CHALMERS

Efficient Computation Of The Strong And Weak Error For Linear SDE

Master's Thesis in Applied Mathematics

Yueleng Wang

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2014

Master's Thesis 2014:10

Abstract

In this thesis the numerical approximation of solutions to stochastic differential equations is considered. Numerical approximation is often necessary in order to obtain solutions to stochastic differential equations as explicit solutions are seldom available. In this thesis we are interested in the error introduced by the numerical method. To investigate how numerical approximation behaves we investigate both the Ornstein-Uhlenbeck equation and the geometric Brownian motion. For these equations we find explicit expressions for the error. For more complicated equations the error can be approximated by Monte-Carlo simulation, but this method is known to converge very slow. In this thesis, we apply this new method to compute the weak error and strong error efficiently which is the main theme of the thesis. But we stress that this is possible because we restrict the setting to linear SDE.

Acknowledgements

I would like to express my gratitude to my supervisors Stig Larsson and Adam Andersson for thier help and guidance. They have generously shared their deep mathematical knowledge during my half year master thesis period.

I am also very grateful for the opportunity and environment Mathematical Sciences of Chalmers University Technology provided for my two year mathematics study.

I thank Patrik Albin, Serik Sagitov for their excellent teachings for the courses in probability and statistics. Finally, I want to thank my parents for their patience and support.

Yueleng Wang
Gothenburg, October 2014

Contents

1

Introduction

Ordinary differential equations(ODE) are used to describe the evolutionary dynamical systems. Stochastic differential equations(SDE) arise when random noises are introduced into differential equations and are widely used in finance, engineering, biology, and physics to describe models. The solution to an ordinary differential equation is a fixed deterministic path, while the solution to a stochastic differential equation on the other hand is a random process. This means that for any specific time the solution is a random variable.

Most ordinary differential equations can not be sovled explicitly. We have the same problem for the stochastic differential equations. Therefore in most cases, we have to rely on numerical methods. We consider for instance the Ornstein-Uhlenbeck equation

$$\begin{aligned} X(0) &= X_0 \\ \mathrm{d}X(t) &= \mu X(t)\,\mathrm{d}t + \sigma\,\mathrm{d}W(t), \quad t \in [0,T], \end{aligned}$$

and its numerical approximation by the implicit Euler method,

$$\begin{aligned} \widetilde{X}_0 &= X_0 \\ \widetilde{X}_n &= \widetilde{X}_{n-1} + \mu k \widetilde{X}_n + \sigma \Delta W^n, \quad n \in \{1,...N\}. \end{aligned}$$

Here, X_0 is the initial value, k is the step size, μ and σ are drift and diffusion constants respectively, $\Delta W^n := W(t_n) - W(t_{n-1})$ is the increment of a standard real valued Wiener process, $\{W(t)\}_{t\in[0,T]}$.

The numerical method considered is an iteration scheme, where computed values of the solution at past time step is nested to produce a new value one time step ahead. We consider the backward Euler method which uses one past time step and the BDF2 scheme which uses two past time steps. One is often interested in computing an error between the numerical solution and the exact solution. The Monte-Carlo method is often used to approximate the error numerically. In the Monte-Carlo method we compute a huge amount of random variables $\widetilde{X}(t_N,w_j), \omega_j \in \Omega$, where Ω is the probability space, N is the number of time steps used for the discretization, i.e. $t_N = T$. We approximate the expected value $\mathbb{E}(g(X_T))$ by the average $\frac{1}{M}\sum_{j=1}^{M} g(\widetilde{X}(t_N,\omega_j))$. The total error is then the sum of the *discretization error* and the *statistical error* (*sampling error*):

$$\begin{aligned} &\Big|\frac{1}{M}\sum_{j=1}^{M} g(\widetilde{X}(t_N,w_j)) - \mathbb{E}\big(g(X(T))\big)\Big| \\ &\leq \Big|\mathbb{E}\big(g(\widetilde{X}_{t_N})\big) - \mathbb{E}\big(g(X(T))\big)\Big| + \Big|\frac{1}{M}\sum_{j=1}^{M} g(\widetilde{X}(t_N,w_j)) - \mathbb{E}\big(g(\widetilde{X}_{t_N})\big)\Big|. \end{aligned} \tag{1.1}$$

Normally, we are interested the order of convergence, i.e., the speed that the error converges to 0 as the step size k becomes smaller, showing the goodness of a certain method. Technically, we shall say that a time discrete approximation $\widetilde{X}_{t_N}$ converges in the weak sense with order $\beta > 0$ if for any smooth function g there exists a finite constant K and a positive constant $\delta_0 > 0$ such that

$$\big|\mathbb{E}(g(X_T)) - \mathbb{E}(g(\widetilde{X}_{t_N})\big| \leq K\delta^{\beta}, \quad \delta \in (0, \delta_0). \tag{1.2}$$

We say that $\widetilde{X}_{t_N}$ converges in the strong sense with order $\beta > 0$ if $\exists K, \delta_0 > 0$:

$$\sqrt{\mathbb{E}\big(\big|\widetilde{X}_{t_N} - X_T\big|^2\big)} \leq K\delta^{\beta}, \quad \delta \in (0, \delta_0). \tag{1.3}$$

In this thesis we avoid using Monte-Carlo method in the computation of the weak and strong error by restricting us to linear equations and by deriving explicit formulus for the error. This is good as the Monte-Carlo method converges very slow and makes it pratically impossible to obtain high accuracy. There is an experiment in section 4.1 using the Monte-Carlo method to compute error for the Ornstein-Uhlenbeck equation computed by the implicit Euler method. It does not show good result due to the memory restriction.

2

Linear SDE

2.1 Introduction to Brownian Motion and Itô Calculus

Defining Brownian Motion

In mathematics, Brownian motion is described by the Wiener process; a continuous time stochastic process named in honor of Norbert Wiener. We consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where $\mathcal{F}$ is a σ-algebra and $\mathbb{P}$ is a probability measure. A stochastic process is a mapping $Y : \Omega \times [0,T] \to \mathbb{R}$ such that $Y(t) : \Omega \to \mathbb{R}$ is measurable. A filtration is an increasing family $\{\mathcal{F}_t\}_{t\in[0,T]}$ of σ-algebras and the process Y is said to be adapted to $\{\mathcal{F}_t\}_{t\in[0,T]}$ if Y_t is $\mathcal{F}_t$-measurable for every $t \in [0,T]$. The filtration $\{\widetilde{\mathcal{F}}_t\}_{t\in[0,T]}$ given by $\widetilde{\mathcal{F}}_t := \sigma(Y_s : s \leq t)$ is called the filtration generated by Y. Clearly Y is adapted to $\{\widetilde{\mathcal{F}}_t\}_{t\in[0,T]}$. By $N(\mu,\sigma^2)$ we denote the normal distribution with mean μ and variance σ^2.

Definition 2.1.1. *A stochastic process $\{W(t)\}_{t\in[0,T]}$ adapted to a filtration $\{\mathcal{F}_t\}_{t\in[0,T]}$ is called a standard Brownian motion (or standard Wiener process) if the following three conditions hold:*

1. $W(0) = 0$;
2. $W(t) - W(s) \sim N(0,|t-s|)\ \forall\ s,t \in [0,T]$, $W(t)-W(s)$ *is independent of* $\mathcal{F}_{\min\{s,t\}}$.
3. *The trajectories* $t \mapsto W(t)$ *are* $\mathbb{P} - a.s.$ *continuous.*

An alternative characterisation of the Wiener process is the so-called Lévy characterisation that say that the Wiener process is a continuous martingale with $W(0) = 0$ and quadratic variation

$$[W(t),W(t)] = \lim_{\delta_i \to 0} \sum_{i=1}^{n} \left(W(t_i^n) - W(t_{i-1}^n)\right)^2 = t,$$

where the limit is taken over partitions: $0 = t_0^n < t_1^n < \dots < t_n^n = t$ with $\delta_n = \max_{1\leq i\leq n}(t_i^n - t_{i-1}^n)$.

Definition of the Itô Integral

A process $X = \{X(t), 0 \leq t \leq T\}$ is called a simple adapted process if there exist times $0 = t_0 < t_1 < t_2 ... < t_n = T$ and random variables $\xi_0, \xi_1, ... \xi_{n-1}$, such that ξ_0 is a constant, ξ_i is F_{t_i} measurable, and $\mathbb{E}(\xi_i^2) < \infty$, $i = 0,...,n-1$, such that

$$X(t) = \xi_0 I_0(t) + \sum_{i=0}^{n-1} \xi_i I_{(t_i,t_{i+1}]}(t). \tag{2.1}$$

For simple adapted processes, we define the Itô integral $\int_0^T X(t)\,\mathrm{d}W(t)$ as the random variable:

$$\int_0^T X(t)\,\mathrm{d}W(t) = \sum_{i=0}^{n-1} \xi_i \big(W(t_{i+1}) - W(t_i)\big).$$

Let $L^2_{\mathcal{F}}(\Omega \times [0,T]; \mathbb{R})$ denote the closure of the simple adapted processes in the Hilbert space $L^2(\Omega \times [0,T]; \mathbb{R})$. We call $Y \in L^2_{\mathcal{F}}(\Omega \times [0,T]; \mathbb{R})$ a predictable square integrable process.

Theorem 2.1.2. *For predictable square integrable processes $Y \in L^2_{\mathcal{F}}(\Omega \times [0,T]; \mathbb{R})$ the Itô integral $\int_0^T Y(t)\,\mathrm{d}W(t)$ is a well defined random variable and the following properties hold.*

1. *Zero mean:*

$$\mathbb{E}\Big(\int_0^T Y(t)\,\mathrm{d}W(t)\Big) = 0;$$

2. *Isometry:*

$$\mathbb{E}\Big(\int_0^T Y(t)\,\mathrm{d}W(t)\Big)^2 = \int_0^T \mathbb{E}\big(Y^2(t)\big)\,\mathrm{d}t.$$

Proof. We first prove the theorem for simple adapted process as defined in (2.1). First 1. hold since

$$\mathbb{E}\sum_{i=0}^{n-1} \xi_i \big(W(t_{i+1}) - W(t_i)\big) = \sum_{i=0}^{n-1} \mathbb{E}(\xi_i)\mathbb{E}\big(W(t_{i+1}) - W(t_i)\big) = 0.$$

To prove the isometry property for simple adapted process we see that

$$\begin{aligned}\mathbb{E}\Big(\sum_{i=0}^{n-1} \xi_i \big(W(t_{i+1}) - W(t_i)\big)\Big)^2 &= \sum_{i=0}^{n-1} \mathbb{E}\Big(\xi_i^2 \big(W(t_{i+1}) - W(t_i)\big)^2\Big) \\ &\quad + 2\sum_{i<j} \mathbb{E}\Big(\xi_i \xi_j \big(W(t_{i+1}) - W(t_i)\big)\big(W(t_{j+1}) - W(t_j)\big)\Big).\end{aligned}$$

We notice that

$$\mathbb{E}\Big(\xi_i \xi_j \big(W(t_{i+1}) - W(t_i)\big)\big(W(t_{j+1}) - W(t_j)\big)\Big) = 0.$$

Using the martingale property of Brownian motion yields

$$\begin{aligned}
\sum_{i=0}^{n-1}\mathbb{E}\Big(\xi_i^2\big(W(t_{i+1})-W(t_i)\big)^2\Big) &= \sum_{i=0}^{n-1}\mathbb{E}\Big(\mathbb{E}\Big(\xi_i^2\big(W(t_{i+1})-W(t_i)\big)^2\big|\mathcal{F}_{t_i}\Big)\Big)\\
&= \sum_{i=0}^{n-1}\mathbb{E}\Big(\xi_i^2\mathbb{E}\big(W(t_{i+1})-W(t_i)\big)^2\big|\mathcal{F}_{t_i}\Big) = \sum_{i=0}^{n-1}\mathbb{E}(\xi_i)(t_{i+1}-t_i)\\
&= \int_0^T \mathbb{E}\big(X^2(t)\big)\,\mathrm{d}t.
\end{aligned}$$

As $L^2_{\mathcal{F}}(\Omega\times[0,T];\mathbb{R})$ is the closure of all simple adapted processes in $L^2(\Omega\times[0,T];R)$ there exists for every $Y\in L^2_{\mathcal{F}}(\Omega\times[0,T];R)$ a Cauchy Sequence Y^n of simple adapted processes such that $Y^n\to Y$ in $L^2_{\mathcal{F}}(\Omega\times[0,T];\mathbb{R})$. By the isometry $\{\int_0^T Y^n\,\mathrm{d}W\}_{n\in N}$ is a Cauchy Sequence in $L^2(\Omega;\mathbb{R})$ and as this space is complete there exist a unique limit $\int_0^T Y\,\mathrm{d}W$. Clearly the isometry holds for the limit.

□

Itô's Lemma

We next present the Itô lemma. It is proved by a Taylor expansion.

Lemma 2.1.3. *Let X satisfy*

$$\begin{aligned}
\mathrm{d}X(t) &= Y(t)\,\mathrm{d}t + Z(t)\,\mathrm{d}W(t)\\
X(0) &= X_0,
\end{aligned}$$

where $Y(t)$, $Z(t)\in L^2_{\mathcal{F}}(\Omega\times[0,T];\mathbb{R})$. Assume that $u(x,t)$ is a smooth function of the independent variables x and t. Then

$$\mathrm{d}u\big(X(t),t\big) = \frac{\partial}{\partial t}u\big(X(t),t\big)\,\mathrm{d}t + \frac{\partial}{\partial x}u\big(X(t),t\big)\,\mathrm{d}X(t) + \frac{1}{2}\frac{\partial^2}{\partial x^2}u\big(X(t),t\big)Z^2(t)\,\mathrm{d}t.$$

2.2 Linear SDE

Linear SDE form a class of SDE that can be solved explicitly. Consider the following general linear SDE in one dimension:

$$\begin{aligned}
\mathrm{d}X(t) &= \big(\alpha(t)+\beta(t)X(t)\big)\,\mathrm{d}t + \big(\gamma(t)+\delta(t)X(t)\big)\,\mathrm{d}W(t), \quad t\in[0,T]\\
X(0) &= X_0,
\end{aligned}\tag{2.2}$$

where α, β,γ,δ are given functions of t in $[0,T]$. For simplicity we assume them bounded. In this chapter we investigate some linear SDE and the way to solve them. Equation (2.2) is a short hand notation of the following integral equation:

$$\begin{aligned} X(t) &= X_0 + \int_0^t \big(\alpha(t) + \beta(t)X(t)\big)\,\mathrm{d}s + \int_0^t \big(\gamma(t) + \delta(t)X(t)\big)\,\mathrm{d}W(s), \quad t \in [0,T], \\ X(0) &= X_0, \end{aligned} \tag{2.3}$$

The motivations of numerical methods come from (2.3).

To have the presentation as self contained as possible we show uniqueness of solutions. For existence we will derive the solution for two special cases that we treat in this thesis.

Lemma 2.2.1. *(Gronwall's Inequality) Let $y : [0,T] \mapsto \mathbb{R}$ be a continuous function such that for some $\lambda, \mu > 0$ it holds $y(t) \le \lambda + \mu \int_0^t y(s)\,\mathrm{d}s$, then*

$$y(t) \le \lambda e^{\mu t}, \quad t \in [0,T].$$

Let X^1,X^2 be solutions to (2.2), i.e. let

$$\mathrm{d}X_t^i = \big(\alpha(t) + \beta(t)X_t^i\big)\,\mathrm{d}t + \big(\gamma(t) + \delta(t)X_t^i\big)\,\mathrm{d}W(t), \quad X^i(0) = X_0, i = 1,2. \tag{2.4}$$

The difference $\Delta = X^1 - X^2$ satisfies the equation

$$\begin{aligned} \mathrm{d}\Delta(t) &= \beta(t)\Delta(t)\,\mathrm{d}t + \delta(t)\Delta(t)\,\mathrm{d}W(t). \\ \Delta(0) &= 0 \end{aligned} \tag{2.5}$$

The integral form is written as

$$\Delta(t) = \int_0^t \beta(s)\Delta(s)\,\mathrm{d}s + \int_0^t \delta(s)\Delta(s)\,\mathrm{d}W(s).$$

Take expectation for both sides, we get by the Cauchy Schwartz inequality and the Itô isometry:

$$\begin{aligned} \mathbb{E}|\Delta(t)|^2 &\le 2\mathbb{E}\Big|\int_0^t \beta(s)\Delta(s)\,\mathrm{d}s\Big|^2 + 2\mathbb{E}\Big|\int_0^t \delta(s)\Delta(s)\,\mathrm{d}W(s)\Big|^2 \\ &\le 2\int_0^t \beta^2(s)\,\mathrm{d}s \int_0^t \mathbb{E}\big[\Delta^2(s)\big]\,\mathrm{d}s + 2\int_0^t \delta^2(s)\mathbb{E}\big[\Delta^2(s)\big]\,\mathrm{d}s. \end{aligned}$$

We let $b := \int_0^T \beta^2(s)\,\mathrm{d}s$ and $d := \max_{t\in[0,T]} |\delta(t)|$ we get,

$$\begin{aligned} \mathbb{E}|\Delta(t)|^2 &\le 2b\int_0^t \mathbb{E}\big[\Delta^2(s)\big]\,\mathrm{d}s + 2d^2\int_0^t \mathbb{E}\big[\Delta^2(s)\big]\,\mathrm{d}s \\ &= (2b + 2d^2)\int_0^t \mathbb{E}\big[\Delta^2(s)\big]\,\mathrm{d}s. \end{aligned}$$

By using the Gronwall Lemma we have the inequality,

$$\mathbb{E}|\Delta(t)|^2 \le \Delta(0)e^{(2b+2d^2)t} \le \Delta(0)e^{(2b+2d^2)T} = 0$$

Thus we get $\Delta(t) \equiv 0$. This finishes the proof of uniqueness for linear SDEs.

2.3 Ornstein-Uhlenbeck Process

We here consider the scalar Onstein Uhlenbeck equation:

$$\mathrm{d}X(t) = \mu X(t)\,\mathrm{d}t + \sigma\,\mathrm{d}W(t), \quad t \in (0,T], \quad X(0) = X_0, \tag{2.6}$$

$\mu, \sigma \in \mathbb{R}$ and $\{W(t)\}_{t\in[0,T]}$ is a standard Wiener process. To solve the SDE consider the process $Y(t) = X(t)e^{-\mu t}$. Using the Itô Lemma we obtain:

$$\begin{aligned}\mathrm{d}Y(t) &= e^{-\mu t}\,\mathrm{d}X(t) - \mu e^{-\mu t}X(t)\,\mathrm{d}t = e^{-\mu t}\big(-\mu X(t)\,\mathrm{d}t + \mathrm{d}X(t)\big)\\ &= \sigma\,\mathrm{d}W(t).\end{aligned}$$

This gives

$$Y(t) = Y(0) + \int_0^t \sigma e^{-\mu s}\,\mathrm{d}W(t), \quad t \in [0,T].$$

Now the solution for $X(t)$ is

$$\begin{aligned}X(t) &= e^{\mu t}\big(X(0) + \int_0^t \sigma e^{-\mu s}\,\mathrm{d}W(t)\big)\\ &= e^{\mu t}X_0 + \sigma\int_0^t e^{\mu(t-s)}\,\mathrm{d}W(t), \quad t \in [0,T].\end{aligned}$$

It is unique by the argument in section 2.2.

2.4 Geometric Brownian Motion

The scalar geometric Brownian motion is given by

$$\mathrm{d}X(t) = \mu X(t)\,\mathrm{d}t + \sigma X(t)\,\mathrm{d}W(t), \quad t \in [0,T], \quad X(0) = X_0,$$

$\mu, \sigma \in \mathbb{R}$ and $\{W(t)\}_{t\in[0,T]}$ is a standard Wiener process. We apply Ito's Lemma to $u = \ln X(t)$ to get that

$$\begin{aligned}\mathrm{d}u(t) &= \frac{1}{X(t)}\,\mathrm{d}X(t) + \frac{1}{2}\frac{\sigma^2X^2(t)}{X^2(t)}\,\mathrm{d}t\\ &= \Big(\mu - \frac{\sigma^2}{2}\Big)\,\mathrm{d}t + \sigma\,\mathrm{d}W(t)\end{aligned}$$

which by integratiion is solved by:

$$u(t) = u(0) + \Big(\mu - \frac{\sigma^2}{2}\Big)t + \sigma W(t), \quad t \in [0,T],$$

so that

$$X(t) = X_0\exp\Big\{\Big(\mu - \frac{\sigma^2}{2}\Big)t + \sigma W(t)\Big\}, \quad t \in [0,T].$$

3

Numerical Schemes

In this section we present some of the standard numerical schemes for SDE. Except for the implicit and explicit Euler scheme we present a multi-step scheme.

Consider the one-dimensional stochastic ordinary differential equation

$$\begin{aligned} \mathrm{d}X(t) &= \mu\big(X(t),t\big)\,\mathrm{d}t + \sigma\big(X(t),t\big)\,\mathrm{d}W(t), \quad t \in [0,T], \\ X(0) &= X_0, \end{aligned} \tag{3.1}$$

where $\{W(t)\}_{t\in[0,T]}$ is standard Wiener process, $\sigma(x,t), \mu(x,t)$ are called diffusion coefficient and drift coefficient respectively.

3.1 Euler-Maruyama Method

Let $0 = t_0 < t_1 < ... < t_n < t_{n+1} < ... < t_N = T$ be a uniform mesh, let $k = t_{n+1} - t_n \leq 1 \;\forall n$. It holds that

$$X(t_n) = X(t_{n-1}) + \int_{t_{n-1}}^{t_n} \mu\big(X(s),s\big)\,\mathrm{d}s + \int_{t_{n-1}}^{t_n} \sigma\big(X(s),s\big)\,\mathrm{d}W(s), \quad n \in \{1,...,N\}.$$

We define an approximation of X by the explicit Euler-Maruyama method as follows

$$\begin{aligned} \widetilde{X}_0 &= X_0 \\ \widetilde{X}_n &= \widetilde{X}_{n-1} + \mu(\widetilde{X}_{n-1}, t_{n-1}) \int_{t_{n-1}}^{t_n} \mathrm{d}s + \sigma(\widetilde{X}_{n-1}, t_{n-1}) \int_{t_{n-1}}^{t_n} \mathrm{d}W(s), \quad n \in \{1,...,N\}, \end{aligned}$$

here $\int_{t_{n-1}}^{t_n} \mathrm{d}s = k, \int_{t_{n-1}}^{t_n} \mathrm{d}W(s) = W(t_n) - W(t_{n-1}) = \sqrt{k}\xi_{n-1}$, where $\xi_{n-1} \sim N(0,1)$, we let $\Delta W^{n-1} := W(t_n) - W(t_{n-1})$, so

$$\begin{aligned}&\widetilde{X}_0 = X_0\\&\widetilde{X}_n = \widetilde{X}_{n-1} + \mu(\widetilde{X}_{n-1},t_{n-1})k + \sigma(\widetilde{X}_{n-1}, t_{n-1})\Delta W^n, \quad n \in \{1,...,N\}.\end{aligned}$$

Often an implicit method is prefered as it has better stability properties. For the implicit Euler-Maruyama method, the drift part is implicit and the diffusion part is explicit so the method is given as follows:

$$\begin{aligned}&\widetilde{X}_0 = X_0\\&\widetilde{X}_n = \widetilde{X}_{n-1} + \mu(\widetilde{X}_n,t_n)k + \sigma(\widetilde{X}_{n-1}, t_{n-1})\Delta W^n, \quad n \in \{1,...,N\}.\end{aligned} \tag{3.2}$$

3.2 Multi-step Schemes

We consider a stochastic linear m-method, which for $l = m,...,N$ takes the form

$$\sum_{j=0}^{m} \alpha_j X_{l-j} = k \sum_{j=0}^{m} \beta_j f(X_{l-j},t_{l-j}) + \sum_{j=1}^{m} \Gamma_j(X_{l-j}, t_{l-j})\Delta W^{l-j+1}.$$

We set $\alpha_0 = 1$. We requre given initial values X_0,...X_{m-1}. As in the deterministic case, usually only X_0 is given by initial value problem and the values X_1,...X_{m-1} need to be computed numerically. This can be done by suitable one-step method. Every diffusion term $\Gamma_j(x,t)\Delta W^{l-j+1}$ is an appropriate function Γ_j of x and t multiplied by a one-dimensional standard Wiener integral over $[t_{l-j}, t_{l-j+1}]$.

Backward Differentiation Formula 2 steps, BDF2. The scheme is given as follows:

$$\begin{aligned}&\widetilde{X}_n - \frac{4}{3}\widetilde{X}_{n-1} + \frac{1}{3}\widetilde{X}_{n-2}\\=&\frac{2}{3}k\mu(\widetilde{X}_n,t_n) + \sigma(\widetilde{X}_{n-1},t_{n-1})\Delta W^n - \frac{1}{3}\sigma(\widetilde{X}_{n-2},t_{n-2})\Delta W^{n-1},\end{aligned}$$

here $\alpha_0 = 1$, $\alpha_1 = -\frac{3}{4}$, $\alpha_2 = \frac{1}{3}$, $\beta_0 = \frac{2}{3}$, $\beta_1 = \beta_2 = 0$ and

$$\Gamma_1(x,t)\Delta W^n = \sigma(x,t)\Delta W^n, \quad \Gamma_2(x,t)\Delta W^{n-1} = -\frac{1}{3}\sigma(x,t)\Delta W^{n-1},$$

where ΔW^n as usual equals to $W(t_n) - W(t_{n-1})$ and X_1 is usually given by one-step scheme such as explicit Euler scheme or implicit Euler scheme.

4

Weak And Strong Error For The Ornstein-Uhlenbeck Process

In this section we apply the implicit-Euler scheme and the BDF2 scheme to the Ornstein-Uhlenbeck process, and write down the formula of X_{N_k} generated by the two schemes and then compute the weak error and strong error efficiently. By doing this, we then investigate the convergence order of the two schemes.

We here consider the scalar Onstein-Unlenbeck process

$$\mathrm{d}X(t) = \mu X(t)\,\mathrm{d}t + \sigma\,\mathrm{d}W(t), \quad t \in [0,T], \quad X(0) = X_0, \tag{4.1}$$

$\mu, \sigma \in \mathbb{R}$ and $\{W(t)\}_{t\in\mathbb{R}}$ is a standard Wiener process. We know from Section 2.3 that the solution is given by:

$$X(t) = e^{\mu t}X_0 + \sigma \int_0^t e^{\mu(t-s)}\,\mathrm{d}W(s), \quad t \in [0,T].$$

We observe that the exact solution $X(t)$ is normally distributed:

$$X(t) \sim N(\hat{\mu},\hat{\sigma}^2),$$

for some $\hat{\mu}, \hat{\sigma}$, which we next compute.

$$\hat{\mu} := \mathbb{E}\big(X(T)\big) = e^{\mu T}X_0. \tag{4.2}$$

To compute the variance it suffices to consider the stochastic integral. By the Itô isometry, it holds that

$$\begin{aligned}\hat{\sigma}^2 :&= \mathbb{V}\mathrm{ar}\big(X(T)\big) = \mathbb{E}\Big|\sigma \int_0^T e^{\mu(T-s)}\,\mathrm{d}W(s)\Big|^2 \\ &= \sigma^2 \int_0^T e^{2\mu(T-s)}\,\mathrm{d}s = \frac{\sigma^2}{2\mu}(e^{2\mu T} - 1).\end{aligned} \tag{4.3}$$

The fact that we know the distribution of $X(T)$ will be used.

4.1 Implicit Euler Scheme

The implicit Euler method for the Ornstein-Uhlenbeck process reads, find

$$\begin{aligned}\widetilde{X}_0 &= X_0\\ \widetilde{X}_n &= \widetilde{X}_{n-1} + \mu k \widetilde{X}_n + \sigma \Delta W^n,\end{aligned}$$

here, $n \in \{1,...,N_k\}$ and $N_k = \frac{T}{k}$. Equivalently We can write it as:

$$\begin{aligned}\widetilde{X}_0 &= X_0\\ \widetilde{X}_n &= (1-\mu k)^{-1}(\widetilde{X}_{n-1} + \sigma \Delta W^n).\end{aligned} \tag{4.4}$$

We are insterested in the difference between the exact solution and the numerical solution. Two kinds of errors are considered: the strong error and the weak error.

$$\begin{aligned}\text{Strong error} :=& \quad \sqrt{\mathbb{E}\big|X(T) - \widetilde{X}_{N_k}\big|^2} = \big|\big|X(T) - \widetilde{X}_{N_k}\big|\big|_{L^2(\Omega,\mathbb{R})}\\ \text{Weak error} :=& \quad \Big|\mathbb{E}\big[\varphi\big(X(T)\big)\big] - \mathbb{E}\big[\varphi\big(\widetilde{X}_{N_k}\big)\big]\Big|,\end{aligned} \tag{4.5}$$

The continuous function φ is called the test function. To start with we let φ be $\varphi(x) = x^2$ and we try to compute the weak error by definiton (4.5). From (4.3) for the exact value, we computed $\mathbb{E}\big[\varphi\big(X(T)\big)\big] = \mathbb{V}\text{ar}\big(X(T)\big)$. To compute $\mathbb{E}\big[\varphi\big(\widetilde{X}_{N_k}\big)\big]$ we start with noting that

$$\begin{aligned}\widetilde{X}_n &= (1-\mu k)^{-1}(\widetilde{X}_{n-1} + \sigma \Delta W^n)\\ &= (1-\mu k)^{-1}\big((1-\mu k)^{-1}\widetilde{X}_{n-1} + (1-\mu k)^{-1}\sigma\Delta W^{n-1} + \sigma \Delta W^n\big)\\ &= \dots\\ &= (1-\mu k)^{-n} X_0 + \sigma \sum_{j=1}^{n} (1-\mu k)^{-(n-j+1)} \Delta W^j.\end{aligned} \tag{4.6}$$

It is clear that $\widetilde{X}_{N_k}$ is a normally distributed random variable with parameters $\tilde{\mu}$ and $\tilde{\sigma}^2$, i.e., $\widetilde{X}_{N_k} \sim N(\tilde{\mu},\tilde{\sigma}^2)$. We compute its mean and variance.

$$\tilde{\mu} := \mathbb{E}[\widetilde{X}_{N_k}] = (1-\mu k)^{-N_k} X_0.$$

For the variance we get that

$$\mathbb{E}[\widetilde{X}_{N_k}^2] = (1-\mu k)^{-2N_k} X_0^2 + \sigma^2 \sum_{j=1}^{N_k} (1-\mu k)^{2(N_k-j+1)} \mathbb{E}[(\Delta W^j)^2],$$

and as $\Delta W^j \sim N(0,k)$, it holds that $\mathbb{E}[(\Delta W^j)^2] = k$ and for $q := (1-\mu k)^{-2}$ it holds

$$\begin{aligned}
\mathbb{E}[\widetilde{X}_{N_k}^2] &= (1-\mu k)^{-2N_k} X_0^2 + \sigma^2 k \cdot \sum_{j=1}^{N_k} (1-\mu k)^{2(N_k-j+1)}, \\
\tilde{\sigma}^2 : &= \mathbb{V}\mathrm{ar}(\widetilde{X}_{N_k}) = \mathbb{E}\big[\widetilde{X}_{N_k}^2\big] - \mathbb{E}^2\big[\widetilde{X}_{N_k}\big] \\
&= \sigma^2 k \sum_{j=1}^{N_k} (1-\mu k)^{-2(N_k-j+1)} = \sigma^2 k \frac{q^{N_k+1}-q}{1-q}.
\end{aligned} \tag{4.7}$$

We can now compute explicitly for every step size $k > 0$. Next we compute the strong error. By

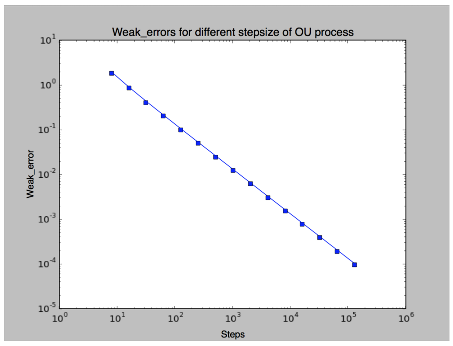
$$\begin{aligned}
&||X(T) - \widetilde{X}_{N_k}||^2_{L^2(\Omega,\mathbf{R})} \\
&\quad = \mathbb{E}\Bigg[\bigg|\big(e^{\mu T} - (1-\mu k)^{-N_k}\big) X_0 \\
&\quad + \sigma \sum_{j=1}^{N_k} \int_{t_{j-1}}^{t_j} \big(e^{\mu(T-s)} - (1-\mu k)^{-(N_k-j+1)}\big)\,\mathrm{d}W(s)\bigg|^2\Bigg] \\
&\quad = \big(e^{\mu T} - (1-\mu k)^{-N_k}\big)^2 X_0^2 \\
&\quad + \sigma^2 \sum_{j=1}^{N_k} \int_{t_{j-1}}^{t_j} \big(e^{\mu(T-s)} - (1-\mu k)^{-(N_k-j+1)}\big)^2\,\mathrm{d}s \\
&\quad = \big(e^{\mu T} - (1-\mu k)^{-N_k}\big)^2 X_0^2 + \frac{\sigma^2}{2\mu}(e^{2\mu T}-1) \\
&\quad - \frac{2\sigma^2}{\mu}(e^{\mu k}-1) \sum_{j=1}^{N_k} e^{\mu(T-t_j)} (1-\mu k)^{-(N_k-j+1)} + \frac{\sigma^2\big((1-\mu k)^{-2N_k}-1\big)}{2\mu - \mu^2 k}.
\end{aligned} \tag{4.8}$$

Proposition 4.1.1. *The implicit Euler scheme for the Ornstein-Uhlenbeck equation* (4.1) *has with $\varphi(x) = x^2$ the following weak error:*

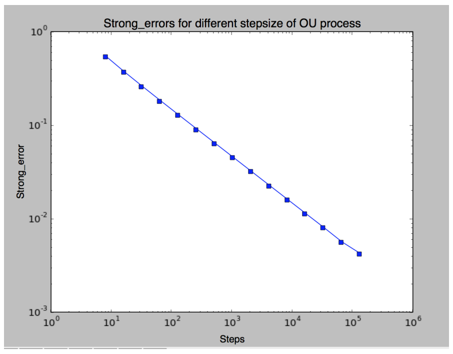
$$\begin{aligned}
\text{Weak error} : &= \big|\mathbb{E}[X^2(T)] - \mathbb{E}[\widetilde{X}_{N_k}^2]\big| \\
&= \bigg| e^{2\mu T} X_0^2 + \frac{\sigma^2}{2\mu}(e^{2\mu T}-1) - \sigma^2 k \sum_{j=1}^{N_k} (1-\mu k)^{-2(N_k-j+1)} \bigg|,
\end{aligned}$$

and the strong error is given by (4.8)*.*

Now we computed the strong error explicitly. We next show the error plots for the strong error and weak error with $\varphi(x) = x^2$. The python code is available in appendix A.1.



(a)



(b)

Figure 4.1

N	Weak Err	N	Weak Err	N	Strong Err	N	Strong Err
8	1.87004927	2048	0.00619554	8	0.54176103	2048	0.03193753
16	0.85819789	4096	0.00309684	16	0.37164784	4096	0.02258086
32	0.41206586	8192	0.00154819	32	0.25903532	8192	0.01596646
64	0.20201207	16384	0.00077404	64	0.18188200	16384	0.01128816
128	0.10002864	32768	0.00038700	128	0.12816384	32768	0.00799037
256	0.04977334	65536	0.00019350	256	0.09046911	65536	0.00560115
512	0.02482684	131072	0.00009675	512	0.06391625	131072	0.00421570
1024	0.01239851			1024	0.04517619		

Table 4.1

We see from Table 4.1 that the weak error of the impilcit Euler method gives order 1 and strong order $\frac{1}{2}$.

We next try the Monte-Carlo method for the numerical solutions and investigate the implementation as introduced in the Introduction section. For the practice of Monte-Carlo method, we simulate sample paths $\widetilde{X}(t_{N_k},\omega_j)$, $\omega_j \in \Omega$ where Ω is the probability space. We approximate the expected value $\mathbb{E}\big(\varphi(X_T)\big)$ by the average $\frac{1}{M}\sum_{j=1}^{M}\varphi\big(\widetilde{X}(t_N,\omega_j)\big)$. The codes are listed in Appendix A.2, and Table 4.2 shows the results.

Simulation 1				Simulation 2			
N	$\varphi(x) = x^2$	N	$\varphi(x) = x^2$	N	$\varphi(x) = x^2$	N	$\varphi(x) = x^2$
8	1.15724579	512	0.01215363	8	1.20616020	512	0.00613718
16	0.58119065	1024	0.03901456	16	0.57805450	1024	0.01197015
32	0.28571545			32	0.30094291		
64	0.16702714			64	0.15939946		
128	0.07241882			128	0.08599735		
256	0.03421214			256	0.02603705		

Table 4.2

We see from the two Monte-Carlo simulations, the convergence is obvious. But when the N becomes larger the behavior is bad. For example we see the results for $N = 512$ and $N = 1024$, the weak error becomes inconsistent. This is due to the Monte-Carlo simulation which requires terribly large M(number of paths simulated) to make the statistical error $\left|\frac{1}{M}\sum_{j=1}^{M} g\big(\widetilde{X}_{N_k}(\omega_j)\big) - \mathbb{E}\big(g(\widetilde{X}_{N_k})\big)\right|$ in (1.1) almost equal to 0. $M = 400000$ is actually very big, but not big enough. This method is useful given that if we have a large computer, but works not very good in the personal computer.

We look back in the previous calculation (Figure.4.1 and Table.4.1). We get precise results if $\varphi(x) = x^2$. But if $\varphi(x)$ is a more complicated, such as $\varphi(x) = x^3, \sin(x), e^x$ and so on. We can approximate $\mathbb{E}\big(\varphi(X(T))\big)$ and $\mathbb{E}\big(\varphi(X_{N_k})\big)$ by quadrature and the fact that if f is the probability density function of X it holds that

$$\mathbb{E}\big(\varphi(X)\big) = \int_{-\infty}^{\infty} \varphi(x) f(x)\,\mathrm{d}x. \tag{4.9}$$

Fortunately we know everything about the two random variables $\widetilde{X}_{N_k}$,$X(T)$ by the calculations above. These two random variables are both normally distributed and we have already computed the mean and the variance.

Proposition 4.1.2. *The implicit Euler scheme for the Ornstein-Uhlenbeck equation* (4.1) *has with general $\varphi(x)$ the following weak error:*

$$\begin{aligned}\text{Weak error} :&= \big|\mathbb{E}[\varphi(X(T))] - \mathbb{E}[\varphi(\widetilde{X}_{N_k})]\big| \\ &= \Big|\int_{-\infty}^{\infty} \varphi(x) f_{\{e^{\mu T}X_0, \sigma^2(e^{2\mu T}-1)/2\mu\}}\,\mathrm{d}x - \int_{-\infty}^{\infty} \varphi(x) f_{\{m,s^2\}}\,\mathrm{d}x\Big|,\end{aligned}$$

where f_{m,s^2} is the probability density function of $\widetilde{X}_{N_k}$ with mean $m = (1-\mu k)^{-N_k}X_0$ and variance $s^2 = \sigma^2 k \sum_{j=1}^{N_k}(1-\mu k)^{-2(N_k-j+1)}$

Using truncation and quadrature we approximate the expected values as follows:

$$\begin{aligned}\mathbb{E}\big(\varphi(X)\big) &\approx \int_{-L}^{L} \varphi(x) f(x)\,\mathrm{d}x \\ &\approx \sum_{i=-n}^{n} \varphi(x_i) f_{\{\mu,\sigma^2\}}(x_i)\Delta x,\end{aligned}$$

where $n \cdot \Delta x = L$.

The results are listed in the Table 4.3. We see from the results that the quadrature does no effect the results in the first 8 decimal digits, i.e. it's not an important factor.

TestFun:$\varphi(x) = x^3$		TestFun:$\varphi(x) = x^4$		TestFun:$\varphi(x) = \sin(x)$	
N	Weak Error	N	Weak Error	N	Weak Error
8	13.29561919	8	94.96479537	8	0.05188570
16	5.96250833	16	41.48893976	16	0.02786032
32	2.83279070	32	19.47854602	32	0.01434445
64	1.38172574	64	9.44711924	64	0.00726774
128	0.68247932	128	4.65331482	128	0.00365675
256	0.33917798	256	2.30943122	256	0.00183398
512	0.16907777	512	1.15045009	512	0.00091837
1024	0.08441160	1024	0.57416374	1024	0.00045953

Table 4.3: Weak errors by three different test functions

Which shows the same convergence speed as Table 4.1, i.e., weak order 1. This is a good method because we can compute weak error of any test functions. But the condition is that we should know the density function of X_{N_K} and compute by (4.9). We have listed codes to calculate the results for weak errors of different test functions in Appendix A.3.

4.2 BDF2 Scheme

The algorithm is given in previous chapter and the algorithm for Ornstein-Uhlenbeck process is given as follows:

$$\begin{gathered}\widetilde{X}_n - \frac{4}{3}\widetilde{X}_{n-1} + \frac{1}{3}\widetilde{X}_{n-2} = \frac{2}{3}k\mu\widetilde{X}_n + \sigma\Delta W^n - \frac{1}{3}\sigma\Delta W^{n-1} \\ \widetilde{X}_0 = X_0, \widetilde{X}_1 = (X_0 + \sigma\Delta W^1)(1-k\mu)^{-1}.\end{gathered} \tag{4.10}$$

We want to write the explicit representation formula for BDF2 scheme similar to (4.6) for the impicit Euler scheme. To do so some more effort is needed and basic theory for difference equations used, see Appendix A.8.

We now focus on the difference equations (4.10) given by BDF2. Firstly we change it to the homogeneous difference equation:

$$(1-\frac{2}{3}k\mu)\widetilde{X}_n - \frac{4}{3}\widetilde{X}_{n-1} + \frac{1}{3}\widetilde{X}_{n-2} = 0.$$

The corresponding characteristic polynomial $\rho(z)$ has the roots:

$$\zeta_1 = \frac{1}{3}\Big(2+\sqrt{1+2k\mu}\Big)\big(1-\frac{2}{3}k\mu\big)^{-1}$$
$$\zeta_2 = \frac{1}{3}\Big(2-\sqrt{1+2k\mu}\Big)\big(1-\frac{2}{3}k\mu\big)^{-1},$$

then we determine the coefficients: $\{b_1^1,b_2^1\}$ and $\{b_1^2,b_2^2\}$ corresponds to the fundamental solution $Z^{:,0}$,$Z^{:,1}$. As mentioned in Appendix A.8 this can be solved by the system (A.6). We first get that:

$$b_1^1 = -\frac{\zeta_2}{\zeta_1-\zeta_2}, \quad b_2^1 = \frac{\zeta_1}{\zeta_1-\zeta_2},$$
$$b_1^2 = \frac{1}{\zeta_1-\zeta_2}, \quad b_2^2 = -b_1^2 = -\frac{1}{\zeta_1-\zeta_2}.$$

So the fundamental solution $Z^{:,0}$,$Z^{:,1}$ to the equation (A.2) are given by:

$$Z^{:,0} = b_1^1 Y_1 + b_2^1 Y_2$$
$$Z^{:,1} = b_1^2 Y_1 + b_2^2 Y_2.$$

where Y_1, Y_2 are two sequences of solutions to the homogeneous equation (A.5). Thus the solution to the BDF2 difference equation (4.10) is by (A.4) given as:

$$\widetilde{X}_n = Z^{n,0}\widetilde{X}_0 + Z^{n,1}\widetilde{X}_1 + \frac{\sigma}{a_2}\sum_{\eta=2}^{n} Z^{n-\eta+1,1}(\Delta W^\eta - \frac{1}{3}\Delta W^{\eta-1}).$$

This gives:

$$\begin{aligned}\widetilde{X}_{N_k} &= \big(Z^{N_k,0} + Z^{N_k,1}(1-k\mu)^{-1}\big)X_0 + \sigma\big(Z^{N_k,1}(1-k\mu)^{-1} - \frac{1}{3a_2}Z^{N_k-1,1}\big)\Delta W^1 \\ &\quad + \frac{\sigma}{a_2}\sum_{\eta=2}^{N_k-1}(Z^{N_k-\eta+1,1} - \frac{1}{3}Z^{N_k-\eta,1})\Delta W^\eta + \frac{\sigma}{a_2}Z^{1,1}\Delta W^{N_k}.\end{aligned} \tag{4.11}$$

It's a normally distributed random variable $\widetilde{X}_{N_k} \sim N(\hat{\mu},\hat{\sigma}^2)$, for some $\hat{\mu},\hat{\sigma}$. The second moment of $\widetilde{X}_{N_k}$ is now calculated:

$$\begin{aligned}\mathbb{E}(\widetilde{X}_{N_k}^2) &= \big(Z^{N_k,0} + Z^{N_k,1}(1-k\mu)^{-1}\big)^2 X_0^2 + \sigma^2 k\big(Z^{N_k,1}(1-k\mu)^{-1} - \frac{1}{3a_2}Z^{N_k-1,1}\big)^2 \\ &\quad + \frac{\sigma^2}{a_2^2}k\sum_{\eta=2}^{N_k-1}(Z^{N_k-\eta+1,1} - \frac{1}{3}Z^{N_k-\eta,1})^2 + \frac{\sigma^2}{a_2^2}k(Z^{1,1})^2.\end{aligned}$$

Proposition 4.2.1. *The weak error with test function $\varphi(x) = x^2$ when applying BDF2 scheme to Ornstein-Uhlenbeck equation* (4.1) *is given by*

$$\begin{aligned}
&\text{Weak error}\\
&= \Big|\big(Z^{N_k,0} + Z^{N_k,1}(1-k\mu)^{-1}\big)^2 X_0^2 + \sigma^2 k\big(Z^{N_k,1}(1-k\mu)^{-1} - \frac{1}{3a_2}Z^{N_k-1,1}\big)^2\\
&+ \frac{\sigma^2}{a_2^2}k\sum_{\eta=2}^{N_k-1}(Z^{N_k-\eta+1,1} - \frac{1}{3}Z^{N_k-\eta,1})^2 + \frac{\sigma^2}{a_2^2}k(Z^{1,1})^2 - e^{2\mu T}X_0^2 - \frac{\sigma^2}{2\mu}(e^{2\mu T}-1)\Big|.
\end{aligned}$$

We can see results in the following Table 4.4.

N	Weak Err BDF2	N	Weak Err BDF2
8	0.4291553104	2048	0.0000068541
16	0.1087904715	4096	0.0000017138
32	0.0275711081	8192	0.0000004285
64	0.0069522604	16384	0.0000001071
128	0.0017463502	32768	0.0000000268
256	0.0004376781	65536	0.0000000067
512	0.0001095593	131072	0.0000000017
1024	0.0000274075		

Table 4.4

The results are better than Table.4.1, and gives weak convegence order of 2. The python code is available in appendix A.4.

Next we compute the weak error using different test funtions $\varphi(x) = x^3, \varphi(x) = x^4, \varphi(x) = \sin(x)$ with *Quadrature* and *Truncation*. The codes are available in appendix A.5 and the results are given in Table 4.5:

TestFun:$\varphi(x) = x^3$		TestFun:$\varphi(x) = x^4$		TestFun:$\varphi(x) = \sin(x)$	
N	Weak Error	N	Weak Error	N	Weak Error
8	2.97807288	8	20.65558451	8	0.01493503
16	0.74911201	16	5.14944126	16	0.00399011
32	0.18946030	32	1.29930808	32	0.00102476
64	0.04774725	64	0.32724302	64	0.00025927
128	0.01199183	128	0.08217360	128	0.00006518
256	0.00300530	256	0.02059268	256	0.00001634
512	0.00075227	512	0.00515458	512	0.00000409
1024	0.00018819	1024	0.00128946	1024	0.00000102

Table 4.5

The results give weak order 2. As we know the exact representation of $\widetilde{X}_{N_k}$, the strong error is calculated by the definition (4.5). By Itô isometry it holds that

$$\begin{aligned}
&||X(T) - \widetilde{X}_{N_k}||^2_{L^2(\Omega,\mathbb{R})} \\
&= \mathbb{E}\Bigg[\Bigg|\big(e^{\mu T} - Z^{N_k,0} - Z^{N_k,1}(1-k\mu)^{-1}\big)X_0 \\
&+ \sigma \int_0^{t_1} \big(e^{\mu(T-s)} - Z^{N_k,1}(1-k\mu)^{-1} + \frac{1}{a_2} Z^{N_k-1,1}\big)\,\mathrm{d}W(s) \\
&+ \sigma \sum_{\eta=2}^{N_k-1} \int_{t_{\eta-1}}^{t_\eta} \big(e^{\mu(T-s)} - \frac{1}{a_2}(Z^{N_k-\eta+1,1} - \frac{1}{3} Z^{N_k-\eta+2,1})\big)\,\mathrm{d}W(s) \\
&+ \sigma \int_{t_{N_k-1}}^{T} (e^{\mu(T-s)} - \frac{1}{a_2})\,\mathrm{d}W(s)\Bigg|^2\Bigg] \\
&= \big(e^{\mu T} - Z^{N_k,0} - Z^{N_k,1}(1-k\mu)^{-1}\big)^2 X_0^2 \\
&+ \sigma^2 \int_0^{t_1} \big(e^{\mu(T-s)} - Z^{N_k,1}(1-k\mu)^{-1} + \frac{1}{a_2} Z^{N_k-1,1}\big)^2\,\mathrm{d}s \\
&+ \sigma^2 \sum_{\eta=2}^{N_k-1} \int_{t_{\eta-1}}^{t_\eta} \big(e^{\mu(T-s)} - \frac{1}{a_2}(Z^{N_k-\eta+1,1} - \frac{1}{3} Z^{N_k-\eta+2,1})\big)^2\,\mathrm{d}s \\
&+ \sigma^2 \int_{t_{N_k-1}}^{T} (e^{\mu(T-s)} - \frac{1}{a_2})^2\,\mathrm{d}s,
\end{aligned} \tag{4.12}$$

Proposition 4.2.2. *The strong error when applying BDF2 scheme to Ornstein-Uhlenbeck equation* (4.1) *is given by* (4.12).

All the integrals are of the same form:

$$\begin{aligned}
&\int_{t_{\eta-1}}^{t_\eta} \left(e^{\mu(T-s)} - \xi\right)^2 \mathrm{d}s \\
&= \int_{t_{\eta-1}}^{t_\eta} e^{2\mu(T-s)} \,\mathrm{d}s - \int_{t_{\eta-1}}^{t_\eta} 2\xi e^{\mu(T-s)} \,\mathrm{d}s + \xi^2 k \\
&= \frac{1}{2\mu} e^{2\mu(T-t_\eta)}(e^{2k\mu} - 1) - \frac{2\xi}{\mu} e^{\mu(T-t_\eta)}(e^{k\mu} - 1) + k\xi^2.
\end{aligned}$$

We compute the strong error in python and the codes are listed in appendix A.6 and the result in Table 4.6.

N	Strong Err BDF2	N	Strong Err BDF2
8	0.2161990356	2048	0.0036008372
16	0.1171815398	4096	0.0025011615
32	0.0640923077	8192	0.0017524335
64	0.0355780168	16384	0.0012333973
128	0.0204773753	32768	0.0008701148
256	0.0124058066	65536	0.0006145010
512	0.0079181794	131072	0.0004343732
1024	0.0052673063		

Table 4.6

Surprisingly, BDF2 method gives strong convergence order $\frac{1}{2}$ in spite of its good behavior for weak convergence with order 2.

5

Weak Error For The Geometric Brownian Motion

In this Section we apply implicit-Euler scheme and BDF2 scheme to geometric Brownian motion. We manage to get analogous results as in Section 4 for the implicit Euler scheme but show that for the BDF2 scheme severe complications appears that made us not able to treat this case.

We recall that the scalar geometric Brownian motion is given by

$$\mathrm{d}X(t) = \mu X(t)\,\mathrm{d}t + \sigma X(t)\,\mathrm{d}W(t), \quad ,t \in [0,T], \quad X(0) = X_0,$$

$t \in [0,t]$, $\mu,\sigma \in \mathbb{R}$ and $\{W(t)\}_{t\in[0,T]}$ is a standard Wiener process. Recall that the exact solution is given by:

$$X(t) = X_0 e^{[(\mu - \frac{\sigma^2}{2})t + \sigma W(t)]} \quad t \in [0,T].$$

5.1 The Implicit Euler Scheme

The implicit Euler method (3.2) for the geometric Brownian motion reads:

$$\begin{aligned} \widetilde{X}_0 &= X_0 \\ \widetilde{X}_n &= \widetilde{X}_{n-1} + \mu k \widetilde{X}_n + \sigma \widetilde{X}_{n-1} \Delta W^{n-1}, \quad n \in \{1,...,N_k\}, \end{aligned}$$

we get that the approximation satisfies:

$$\widetilde{X}_n = (1-\mu k)^{-1}(1+\sigma\Delta W^n)\widetilde{X}_{n-1}. \tag{5.1}$$

By iteration of (5.1) we get:

$$\widetilde{X}_n = (1-\mu k)^{-n}\prod_{i=1}^{n}(1+\sigma\Delta W^i)\widetilde{X}_0.$$

The expectation of the random variable $\widetilde{X}_n$ is given by:

$$\begin{aligned}
\mathbb{E}(\widetilde{X}_n) &= (1-\mu k)^{-n}\widetilde{X}_0\mathbb{E}\Big[\prod_{i=1}^{n}(1+\sigma\Delta W^i)\Big]\\
&= \big\{(1+\sigma\Delta W^i) \text{ is independent for each } i\big\}\\
&= (1-\mu k)^{-n}\widetilde{X}_0\prod_{i=1}^{n}\mathbb{E}\Big[1+\sigma\Delta W^i\Big]\\
&= (1-\mu k)^{-n}\widetilde{X}_0.
\end{aligned}$$

The second moment of the $\widetilde{X}_n$ is given by:

$$\begin{aligned}
\mathbb{E}(\widetilde{X}_n^2) &= (1-\mu k)^{-2n}\widetilde{X}_0^2\mathbb{E}\Big[\prod_{i=1}^{n}(1+\sigma\Delta W^i)^2\Big]\\
&= \big\{(1+\sigma\Delta W^i)^2 \text{ is independent for each } i\big\}\\
&= (1-\mu k)^{-2n}\widetilde{X}_0^2\prod_{i=1}^{n}\mathbb{E}\Big[1+\sigma\Delta W^i\Big]^2\\
&= (1-\mu k)^{-2n}(1+\sigma^2 k)^n\widetilde{X}_0^2.
\end{aligned}$$

Actually we can compute the expectation of any moment function $\varphi(x) = x^n$. The expectation of nth moment of $\widetilde{X}_{N_k}$ is given by:

$$\begin{aligned}
\mathbb{E}(\widetilde{X}_{N_k}^n) &= \mathbb{E}\left[(1-\mu k)^{-N_k}\widetilde{X}_0\prod_{i=1}^{N_k}(1+\sigma\Delta W^i)\right]^n\\
&= (1-\mu k)^{-nN_k}\widetilde{X}_0^n\mathbb{E}\left[\prod_{i=1}^{N_k}(1+\sigma\Delta W^i)^n\right]\\
&= \big\{(1+\sigma\Delta W^i)^n \text{ is independent for each } i, \text{ actually i.i.d}\big\}\\
&= (1-\mu k)^{-nN_k}\widetilde{X}_0^n\prod_{i=1}^{N_k}\mathbb{E}\Big[(1+\sigma\Delta W^i)^n\Big].
\end{aligned}$$

To compute the expectation of the nth moment of a normally distributed random variable numerically, we use for simplicity *Quadrature* and *Truncation* to approximate it.

$$\mathbb{E}\Big[(1+\sigma\Delta W^i)^n\Big] = \int_{-\infty}^{\infty} x^n f_{\{1,\sigma^2 k\}}(x)\,\mathrm{d}x,$$

here

$$f_{\{1,\sigma^2 k\}}(x) = \frac{1}{\sqrt{2\pi k}\sigma} e^{-\frac{(x-1)^2}{2\sigma^2 k}},$$

$$\begin{aligned}\mathbb{E}[\widetilde{X}^n_{N_k}] &= (1-\mu k)^{-nN_k} \widetilde{X}^n_0 \prod_{i=1}^{N_k} \mathbb{E}\Big[(1+\sigma\Delta W^i)^n\Big] \\ &= (1-\mu k)^{-nN_k} \widetilde{X}^n_0 \Big(\int_{-\infty}^{\infty} x^n f_{\{1,\sigma^2 k\}}(x)\,\mathrm{d}x\Big)^{N_k},\end{aligned} \tag{5.2}$$

We calculated the expectation of nth moment of the exact solution by (4.9) and calculate numerically by *Quadrature* and *Truncation*. Then we have the weak error by definition.

Proposition 5.1.1. *The weak error with test function $\varphi(x) = x^n$ when applying implicit Euler scheme to geometric Brownian motion is given by*

$$\begin{aligned}\text{Weak error} :=& \Big|\mathbb{E}[X^n(T)] - \mathbb{E}[\widetilde{X}^n_{N_k}]\Big| \\ =& \Big|(1-\mu k)^{-nN_k} \widetilde{X}^n_0 \Big(\int_{-\infty}^{\infty} x^n f_{\{1,\sigma^2 k\}}(x)\,\mathrm{d}x\Big)^{N_k} \\ &- X^n_0 e^{n(\mu - \frac{\sigma^2}{2})t} \int_{-\infty}^{\infty} e^{n\sigma x} f_{\{0,t\}}(x)\,\mathrm{d}x\Big|.\end{aligned}$$

Note that we have only treated weak error for polynomial test functions of implicit Euler scheme for the geometric Brownian motion. The codes are listed in appendix A.7, and the results are given in the Table 5.1.

TestFun:$\varphi(x) = x$		TestFun:$\varphi(x) = x^2$		TestFun:$\varphi(x) = x^3$	
N	Weak Error	N	Weak Error	N	Weak Error
8	0.19200354	8	1.64604449	8	88.48843841
16	0.09012214	16	0.72015554	16	57.07765020
32	0.04372726	32	0.33640219	32	32.70247059
64	0.02154535	64	0.16249561	64	17.56098572
128	0.01069490	128	0.07984578	128	9.10841073
256	0.00532823	256	0.03957528	256	4.63970097
512	0.00265933	512	0.01970110	512	2.34168679
1024	0.00132848	1024	0.00982895	1024	1.17636176
2048	0.00066394	2048	0.00490908	2048	0.58956796

Table 5.1

The table shows that the implicit Euler scheme gives order 1 for weak convergence for linear SDE for the given test functions.

5.2 BDF2 Scheme

The BDF2 method is given as follows for the geometric Brownian motion (3.1):

$$\widetilde{X}_n - \frac{4}{3}\widetilde{X}_{n-1} + \frac{1}{3}\widetilde{X}_{n-2} = \frac{2}{3}k\mu\widetilde{X}_n + \sigma\widetilde{X}_{n-1}\Delta W^n - \frac{1}{3}\sigma\widetilde{X}_{n-2}\Delta W^{n-1},$$

i.e.

$$(1-\frac{2}{3}k\mu)\widetilde{X}_n - (\frac{4}{3}+\sigma\Delta W^n)\widetilde{X}_{n-1} + (\frac{1}{3}+\frac{1}{3}\sigma\Delta W^{n-1})\widetilde{X}_{n-2} = 0, n \in \{2,...,N_k\}.$$

Roots for the corresponding polynomial

$$\zeta_{1,2} = \frac{(\frac{4}{3}+\sigma\Delta W^n) \pm \sqrt{(\frac{4}{3}+\sigma\Delta W^n)^2 - 4(1-\frac{2}{3}k\mu)(\frac{1}{3}+\frac{1}{3}\sigma\Delta W^{n-1})}}{2(1-\frac{2}{3}k\mu)}.$$

Problem appears for this difference equation. In the previous chapter, we applied BDF2 method to Ornstein-Uhlenbeck process and get a difference equation with constant coefficients. Here, we get the coefficients $1-\frac{2}{3}k\mu$, $\frac{4}{3}+\sigma\Delta W^n$, $\frac{1}{3}+\frac{1}{3}\sigma\Delta W^{n-1}$ varying with n. As the general theory of difference equation for constant coefficients does not hold here, we may have to rely on the Monte-Carlo simulation.

6

Summary

In this thesis, we investigate two different numerical methods for linear SDE (Ornstein-Uhlenbeck equation and geometric Brownian motion), we avoid using Monte-Carlo simulation to get the weak error and strong error between numerical result and the exact solution. Instead, we write down the exact formula of the final time of both the exact solution and the approximate solution and by computing their probability distributions we can compute the errors in an efficient way. We found that implicit Euler method gives convergence order 1 for the weak error and convergence order $\frac{1}{2}$ for the strong, while the BDF2 method gives convergence order 2 for weak error but $\frac{1}{2}$ for strong. In dealing with geometric Brownian motion, our method has its limitations. By applying the implicit Euler method to geometric Brownian motion, we successfully calculated the formula for X_{N_k}:

$$X_{N_k} = (1-\mu k)^{-N_k} \prod_{i=1}^{N_k} (1+\sigma \Delta W^i) \widetilde{X}_0.$$

But it is hard to calculate the distribution of X_{N_k}. Although we can calculate the expectations of polynomical functions of X_{N_k}, we cannot calculate expectations of other continuous functions like $\mathbb{E}\big(\sin(X_{N_k})\big)$. Furthermore, we even cannot calculate the explicit expression of X_{N_k} by applying BDF2 method to geometric Brownian motion. Finally, with the limitations mentioned above, I still think this method has its value and needs further research.

Bibliography

[1] A. Andersson, R. Kruse, and S. Larsson, *Duality in refined Watanabe-Sobolev spaces and weak approximation of SPDE*, arXiv:1312.5893[math.PR] (preprint 2013).

[2] E. Buckwar and R. Winkler, *Multistep methods for SDEs and their application to problems with small noise*, SIAM J. Numer. Anal. **44** (2006), no. 2, 779–803 (electronic). MR2218969 (2007k:60214)

[3] S. Hsu, *Ordinary differential equations with applications*, World Scientific Publishing (2013).

[4] F. Klebaner, *Introduction to stochastic calculus with applications, Second Edition*, Imperial College Press (2004).

[5] P. Kloeden and E. Platen, *Numerical solution of stochastic differential equations*, Springer (1995).

[6] S. Larsson, *Numerical methods for stochastic ODEs, Lecture Notes*, Chalmers University of Technology (2013).

[7] G. Meyer, *Numerical methods in finance, Lecture Notes*, Georgia Institute of Technology (2012).

A

Appendices

A.1

```
#This file deals with weak errors of OU process
#by Implicit Euler scheme.
#With text functions phi(x)=x^2

import numpy as np
import matplotlib.pyplot as plt
# parameters
mu = 1.
sigma = 1.

t = 0.  # initial time
T = 1.  # final time
x0 = 1. # initial value
N_list = 8.*2**np.arange(15) # step sizes

def sol2mom(T):
#This function computes the second moment of the exact solution
    return np.exp(2.*mu*T)*x0**2 + \
    sigma**2*(np.exp(2*mu*T) - 1)/(2*mu)

def num2mom(T,N):
    dt = T/N
    return (1 - mu*dt)**(-2*N)*x0**2 + sigma**2*dt\
            *np.sum( (1-mu*dt)**(-2*(np.arange(N) + 1)))

#computing exact value
exact = sol2mom(T)
error = np.zeros((2, np.size(N_list)))
```

```
#computing weak errors
for j in range(len(N_list)):
    error[0,j] = np.abs(exact - num2mom(T,N_list[j]))
#computing strong errors
def Euler_strong(T,N):
    dt=T/N
    err = (np.exp(mu*T) - (1. - mu*dt)**(-N))**2*x0**2 +\
            sigma**2/(2*mu)*(np.exp(2*mu*T)-1.) - \
            2.*sigma**2/mu*(np.exp(mu*dt)-1.)\
            * np.sum( np.exp(mu*dt*np.arange(N-1,-1,-1))\
            *(1-mu*dt)**(-(np.arange(N-1,-1,-1)+1)))\
            + sigma**2*((1-mu*dt)**(-2*N)-1)/(2*mu-mu**2*dt)
    return np.sqrt(err)

for j in range(len(N_list)):
    error[1,j] = np.sqrt(Euler_strong(T,N_list[j]))
# output
print '\n\nEuler - Maruyama'
print '       N    Weak Err   Strong Err'
for j in range(len(N_list)):
    print '%6d  %11.8f %11.8f'  \
            % (N_list[j], error[0,j], error[1,j])

plt.loglog(N_list, error[0,:],'-s', color="blue", linewidth=1)
plt.xlabel("Steps")
plt.ylabel("Weak_error")
plt.title("Weak_errors for different stepsize of OU process")
plt.show()

plt.loglog(N_list, error[1,:],'-s', color="blue", linewidth=1)
plt.xlabel("Steps")
plt.ylabel("Strong_error")
plt.title("Strong_errors for different stepsize of OU process")
plt.show()
```

A.2

```
"""

This program deals with the SODE:

dX(t) = mu X(t) dt + sigma dW(t), X(0) = x0.

The program uses Monte-Carlo simulations to
approximate expected values.
Test function: phi(x) = X ** 2
```

```
"""
"""

This program deals with the SODE:

dX(t) = mu X(t) dt + sigma dW(t), X(0) = x0.

The program uses Monte-Carlo simulations to
approximate expected values.

"""
import numpy as np

# parameter list
mu = 1.
sigma= 1.

t = 0.  # initial time
T = 1.  # final time

x0 = 1. # initial value

N_list = 8*2**np.arange(8) # np.arange(6)=[0,1,2,3,4,5]
M = 400000

# test function for weak error
def phi(x):
    return x**2



# err1 Euler, err2 BDF2, err3 MOD. BDF2.
# w1 phil1, W2 phil2, W3 phil3

w_err = np.zeros(len(N_list))


# exact solution
#X_ex = np.exp(mu*T)*x0 + sigma*\
#         np.sqrt((1-np.exp(2*mu*T))\
#     /(-2*mu))*np.random.normal(size=(1,M))

mean = x0**2 * np.exp(2 * mu * T) + (sigma **2)/(2 * mu) * \
              (np.exp(2 * mu * T) - 1) #By easy calculation.


print '\nParameter values:'
print 'mu = %4.5f, sigma = %4.5f, T = %2.2f, x0 = %4.5f' \
% (mu, sigma, T, x0)
print 'M = %d, max_N = %d' % (M, N_list[-1])

print  '\nExact values:'
print  'mean = %7.8f,' % mean,


print "\nStarting Monte-Carlo computation of weak errors..."
print "Computing errors for",
# calculating the exact and numerical solution
j = 0
for N in N_list:

    print 'N = %d,' % N,
    dt = T/N # temporal step size

    # setting initial value for EM
    X_num = np.ones((1,M))*x0

    # numerical schemes
    for k in range(N-1):
        # range(p)=[0,1,...,p-1]
        # simulating random numbers for Wiener increment
        dW_new = np.sqrt(dt)*np.random.normal(size=(1,M))
        # implicit Euler-Maruyama
        X_num = (X_num + sigma*dW_new )/(1 - mu*dt)


    #the weak errors
    w_err[j] = np.abs( mean - np.mean(phi(X_num)))
    j = j+ 1

#output
print '\n\nEuler - Maruyama'
print '   N  Weak Err'
for j in range(len(N_list)):
    print '%4d  %0.8f' \
            % (N_list[j], w_err[j])
```

A.3

```
#Python
#This file is considered as using Truncation and Quadrature to
#compute expectation of phil(x).
# Solving Equation Onrstein-Uhlenbeck process.
import numpy as np
import math
```

```
# parameters
mu = 1.
sigma = 1.

t = 0.  # initial time
T = 1.  # final time

x0 = 1. # initial value

N_list = 8*2**np.arange(8)
#N_list = [  8  16  32  64 128 256 512 1024]

# Test Functions
def phi1(x):
    return   x**3
def phi2(x):
    return   x**4
def phi3(x):
    return   np.sin(x)

w1_err1 = np.zeros(len(N_list))
w2_err1 = np.zeros(len(N_list))
w3_err1 = np.zeros(len(N_list))
w1_err2 = np.zeros(len(N_list))
w2_err2 = np.zeros(len(N_list))
w3_err2 = np.zeros(len(N_list))

# w1 means test1;  w2 means test2;
# err1 means Quadrature left point;
# err2 means Quadrature midpoint.
# exact solution
# X_ex = np.exp(mu*T)*x0 + sigma*\
#        np.sqrt((1-np.exp(2*mu*T))\
#        /(-2*mu))*np.random.normal(size=(1,M))
# In terms of Variance and the Expectation,
# because we already know that the solution
# is normal distributed.
# integration to compute the expectation

def normal_density(x):
    g_x = 1/(sigma*np.sqrt((np.exp(2*mu*T)-1)/(2*mu))*\
        np.sqrt(2*math.pi))*np.exp(-(x-np.exp(mu*T)*x0)\
        **2/(sigma**2*(np.exp(2*mu*T)-1)/mu))
    return g_x

L = 20.
cuts = 500
# How many small intervals between [mu-L,mu+L] integrated.
dx = 2*L/cuts
```

```
i = 1
integration1 = 0.
integration2 = 0.
integration3 = 0.
x = np.exp(mu*T)*x0 - L
while i <= cuts:
    integration1 = integration1 + phi1(x)*normal_density(x)*dx
    integration2 = integration2 + phi2(x)*normal_density(x)*dx
    integration3 = integration3 + phi3(x)*normal_density(x)*dx
    x = x + dx
    i = i + 1

mean1 = integration1
mean2 = integration2
mean3 = integration3

#mean1 = np.mean(phi1(X_ex))
#mean2 = np.mean(phi2(X_ex))
#mean3 = np.mean(phi3(X_ex))

print '\nParameter values:'  # \n represents enter.
print 'mu = %4.5f, sigma = %4.5f, T = %2.2f, x0 = %4.5f'\
     % (mu, sigma, T, x0)
print 'max_N = %d' % (N_list[-1])


print  '\nExact values:'
print  'Mean1 = %7.8f,' % mean1,
print  'Mean2 = %7.8f,' % mean2,
print  'Mean3 = %7.8f' % mean3

print "\n"
print "Computing errors for",
# calculating the numerical solution

j = 0
for N in N_list:
    print 'N = %d,' % N,
    mu_new=0.
    sigma_new=0.
    k = T/N
    mu_new =(1-mu*k)**(-N)*x0
    I = np.arange(N)
    for i in I:
        sigma_new =sigma_new + (1-mu*k)**(-2*N+2*(i+1)-2)
    sigma_new = sigma_new * sigma** 2 * k
    sigma_new = np.sqrt(sigma_new)
    def normal_density_numerical(x):
        f_x= 1/np.sqrt(2*math.pi*sigma_new**2) * \
        np.exp(-(x-mu_new)**2/(2*sigma_new**2))
        return f_x
    integration1_numerical = 0.
    integration2_numerical = 0.
    integration3_numerical = 0.
    x = mu_new - L
    #expectation with respect to different test functions.
    i = 1
    while i <= cuts:
        integration1_numerical = integration1_numerical + \
        phi1(x)*normal_density_numerical(x)*dx
        integration2_numerical = integration2_numerical + \
        phi2(x)*normal_density_numerical(x)*dx
        integration3_numerical = integration3_numerical + \
        phi3(x)*normal_density_numerical(x)*dx
        x = x + dx
        i = i + 1
    #error
    w1_err1[j] = np.abs( mean1 - integration1_numerical)
    w2_err1[j] = np.abs( mean2 - integration2_numerical)
    w3_err1[j] = np.abs( mean3 - integration3_numerical)

    #Quadrature middle point
    integration1_numerical = 0.
    integration2_numerical = 0.
    integration3_numerical = 0.
    i = 1
    x = mu_new - L
    while i <= cuts:
        integration1_numerical = integration1_numerical + \
        phi1(x+1/2*dx)*normal_density_numerical(x+1/2*dx)*dx
        integration2_numerical = integration2_numerical + \
        phi2(x+1/2*dx)*normal_density_numerical(x+1/2*dx)*dx
        integration3_numerical = integration3_numerical + \
        phi3(x+1/2*dx)*normal_density_numerical(x+1/2*dx)*dx
        x = x + dx
        i = i + 1
    #error
    w1_err2[j] = np.abs(mean1 - integration1_numerical)
    w2_err2[j] = np.abs(mean2 - integration2_numerical)
    w3_err2[j] = np.abs(mean3 - integration3_numerical)

    j=j+1


# output
print '\n\nEuler - Maruyama  Left point Quadrature'
print '   N  Testfun1    Testfun2    Testfun3'
for j in range(len(N_list)):
    print '%4d  %0.8f   %0.8f   %0.8f' \
            % (N_list[j], w1_err1[j], w2_err1[j], w3_err1[j])


print '\n\nEuler - Maruyama  middle point Quadrature'
print '   N  Testfun1    Testfun2    Testfun3'
for j in range(len(N_list)):
    print '%4d  %0.8f   %0.8f   %0.8f' \
            % (N_list[j], w1_err2[j], w2_err2[j], w3_err2[j])
```

A.4

```

#This file deals with weak errors of OU process
#by BDF2 scheme.
#With text functions phi(x)=x^2

import numpy as np

# parameters
mu = 1.
sigma = 1.

t = 0.  # initial time
T = 1.  # final time

x0 = 1. # initial value

N_list = 8.*2**np.arange(15) # step sizes


# coefficient functions for exact representation of BDF2
def v0(i,dt):
    # roots of characteristic polynomial
    z1 = (2 + np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    z2 = (2 - np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    # linear coefficients
    lamb1 = -z2/(z1 - z2)
    lamb2 =  z1/(z1 - z2)
    return lamb1*z1**i + lamb2*z2**i

def v1(i,dt):
    # roots of characteristic polynomial
    z1 = (2 + np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    z2 = (2 - np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    # linear coefficients
    lamb1 = 1/(z1 - z2)
    lamb2 = -lamb1
    return np.where(i < 0., 0., lamb1*z1**i + lamb2*z2**i)

def v_eta(i,eta,dt):
    return v1(i-eta+1,dt)/(1-2./3.*dt*mu)

def sol2mom(T):
# This function computes the second moment of the exact solution
    return np.exp(2.*mu*T)*x0**2 + sigma**2*\
    (np.exp(2*mu*T) - 1)/(2*mu)

def BDF2_2mom(T,N):
# This function computes the second moment of the BDF2-Maruyama
# approximation
    dt = T/N
    return (v0(N,dt)*x0 + v1(N,dt)*x0/(1-dt*mu))**2\
            +dt*sigma**2*( (v1(N,dt)/(1-dt*mu) - \
                v_eta(N,2,dt)/3.)**2\
            +np.sum( ( v_eta(N, np.arange(N-1,1,-1),dt) - \
            v_eta(N, np.arange(N,2,-1),dt)/3. )**2) \
            + v_eta(N,N,dt)**2 )

exact = sol2mom(T)

w_err = np.zeros( len(N_list))

for j in range(len(N_list)):
    w_err[j] = np.abs(exact - BDF2_2mom(T,N_list[j]))

# output
print '\nWeak errors'
print '     N     BDF2 '
for j in range(len(N_list)):
    print '%6d  %0.10f' \
            % (N_list[j], w_err[j])
```

A.5

```
#Python
#This file is considered as using Truncation and Quadrature to
#compute expectation of phil(x).
#Solving Equation Onrstein-Uhlenbeck process.
#The numerical scheme here is BDF2 scheme.

import numpy as np
import math
# parameters
```

```
mu = 1.
sigma = 1.
t = 0.  # initial time
T = 1.  # final time
x0 = 1. # initial value

N_list = 8*2**np.arange(8)
#np.arange(8)=[0,1,2,3,4,5,6,7]
#N_list = [  8   16   32   64 128 256 512 1024]

# Test Functions
def phi1(x):
    return   x**3

def phi2(x):
    return   x**4

def phi3(x):
    return   np.sin(x)

w1_err1 = np.zeros(len(N_list))
w2_err1 = np.zeros(len(N_list))
w3_err1 = np.zeros(len(N_list))

def normal_density(x):
    g_x = 1/(sigma*np.sqrt((np.exp(2*mu*T)-1)/(2*mu))*\
        np.sqrt(2*math.pi))*np.exp(-(x-np.exp(mu*T)*x0)**2\
        /(sigma**2*(np.exp(2*mu*T)-1)/mu))
    return g_x

L = 40.
cuts = 5000
# How many small intervals between [mu-L,mu+L] integrated.
dx = 2*L/cuts
i = 1
integration1 = 0.
integration2 = 0.
integration3 = 0.
x = np.exp(mu*T)*x0 - L
while i <= cuts:
    integration1 = integration1 + phi1(x)*normal_density(x)*dx
    integration2 = integration2 + phi2(x)*normal_density(x)*dx
    integration3 = integration3 + phi3(x)*normal_density(x)*dx
    x = x + dx
    i = i + 1

mean1 = integration1
mean2 = integration2
mean3 = integration3
```

```
print '\nParameter values:'  # \n represents enter.
print 'mu = %4.5f, sigma = %4.5f, T = %2.2f, x0 = %4.5f' \
% (mu, sigma, T, x0)
print 'max_N = %d' % ( N_list[-1])

print  '\nExact values:'
print  'Mean1 = %7.8f,' % mean1,
print  'Mean2 = %7.8f,' % mean2,
print  'Mean3 = %7.8f' % mean3

print "\nStarting Monte-Carlo computation of weak errors..."
print "Computing errors for",

# calculating the numerical solution

def v0(i,dt):
    # roots of characteristic polynomial
    z1 = (2 + np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    z2 = (2 - np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    # linear coefficients
    lamb1 = -z2/(z1 - z2)
    lamb2 =  z1/(z1 - z2)
    return lamb1*z1**i + lamb2*z2**i

def v1(i,dt):
    # roots of characteristic polynomial
    z1 = (2 + np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    z2 = (2 - np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    # linear coefficients
    lamb1 = 1/(z1 - z2)
    lamb2 = -lamb1
    return np.where(i < 0., 0., lamb1*z1**i + lamb2*z2**i)

def v_eta(i,eta,dt):
    return v1(i-eta+1,dt)/(1-2./3.*dt*mu)

# functions for explicit computations of second moments

def BDF2_2mom(T,N):
#This function computes the second moment of the BDF2-Maruyama
#approximation
    dt = T/N
    return (v0(N,dt)*x0 + v1(N,dt)*x0/(1-dt*mu))**2\
           +dt*sigma**2*( (v1(N,dt)/(1-dt*mu) - \
               v_eta(N,2,dt)/3.)**2\
           +np.sum( ( v_eta(N, np.arange(N-1,1,-1),dt) - \
           v_eta(N, np.arange(N,2,-1),dt)/3. )**2)\
            + v_eta(N,N,dt)**2)
```

```
def BDF2_mean(T,N):
    dt = T/N
    return v0(N,dt)*x0 + v1(N,dt)*x0/(1-dt*mu)

j=0
for N in N_list:
    print 'N = %d,' % N,
    mu_new=0.
    sigma_new=0.
    k = T/N
    mu_new = BDF2_mean(T,N)
    sigma_new = np.sqrt(BDF2_2mom(T,N) -  BDF2_mean(T,N)**2)
    def normal_density_numerical(x):
        f_x= 1/np.sqrt(2*math.pi*sigma_new**2) * \
        np.exp(-(x-mu_new)**2/(2*sigma_new**2))
        return f_x
    integration1_numerical = 0.
    integration2_numerical = 0.
    integration3_numerical = 0.
    x = mu_new - L
    i = 1
    while i <= cuts:
        integration1_numerical = integration1_numerical +\
        phi1(x)*normal_density_numerical(x)*dx
        integration2_numerical = integration2_numerical +\
        phi2(x)*normal_density_numerical(x)*dx
        integration3_numerical = integration3_numerical +\
        phi3(x)*normal_density_numerical(x)*dx
        x = x + dx
        i = i + 1
    #error
    w1_err1[j] = np.abs( mean1 - integration1_numerical)
    w2_err1[j] = np.abs( mean2 - integration2_numerical)
    w3_err1[j] = np.abs( mean3 - integration3_numerical)
    j=j+1

# output
print '\n\n BDF2  Left point Quadrature'
print '   N   Testfun1    Testfun2    Testfun3'
for j in range(len(N_list)):
    print '%4d  %0.8f   %0.8f   %0.8f' \
            % (N_list[j], w1_err1[j], w2_err1[j], w3_err1[j])
```

A.6

```
#This file deals with strong errors of OU process
```

```
#by BFD2 scheme.

import numpy as np

mu = 1.
sigma = 1.

t = 0.  #initial time
T = 1.  #final time
x0 = 1. #initial value

N_list = 8.*2**np.arange(15)
#np.arange(15)=[0,1,2,3,4,5,6,7,...,14]

# coefficient functions for exact representation of BDF2
def v0(i,dt):
    # roots of characteristic polynomial
    z1 = (2 + np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    z2 = (2 - np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    # linear coefficients
    lamb1 = -z2/(z1 - z2)
    lamb2 =  z1/(z1 - z2)
    return lamb1*z1**i + lamb2*z2**i

def v1(i,dt):
    # roots of characteristic polynomial
    z1 = (2 + np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    z2 = (2 - np.sqrt(1. + 2.*dt*mu))/(3. - 2.*dt*mu)
    # linear coefficients
    lamb1 = 1/(z1 - z2)
    lamb2 = -lamb1
    return np.where(i < 0., 0., lamb1*z1**i + lamb2*z2**i)
#v_eta/a_2
def v_eta(i,eta,dt):
    return v1(i-eta+1,dt)/(1-2./3.*dt*mu)


def xi(n,dt,N):
    #if n == 1:
    #   return v1(N,dt)/(1-dt*mu)-v_eta(N,2,dt)/3
    #elif n == N:
    #   return 1/(1-2./3.*dt*mu)
    #else:
        return v_eta(N,n,dt)-v_eta(N,n-1,dt)/3

def ksi(dt,N):
    return  v1(N,dt)/(1-dt*mu)-v_eta(N,2,dt)/3

def kxi(dt,N):
    return  1/(1-2./3.*dt*mu)


def int_n(n,dt,N):
    return  1/(2*mu)*np.exp(2*mu*(T-n*dt))*(np.exp(2*dt*mu)-1)\
            -(2*xi(n,dt,N)/mu)*np.exp(mu*(T-n*dt))*\
            (np.exp(dt*mu)-1)+dt*xi(n,dt,N)**2
def int_1(dt,N):
    return  1/(2*mu)*np.exp(2*mu*(T-1*dt))*(np.exp(2*dt*mu)-1)\
            -(2*ksi(dt,N)/mu)*np.exp(mu*(T-1*dt))*\
            (np.exp(dt*mu)-1)+dt*ksi(dt,N)**2
def int_N(dt,N):
    return  1/(2*mu)*np.exp(2*mu*(T-N*dt))*(np.exp(2*dt*mu)-1)\
            -(2*kxi(dt,N)/mu)*np.exp(mu*(T-N*dt))*\
            (np.exp(dt*mu)-1)+dt*kxi(dt,N)**2

def BDF2_Strong_Error(T,N):
    dt = T/N
    return (np.exp(mu*T)-v0(N,dt)-v1(N,dt)/(1-dt*mu))**2*x0**2\
           +sigma**2*( np.sum(int_n(np.arange(2,N),dt,N)) \
            + int_1(dt,N) + int_N(dt,N) )

s_err = np.zeros(len(N_list))

for j in range(len(N_list)):
    s_err[j] = np.sqrt(BDF2_Strong_Error(T,N_list[j]))

# output
print '\nStrong errors'
print '     N      BDF2 '
for j in range(len(N_list)):
    print '%6d  %0.10f' \
            % (N_list[j], s_err[j])
```

A.7

```
#Python
#This file is considered using Truncation and Quadrature to
#compute expectation of phil(x), where phil(x) is of polynomial
#growth.
#Solving Black-Scholes Stochastic Equations
#Black Scholes Equation: dX(t) = mu*X(t)*dt + sigma*X(t)*dW(t)
#The numerical method we ues is implicit Euler Maruyama scheme.

import numpy as np
import math
# parameters
```

```
mu = 1.
sigma = 1.
T = 1. #final time
x0 = 1. #initial value
N_list= 8*2**np.arange(18)

#Test functions
def phil1(x):
    return x
def phil2(x):
    return x**2
def phil3(x):
    return x**3

w_err1 = np.zeros(len(N_list))
w_err2 = np.zeros(len(N_list))
w_err3 = np.zeros(len(N_list))

def normal_density(x):
    new_mu = (mu - sigma**2/2)*T
    new_sigma = sigma*np.sqrt(T)
    g_x = 1/(np.sqrt(2*math.pi)*new_sigma) * \
    np.exp(-(x-new_mu)**2/(2*new_sigma**2))
    return g_x

L=40.
cuts = 1000
dx = 2*L/cuts
i = 1
integration1 = 0.
integration2 = 0.
integration3 = 0.
x = (mu - sigma**2/2)*T - L
while i <= cuts:
    integration1 = integration1 + phil1(x0*np.exp(x))*\
    normal_density(x)*dx
    integration2 = integration2 + phil2(x0*np.exp(x))*\
    normal_density(x)*dx
    integration3 = integration3 + phil3(x0*np.exp(x))*\
    normal_density(x)*dx
    x = x + dx
    i = i + 1

mean1 = integration1
mean2 = integration2
mean3 = integration3

print '\nParameter values:'  # \n represents enter.
print 'mu = %4.5f, sigma = %4.5f, T = %2.2f, x0 = %4.5f' \
% (mu, sigma, T, x0)
print 'max_N = %d' % (N_list[-1])
print  '\nExact values:'
print  'Mean1 = %7.8f,' % mean1,
print  'Mean2 = %7.8f,' % mean2,
print  'Mean3 = %7.8f'  % mean3

print "\n"
print "Computing errors for",
j = 0
for N in N_list:
    print 'N = %d,' %N,
    k = T/N
    Expectation = (1 - mu*k)**(-N)*x0
    Expectation_2mon = (1 - mu*k)**(-2*N) * x0**2 * \
    (1+sigma**2*k)**N

    def normal_density_numerical(x):
        new_mu = 1.
        new_sigma = sigma*np.sqrt(k)
        g_x = 1/(np.sqrt(2*math.pi)*new_sigma) * \
        np.exp(-(x-new_mu)**2/(2*new_sigma**2))
        return g_x
    Integration = 0.
    L = 40.
    cuts = 100000
    dx = 2 * L/cuts
    i = 1
    x = 1. - L
    while i <= cuts:
        Integration = Integration + x**3*\
        normal_density_numerical(x)*dx
        x = x + dx
        i = i + 1
    Expectation_3mon = (1 - mu*k)**(-3*N) * x0**3 * \
    (Integration)**N

    w_err1[j] = np.abs(mean1 - Expectation)
    w_err2[j] = np.abs(mean2 - Expectation_2mon)
    w_err3[j] = np.abs(mean3 - Expectation_3mon)
    j=j+1

# output
print '\n\nEuler - Maruyama ;Quadrature'
print '   N  Testfun1    Testfun2    Testfun3'
for j in range(len(N_list)):
    print '%4d  %0.8f   %0.8f   %0.8f' \
            % (N_list[j], w_err1[j], w_err2[j], w_err3[j])
```

A.8 Difference Equation

Consider the difference equation:

$$\begin{gathered}\sum_{j=0}^{m} a_j Y^{n-m+j} = f_n, \quad n = m,m+1,m+2,\dots, \\ Y^n = y_n, \quad n = 0,1,2,\dots,m-1,\end{gathered} \tag{A.1}$$

where $m \in \mathbb{N}^+$, $a_0,\dots,a_m \in \mathbb{R}$ with $a_m \neq 0$.

We want to find the explicit presentation of $Y_n, n = 1,2,\dots,$. The solution can be found in terms of the system of fundamental solutions $(Z^{n,\eta})\, n \in \mathbb{N}, \eta = 0,\dots,m-1$. For every $\eta \in \{0,\dots,m-1\}$, the squence $(Z^{n,\eta})$ solves the following equation:

$$\begin{gathered}\sum_{j=0}^{m} a_j Y^{n-m+j} = 0, \quad n = m,m+1,m+2,\dots, \\ Y^n = \delta_{n,\eta}, \quad n = 0,1,2,\dots,m-1,\end{gathered} \tag{A.2}$$

here, $\delta_{n,\eta}$ is called the Kronecker delta, satisfying

$$\delta_{n,\eta} = \begin{cases} 1 & \text{if } n = \eta \\ 0 & \text{if } n \neq \eta \ . \end{cases} \tag{A.3}$$

The solution of Y^n is given by

$$Y^n = \begin{cases} \sum_{\eta=0}^{m-1} y_\eta Z^{n,\eta} + \sum_{\eta=m}^{n} \frac{1}{a_m} Z^{n-\eta+m-1,m-1} f_\eta & \text{if } n \geq m \\ \sum_{\eta=0}^{n} y_\eta Z^{n,\eta} & \text{if } n \leq m-1 \ . \end{cases} \tag{A.4}$$

Formula (A.4) is easy to check for $n = 0,\dots,m-1$, we check for $n = m$, we can easily calculate from (A.1) that

$$\begin{aligned} Y^m &= \frac{1}{a_m}\Big(f_m - \sum_{j=0}^{m-1} a_j Y^j\Big) \\ &= \frac{f_m}{a_m} - \frac{a_0}{a_m}Y^0 - \frac{a_1}{a_m}Y^1 - \dots - \frac{a_{m-1}}{a_m}Y^{m-1}, \end{aligned}$$

we get from (A.4) that

$$\begin{aligned} Y^m &= \sum_{\eta=0}^{m-1} y_\eta Z^{m,\eta} + \frac{1}{a_m} Z^{m-1,m-1} f_m \\ &= y_0 Z^{m,0} + y_1 Z^{m,1} + \dots + y_m Z^{m,m-1} + \frac{1}{a_m} Z^{m-1,m-1} f_m, \end{aligned}$$

we calculted from (A.2) that $Z^{m,0} = -\frac{a_0}{a_m}$, $Z^{m,1} = -\frac{a_1}{a_m}$,...,$Z^{m,m-1} = -\frac{a_{m-1}}{a_m}$, satisfied.

For general case ($n \geq m+1$), we see from (A.4), the first sum $\sum_{\eta=0}^{m-1} y_\eta Z^{n,\eta}$ is a linear combination of solutions to the homogeneous problem, which solves:

$$\sum_{j=0}^{m} a_j Y^{n-m+j} = 0, \quad n = m,m+1,...$$
$$Y^n = y_n, \quad n = 0,1,2,...,m-1,$$

the second sum $\sum_{\eta=m}^{n} \frac{1}{a_m} Z^{n-\eta+m-1,m-1} f_\eta$, sovles the inhomogeneous problem, but with zero initial conditons:

$$\sum_{j=0}^{m} a_j Y^{n-m+j} = f_n, \quad n = m,m+1,...$$
$$Y^n = 0, \quad n = 0,1,2,...,m-1.$$

Next we check if the second sum $\sum_{\eta=m}^{n} \frac{1}{a_m} Z^{n-\eta+m-1,m-1} f_\eta$ indeed solves the above difference equation ($n \geq m+1$). We have

$$\sum_{j=0}^{m} a_j \sum_{\eta=m}^{n-m+j} \frac{1}{a_m} Z^{n-m+j-\eta+m-1,m-1}$$
$$= \sum_{\eta=m}^{n} \frac{1}{a_m} f_\eta \sum_{j=0}^{m} \mathbb{I}_{\{m,m+1,...n-m+j\}}(\eta) Z^{n+j-\eta-1,m-1}.$$

Now for the sum with $\eta = n$:

$$\frac{1}{a_m} f_n \sum_{j=0}^{m} \mathbb{I}_{\{m,m+1,...,n-m+j\}}(n) a_j Z^{j-1,m-1} = \frac{1}{a_m} f_m a_m = f_m.$$

For other sums $\eta < n$ we notice that

$$Z^{n+j-\eta-1,m-1} = 0,$$

whenever $n - m + j < \eta < n$. Hence, $\mathbb{I}$ disappears and thus becomes

$$\sum_{j=0}^{m} \mathbb{I}_{\{m,m+1,...,n-m+j\}}(\eta) a_j Z^{n+j-\eta-1,m-1} = \sum_{j=0}^{m} a_j Z^{n+j-\eta-1,m-1} = 0,$$

for all $m \leq \eta \leq n-1$, thus verified.

Next we determine the fundamental solutions $(Z^{n,\eta})_{n\in\mathbb{N},\eta\in\{0,...,m-1\}}$ and then the solution for general difference equation is given as (A.4). At the very beginning, we should solve the homogeneous difference equation and then we extend it to the fundamental solution $Z^{n,\eta}$. The homogeneous equation is given as follows:

$$\sum_{j=0}^{m} a_j Y^{n-m+j} = 0, \quad n = m,m+1,m+2,..., \tag{A.5}$$

which results to a corresponding characteristic polynomial given by

$$\rho(z) = \sum_{j=0}^{m} a_j z^j, z \in \mathbb{C}.$$

Let ζ_1,...,ζ_k be the roots of the characteristic polynomial, with multiplicity s_1,...,s_k ($s_1+s_2+...+s_k = m$). The following sequences are linearly independent and span the solution space of the homogeneous equation:

$$\begin{aligned}
&(\zeta_1^n)_{n=0}^{\infty}, (n\zeta_1^n)_{n=0}^{\infty},...,(n^{s_1-1}\zeta_1^n)_{n=0}^{\infty},\\
&(\zeta_2^n)_{n=0}^{\infty}, (n\zeta_2^n)_{n=0}^{\infty},...,(n^{s_2-1}\zeta_2^n)_{n=0}^{\infty},\\
&\qquad\qquad ...\\
&(\zeta_k^n)_{n=0}^{\infty}, (n\zeta_k^n)_{n=0}^{\infty},...,(n^{s_k-1}\zeta_k^n)_{n=0}^{\infty}.
\end{aligned}$$

We pick this sequence $(n\zeta_1^n)_{n=0}^{\infty}$ to show to the validity of the statement. We get that

$$\begin{aligned}
Y_i &= (Y_i^0,Y_i^1,Y_i^2,...), \quad i = 1,...,m\\
&= (0,\zeta_1^1,2\zeta_1^2,...,n\zeta_1^n,...).
\end{aligned}$$

Insert this homogeneous solution to the left hand side of the (A.5), we get

$$\begin{aligned}
\sum_{j=0}^{m} a_j Y^{n-m+j} &= \sum_{j=0}^{m} a_j j \zeta_1^{n-m+j}\\
&= \zeta_1^{n-m+1} \sum_{j=0}^{m} a_j j \zeta_1^{j-1}\\
&= \zeta_1^{n-m+1} \cdot 0 = 0,
\end{aligned}$$

satisfied. Note that any linear combinations of those homogeneous solutions can be a solution thus formulate the whole space.

Now we focus on the formulation of the fundamental solutions, i.e. solutions to (A.2). We should find one (and the only one) combination of those homogeneous sulotions (m squences) satisfying the initial data given by the fundamental equaitions (A.2). For simplicity we are here only to consider the characteristic polynomial with ζ_1,...,ζ_m to be the roots each have multiplicity 1. (Actually, what we are dealing with for the BDF2 scheme is a second order difference equation with two different roots each with multiplicity 1).

As

$$\begin{aligned}
Z^{:,0} &= (Z^{0,0},Z^{1,0},Z^{2,0},...,Z^{m-1,0},Z^{m,0},...)\\
&= (1,0,0,...,0,Z^{m,0},...)\\
Z^{:,1} &= (Z^{0,1},Z^{1,1},Z^{2,1},...,Z^{m-1,1},Z^{m,1},...)\\
&= (0,1,0,...,0,Z^{m,1},...)\\
&...\\
Z^{:,m-1} &= (Z^{0,m-1},Z^{1,m-1},Z^{2,m-1},...,Z^{m-1,m-1},Z^{m,m-1},...)\\
&= (0,0,0,...,1,Z^{m,m-1},...),
\end{aligned}$$

we assume $Z^{:,i-1} = \sum_{j=1}^{m} b_j^i Y_j, i = 1,...,m$. Now the question is to determine the coefficients of $(b_j^i)_{j=1}^m$ for any given $i \in \{1,...,m\}$. After all we have to determine m^2 coefficients. The following matrix shows how it works:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \zeta_1 & \zeta_2 & \cdots & \zeta_m \\ \vdots & \vdots & \ddots & \vdots \\ \zeta_1^{m-1} & \zeta_2^{m-1} & \cdots & \zeta_m^{m-1} \end{pmatrix} \begin{pmatrix} b_1^i \\ b_2^i \\ \vdots \\ b_m^i \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \leftarrow i\text{th slot.} \tag{A.6}$$

By soloving the above m such linear equations we have the expression for $Z^{:,i-1} = \sum_{m}^{j=1} b_{j=1}^{i} Y_j$, $i = 1,...,m$. And by formula (A.4), we solved the difference equation.