continuously differentiable with partial derivatives $u_x(x,t), u_{xx}(x,t)$ and $u_t(x,t)$. Define $Y_t := u(X_t, t)$. Then

$$\begin{aligned} Y_t - Y_0 = \int_0^t u_t(X_s, s) + u_x(X_s, s)a(X_s)\ + \\ \frac{1}{2} u_{xx}(X_s, s)b(X_s)^2\,\mathrm{d}s + \int_0^t u_x(X_s, s)b(X_s)\,\mathrm{d}W_s \end{aligned}$$

where equality is P-a.s. for $t \in [0, T]$.

Proof See [2]. □

Let us consider some examples:

Example 3.8 Let $X_t = W_t$ and u: $x \mapsto x^2$. Then

$$\begin{aligned} W_t^2 &= \int_0^t \mathrm{d}s + \int_0^t 2W_s\,\mathrm{d}W_s \\ W_t^2 - t &= \int_0^t 2W_s\,\mathrm{d}W_s \quad \text{by rearrangement.} \end{aligned}$$

This is the same result as in lemma 2.7 where we used the definition to calculate this stochastic integral in the Itô sense. □

Example 3.9 Let $X_t - X_0 = \int_0^t \mu X_s \,\mathrm{d}s + \int_0^t \sigma X_s \,\mathrm{d}W_s$. μ, $\sigma > 0$ constants and u: $x \mapsto \ln(x)$ $(x \neq 0)$. Then

$$u_x(x) = \frac{1}{x} \quad \text{and} \quad u_{xx}(x) = -\frac{1}{x^2}$$

$$\begin{aligned} u(X_t) - u(X_0) &= \int_0^t u_x(X_s)\mu X_s + \frac{1}{2}u_{xx}(X_s)(\sigma X_s)^2 \,\mathrm{d}s + \int_0^t u_x(X_s)\sigma X_s \,\mathrm{d}W_s \\ &= \int_0^t \mu - \frac{1}{2}\sigma^2 \,\mathrm{d}s + \int_0^t \sigma \,\mathrm{d}W_s = (\mu - \frac{1}{2}\sigma^2)t + \sigma W_t. \end{aligned}$$

This just means:

$$\ln X_t = \ln X_0 + (\mu - \frac{1}{2}\sigma^2)t + \sigma W_t$$

where X_0 can be set as an initial value (can be a random variable under some conditions). This example will be of importance in the next chapter. □

3.3 Stochastic Taylor expansions

Given a deterministic and smooth function $X(t)$ and $\frac{\mathrm{d}}{\mathrm{d}t}X(t) = a(X(t))$ we can express it using the fundamental theorem of calculus as:

$$X(t) - X(0) = \int_0^t a(X(s)) \,\mathrm{d}s.$$

We can reapply the theorem on $a(X(s))$ and use the chain rule to get:

$$\begin{aligned} X(t) - X(0) &= \int_0^t a(X(0)) \ + \ \int_0^s a'(X(r)) \cdot a(X(r)) \,\mathrm{d}r \,\mathrm{d}s \\ &= a(X(0))t \ + \int_0^t \int_0^s a'(X(r)) \cdot a(X(r)) \,\mathrm{d}r \,\mathrm{d}s. \end{aligned}$$

Indeed we could show this representation using the Taylor expansion and by plugging in the derivative of $X(t)$.

The second integral is of quadratic order ($\mathcal{O}(t^2)$) (Where we assumed that the derivatives are bounded). If we truncate the second integral we can give a linear approximation of X(t) for t small, where $x_0 = X(0)$ is a given initial value. This is the simplest approximation for (deterministic) ordinary differential equations, the so-called *explicit Euler-scheme*.

We want to construct similar approximation schemes in the stochastic setting. Let X_t be a stochastic process and its representation as Itô process, where a:$\mathbb{R} \to \mathbb{R}$, b:$\mathbb{R} \to \mathbb{R}$ are smooth and have bounded derivatives (see 3.4):

$$X_t - X_0 = \int_0^t a(X_s) \,\mathrm{d}s + \int_0^t b(X_s) \,\mathrm{d}W_s.$$

Later we will call such equations *stochastic differential equations*, where $X_0 = x_0$ is a (possibly random) initial value.
Applying the lemma of Itô on $a(X_s)$ and $b(X_s)$ gives us:

$$a(X_s) = a(X_0) + \int_0^s a_x(X_r)a(X_r) + \frac{1}{2}a_{xx}(X_r)b(X_r)^2 \,\mathrm{d}r + \int_0^s a_x(X_r)b(X_r)\,\mathrm{d}W_r$$

$$b(X_s) = b(X_0) + \int_0^s b_x(X_r)a(X_r) + \frac{1}{2}b_{xx}(X_r)b(X_r)^2 \,\mathrm{d}r + \int_0^s b_x(X_r)b(X_r)\,\mathrm{d}W_r$$

and by plugging in to the original equation we get:

$$\begin{aligned} X_t - X_0 = &\int_0^t a(X_0) + \int_0^s a_x(X_r)a(X_r) + \frac{1}{2}a_{xx}(X_r)b(X_r)^2 \,\mathrm{d}r + \int_0^s a_x(X_r)b(X_r)\,\mathrm{d}W_r\,\mathrm{d}s \\ &+ \int_0^t b(X_0) + \int_0^s b_x(X_r)a(X_r) + \frac{1}{2}b_{xx}(X_r)b(X_r)^2 \,\mathrm{d}r + \int_0^s b_x(X_r)b(X_r)\,\mathrm{d}W_r\,\mathrm{d}W_s \end{aligned}$$

This means:

$$\begin{aligned} X_t = X_0 &+ \int_0^t a(X_0)\,\mathrm{d}s + \int_0^t\int_0^s a_x(X_r)a(X_r) + \frac{1}{2}a_{xx}(X_r)b(X_r)^2 \,\mathrm{d}r\,\mathrm{d}s \\ &+ \int_0^t\int_0^s a_x(X_r)b(X_r)\,\mathrm{d}W_r\,\mathrm{d}s \\ &+ \int_0^t b(X_0)\,\mathrm{d}W_s + \int_0^t\int_0^s b_x(X_r)a(X_r) + \frac{1}{2}b_{xx}(X_r)b(X_r)^2 \,\mathrm{d}r\,\mathrm{d}W_s \\ &+ \int_0^t\int_0^s b_x(X_r)b(X_r)\,\mathrm{d}W_r\,\mathrm{d}W_s \end{aligned}$$

This gives us the simplest *stochastic Taylor expansion*:

Proposition 3.10 (1. stochastic Taylor expansion)
Given the Itô process X_t as above. Then

$$\begin{aligned} X_t - X_0 &= \int_0^t a(X_0)\,\mathrm{d}s + \int_0^t b(X_0)\,\mathrm{d}W_s + R \\ &= a(X_0)t + b(X_0)W_t + R \end{aligned}$$

where R is the remainder term, containing in this case integrals of order t and above.

Note that integrals of the form $\int_0^t\int_0^s \mathrm{d}W_r\,\mathrm{d}s$ have higher order than t (in $L^2[\Omega]$). More important: We can give an estimate for the last integral in the remainder term:

$$\mathbb{E}[(\int_0^t\int_0^s b_x(X_r)b(X_r)\,\mathrm{d}W_r\,\mathrm{d}W_s)^2]^{\frac{1}{2}} = \mathbb{E}[\int_0^t\int_0^s (b_x(X_r)b(X_r))^2\,\mathrm{d}r\,\mathrm{d}s]^{\frac{1}{2}} = \mathcal{O}(t).$$

Where we used the boundedness of b and its derivative and the Itô-isometry twice.

The above expansion is insofar discrepant as it contains integrals of order t in both, the main term and the remainder term. This is not true in the deterministic case. However we can reapply the Itô lemma on $b_x(X_r)b(X_r)$ and plug it into the above equation. This gives us the second *stochastic Taylor expansion*:

Proposition 3.11 (2. stochastic Taylor expansion)
Given the Itô process X_t as above. Then

$$\begin{aligned} X_t - X_0 &= \int_0^t a(X_0)\,\mathrm{d}s + \int_0^t b(X_0)\,\mathrm{d}W_s + \int_0^t\int_0^s b_x(X_0)b(X_0)\mathrm{d}W_r\,\mathrm{d}W_s + R \\ &= a(X_0)t + b(X_0)W_t + \frac{1}{2}b_x(X_0)b(X_0)(W_t^2 - t) + R \end{aligned}$$

where R is the remainder term, containing in this case integrals of order strictly higher than t.

Through iterative usage of the Itô lemma we got an expansion which can be seen as a generalization of the Taylor expansion from deterministic calculus. Taylor expansions can be used to construct numerical schemes for ordinary differential equations. In the next chapter we will define stochastic differential equations as Itô processes and in chapter 5 we will use stochastic Taylor expansions to construct numerical schemes for stochastic differential equations.

Chapter 4

Stochastic Differential Equations

In this chapter we will define what we mean when we say that X_t is a *solution* (or *solution process*) of a stochastic differential equation and we will discuss the conditions under which the solution exists and its uniqueness. Furthermore we will give some examples of stochastic differential equations which can be solved analytically.

4.1 Existence and uniqueness of solutions

Definition 4.1 (Stochastic differential equation)
Let a:$\mathbb{R} \times [0,T] \to \mathbb{R}$ and b:$\mathbb{R} \times [0,T] \to \mathbb{R}$ be measurable functions. A stochastic differential equation (SDE) has the following form:

$$\text{SDE} \quad \begin{cases} \mathrm{d}X_t = a(X_t,t)\mathrm{d}t + b(X_t,t)\mathrm{d}W_t \\ X_0 = x_0 \ \textit{(initial value)} \end{cases}$$

for $t \in [0,T]$. We will give it the following (Itô-)representation:

$$X_t = x_0 + \int_0^t a(X_s,s)\,\mathrm{d}s + \int_0^t b(X_s,s)\,\mathrm{d}W_s \quad P\text{-a.s.}$$

and the following has to hold:

1. *x_0 is $\mathcal{F}_0$-measurable and $\mathbb{E}[|x_0|^2] < \infty$ (can be constant).*
2. *$a(X_t,t)$ and $b(X_t,t)$ as in definition 3.4.*

Lemma 4.2 (Existence and Uniqueness)
Given a stochastic differential equation as before. The solution X_t exists and is unique (P-a.s) if there are constants C, $L \in \mathbb{R}$ s.t:

1. $|a(x,t) - a(y,t)| \leq L|x-y|$
   $|b(x,t) - b(y,t)| \leq L|x-y|\ \forall x,y \in \mathbb{R}, t \in [0,T]$ *(globally Lipschitz)*
2. $|a(x,t)| \leq C(1+|x|)$
   $|b(x,t)| \leq C(1+|x|)\ \forall x \in \mathbb{R}, t \in [0,T]$ *(linear growth-bound)*

In the autonomous case, where $a(x,t) = a(x)$ and $b(x,t) = b(x)$, the Lipschitz-condition implies the second condition.

Proof See [2]. □

4.2 Analytically solvable equations

There are not many stochastic differential equations which have an analytical (closed-form) solution. We will present the *geometric Brownian motion* and the *Ornstein-Uhlenbeck process* which have applications in physics and mathematical finance. Both are solutions of stochastic differential equations. Knowing the exact solution allows us to test how good our approximation schemes are using simulation. This will be the subject of chapter 5.

4.2.1 Geometric Brownian motion

Proposition 4.3 (Geometric Brownian motion)
The stochastic differential equation is given by:

$$\mathrm{d}X_t = \mu X_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t, \quad t \in [0,T], \quad X_0 = x_0$$
$$X_t = x_0 + \int_0^t \mu X_s \,\mathrm{d}s + \int_0^t \sigma X_s \,\mathrm{d}W_s \quad P\text{-a.s.}$$

where x_0, μ, $\sigma \in \mathbb{R}$ *constant and* $\sigma > 0$. *Its solution is:*

$$X_t = x_0 \exp((\mu - \frac{\sigma^2}{2})t + \sigma W_t).$$

If we set $x_0 = 1$, then $X_t \sim \exp(Z_t)$ where $Z_t \sim \mathcal{N}((\mu - \frac{1}{2}\sigma^2)t, \sigma t)$. This means that X_t is lognormally distributed $\forall\, t \in [0,T]$.

Proof See example 3.9 and take the exponent. $\square$

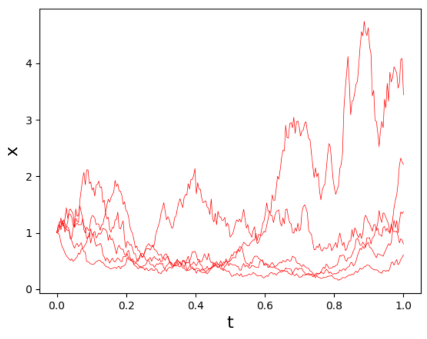

Figure 4.1: 5 samples of the geometric Brownian motion with $x_0 = 1, \mu = 1, \sigma = 1.5$.

4.2.2 Ornstein-Uhlenbeck process

Proposition 4.4 (Ornstein-Uhlenbeck process)
The stochastic differential equation is given by:

$$\mathrm{d}X_t = -\beta X_t \mathrm{d}t + \sigma \mathrm{d}W_t, \quad t \in [0,T], \quad X_0 = x_0$$
$$X_t = x_0 + \int_0^t -\beta X_s \,\mathrm{d}s + \int_0^t \sigma \,\mathrm{d}W_s \quad P\text{-a.s.}$$

where β, $\sigma \in \mathbb{R}$ constant and β, $\sigma > 0$. We will choose x_0 either to be constant or a normal-distributed random variable. Its solution is:

$$X_t = x_0 \exp(-\beta t) + \sigma \int_0^t \exp(-\beta(t-s))\,\mathrm{d}W_s.$$

However we are not able to simulate the stochastic integral $\int_0^{t_n} \exp(-\beta(t_n - s))\,\mathrm{d}W_s$ exactly given the time step t_n. We will use the following approximation and consider it as the exact solution, given a partition P_n and a discretized Wiener process: $\sum_{k=0}^{n-1} \exp(-\beta(t_n - t_k))(W_{t_{k+1}} - W_{t_k})$.

Proof Again we will use Itô's lemma: Let u: $(x,t) \mapsto x\exp(\beta t)$. Then

$u_x(x,t) = \exp(\beta t),\ u_{xx}(x,t) = 0,\ u_t(x,t) = x\beta\exp(\beta t)$

$$u(X_t,t) - u(X_0,0) = \int_0^t u_t(X_s,s) - \beta X_s \cdot u_x(X_s,s) + \frac{1}{2}u_{xx}(X_s,s)\sigma^2 \,\mathrm{d}s + \int_0^t \sigma u_x(X_s,s)\,\mathrm{d}W_s$$

$$X_t \exp(\beta t) - X_0 \exp(\beta 0) = \int_0^t \beta X_s \exp(\beta s) - \beta X_s \exp(\beta s) + 0 \,\mathrm{d}s + \int_0^t \sigma \exp(\beta s)\,\mathrm{d}W_s$$

$$X_t = x_0 \exp(-\beta t) + \sigma \int_0^t \exp(-\beta(t-s))\,\mathrm{d}W_s. \qquad \square$$

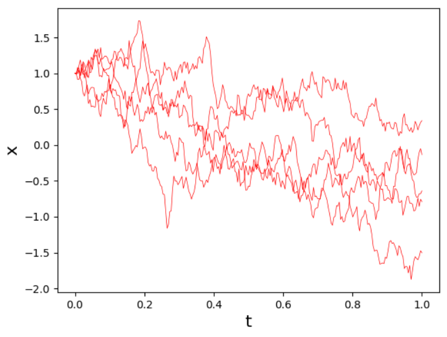

Figure 4.2: 5 samples of the Ornstein-Uhlenbeck process with $x_0 = 1, \beta = 1, \sigma = 1.5$.

Chapter 5

Numerical Analysis

In this chapter we will discuss numerical methods for solving stochastic differential equations based on the stochastic Taylor expansions which have been presented in section 3.3. These are also called *time discrete approximations* since we will give the approximative values on points of the time interval [0,T]. We will consider strongly converging schemes only. Strong means in this context that we want to approximate the *sample paths* of the true solution X_t of a stochastic differential equation. On the other hand, weakly converging schemes deal with the approximation of some funcionals of the true solution X_t, for example the expected value or the variance. Obviously, strongly converging schemes are also weakly converging.
These methods will be discussed: *Euler-Maruyama-scheme* and the *Milstein-scheme*.[1] We will prove the convergence of these methods and give the convergence order analytically. In the final sections of this chapter we will discuss the results and determine the convergence order numerically using simulations.

5.1 Strong convergence and error analysis

For the moment, let us consider a general setting. Let X_t be the solution of a stochastic differential equation and let $\overline{X}_t$ denote its approximation on [0,T]. We want that the sample paths on [0,T] of the approximation are close to the sample paths of the true solution (pathwise approximation). There are many different criterions to measure the closeness or the approximation error:

1. $\mathbb{E}[|X_T - \overline{X}_T|]$ (absolute error at T)
2. $\mathbb{E}[(X_T - \overline{X}_T)^2]^{\frac{1}{2}}$ (squared-mean error at T)
3. $\sup_{t\in[0,T]} \mathbb{E}[|X_t - \overline{X}_t|]$ (uniform absolute error)
4. $\sup_{t\in[0,T]} \mathbb{E}[(X_t - \overline{X}_t)^2]^{\frac{1}{2}}$ (uniform squared-mean error)

The first two criterions measure the error at the final time-point T. We will use the second criterion in our numerical simulations, since it is a bound for the

[1]There is a third method which we also implemented: the *Wagner-Platen-scheme*, but we will not give a discussion of this scheme and refer to [8] for the details, the scheme is also called *order 1.5 strong stochastic Taylor scheme* and it is also based on the stochastic Taylor expansion. See appendix B for the implementations and appendix C for graphics.

absolute error at T: $\mathbb{E}[|X_T - \overline{X}_T|] \leq \mathbb{E}[(X_T - \overline{X}_T)^2]^{\frac{1}{2}}$ and the implementation is simple. We assume that the approximation error is determined at the final time point T. However we will use the uniform squared-mean error criterion in our convergence proofs.

Now let $\overline{X}_t^n$ define a sequence of approximate solutions on partitions P_n of [0,T]. This means for each n we have a time-grid with n steps and step-size $\frac{T}{n}$. Now define the approximation error e_n for each n:

$$e_n := \mathbb{E}[(X_T - \overline{X}_T^n)^2]^{\frac{1}{2}}.$$

Obviously we want that $e_n \to 0$ as $n \to \infty$ holds for our approximation schemes. However for practical applications we are interested in how fast different approximation schemes converge to the true solution X_t for increasing n (where we use the criterion as norm). We want also to be able to compare different approximation schemes regarding their convergence speed.

Let us consider the following definition:

Definition 5.1 (Strong convergence and strong convergence order)
The approximation $\overline{X}_t^n$ converges to X_t in the strong sense if there are constants K and $\gamma > 0$ s.t:

$$e_n \leq K \cdot n^{-\gamma}$$

The convergence speed is determined by γ. Thus we will call γ the strong order of convergence *and we will say that $\overline{X}_t^n$ converges strongly with order γ to the true solution X_t for increasing n.*

It will be important to have an estimate for γ for our approximation schemes. This can be given either analytically or empirically through simulation. We will discuss both. For the empirical estimation we can justify heuristically:

$$\begin{aligned}
&\log(e_n) = \log(K) - \gamma \log(n) \\
&\log(e_{n+1}) = \log(K) - \gamma \log(n+1) \\
&\log(e_{n+1}) - \log(e_n) = -\gamma \cdot (\log(n+1) - \log(n)) \\
&\frac{\log(e_{n+1}) - \log(e_n)}{\log(n+1) - \log(n)} = -\gamma
\end{aligned}$$

We can plot the points $(\log(n), \log(e_n))$ for all our n and draw a regression line. The slope of the line is then an estimate for $-\gamma$. In our simulations we will choose the values of n to be powers of 2. This will allow us to use the refinement algorithm for the discretizations, see algorithm 2.5. In this case it makes sense to take the logarithm to base 2.

5.2 Strongly converging schemes

In this section we will present the Euler-Maruyama-scheme and the Milstein-scheme. Both schemes can be constructed by truncating stochastic Taylor expansions, which have been discussed in section 3.3. We will give the algorithm and prove their convergence properties. We will show that the Euler-scheme has strong order of convergence 0.5, and the Milstein-scheme has strong order of convergence 1 under some conditions on the coefficients.

Let us fix the following setting. We have given a stochastic differential equation:

$$\mathrm{d}X_t = a(X_t)\mathrm{d}t + b(X_t)\mathrm{d}W_t, \quad t \in [0,T], \quad X_0 = x_0$$

$$X_t = x_0 + \int_0^t a(X_s)\,\mathrm{d}s + \int_0^t b(X_s)\,\mathrm{d}W_s \quad \text{P-a.s.}$$

where the functions $a : \mathbb{R} \mapsto \mathbb{R}$, $b : \mathbb{R} \mapsto \mathbb{R}$ and the initial value x_0 are defined as in 3.4, and fulfill the properties of 4.2. Especially x_0 is square-integrable. Then since $X_t \in L_2[\Omega]$, $\mathbb{E}[X_t^2] < \infty\ \forall t \in [0,T]$.
Let $X_t : [0,T] \times \Omega \mapsto \mathbb{R}$ be the unique solution process of the stochastic differential equation. Let [0,T] be a time interval, T finite. Let $P_n = \{t_0 = 0, \ldots, t_n := T\}$ be a discretization of [0,T] with n steps and constant stepsize $\frac{T}{n}$. Let $\overline{X}^n_{t_k}$ denote the approximative solution (for given step-amount n) on the grid-points t_k for k = 0,. . . ,n.

5.2.1 Euler-Maruyama-scheme

The Euler-Maruyama-scheme can be obtained from the 1. stochastic Taylor expansion, see proposition 3.10, by giving the expansion for each $[t_k, t_{k+1}]$, $k = 0, \ldots, n-1$, and by truncating the remainder term.

Algorithm 5.2 (Euler-Maruyama-scheme)

```
1. Initialize $\overline{X}^n_{t_0} = x_0$
2. For k = 0 to n-1
      a) Generate $\triangle W_k \sim \mathcal{N}(0, \frac{T}{n})$
      b) Set $\overline{X}^n_{t_{k+1}} = \overline{X}^n_{t_k} + a(\overline{X}^n_{t_k})\triangle t_k + b(\overline{X}^n_{t_k})\triangle W_k$
3. Linear interpolation.
```

Where $\triangle t_k := t_{k+1} - t_k = \frac{T}{n}$ and $\triangle W_k = W_{t_{k+1}} - W_{t_k} \sim \mathcal{N}(0, \frac{T}{n})$ for k=0,. . . ,n-1.

This algorithm gives us values of the approximation on the discretization points and we can use linear interpolation if needed.

Theorem 5.3 (Strong convergence of the Euler-Maruyama-scheme)
Let L_a, L_b be constants. Let $a : \mathbb{R} \mapsto \mathbb{R}$ and $b : \mathbb{R} \mapsto \mathbb{R}$ satisfy $\forall x, y \in \mathbb{R}$:

1. $|a(x) - a(y)| \le L_a|x-y|$
   $|b(x) - b(y)| \le L_b|x-y|$
2. $a(x)^2 \le C(1+x^2)$
   $b(x)^2 \le C(1+x^2)$

Let $\overline{X}^n_{t_k}$ be an Euler-Maruyama approximation of the above equation for k = 0,. . . ,n with step size $\frac{T}{n}$. Let $\overline{X}^n_t$ be the constant extension[2] of the approximation. This means we will define:

$$\overline{X}^n_t := \overline{X}^n_{t_k} \quad \text{for } t \in [t_k, t_{k+1})$$

then:

$$\sup_{t\in[0,T]} \mathbb{E}[(X_t - \overline{X}^n_t)^2]^{\frac{1}{2}} = \sup_{t\in[0,T]} \|X_t - \overline{X}^n_t\|_{L^2[\Omega]} \le K \cdot n^{-\frac{1}{2}}$$

for some constant K.

[2]It is also possible to take linear interpolation instead, however the proof is simpler this way.

This means that the Euler-Maruyama approximation is converging strongly in $L_2[\Omega]$ to the true solution process with order 0.5 for t$\in [0,T]$. This is an uniform error-bound since the bound does not depend on t.

Proof Fix t$\in [t_k, t_{k+1}]$. Then:

$$\begin{aligned}
\|X_t - \overline{X}_t^n\|_{L^2[\Omega]} &= \| \int_0^t a(X_s)\,\mathrm{d}s + \int_0^t b(X_s)\,\mathrm{d}W_s - \int_0^{t_k} a(\overline{X}_s^n)\,\mathrm{d}s - \int_0^{t_k} b(\overline{X}_s^n)\,\mathrm{d}W_s \|_{L^2[\Omega]} \\
&= \| \int_{t_k}^t a(X_s)\,\mathrm{d}s + \int_0^{t_k} a(X_s) - a(\overline{X}_s^n)\,\mathrm{d}s \\
&+ \int_{t_k}^t b(X_s)\,\mathrm{d}W_s + \int_0^{t_k} b(X_s) - b(\overline{X}_s^n)\,\mathrm{d}W_s \|_{L^2[\Omega]}. \\
&\leq \| \int_{t_k}^t a(X_s)\,\mathrm{d}s \|_{L^2[\Omega]} + \| \int_0^{t_k} a(X_s) - a(\overline{X}_s^n)\,\mathrm{d}s \|_{L^2[\Omega]} \\
&+ \| \int_{t_k}^t b(X_s)\,\mathrm{d}W_s \|_{L^2[\Omega]} + \| \int_0^{t_k} b(X_s) - b(\overline{X}_s^n)\,\mathrm{d}W_s \|_{L^2[\Omega]}.
\end{aligned}$$

Where we used the triangle-inequality.
Now we will take the square and use:
$(a+b+c+d)^2 \leq 4(a^2+b^2+c^2+d^2) \quad \forall a,b,c,d \in \mathbb{R}$.

$$\begin{aligned}
\|X_t - \overline{X}_t^n\|_{L^2[\Omega]}^2 \leq 4(&\| \int_{t_k}^t a(X_s)\,\mathrm{d}s \|_{L^2[\Omega]}^2 + \| \int_0^{t_k} a(X_s) - a(\overline{X}_s^n)\,\mathrm{d}s \|_{L^2[\Omega]}^2 \\
&+ \| \int_{t_k}^t b(X_s)\,\mathrm{d}W_s \|_{L^2[\Omega]}^2 + \| \int_0^{t_k} b(X_s) - b(\overline{X}_s^n)\,\mathrm{d}W_s \|_{L^2[\Omega]}^2)
\end{aligned}$$

Now let us estimate each integral separately.

$$\begin{aligned}
\| \int_{t_k}^t a(X_s)\,\mathrm{d}s \|_{L^2[\Omega]}^2 = \mathbb{E}[(\int_{t_k}^t a(X_s)\,\mathrm{d}s)^2] &\leq \mathbb{E}[\int_{t_k}^t a(X_s)^2\,\mathrm{d}s \cdot \int_{t_k}^t \mathrm{d}s] \\
&\leq \frac{T}{n} \int_{t_k}^t \mathbb{E}[a(X_s)^2]\,\mathrm{d}s \leq C \cdot \frac{T}{n} \int_{t_k}^t \mathbb{E}[1+X_s^2]\,\mathrm{d}s \\
&\leq C \cdot (\frac{T}{n})^2 \cdot M
\end{aligned}$$

where $M := \sup_{s\in[0,T]} \mathbb{E}[1+X_s^2] < \infty$.

We used Cauchy-Schwarz, Fubini and the linear growth-bound.

$$\begin{aligned}
\| \int_0^{t_k} a(X_s) - a(\overline{X}_s^n)\, \mathrm{d}s \|_{L^2[\Omega]}^2 &= \mathbb{E}[(\int_0^{t_k} a(X_s) - a(\overline{X}_s^n)\, \mathrm{d}s)^2] \\
&\leq \mathbb{E}[\int_0^{t_k} (a(X_s) - a(\overline{X}_s^n))^2\, \mathrm{d}s \cdot \int_0^{t_k} \mathrm{d}s] \\
&\leq T \cdot L_a^2 \cdot \int_0^{t_k} \mathbb{E}[(X_s - \overline{X}_s^n)^2]\, \mathrm{d}s
\end{aligned}$$

We used again Cauchy-Schwarz, Fubini and the Lipschitz-condition.

$$\begin{aligned}
\| \int_{t_k}^{t} b(X_s)\, \mathrm{d}W_s \|_{L^2[\Omega]}^2 &= \mathbb{E}[(\int_{t_k}^{t} b(X_s)\, \mathrm{d}W_s)^2] = \mathbb{E}[\int_{t_k}^{t} b(X_s)^2\, \mathrm{d}s] \\
&= \int_{t_k}^{t} \mathbb{E}[b(X_s)^2]\, \mathrm{d}s \qquad \leq C \int_{t_k}^{t} \mathbb{E}[1 + X_s^2]\, \mathrm{d}s \\
&\leq C \cdot \frac{T}{n} \cdot M
\end{aligned}$$

We used the Itô-isometry, Fubini and the linear growth-bound.

$$\begin{aligned}
\| \int_0^{t_k} b(X_s) - b(\overline{X}_s^n)\, \mathrm{d}W_s \|_{L^2[\Omega]}^2 &= \mathbb{E}[(\int_0^{t_k} b(X_s) - b(\overline{X}_s^n)\, \mathrm{d}W_s)^2] \\
&= \mathbb{E}[\int_0^{t_k} (b(X_s) - b(\overline{X}_s^n))^2\, \mathrm{d}s] \\
&= \int_0^{t_k} \mathbb{E}[(b(X_s) - b(\overline{X}_s^n))^2]\, \mathrm{d}s \\
&\leq L_b^2 \cdot \int_0^{t_k} \mathbb{E}[(X_s - \overline{X}_s^n)^2]\, \mathrm{d}s
\end{aligned}$$

We used again the Itô-isometry, Fubini and the Lipschitz-condition.
We just showed:

$$\begin{aligned}
\|X_t - \overline{X}_t^n\|_{L^2[\Omega]}^2 &\leq 4CM\frac{T}{n}(1 + \frac{T}{n}) + 4(TL_a^2 + L_b^2) \int_0^{t_k} \|X_s - \overline{X}_s^n\|_{L^2[\Omega]}^2\, \mathrm{d}s \\
&\leq 4CM\frac{T}{n}(1 + \frac{T}{n}) + 4(TL_a^2 + L_b^2) \int_0^{t} \|X_s - \overline{X}_s^n\|_{L^2[\Omega]}^2\, \mathrm{d}s
\end{aligned}$$

Since t$\in [t_k, t_{k+1}]$.
Now we will use the Gronwall-inequality, see appendix A. This gives us:

$$\|X_t - \overline{X}^n_t\|^2_{L^2[\Omega]} \leq 4CM\frac{T}{n}(1+\frac{T}{n})\cdot\exp(4T(TL_a^2 + L_b^2))$$

Now we will use that a term of the form: $\frac{C_1}{n} + \frac{C_2}{n^2}$ can be estimated by $\frac{C_3}{n}$ from above for some constants. Taking the square-root gives us:

$$\|X_t - \overline{X}^n_t\|_{L^2[\Omega]} \leq K \cdot n^{-\frac{1}{2}}$$

for some constant K and t$\in [0,T]$. This completes the proof. □

5.2.2 Milstein-scheme

The Milstein-scheme can be obtained by truncating the remainder term of the 2. stochastic Taylor expansion, see proposition 3.11.

Algorithm 5.4 (Milstein-scheme)

```
1. Initialize $\overline{X}^n_{t_0} = x_0$
2. For k = 0 to n-1
        a) Generate $\triangle W_k \sim \mathcal{N}(0,\frac{T}{n})$
        b) Set $\overline{X}^n_{t_{k+1}} = \overline{X}^n_{t_k} + a(\overline{X}^n_{t_k})\triangle t_k + b(\overline{X}^n_{t_k})\triangle W_k$
            $+ \frac{1}{2}b(\overline{X}^n_{t_k})b_x(\overline{X}^n_{t_k})((\triangle W_k)^2 - \triangle t_k)$
3. Linear interpolation.
```

Where $\triangle t_k := t_{k+1} - t_k = \frac{T}{n}$ and $\triangle W_k = W_{t_{k+1}} - W_{t_k} \sim \mathcal{N}(0,\frac{T}{n})$ for k=0,...,n-1.

Theorem 5.5 (Strong convergence of the Milstein-scheme)
Let $a : \mathbb{R} \mapsto \mathbb{R}$ and $b : \mathbb{R} \mapsto \mathbb{R}$ be twice continuosly differentiable and satisfy $\forall x, y \in \mathbb{R}$ the Lipschitz conditions. The additional assumption here is that the derivatives exists and are also Lipschitz.

Let $\overline{X}^n_{t_k}$ be an Milstein approximation of the above equation for k = 0,...,n with step size $\frac{T}{n}$. Let $\overline{X}^n_t$ be the constant extension of the approximation. This means we will define:

$$\overline{X}^n_t := \overline{X}^n_{t_k} \quad \textit{for } t \in [t_k, t_{k+1})$$

then:

$$\sup_{t\in[0,T]} \mathbb{E}[(X_t - \overline{X}^n_t)^2]^{\frac{1}{2}} = \sup_{t\in[0,T]} \|X_t - \overline{X}^n_t\|_{L^2[\Omega]} \leq K \cdot n^{-1}$$

for some constant K.

This means that the Milstein approximation is converging strongly in $L_2[\Omega]$ to the true solution process with order 1 for t$\in [0,T]$. This is also an uniform error-bound since the bound does not depend on t.

Proof The proof is omitted. See [4] for the details and further references.

5.3 Numerical results

In this last section we want to illustrate our approximation schemes and verify the convergence orders numerically. The implementation of the algorithms can be seen in appendix B.
The following figures show sample paths of the geometric Brownian motion and the approximations using the Euler- and the Milstein-schemes for step amounts n = 32, 64. We used the refinement algorithm (algorithm 2.5) in order to generate the discretized Wiener process with 64 steps given the discretized Wiener process with 32 steps. Note also that the true solution is also given only on a discrete amount of points $(t_0, \dots, t_n)$. We then used linear interpolation for both, the approximate solution and the true sample path. We then made the approximation error visible by coloring the area between the approximation and the true sample path.

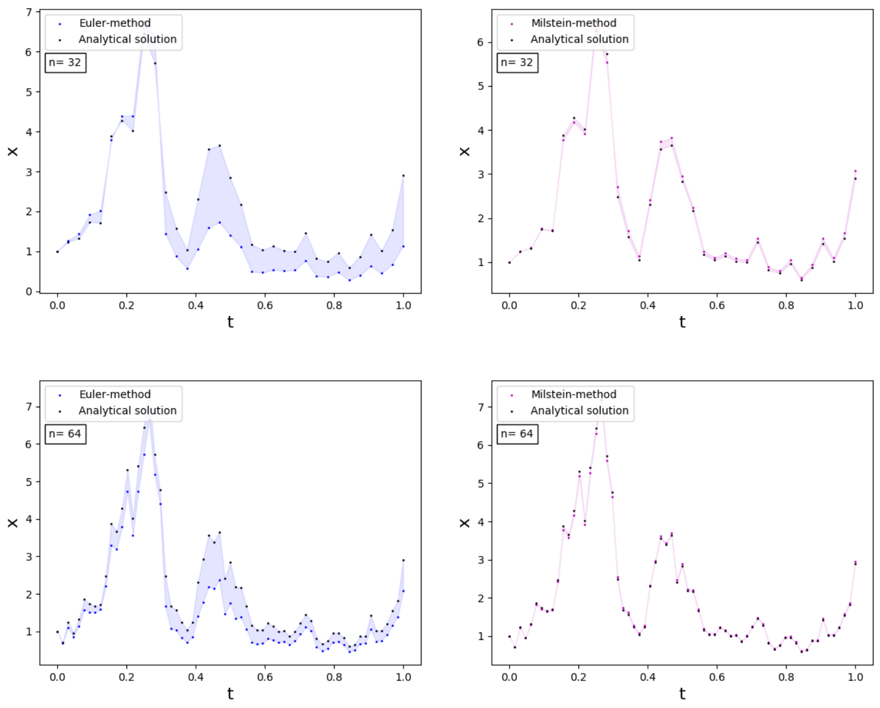

Figure 5.1: Approximation (colored dots) of the sample path of the geometric Brownian motion and the true sample path (black dots) for n = 32, 64.

We did the same analysis for the Ornstein-Uhlenbeck process. Note that this process has a constant diffusion term $(b(x) = \sigma)$. Thus the additional term of the Milstein-schem vanishes and the Euler- and the Milstein-scheme give the same results. Indeed the error-bound of the Milstein-scheme also applies to the Euler-Scheme for stochastic differential equations with constant diffusion terms. The Euler-scheme has convergence order 1.0 in this particular case. In the following figures we can see that both schemes yield the same result:

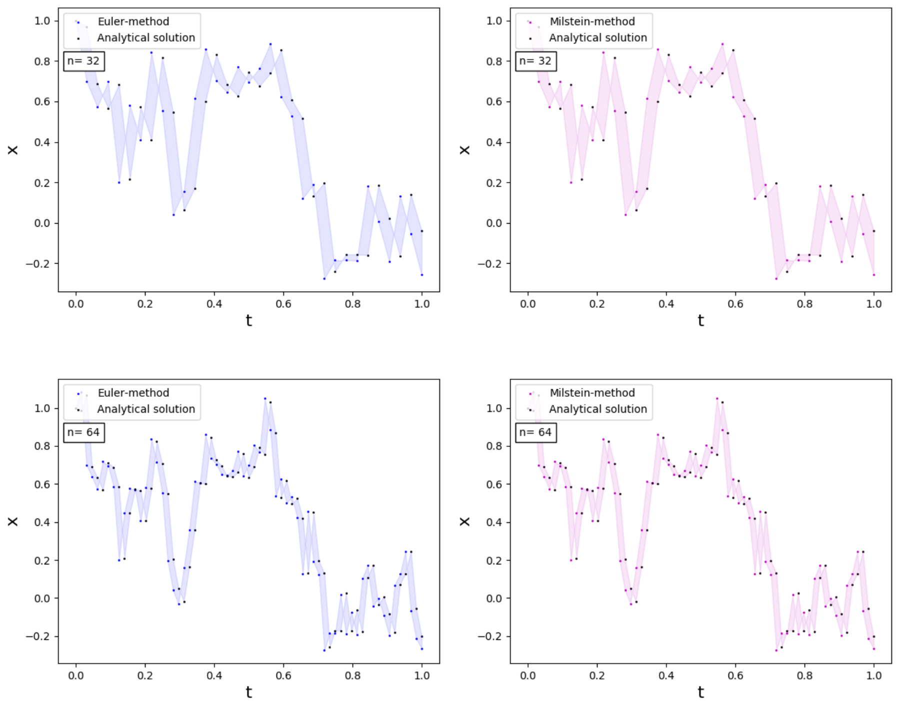

Figure 5.2: Approximation (colored dots) of the sample path of the Ornstein-Uhlenbeck process and the true sample path (black dots) for n = 32, 64.

The next figures (figure 5.3) verify that the approximation error e_n is tending to 0 for increasing step amount n for both, the geometric Brownian motion and the Ornstein-Uhlenbeck process.
The approximation errors e_n have been calculated for $n = 2^3, \dots, 2^{10}$. Recall:

$$e_n := \mathbb{E}[(X_T - \overline{X}_T^n)^2]^{\frac{1}{2}}.$$

We then took the Monte-Carlo estimates using m = 100'000 simulations for each n. The estimate $\hat{e}_n$ is defined as follows:

$$\hat{e}_n := \Big(\frac{1}{m}\sum_{k=1}^{m}(X_T^k - \overline{X}_T^{n,k})^2\Big)^{\frac{1}{2}}$$

for given sample paths $X_t^1, \dots, X_t^m$ and the corresponding approximations $\overline{X}_t^{n,1}, \dots, \overline{X}_t^{n,m}$. We did this analysis using the Euler- and the Milstein-scheme and for each step amount n which we set previously.

The final figures (figure 5.4) show the log-log plot for the estimates of the approximation error and the step amount n. As was discussed in section 5.1 the (absolute value) of the slope of the regression line is an estimate for the convergence order. The estimated convergence orders for the geometric Brownian motion match the analytical results. (0.5 for the Euler-scheme and

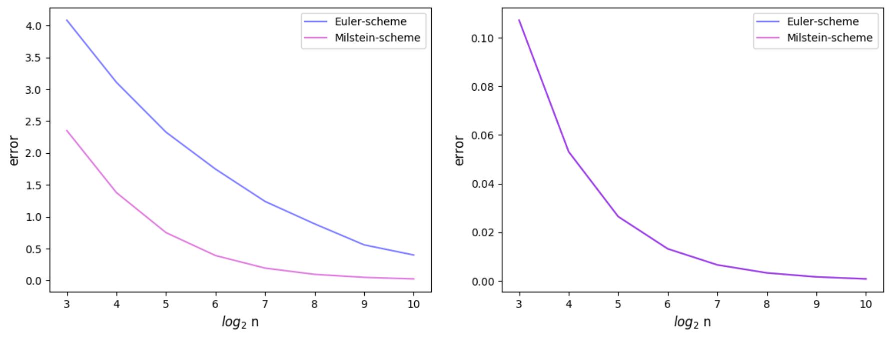

Figure 5.3: Monte-Carlo estimates of the approximation error and the step amount n (logarithmized) for the geometric Brownian motion (left) and the Ornstein-Uhlenbeck process (right).

1.0 for the Milstein-Scheme). The same analysis on the Ornstein-Uhlenbeck process yields an order of 1.0 for both schemes. However this is always the case if the diffusion term of the stochastic differential equation is constant, as we discussed previously. Our estimates for the convergence orders are:

1. Geometric Brownian motion:
   a) Euler: 0.482.
   b) Milstein: 0.950.
2. Ornstein-Uhlenbeck process:
   a) Euler: 1.003.
   b) Milstein: 1.003.

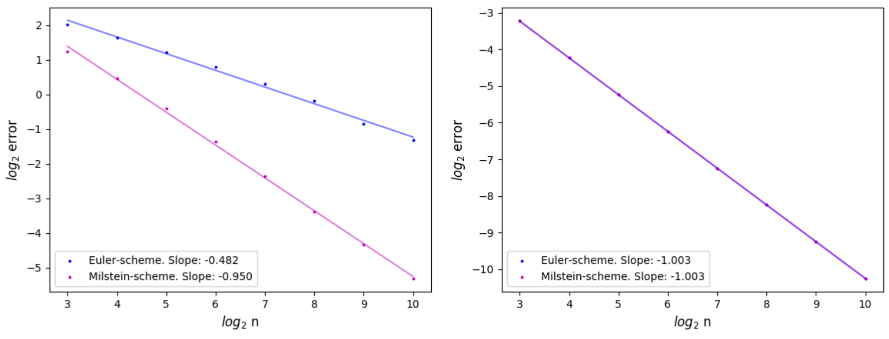

Figure 5.4: Log-log plot of the approximation error and the step amount n for the geometric Brownian motion (left) and the Ornstein-Uhlenbeck process (right).

5.4 Conclusion

We conclude that even for a low amount of discretization steps n, the Milstein-scheme gives an impressive approximation for the true sample paths. The implementation is not difficult and should always be prefered over the Euler-scheme. More graphics can be seen in appendix C. We also verified the (theoretical) convergence orders numerically.

However we remark that the conditions for both schemes (globally Lipschitz) are quite strong and therefore the schemes do not allow for coefficients which do not fulfill these conditions. There are many stochastic differential equations which have non-Lipschitz coefficients. Further research can be done by finding the convergence proofs which works for weaker conditions. Better (in terms of higher convergence order) schemes, if necessary, can be obtained by iteratively applying the Itô-lemma on the coefficients and by truncating these stochastic Taylor expansions (given that the coefficients are smooth enough). However these higher-order schemes become complex very quickly.

One might think that there is also a generalization of the Heun-method for stochastic differential equations, since also the Euler-Maruyama method is a generalization of a well-known method from deterministic analysis. The Heun-method is based on the trapezoid-rule from numerical integration theory. Since we are operating with Itô stochastic integrals, we cannot apply the trapezoid rule (recall the definition of the Itô stochastic integral). However there exist implicit schemes and also derivative-free schemes (Runge-Kutta) for stochastic differential equations and we refer to [8] for their treatment.

We end the thesis with the final remark that numerical schemes for stochastic differential equations are obtained in a similair way as their deterministic counterparts. We also used methods which are analogous to the Taylor expansion in order to construct our approximation schemes, by truncating these expansions. However we cannot blindly generalize methods from (deterministic) numerical analysis. We do need to take the stochastic setting into our consideration. In our case, this was the Itô stochastic calculus. Therefore it is inevitable to have profound knowledge in stochastic analysis, if one wishes to construct numerical schemes for stochastic differential equations.

Appendix A

Gronwall-inequality

Let φ and f be nonnegative and continuous functions defined on [0,T]. Let C denote a constant. If:

$$\varphi(t) \leq C + \int_0^t f(s)\varphi(s)\,\mathrm{d}s \quad \text{for all } t \in [0,T]$$

then:

$$\varphi(t) \leq C \cdot \exp\Big(\int_0^t f(s)\,\mathrm{d}s\Big) \quad \text{for all } t \in [0,T].$$

Proof See [7]. □

Appendix B

Code for algorithms and graphics

We used the programming environment Python 2.7 for the implementation of the algorithms. Especially the packages Matplotlib and Numpy have been very helpful. Informations can be found online. Contact amr.umeri@outlook.com for the source-code.

```python
###############################################################################
###This is the main module which consists of all algorithms.
###Can be imported as package into other scripts.
### - Discretization of the Wiener process
### - Discretization of the time-interval
### - Refinement algorithm for the Wiener process (see thesis for explanation)
### - Numerical methods: Euler-Maruyama, Milstein and Wagner-Platen
###Author: Amr Umeri
###NumericalSDE.py
###Python 2.7
###############################################################################

from math import sqrt
import numpy as np
from scipy.stats import norm

#Global constant T. Can be changed
T=1
#Generates the increments for the Wiener process starting with 0
#n equals the amount of discetization points excluding 0
def wiener(n):
    dt = float(T)/n
    # generate a sample of n numbers from a
    # normal distribution and insert 0 as starting value
    rval = np.insert(norm.rvs(size=n, scale=sqrt(dt)),0,0)

    # This computes the Wiener process by forming the cumulative sum of
    # the random samples and returns its values
    return np.cumsum(rval)

#returns a time-grid of [0, T] with n+1 (including 0) discretization points
def timegrid(n):
    return np.linspace(0.0, T, n+1)

#returns a finer version of a given Wiener process
#see thesis for explanations
def refineWiener(a):
    n=a.size-1
    dt=float(T)/n

    rval=np.empty(a.size*2-1)

    for k in range((a.size-1)):
        rval[2*k] = a[k]
        rval[2*k+1] = norm.rvs(size=1, loc=(a[k]+a[k+1])/2, scale=sqrt(dt/4))
        ##prove it

    rval[a.size*2-2] = a[a.size-1]

    return rval

#Euler-scheme
def sde_euler(x0, a, b, w):
    n = w.size-1
    dt = float(T)/n
    Xval = np.zeros(n+1)
    Xval[0] = x0
    for k in range(0,n):
        a_val           =   a(Xval[k])
        b_val           =   b(Xval[k])

        Xval[k+1] = Xval[k] + a_val*dt + b_val*(w[k+1]-w[k])
    return Xval

#Milstein-scheme
def sde_milstein(x0, a, b, b_dv, w):
    n = w.size-1
    dt = float(T)/n
    Xval = np.zeros(n+1)
    Xval[0] = x0
    for k in range(0,n):
        a_val           =   a(Xval[k])
        b_val           =   b(Xval[k])
        b_dv_val        =   b_dv(Xval[k])

        Xval[k+1] = (Xval[k] + a_val*dt + b_val*(w[k+1]-w[k])
                    + float(1)/2*b_val*b_dv_val*((w[k+1]-w[k])**2-dt))
    return Xval



#Wagner-Platen-scheme
def sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w):
    n = w.size-1
    dt = float(T)/n

    Z = np.zeros(n)
    for k in range(0,n):
        Z[k] = 0.5*dt**1.5*((w[k+1]-w[k])/sqrt(dt) + norm.rvs()/sqrt(3))

    Xval = np.zeros(n+1)
    Xval[0] = x0
    for k in range(0,n):
        a_val           =   a(Xval[k])
        b_val           =   b(Xval[k])
        a_dv_val        =   a_dv(Xval[k])
        b_dv_val        =   b_dv(Xval[k])
        a_dvdv_val      =   a_dvdv(Xval[k])
        b_dvdv_val      =   b_dvdv(Xval[k])

        Xval[k+1] = (Xval[k] + a_val*dt + b_val*(w[k+1]-w[k])
                    + float(1)/2*b_val*b_dv_val*((w[k+1]-w[k])**2-dt)
                    + a_dv_val*b_val*Z[k]
                    + float(1)/2*(a_val*a_dv_val + float(1)/2*b_val**2*a_dvdv_val)*dt**2
                    + (a_val*b_dv_val +
                    float(1)/2*b_val**2*b_dvdv_val)*((w[k+1]-w[k])*dt-Z[k])
                    + float(1)/2*b_val*(b_val*b_dvdv_val +
                    (b_dv_val)**2)*(float(1)/3*(w[k+1]-w[k])**2-dt)*(w[k+1]-w[k]))
    return Xval
```

```python
##############################################################################
###Code for figure 2.1
###Generating m paths of a standard Wiener process on the time intervall [0,T]
###SamplingWienerProcess.py
###Python 2.7
##############################################################################
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from NumericalSDE import *

#Parameters for the discretization
n =2**8
#Time discredized [0,T] in total n+1 elements (including starting value 0)
t = timegrid(n)
#m discretized Wiener processes
m = 10
w = np.zeros((n+1,m))
for k in range(0,m):
    w[:,k] = wiener(n)

#Plot the Wiener Processes
for sample_path in w.T:
    plt.plot(t, sample_path,'b',linewidth=0.5)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)
plt.show()

```

```python
##############################################################################
###Code for figure 2.2
###Plots a discretized Wiener process and finer versions of the same process
###using the refinement algorithm (see module NumericalSDE)
###WienerRefinementGraphic.py
###Python 2.7
##############################################################################
import numpy as np
import matplotlib.pylab as plt
from NumericalSDE import *

#Number of steps.
n = 16
#Create an empty array to store the realizations
en = n
x = wiener(en)
y = refineWiener(x)
z = refineWiener(y)

#timegrids
t_1 = timegrid(en)
t_2 = timegrid(2*en)
t_3 = timegrid(4*en)

#plot
plt.plot(t_1, x,'k', linewidth=0.8)
plt.plot(t_2, y,'r', linewidth=0.5, c='b')
plt.plot(t_3, z,'r', linewidth=0.5, c='b')

plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)
plt.show()


```

```python
############################################################################
###Code for figure 3.1
###plots the Wiener process and its corresponding step function
###WienerStepProcessGraphic.py
###Python 2.7
############################################################################
import numpy as np
import matplotlib.pylab as plt
from NumericalSDE import *

#Number of steps.
n = 16**2
n_step = n/(2*16)

#Create an empty array to store the realizations.
w = wiener(n)
t = timegrid(n)

#Step function
w_step = np.zeros(n_step+1)
t_step = timegrid(n_step)


for k in range(0,n_step):
    w_step[k] = w[k*(n/n_step)]
w_step[n_step] = w[n-1]

#plot
plt.plot(t, w,'k', linewidth=0.6)
plt.step(t_step, w_step,'k', linewidth=1,color='b', where='post')

plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)
plt.show()


```

```python
##############################################################################
###Code for figure 4.1
###Generating m paths of the geometric brownian motion (GBM)
###on the time intervall [0,T]
###SDE_SamplingGBM.py
###Python 2.7
##############################################################################

import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from NumericalSDE import *

#################################################
###Geometric brownian motion (SDE)
### dXt = a(Xt)dt + b(Xt)dWt
### X0 = x0
### a(x) = mu*x, b(x) = sigma*x
### mu, sigma constants
### True solution:
###
#################################################
#Parameter
sigma = 1.5
mu = 1.0
#starting value x0
x0 = 1
#Parameters for the discretization
n =2**8
t = timegrid(n)
#m discretized Wiener processes
m = 5
w = np.zeros((n+1,m))
for k in range(0,m):
    w[:,k] = wiener(n)
#m sample paths
Xt = np.zeros((n+1,m))
for k in range(0,m):
    Xt[0,k] = x0
    for j in range(0,n):
        Xt[j+1,k] = Xt[0,k]*np.exp((mu-sigma**2/2)*(t[k+1]) + sigma*w[j+1,k])


#Plot
for sample_path in Xt.T:
    plt.plot(t, sample_path,'r',linewidth=0.5)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)
plt.show()
```

```python
##############################################################################
###Code for figure 4.2
###Generating m paths of the Ornstein-Uhlenbeck process (OU)
###on the time intervall [0,T]
###SDE_SamplingOU.py
###Python 2.7
##############################################################################

import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from NumericalSDE import *

#################################################
### Ornstein-Uhlenbeck process (OU)
### dXt = a(Xt)dt + b(Xt)dWt
### X0 = x0
### a(x) = -beta*x, b(x) = sigma
### beta, sigma positive constants
### True solution:
###
#################################################
#Parameter
sigma = 1.5
beta = 1.0
#starting value x0
x0 = 1
#Parameters for the discretization
n =2**8
t = timegrid(n)
#m discretized Wiener processes
m = 5
w = np.zeros((n+1,m))
for k in range(0,m):
    w[:,k] = wiener(n)
#m sample paths
Xt = np.zeros((n+1,m))
stochIntApprox = np.zeros((n+1,m))
temp = np.zeros(n+1)
for s in range(0,m):
    Xt[0,s] = x0
    for k in range(0,n):
        for j in range(0,k):
            temp[j+1] =(w[j+1,s]-w[j,s])*np.exp((-beta)*((t[k+1])-t[j]))
        stochIntApprox[k+1,s] = np.sum(temp)
        temp = np.zeros(n)
        Xt[k+1,s] = Xt[0,s]*np.exp((-beta)*(t[k+1])) + sigma*stochIntApprox[k+1,s]


#The code below is needed to test if the approximated stoch. integral is
#a good approximation or not (can be removed)
##def a(x):
##    return -beta*x
##def b(x):
##    return sigma
##Yt  = sde_euler(x0, a, b, w[:,1])
##plt.scatter(t, Yt, 1, c='b')

#Plot
for sample_path in Xt.T:
    plt.plot(t, sample_path,'r',linewidth=0.5)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)
plt.show()
```

```python
############################################################################
###Plots the true solution and the approximation on n (and 2*n) time-points for the
###Geometric brownian motion
###Euler-Maruyama, Milstein (optional: Wagner-Platen)
###SDE_ApproximationGBM.py
###Python 2.7
############################################################################

import numpy as np
import matplotlib.pylab as plt
from NumericalSDE import *

################################################
###Geometric brownian motion (SDE)
### dXt = a(Xt)dt + b(Xt)dWt
### X0 = x0
### a(x) = mu*x, b(x) = sigma*x
### mu, sigma constants
### True solution:
###
################################################
#Parameter
sigma = 2.0
mu = 1.0
#functions a, b
def a(x):
    return mu*x
def b(x):
    return sigma*x
#derivativs of a, b
def a_dv(x):
    return mu
def b_dv(x):
    return sigma
def a_dvdv(x):
    return 0
def b_dvdv(x):
    return 0
#starting value x0
x0 = 1
################################################
# Number of steps.
n = 32
#Wiener process w, discretization of [0,T] t
w = wiener(n)
t = timegrid(n)
#Finer versions(using refinmenent)
w2 = refineWiener(w)
t2 = timegrid(2*n)
################################################
##############Numerical solutions##############
################################################
#Numerical solution: Euler-scheme
Yt_euler = sde_euler(x0,a,b,w)
Yt2_euler = sde_euler(x0,a,b,w2)
#Numerical solution: Milstein-scheme
Yt_milstein = sde_milstein(x0, a, b, b_dv, w)
Yt2_milstein = sde_milstein(x0, a, b, b_dv, w2)
#Numerical solution: Wagner-Platen-scheme
#Yt_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w)
#Yt2_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w2)

#Analytical solution (Geometric brownian motion) (adapt this for other SDEs.)
Xt=np.zeros(n+1)
Xt2=np.zeros(2*n+1)
Xt[0] = x0
Xt2[0] = x0
```

```python
for k in range(0,n):
    Xt[k+1] = Xt[0]*np.exp((mu-sigma**2/2)*(t[k+1]) + sigma*w[k+1])
for k in range(0,2*n):
    Xt2[k+1] = Xt2[0]*np.exp((mu-sigma**2/2)*(t2[k+1]) + sigma*w2[k+1])

#plot
plt.figure(1)
plt.scatter(t, Yt_euler, 1, c='b', label="Euler-method")
plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
plt.fill_between(t, Xt, Yt_euler, color='b',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

plt.figure(2)
plt.scatter(t, Yt_milstein, 1, c='m', label="Milstein-method")
plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
plt.fill_between(t, Xt, Yt_milstein, color='m',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

##plt.figure(3)
##plt.scatter(t, Yt_wagnerplaten, 1, c='forestgreen', label="Wagner-Platen-method")
##plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
##plt.fill_between(t, Xt, Yt_wagnerplaten, color='forestgreen',alpha=.1, interpolate=False)
##plt.legend(loc='upper left')
##ax = plt.gca()
##plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform = ax.transAxes)
##plt.xlabel('t', fontsize=16)
##plt.ylabel('x', fontsize=16)

plt.figure(4)
plt.scatter(t2, Yt2_euler, 1, c='b', label="Euler-method")
plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
plt.fill_between(t2, Xt2, Yt2_euler, color='b',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

plt.figure(5)
plt.scatter(t2, Yt2_milstein, 1, c='m', label="Milstein-method")
plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
plt.fill_between(t2, Xt2, Yt2_milstein, color='m',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

##plt.figure(6)
##plt.scatter(t2, Yt2_wagnerplaten, 1, c='forestgreen', label="Wagner-Platen-method")
##plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
##plt.fill_between(t2, Xt2, Yt2_wagnerplaten, color='forestgreen',alpha=.1, interpolate=False)
```

```
##plt.legend(loc='upper left')
#ax = plt.gca()
#plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
##plt.xlabel('t', fontsize=16)
##plt.ylabel('x', fontsize=16)

plt.show()

```

```python
##############################################################################
###Plots the true solution and the approximation on n (and 2*n) time-points for the
###Ornstein-Uhlenbeck process
###Euler-Maruyama, Milstein (optional: Wagner-Platen)
###SDE_ApproximationOU.py
###Python 2.7
##############################################################################

import numpy as np
import matplotlib.pylab as plt
from NumericalSDE import *

##################################################
### Ornstein-Uhlenbeck process (OU)
### dXt = a(Xt)dt + b(Xt)dWt
### X0 = x0
### a(x) = -beta*x, b(x) = sigma
### beta, sigma positive constants
### True solution:
###
##################################################
#Parameter
sigma = 1.5
beta = 1.0
#functions a, b
def a(x):
    return -beta*x
def b(x):
    return sigma
#derivativs of a, b
def a_dv(x):
    return -beta
def b_dv(x):
    return 0
def a_dvdv(x):
    return 0
def b_dvdv(x):
    return 0
#starting value x0
x0 = 1
##################################################
# Number of steps.
n = 32
#Wiener process w, discretization of [0,T] t
w = wiener(n)
t = timegrid(n)
#Finer versions(using refinmenent)
w2 = refineWiener(w)
t2 = timegrid(2*n)
##################################################
###############Numerical solutions###############
##################################################
#Numerical solution: Euler-scheme
Yt_euler = sde_euler(x0,a,b,w)
Yt2_euler = sde_euler(x0,a,b,w2)
#Numerical solution: Milstein-scheme
Yt_milstein = sde_milstein(x0, a, b, b_dv, w)
Yt2_milstein = sde_milstein(x0, a, b, b_dv, w2)
#Numerical solution: Wagner-Platen-scheme
Yt_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w)
Yt2_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w2)

#Analytical solution (Ornstein-Uhlenbeck process) (adapt this for other SDEs.)
temp = np.zeros(n+1)
Xt=np.zeros(n+1)
stochInt = np.zeros(n+1)

```

```python
temp2 = np.zeros(2*n+1)
Xt2=np.zeros(2*n+1)
stochInt2 = np.zeros(2*n+1)

Xt[0] = x0
Xt2[0] = x0
stochInt[0] = 0
stochInt2[0] = 0
temp[0] = 0
temp2[0] = 0
for k in range(0,n):
    for j in range(0,k):
        temp[j+1] =(w[j+1]-w[j])*np.exp((-beta)*((t[k+1])-t[j]))
    stochInt[k+1] = np.sum(temp)

    Xt[k+1] = Xt[0]*np.exp((-beta)*(t[k+1])) + sigma*stochInt[k+1]




for k in range(0,2*n):
    #Xt2[k+1] = Xt[0]*np.exp((mu-sigma**2/2)*(t2[k+1]) + sigma*w2[k+1])
    for j in range(0,k):
        temp2[j+1] =(w2[j+1]-w2[j])*np.exp((-beta)*((t2[k+1])-t2[j]))
    stochInt2[k+1] = np.sum(temp2)

    Xt2[k+1] = Xt2[0]*np.exp((-beta)*(t2[k+1])) + sigma*stochInt2[k+1]
#plot
plt.figure(1)
plt.scatter(t, Yt_euler, 1, c='b', label="Euler-method")
plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
plt.fill_between(t, Xt, Yt_euler, color='b',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

plt.figure(2)
plt.scatter(t, Yt_milstein, 1, c='m', label="Milstein-method")
plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
plt.fill_between(t, Xt, Yt_milstein, color='m',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

##plt.figure(3)
##plt.scatter(t, Yt_wagnerplaten, 1, c='forestgreen', label="Wagner-Platen-method")
##plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
##plt.fill_between(t, Xt, Yt_wagnerplaten, color='forestgreen',alpha=.1,
interpolate=False)
##plt.legend(loc='upper left')
##ax = plt.gca()
##plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
##plt.xlabel('t', fontsize=16)
##plt.ylabel('x', fontsize=16)

plt.figure(4)
plt.scatter(t2, Yt2_euler, 1, c='b', label="Euler-method")
plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
plt.fill_between(t2, Xt2, Yt2_euler, color='b',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
```

```python
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

plt.figure(5)
plt.scatter(t2, Yt2_milstein, 1, c='m', label="Milstein-method")
plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
plt.fill_between(t2, Xt2, Yt2_milstein, color='m',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

##plt.figure(6)
##plt.scatter(t2, Yt2_wagnerplaten, 1, c='forestgreen', label="Wagner-Platen-method")
##plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
##plt.fill_between(t2, Xt2, Yt2_wagnerplaten, color='forestgreen',alpha=.1,
interpolate=False)
##plt.legend(loc='upper left')
#ax = plt.gca()
#plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
##plt.xlabel('t', fontsize=16)
##plt.ylabel('x', fontsize=16)

plt.show()

```

```python
###############################################################################
###Code for the Monte-Carlo estimation of the convergence order for the
###geometric browian motion
###MC_ErrorAnalysis_ConvergenceOrderGBM.py
###Python 2.7
###############################################################################

import numpy as np
import matplotlib.pylab as plt
from scipy.stats import linregress
from NumericalSDE import *

##################################################
###Geometric brownian motion (SDE)
### dXt = a(Xt)dt + b(Xt)dWt
### X0 = x0
### a(x) = mu*x, b(x) = sigma*x
### mu, sigma constants
### True solution:
###
##################################################
#Parameter
sigma = 1.5
mu = 1.0
#functions a, b
def a(x):
    return mu*x
def b(x):
    return sigma*x
#derivativs of a, b
#def a_dv(x):
#    return mu
def b_dv(x):
    return sigma
#def a_dvdv(x):
#    return 0
#def b_dvdv(x):
#    return 0
#starting value x0
x0 = 1
##############################################################################
#Number of steps. We will do the Monte-Carlo-analysis for the
#step amounts n in nval() starting with n we will then always
#double the amount of steps.
n = 8
nval = np.array([n, n*2, n*4, n*8, n*16, n*32, n*64, n*128])
nsize = np.size(nval)
#Number of MC simulations for each step amount n
nsim = 20000

###############################################################################
#Arrays for the errors for each simulation and corresponding n.
#we are collecting the errors of the Euler, Milstein and Wagner-Platen-scheme.
error_euler = np.zeros((nsim,nsize))
error_milstein = np.zeros((nsim,nsize))
#error_wagnerplaten = np.zeros((nsim,nsize))
#empty arrays
Yt_euler = np.zeros(nsize)
Yt_milstein = np.zeros(nsize)
#Yt_wagnerplaten = np.zeros(nsize)
Xt_T = np.zeros(nsize)

###############################################################################
#Monte-Carlo-algorithm
for m in range(0,nsim):
    w = wiener(nval[0])
    t = timegrid(nval[0])
    for k in range(0,nsize):
        Yt_euler = sde_euler(x0,a,b,w)
        Yt_milstein = sde_milstein(x0,a,b,b_dv,w)
        #Yt_wagnerplaten = sde_wagnerplaten(x0,a,b,a_dv,b_dv,a_dvdv,b_dvdv,w)
        Xt_T = x0*np.exp((mu-sigma**2/2)*(t[nval[k]]) + sigma*w[nval[k]])

        w = refineWiener(w)
        t = timegrid(nval[k]*2)

        #Squared-mean criterion for the error
        error_euler[m,k] = (Yt_euler[nval[k]] - Xt_T)**2
        error_milstein[m,k] = (Yt_milstein[nval[k]] - Xt_T)**2
        #error_wagnerplaten[m,k] = (Yt_wagnerplaten[nval[k]] - Xt_T)**2

    print str(float(m)/nsim*100) + '%'#progress
##############################################################################

#Monte-Carlo estimates of the errors
mc_error_euler=np.zeros(nsize)
mc_error_milstein=np.zeros(nsize)
#mc_error_wagnerplaten=np.zeros(nsize)
for k in range(0,nsize):
    mc_error_euler[k] = sqrt(np.mean(error_euler[:,k]))
    mc_error_milstein[k] = sqrt(np.mean(error_milstein[:,k]))
    #mc_error_wagnerplaten[k] = sqrt(np.mean(error_wagnerplaten[:,k]))

#Regression (Example: a[0] returns the slope and a[1] returns the intersect)
a = linregress(np.log2(nval), np.log2(mc_error_euler))
b = linregress(np.log2(nval), np.log2(mc_error_milstein))
#c = linregress(np.log2(nval), np.log2(mc_error_wagnerplaten))

#log-log-plot: error estimates and step amount n
plt.figure(1)
plt.scatter(np.log2(nval), np.log2(mc_error_euler), 3, c='b', label="Euler-scheme. Slope: " + str("{0:.3f}".format(a[0])))
plt.scatter(np.log2(nval), np.log2(mc_error_milstein), 3, c='m', label="Milstein-scheme. Slope: " + str("{0:.3f}".format(b[0])))
#plt.scatter(np.log2(nval), np.log2(mc_error_wagnerplaten), 2, c='forestgreen', label="Wagner-Platen-scheme. Slope: " + str("{0:.3f}".format(c[0])))
plt.xlabel('$log_2$ n', fontsize=12)
plt.ylabel('$log_2$ error', fontsize=12)
plt.legend(loc='lower left')
#plot regression line
plt.plot(np.log2(nval), np.log2(nval)*a[0]+a[1], c='b', alpha=0.5)
plt.plot(np.log2(nval), np.log2(nval)*b[0]+b[1], c='m', alpha=0.5)
#plt.plot(np.log2(nval), np.log2(nval)*c[0]+c[1], c='forestgreen')

#Second plot: Error analysis
plt.figure(2)
plt.plot(np.log2(nval), mc_error_euler, c='b', alpha=0.5, label="Euler-scheme")
plt.plot(np.log2(nval), mc_error_milstein, c='m', alpha=0.5, label="Milstein-scheme")
#plt.plot(np.log2(nval), mc_error_wagnerplaten, c='forestgreen', label="Wagner-Platen-scheme")
plt.xlabel('$log_2$ n', fontsize=12)
plt.ylabel('error', fontsize=12)
plt.legend(loc='upper right')

plt.show()

```

```python
################################################################################
###Code for the Monte-Carlo estimation of the convergence order for the
###Ornstein-Uhlenbeck process
###MC_ErrorAnalysis_ConvergenceOrderOU.py
###Python 2.7
################################################################################

import numpy as np
import matplotlib.pylab as plt
from scipy.stats import linregress
from NumericalSDE import *

##################################################
### Ornstein-Uhlenbeck process (OU)
### dXt = a(Xt)dt + b(Xt)dWt
### X0 = x0
### a(x) = -beta*x, b(x) = sigma
### beta, sigma positive constants
### True solution:
###
##################################################
#Parameter
sigma = 1.5
beta = 1.0
#functions a, b
def a(x):
    return -beta*x
def b(x):
    return sigma
#derivativs of a, b
#def a_dv(x):
#    return -beta
def b_dv(x):
    return 0
#def a_dvdv(x):
#    return 0
#def b_dvdv(x):
#    return 0
#starting value x0
x0 = 1
###############################################################################
#Number of steps. We will do the Monte-Carlo-analysis for the
#step amounts n in nval() starting with n we will then always
#double the amount of steps.
n = 8
nval = np.array([n, n*2, n*4, n*8, n*16, n*32, n*64, n*128])
nsize = np.size(nval)
#Number of MC simulations for each step amount n
nsim = 20000

################################################################################
#Arrays for the errors for each simulation and corresponding n.
#we are collecting the errors of the Euler, Milstein and Wagner-Platen-scheme.
error_euler = np.zeros((nsim,nsize))
error_milstein = np.zeros((nsim,nsize))
#error_wagnerplaten = np.zeros((nsim,nsize))
#empty arrays
Yt_euler = np.zeros(nsize)
Yt_milstein = np.zeros(nsize)
#Yt_wagnerplaten = np.zeros(nsize)
Xt_T = np.zeros(nsize)
stochIntApprox = 0

################################################################################
#Monte-Carlo-algorithm
for m in range(0,nsim):
    w = wiener(nval[0])
    t = timegrid(nval[0])
    for k in range(0,nsize):
        Yt_euler = sde_euler(x0,a,b,w)
        Yt_milstein = sde_milstein(x0,a,b,b_dv,w)
        #Yt_wagnerplaten = sde_wagnerplaten(x0,a,b,a_dv,b_dv,a_dvdv,b_dvdv,w)

        for j in range(0,nval[k]):
            stochIntApprox = stochIntApprox + (w[j+1]-w[j])*np.exp((-beta)*((t[nval[k]])-t[j]))


        Xt_T = x0*np.exp((-beta)*(t[nval[k]])) + sigma*stochIntApprox

        w = refineWiener(w)
        t = timegrid(nval[k]*2)
        stochIntApprox = 0

        #Squared-mean criterion for the error
        error_euler[m,k] = (Yt_euler[nval[k]] - Xt_T)**2
        error_milstein[m,k] = (Yt_milstein[nval[k]] - Xt_T)**2
        #error_wagnerplaten[m,k] = (Yt_wagnerplaten[nval[k]] - Xt_T)**2

    print str(float(m)/nsim*100) + '%'#progress
##############################################################################

#Monte-Carlo estimates of the errors
mc_error_euler=np.zeros(nsize)
mc_error_milstein=np.zeros(nsize)
#mc_error_wagnerplaten=np.zeros(nsize)
for k in range(0,nsize):
    mc_error_euler[k] = sqrt(np.mean(error_euler[:,k]))
    mc_error_milstein[k] = sqrt(np.mean(error_milstein[:,k]))
    #mc_error_wagnerplaten[k] = sqrt(np.mean(error_wagnerplaten[:,k]))

#Regression (Example: a[0] returns the slope and a[1] returns the intersect)
a = linregress(np.log2(nval), np.log2(mc_error_euler))
b = linregress(np.log2(nval), np.log2(mc_error_milstein))
#c = linregress(np.log2(nval), np.log2(mc_error_wagnerplaten))

#log-log-plot: error estimates and step amount n
plt.figure(1)
plt.scatter(np.log2(nval), np.log2(mc_error_euler), 3, c='b', label="Euler-scheme. Slope: " + str("{0:.3f}".format(a[0])))
plt.scatter(np.log2(nval), np.log2(mc_error_milstein), 3, c='m', label="Milstein-scheme. Slope: " + str("{0:.3f}".format(b[0])))
#plt.scatter(np.log2(nval), np.log2(mc_error_wagnerplaten), 2, c='forestgreen', label="Wagner-Platen-scheme. Slope: " + str("{0:.3f}".format(c[0])))
plt.xlabel('$log_2$ n', fontsize=12)
plt.ylabel('$log_2$ error', fontsize=12)
plt.legend(loc='lower left')
#plot regression line
plt.plot(np.log2(nval), np.log2(nval)*a[0]+a[1], c='b',alpha=0.5)
plt.plot(np.log2(nval), np.log2(nval)*b[0]+b[1], c='m',alpha=0.5)
#plt.plot(np.log2(nval), np.log2(nval)*c[0]+c[1], c='forestgreen')

#Second plot: Error analysis
plt.figure(2)
plt.plot(np.log2(nval), mc_error_euler, c='b',alpha=0.5, label="Euler-scheme")
plt.plot(np.log2(nval), mc_error_milstein, c='m',alpha=0.5, label="Milstein-scheme")
#plt.plot(np.log2(nval), mc_error_wagnerplaten, c='forestgreen', label="Wagner-Platen-scheme")
plt.xlabel('$log_2$ n', fontsize=12)
plt.ylabel('error', fontsize=12)
plt.legend(loc='upper right')

plt.show()
```

Appendix C

Additional graphics

We included some additional figures in the next pages to illustrate the approximation schemes. Especially we also added some graphics of the Wagner-Platen approximation and its error-analysis. It has strong order 1.5. This scheme has not beed discussed in the main text of the thesis. However we implemented this scheme as well.

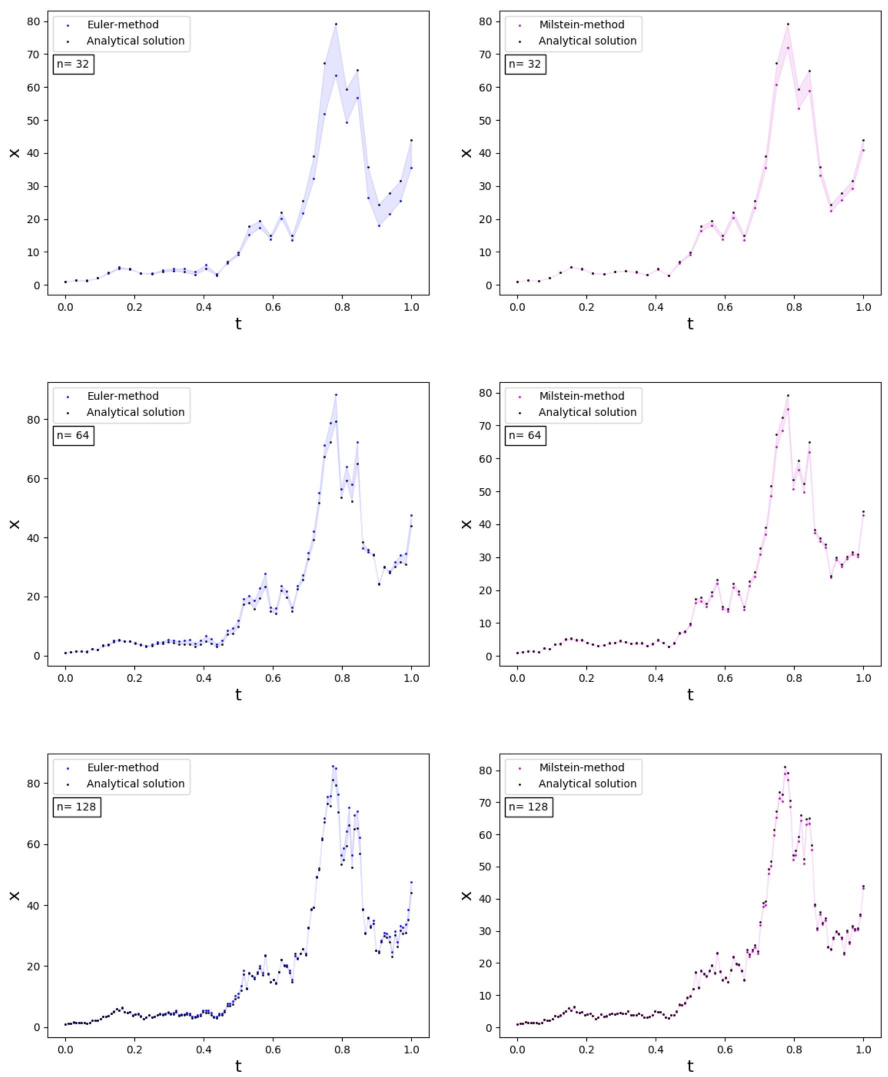

Figure C.1: Approximation (colored dots) of the sample path of the geometric Brownian motion and the true sample path (black dots) for n = 32, 64, 128 using Euler and Milstein.

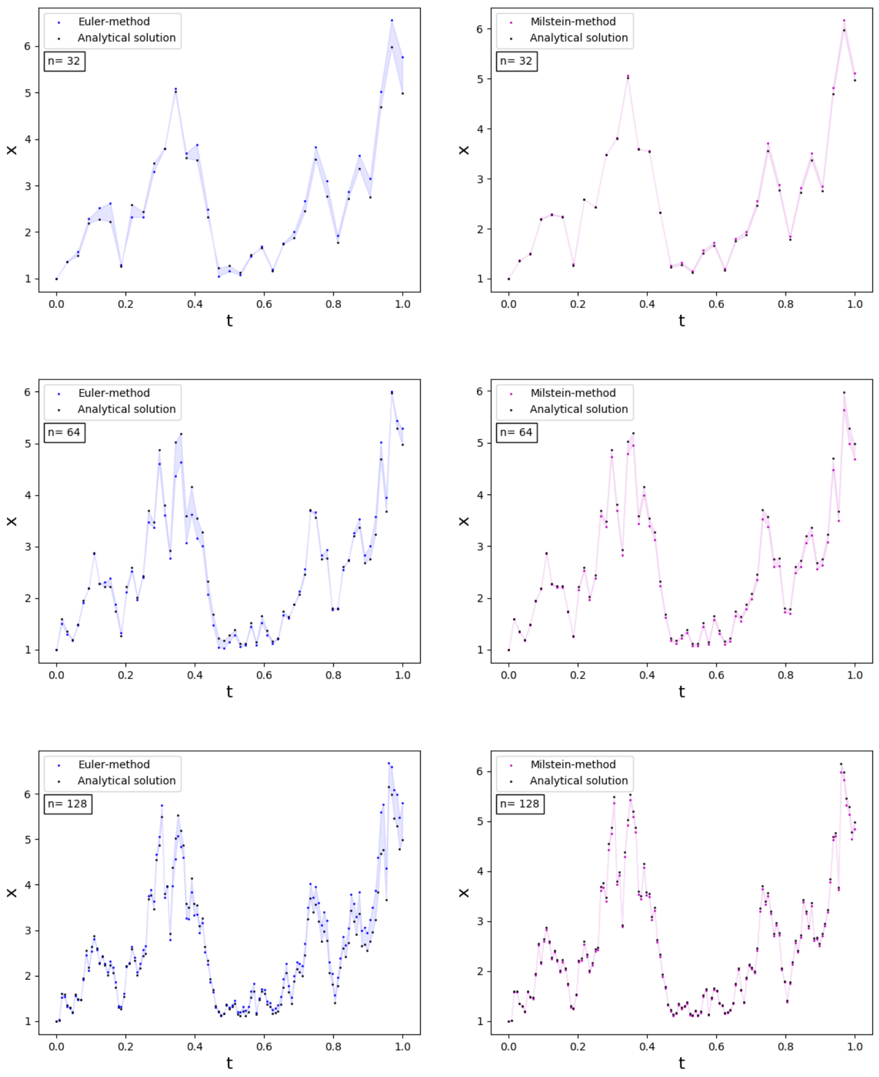

Figure C.2: Approximation (colored dots) of the sample path of the geometric Brownian motion and the true sample path (black dots) for n = 32, 64, 128 using Euler and Milstein.

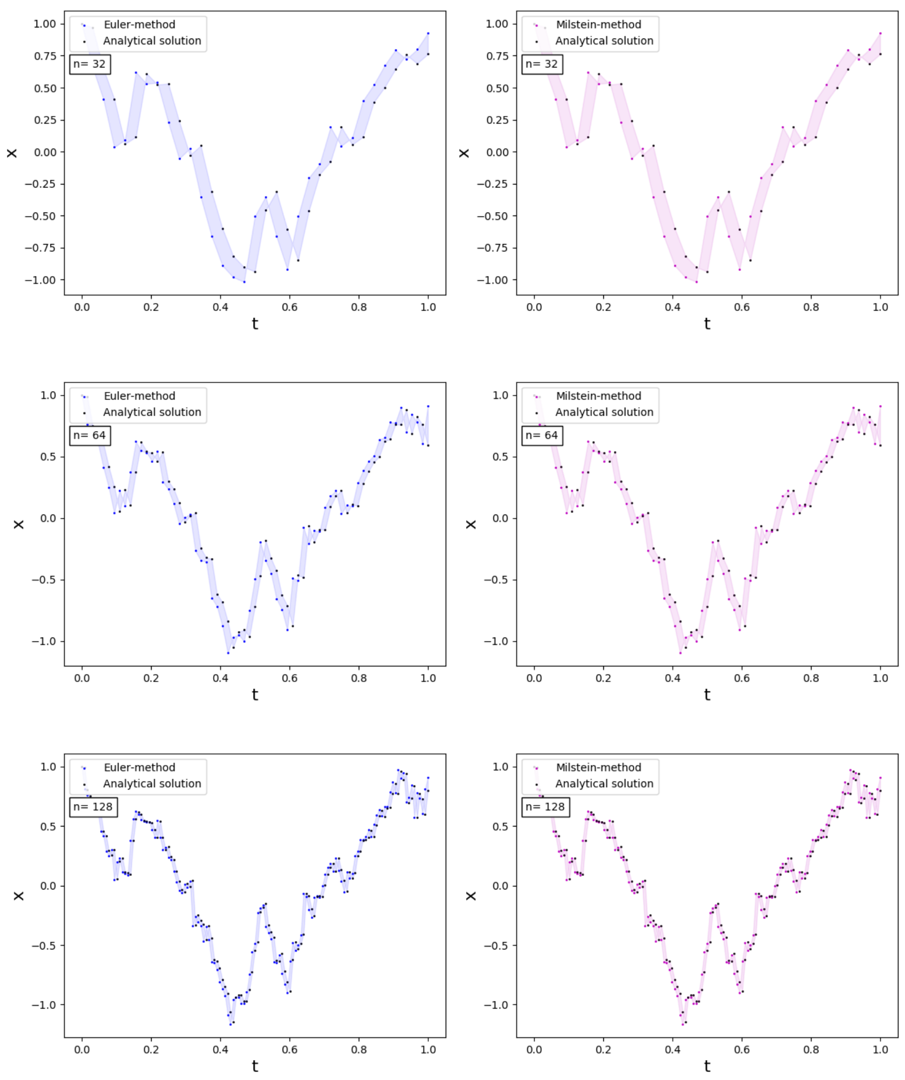

Figure C.3: Approximation (colored dots) of the sample path of the Ornstein-Uhlenbeck process and the true sample path (black dots) for n = 32, 64, 128 using Euler and Milstein.

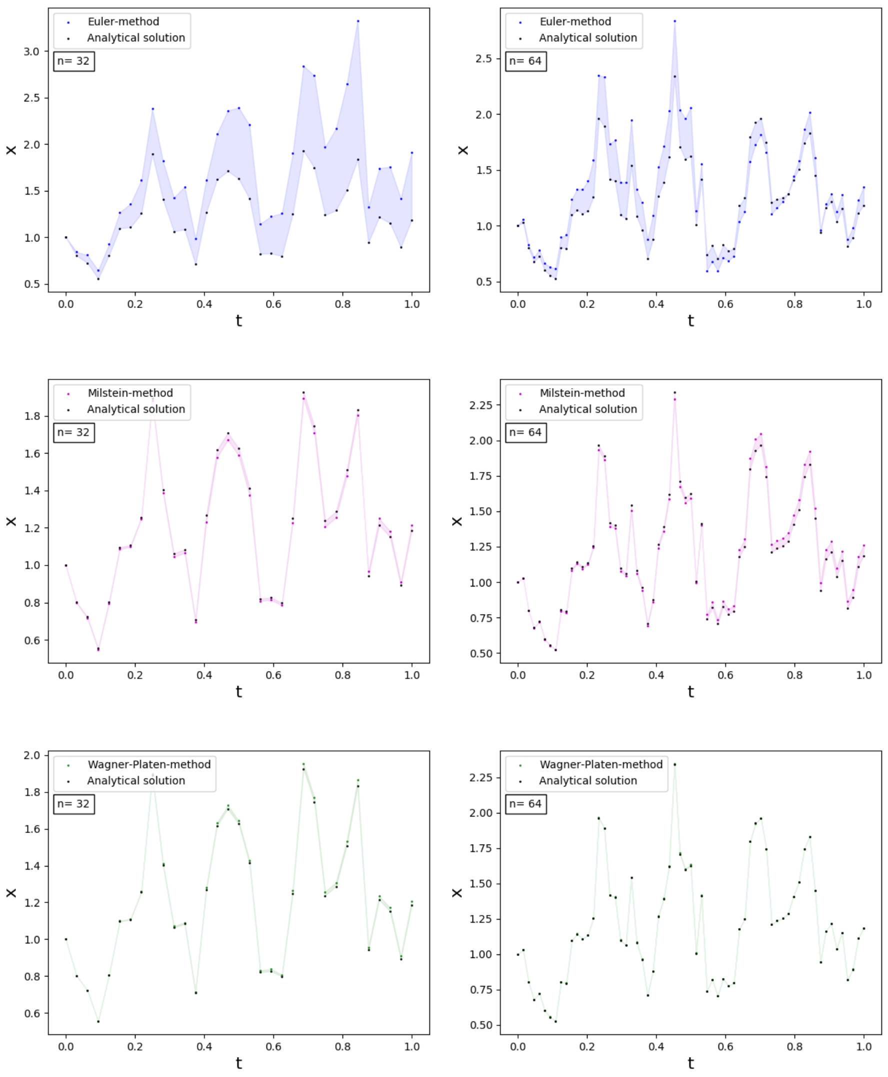

Figure C.4: Approximation (colored dots) of the sample path of the Ornstein-Uhlenbeck process and the true sample path (black dots) for n = 32, 64 using Euler, Milstein and Wagner-Platen.

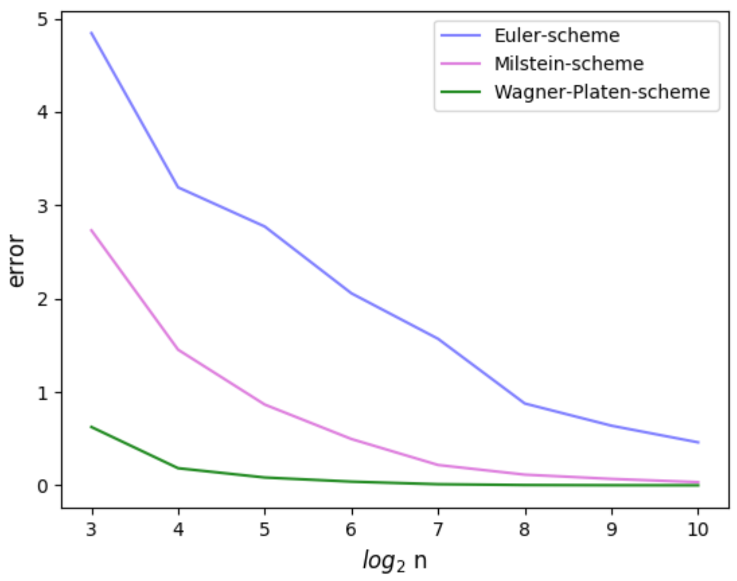

Figure C.5: Monte-Carlo estimates of the approximation error and the step amount n (logarithmized) for the geometric Brownian motion using Euler, Milstein, Wagner-Platen (20'000 simulations).

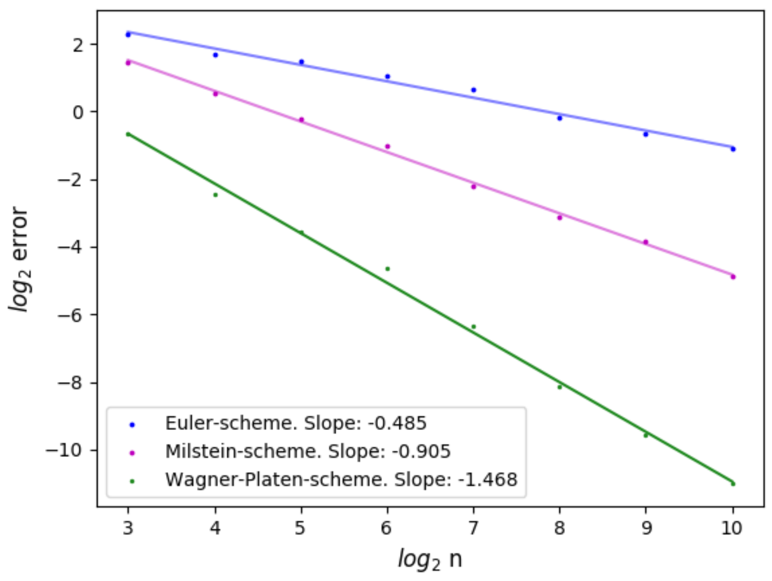

Figure C.6: Log-log plot of the approximation error and the step amount n for the geometric Brownian motion using Euler, Milstein, Wagner-Platen (20'000 simulations).

Bibliography

[1] A. Klenke. *Wahrscheinlichkeitstheorie*. Springer, 2013.

[2] B. Øksendal. *Stochastic Differential Equations - An Introduction with Applications*. Springer, 2003.

[3] D. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM*, 2001.

[4] D. Talay. How to discretize stochastic differential equations. In *Nonlinear Filtering and Stochastic Control*, Cortona, 1982. Centro Internazionale Matematico Estivo, Springer.

[5] E. Süli, D. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[6] I. Karatzas, S. E. Shreve. *Brownian Motion and Stochastic Calculus*. Springer, 1991.

[7] L. C. Evans. *An Introduction to Stochastic Differential Equations*. AMS, 2013.

[8] P. E. Kloeden, E. Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, 1992.

[9] R. L. Schilling, L. Partzsch. *Brownian Motion - an Introduction to Stochastic Processes*. De Gruyter, 2014.

[10] S. Asmussen, P. W. Glynn. *Stochastic Simulation - Algorithms and Analysis*. Springer, 2007.

[11] T. C. Gard. *Introduction to Stochastic Differential Equations*. Pure and Applied Mathematics, 1987.