AMATH 586 Numerical SDE Solvers

Tyler Chen

Abstract

This report is intended as a starting reference for numerical methods to SDEs and contains content similar to [2, 3]. The standard text on this topic is [4], which provides an in-depth approach to stochastic calculus, and numerical methods for SDEs. As a starting point to stochastic calculus in general we refer readers to [1].

1 Introduction

A Stochastic Differential Equation (SDE) is an equation of the form,

$$\mathrm{d}X_t = \mu(X_t, t)\mathrm{d}t + \sigma(X_t, t)\mathrm{d}W_t \tag{1}$$

where W_t denotes a standard Brownian motion [1]. We can write this in integral form as,

$$X_t - X_0 = \int_0^t \mu(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s \tag{2}$$

In general we would like to write our solution X_T as an explicit function of a Brownian motion trajectory. For some processes this is possible, however in general it is not. As a result, numerical methods are required for many SDEs.

SDEs are commonly used to model processes such as molecular dynamics, neurodynamics, financial quantities (stocks, asset prices, interest rates, etc.) SDEs can also be used to efficiently solve high dimensional PDEs numerically, (standard mesh based methods scale exponentially with the dimension of the system).

In this report we first introduce Brownian motion and stochastic calculus. We then discuss how Brownian motion can be approximated numerically and discuss what it means for a numerical SDE method to converge. Next we present simple numerical method and two higher order methods which are obtained using stochastic Taylor expansions. Finally, we conduct some experiments to corroborate the theoretical claims we present earlier in the paper.

2 Brownian motion

There are many equivalent characterizations of Brownian motion. Perhaps the most standard is the following definition.

Definition. *A Brownian motion is a stochastic process* $W = (W_t)_{t\geq 0}$ *defined on some probability space* $(\Omega, \mathcal{F}, \mathbb{P})$ *satisfying,*

1. $W_0 = 0$
2. $(W_d - W_c) \perp\!\!\!\perp (W_b - W_a)$ *for all* $0 \leq a \leq b \leq c \leq d$)
3. $(W_t - W_s) \sim \mathcal{N}(0, t-s)$ *for all* $0 \leq s \leq t$
4. *the map* $t \to W_t(\omega)$ *is continuous for almost all* $\omega \in \Omega$

We attempt to parse this definition and notation. The notation W_t denotes a random variable which depends on the time t. That is, W_t is a function from $\Omega \to \mathbb{R}$. For a fixed $\omega \in \Omega$, $W_t(\omega)$ can be seen as a function of t, and is referred to as a path or a trajectory of the Brownian motion.

Condition 1 means that all paths of the Brownian motion starts start at the origin. Condition 2 means that the random variables $W_d - W_c$ and $W_b - W_a$ are independent of one another. That is, knowing how much the a path of the Brownian motion changes over the time interval $[a, b]$ does not give any information about how much the path changes over the interval $[c, d]$. Condition 3 means that over any time interval $[s, t]$ how much the Brownian motion changes is a normal random variable with mean zero and variance $t - s$. Finally, Condition 4 means that almost all paths of the Brownian motion are continuous.

Brownian motion can be constructed as a scaled random walk. Such a construction can be readily obtained on the internet or in [1]. We will simply note that it may be useful to view Brownian motion as the continuous analog to a discrete random walk.

3 Stochastic Calculus

In this section we aim to provide enough background to allow all statements in the main portion of the paper to be understood. That said, this done in the admittedly unsatisfying way of black box lemma's and heuristics which we do not explain or prove. For a more complete overview of Stochastic Calculus we refer readers to [4, 1].

We first introduce Riemann–Stieltjes integrals.

Definition. *For real valued functions* f *and* g *the Riemann–Stieltjes integral is defined as,*

$$\int_a^b f(x)\mathrm{d}g(x) := \lim_{\|\Pi\|\to 0} \sum_{i=0}^{n-1} f(c_i)(g(x_{i+1}) - g(x_i))$$

where $\Pi = \{a = x_0 < x_1 < \cdots < x_n = b\}$ *is a partition of* $[a, b]$, $\|\Pi\|$ *is the length of the largest subinterval, and* c_i *is any point in* $[x_i, x_{i+1}]$.

We note that if $g(x) = x$ the Riemann–Stieltjes integral is the standard Riemann integral

and that if g is continuously differentiable,

$$f(g(t)) - f(g(0)) = \int_0^t \mathrm{d}f(g(s)) = \int_0^t f'(g(s))g'(s)\mathrm{d}s$$

Brownian motion and many processes involving Brownian motion are not differentiable. Itô's Lemma gives us a way to compute the analogous result for a class of stochastic processes called Itô (drift-diffussion) processes. For our purposes we can think of Itô processes as processes with an integral with respect to t and an integral with respect to W_t.

Lemma (Itô)**.** *For $f : \mathbb{R} \to \mathbb{R}$ sufficiently differentiable and Itô process X_t,*

$$\mathrm{d}f(X_t) = f'(X_t)\mathrm{d}X_t + \frac{1}{2}f''(X_t)\mathrm{d}[X, X]_t \tag{3}$$

Here we have used the notation $[X, X]_t$ to denote the quadratic variation of the process X_t. Heuristically we can compute $\mathrm{d}[X, X]_s$ by expanding $(\mathrm{d}X_s)(\mathrm{d}X_s)$ and simplifying using the identities,

$$\mathrm{d}t\mathrm{d}t = 0, \qquad \mathrm{d}t\mathrm{d}W_t = 0, \qquad \mathrm{d}W_t\mathrm{d}W_t = \mathrm{d}t \tag{4}$$

Itô's Lemma can be generalized to functions and processes of higher dimension.

Lemma (Itô)**.** *For $f : \mathbb{R}^n \to \mathbb{R}^n$ sufficiently differentiable and Itô process $X_t = [X_t^1, X_t^2, \dots X_t^n]^T$,*

$$\mathrm{d}f(X_t) = \sum_{i=1}^{n}\left[\frac{\partial}{\partial x_i}f(X_t)\right]\mathrm{d}X_t^i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[\frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}f(X_t)\right]\mathrm{d}[X^i, X^j]_t \tag{5}$$

Similar to before, we compute $\mathrm{d}[X^i, X^j]_t$ by expanding $(\mathrm{d}X^i)(\mathrm{d}X^j)$ and using the heuristics in (4).

Consider the special case when $n = 2$, $X_t^1 = t$, and $X_t^2 = X_t$. Using our heuristics in (4) we have $\mathrm{d}[X^1, X^2]_t = (\mathrm{d}t)(\mathrm{d}X_t) = 0$ and $\mathrm{d}[X^1, X^1] = (\mathrm{d}t)(\mathrm{d}t) = 0$. Therefore, by (5),

$$\begin{aligned}\mathrm{d}f(t, X_t) &= \frac{\partial}{\partial t}f(t, X_t) + \frac{\partial}{\partial x}f(t, X_t)\mathrm{d}X_t + \frac{1}{2}\frac{\partial^2}{\partial x^2}f(t, X_t)\mathrm{d}[X, X]_t &(6)\\ &= \left(\frac{\partial}{\partial t} + \frac{1}{2}\frac{\partial^2}{\partial x^2}\right)f(t, X_t)\mathrm{d}t + \frac{\partial}{\partial x}f(t, X_t)\mathrm{d}X_t &(7)\end{aligned}$$

4 Example Processes

In this section we provide some results about a few important stochastic processes which we will use in our numerical tests. Both the SDEs we introduce can be solved analytically and the mean and variance can be computed

For constants μ and σ a Geometric Brownian motion satisfies,

$$\mathrm{d}X_t = \mu X_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t, \qquad t \in [0, T] \tag{8}$$

We will solve this equation to build some familiarity with stochastic calculus. To this end, note that $\mathrm{d}X_t = \mu X_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t$ so using the heuristics (4) we have $\mathrm{d}[X, X]_t = \sigma^2 \mathrm{d}t$. We note apply Itô's Lemma to $\ln(x)$ to obtain,

$$\mathrm{d}\ln(X_t) = \frac{1}{X_t}\mathrm{d}X_t - \frac{1}{2}\frac{1}{X_t^2}\mathrm{d}[X, X]_t = \left(\mu - \frac{\sigma^2}{2}\right)\mathrm{d}t + \sigma \mathrm{d}W_t$$

Integrating both sides from T to t we obtain,

$$\ln(X_t) = \int_0^t \mathrm{d}\ln(X_s) = \int_0^t \left(\mu - \frac{\sigma^2}{2}\right)\mathrm{d}s + \int_0^t \sigma \mathrm{d}W_s = \left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t$$

Therefore, the solution to (8) is,

$$X_t = X_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right)$$

For constants θ, μ, σ, an Ornstein–Uhlenbeck (OU) process satisfies,

$$\mathrm{d}X_t = \theta(\mu - X_t)\mathrm{d}t + \sigma \mathrm{d}W_t, \qquad \theta > 0 \tag{9}$$

We note that if X_t is away from μ it will tend towards this value in expectation. Figure 1a shows Euler–Maruyama method applied to (9). As expected, the trajectories all end up "centered" about μ. More precisely, (9) has solution,

$$X_t = X_0 \exp(-\theta t) + \mu(1 - \exp(-\theta t)) + \sigma \int_0^t \exp(-\theta(t - s))\mathrm{d}W_s$$

The mean of X_t is,

$$\mathbb{E}[X_t] = (X_0 - \mu)\exp(-\theta t) + \mu$$

Likewise, the variance of X_t is,

$$\mathbb{E}\left[(X_t - \mathbb{E}[X_t])^2\right] = \frac{\sigma^2}{2\theta}(1 - \exp(-2\theta t))$$

5 Generating Brownian motion Numerically

Given the four properties of Brownian motion, how can we numerically generate trajectories which are representative of a Brownian motion? The most standard way to do this

is to first pick a time mesh on which we will have the values of the Brownian motion. Let $t = [t_0, t_1, \ldots, T]$ where $0 = t_0 < t_1 < \cdots < t_N = T$ be such a mesh. We will generate a numerical realization $\hat{W} = [0, \hat{W}_1, \ldots, \hat{W}_T]$ of a Brownian motion trajectory. To do this we generate the incriments. That is, we start with $\hat{W}_0 = 0$ and recursively define,

$$\hat{W}_{t_{k+1}} = \hat{W}_{t_k} + \Delta\hat{W}_{t_k}$$

where, using property (3) of Brownian motion, we must sample $\Delta\hat{W}_{t_k}$ from a normal distribution with mean 0 and variance $t_{k+1} - t_k$.

Since we only sample at discrete points, we have not generated a "true" trajectory of the Brownian motion. In fact, refining the mesh and generating the interior points is not trivial. However, it is clear that the property 2 is satisfied since we sample or increments independently. Similarly, property 3 is satisfied since the sum of iid normal random variables is a iid random variable with mean equal to the sum of the means, and variance equal to the sum of the variances. Therefore this method gives us trajectories which satisfy the first three properties of Brownian motion.

We now make a few practical observations about implementing this procedure on a computer. First, while we have explicitly defined the vector $\hat{W}$, it may be more efficient to save the vector $\Delta\hat{W}$ since the numerical methods we present involve ΔW_t. Second, if it is slow to sample normal random variables, other simpler distributions coud be used. For instance, we could pick $\Delta W_t \in \{-\sqrt{\mathrm{d}t}, \sqrt{\mathrm{d}t}\}$ with equal probabilities for each choice. In this case the mean and variance will still match Brownian motion, but the higher moments may not (i.e. the increments will no longer be normally distributed). Whether or not this is important is dependent on the application.

6 Convergence of Numerical Methods

A SDE solution may have many distinct trajectories as the solution itself is a random process. Our standard definitions of convergence for IVPs are no longer sufficient.

For some applications we would like to be able to compute trajectories numerically which are representative of the actual trajectories of the solution to the SDE. This gives rise to the following definition convergence.

Definition. *A discrete time approximation $\hat{X}^{(k)}$ is said to converge strongly to the solution X_t at time T if,*

$$\lim_{k\to 0} \mathbb{E}\left[\left|X_T - \hat{X}_T^{(k)}\right|\right] = 0$$

and is said to converge strongly with order m if for sufficiently small k,

$$\mathbb{E}\left[\left|X_T - \hat{X}_T^{(k)}\right|\right] = \mathcal{O}(k^m)$$

Since strong convergence looks at the absolute difference of $\hat{X}_T^{(k)}$ and X_T it tells us about how individual trajectories vary. That is, to compute this expectation we first find the absolute difference of our solution and the exact solution for every trajectory of Brownian motion, and then take the mean.

For other applications it may be sufficient to generate trajectories whose moments (mean, variance, etc.) are the same as the trajectories of the solution to the SDE, even if individual trajectories don't necessarily converge to trajectories of the solution. This gives rise to the notion of weak convergence.

Definition. *A discrete time approximation $\hat{X}^{(k)}$ is said to converge weakly to the solution X_t at time T if for every polynomial p,*

$$\lim_{k\to 0}\left|\mathbb{E}\left[p(X_T)-p\left(\hat{X}_T^{(k)}\right)\right]\right|=0$$

and is said to converge weakly with order m if for sufficiently small k and every polynomial p,

$$\left|\mathbb{E}\left[p(X_T)-p\left(\hat{X}_T^{(k)}\right)\right]\right|=\mathcal{O}(k^m)$$

Since weak convergence looks at the signed difference of $\hat{X}_T^{(k)}$ and X_T the errors can "cancel" with one another.

Note that the term corresponding to $\mathrm{d}t$ is deterministic. This means that if $\sigma \equiv 0$, Equation (2) becomes the initial value problem $\mathrm{d}X_t/\mathrm{d}t = \mu(X_t, t)$ with initial condition at time t given by the value of X_t. In this case both definitions of convergence agree with the standard definition of convergence for IVPs provided in [5].

We now prove the following (unsurprising) theorem following closely to the proof provided in [2].

Theorem. *The weak order of a SDE method is always less than or equal to the strong order of the method.*

Proof. Suppose a method $\hat{X}^{(k)}$ is strongly convergent. Then the set $S := \{\hat{X}_T^{(k)}\}_k \cup \{X_T\}$ is bounded. Therefore, there exists $K > 0$ such that $K > |p'(x)|$ for all x in some finite interval containing S.

Since $|\mathbb{E}Z| \le \mathbb{E}|Z|$ for any random variable Z, and by the Mean Value Theorem,

$$\left|\mathbb{E}\left[p(X_T)-p\left(\hat{X}_T^{(k)}\right)\right]\right| \le \mathbb{E}\left[\left|p(X_T)-p\left(\hat{X}_T^{(k)}\right)\right|\right] \le K\mathbb{E}\left[\left|X_T-\hat{X}_T^{(k)}\right|\right]$$

This proves the result. □

7 A simple Numerical Method

All the numerical methods in this report find an approximate solution to a given SDE on some time grid. More specifically, the methods take in a series of time points and the

value of a Brownian motion trajectory at these time points and output the approximate solution.

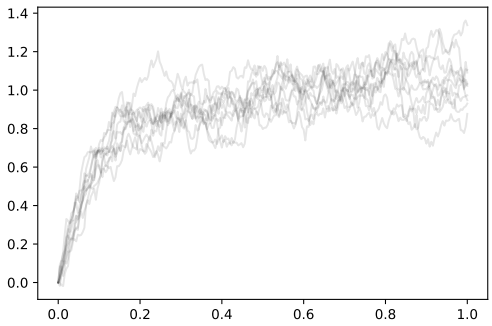
We first introduce the natural generalization of the Forward Euler method. The so called Euler–Maruyama method [3] is given by the relation,

$$\hat{X}_{t+\Delta t} = \hat{X}_t + \mu(t, \hat{X}_t)\Delta t + \sigma(t, \hat{X}_t)\Delta W_t$$

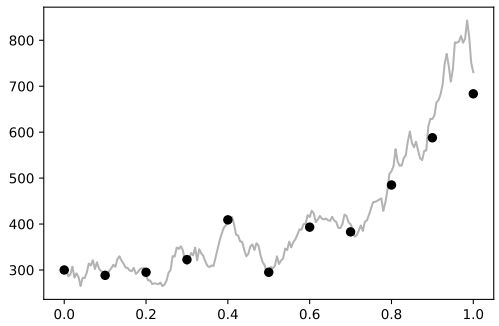
where $\Delta W_t := W_{t+\Delta t} - W_t$.

On SDEs where σ and μ satisfy the appropriate conditions (these condtions are generally satisifed), the Euler-Maruyama method has strong and weak orders of 1/2 and 1 respectively. A commonly cited proof of this result can be found in [4]. However we note that on some SDEs strong convergence of order 1 is attained.

We test the Euler–Maruyama method on the Ornstein–Uhlenbeck equation and Geometric Brownian motion equation. Figure 1 shows the result of two tests. More specifically, Figure 1a shows 10 trajectories of the OU process computed using the Euler-Maruyama method. As expected these trajectories all seem to be heading towards a mean of 1 as time progresses. Figure 1b shows the result of the Euler-Maruyama a large step size in comparison to the exact solution.



(a) 10 trajectories of the OU equation with $\theta = 10$, $\mu = 1$, $\sigma = 1/2$ computed using the Euler–Maruyama method with a step size of $\Delta t = 0.004$.



(b) Exact trajectory of Geometric Brownian motion with $\mu = 1$, $\sigma = 1/2$, and solution computed using the Euler–Maruyama method with a step size of $\Delta t = 1/10$.

Figure 1: Basic numerical tests of Euler–Maruyama method

8 Higher Order Methods

The Euler–Maruyama method is commonly used due to its simplicity and ease of implementation. However, the strong convergence of this method is quite slow. As a result, higher order methods have been developed and studied. We introduce two methods in

the hopes of illustrating how such methods can be derived. Both methods have strong and weak order one. A similar derivation can be found in [2].

By Itô's Lemma (7), if X_T is the solution to (2), for any sufficiently differentiable function f we have,

$$f(T, X_T) = f(t, X_t) + \int_0^t \mathcal{M} f(s, X_s)\mathrm{d}s + \int_0^t \mathcal{S} f(s, X_s)\mathrm{d}W_s \tag{10}$$

where,

$$\mathcal{M} := \frac{\partial}{\partial s} + \mu(s, X_s)\frac{\partial}{\partial x} + \frac{1}{2}\sigma^2(s, X_s)\frac{\partial^2}{\partial x^2}, \qquad \mathcal{S} := \sigma(s, X_s)\frac{\partial}{\partial x}$$

Applying the relation in (10) with $f = \mu$ and $f = \sigma$ gives,

$$\begin{aligned}
\mu(s, X_s) &= \mu(t, X_t) + \int_0^s \mathcal{M}\mu(u, X_u)\mathrm{d}u + \int_0^s \mathcal{S}\mu(u, X_u)\mathrm{d}W_u \\
\sigma(s, X_s) &= \sigma(t, X_t) + \int_0^s \mathcal{M}\sigma(u, X_u)\mathrm{d}u + \int_0^s \mathcal{S}\sigma(u, X_u)\mathrm{d}W_u
\end{aligned}$$

Therefore, inserting these expressions into (2) gives the relations,

$$X_t = X_0 + \mu(t, X_t)\int_0^t \mathrm{d}s + \sigma(t, X_t)\int_0^t \mathrm{d}W_s + R \tag{11}$$

where

$$\begin{aligned}
R = &\int_0^t \int_0^s \mathcal{M}\mu(u, X_u)\mathrm{d}u\mathrm{d}s + \int_0^t \int_0^s \mathcal{S}\mu(u, X_u)\mathrm{d}W_u\mathrm{d}s \\
&+ \int_0^t \int_0^s \mathcal{M}\sigma(u, X_u)\mathrm{d}u\mathrm{d}W_s + \int_0^t \int_0^s \mathcal{S}\sigma(u, X_u)\mathrm{d}W_u\mathrm{d}W_s
\end{aligned} \tag{12}$$

Note the Euler–Maruyama method arises from (11) after dropping all the terms in R and setting $T = t + \Delta t$.

Using the hueristics (4) the $\mathrm{d}W_s\mathrm{d}W_t$ term is the highest order term of R, so a higher order method must keep more of this term. We therefore apply the relation in (10) with $f = \mathcal{S}\sigma$ to obtain,

$$\mathcal{S}\sigma(u, X_u) = \mathcal{S}\sigma(t, X_t) + \int_t^u \mathcal{M}\mathcal{M}\sigma(v, X_v)\mathrm{d}v + \int_t^u \mathcal{S}\mathcal{M}\sigma(v, X_v)\mathrm{d}W_v$$

Therefore,

$$X_t = X_0 + \int_0^t \mu(t, X_t)\mathrm{d}s + \int_0^t \sigma(t, X_t)\mathrm{d}W_s + \int_0^t \int_0^s \mathcal{S}\sigma(t, X_t)\mathrm{d}W_u\mathrm{d}W_v + R \tag{13}$$

where R contains higher order terms (and is different from the R in (11)).

Applying Itô's Lemma to $f(x) = x^2$ we obtained the well known result,

$$\int_0^t \int_t^s \mathrm{d}W_u \mathrm{d}W_s = \int_0^t W_s \mathrm{d}W_s = \frac{1}{2}\left((W_t)^2 - t\right)$$

Dropping all the terms in the new R and setting $T = t + \Delta t$, (13) gives rise to the Milstein method,

$$\hat{X}_{t+\Delta t} = \hat{X}_t + \mu(t, \hat{X}_t)\Delta t + \sigma(t, \hat{X}_t)\Delta W_t + \sigma(t, \hat{X}_t)\partial_x \sigma(t, \hat{X}_t)\left((\Delta W_t)^2 - \Delta t\right)/2$$

Both the Milstein and Forward Euler methods can be seen as Taylor series methods. Using similar approximations to what we have done here arbitrary order methods can be obtained. However, deriving and implementing such methods can easily become unwieldy. Therefore, as with deterministic IVP methods, the order of accuracy of a method must be balanced with the complexity of the method when deciding which method to apply.

As with deterministic IVP Taylor methods, knowing the derivative to a function is necessary. It is sometimes preferable to have methods which do not rely on knowing the derivative of any functions ahead of time.

We can approximate the x-derivative of σ by,

$$\frac{\partial}{\partial x}\sigma(\hat{X}_t, t) = \frac{\sigma(t, \hat{X}_t + \sigma(\hat{X})\sqrt{\Delta t}) - \sigma(t, \hat{X}_t)}{\sigma(\hat{X}, t)\sqrt{\Delta t}} + \mathcal{O}\left(\sigma(t, \hat{X}_t)\sqrt{\Delta t}\right) \tag{14}$$

Using our heuristics (4) we expect substituting the right hand side to give a first order method which is the case (although not proved here).

Using the approximation in (14) yields the strong first order Runge-Kutta method,

$$\begin{aligned}\hat{X}_{t+\Delta t} &= \hat{X}_t + \mu(t, \hat{X}_t)\Delta t + \sigma(t, \hat{X}_t)\Delta W_t + \sigma(t, \hat{X}_t - X^*)\left((\Delta W_t)^2 - \Delta t\right)/(2\sqrt{\Delta t}) \\ X^* &= \hat{X}_t + \sigma(t, \hat{X}_t)\sqrt{\Delta t}\end{aligned}$$

9 Numerical Convergence Experiments

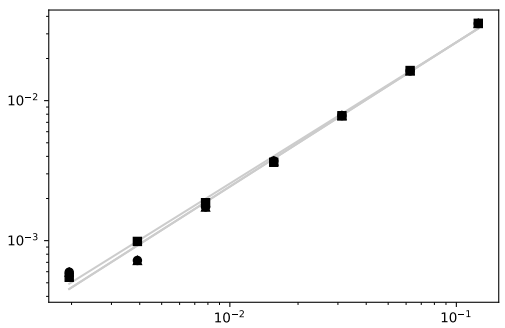
In this section we conduct a few numerical experiments to see if they agree with the theoretical results presented above.

We implement the Euler-Maruyama and Runge-Kutta methods described above in Python using Numpy. These functions take in the Brownian motion in increment form as suggested previously. Note the notation change where we have denoted μ and σ by a and b.

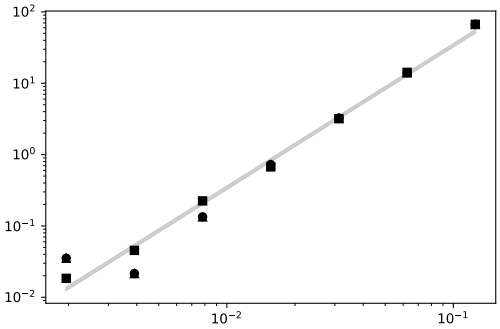
```
def euler_maruyama(a,b,x0,t,dWt):
    N = len(t)
    x = np.zeros((N,len(x0)))
    x[0] = x0
    for i in range(N-1):
        dt = t[i+1]-t[i]
        x[i+1] = x[i] + a(t[i],x[i])*dt + b(t[i],x[i])*dWt[i+1]
    return x
```

```
def runge_kutta(a,b,x0,t,dWt):
    N = len(t)
    x = np.zeros((N,len(x0)))
    x[0] = x0
    for i in range(N-1):
        dt = t[i+1]-t[i]
        x_hat = x[i]+b(t[i],x[i])*np.sqrt(dt)
        x[i+1] = x[i] + a(t[i],x[i])*dt + b(t[i],x[i])*dWt[i+1]
                    + (b(t[i],x_hat)-b(t[i],x[i]))*(dWt[i+1]**2-dt)
                       /(2*np.sqrt(dt))
    return x
```

Figure 2 shows the absolute difference between the mean and variance of the actual solution and the mean and variance of various numerical methods. All three methods tested match the expected orders of weak convergence for the mean and variance of the process.



(a) $\dfrac{\left|\mathbb{E}[X_T] - \mathbb{E}[\hat{X}_T]\right|}{\left|\mathbb{E}[X_T]\right|}$



(b) $\dfrac{\left|\mathbb{E}[(X_T - \mathbb{E}[X_T])^2] - \mathbb{E}[(\hat{X}_T - \mathbb{E}[X_T])^2]\right|}{\left|\mathbb{E}[(X_T - \mathbb{E}[X_T])^2]\right|}$

Figure 2: Statistics based on 2000 trajectories, where $T = 1$ and X_t is a OU process with $\theta = 1$, $\mu = 1$, and $\sigma = 1$ for various $\hat{X}_t$. Circles correspond to the Euler–Maruyama method at various mesh sizes, triangles correspond to the Runge-Kutta method at various mesh sizes, and squares correspond to Euler–Maruyama method with $\Delta W_t \in \{-\Delta t, \Delta t\}$.

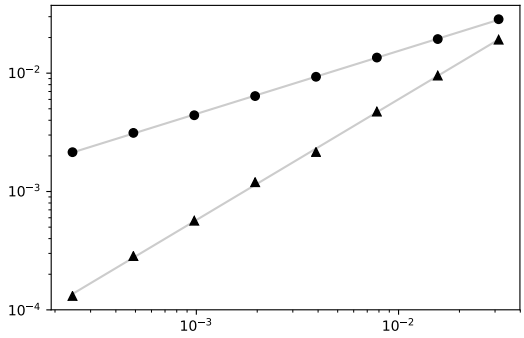


Figure 3: Values of $\mathbb{E}|X_T - \hat{X}_T|/\mathbb{E}|X_T|$ approximated using 1000 trajectories, where $T = 8$ and X_t is a Geometric Brownian motion with $\mu = 1$ and $\sigma = 1/2$ for various $\hat{X}_t$. Circles correspond to the Euler–Maruyama method at various mesh sizes, and triangles correspond to the Runge-Kutta method at various mesh sizes.

Note that the best fit lines in Figure 2a have slopes of 1.0308, 1.0308, and 1.0308 respectively. This is the expected weak convergence of order one.

Interestingly, the best fit lines in Figure 2b have slopes of 1.9982, 1.9982, and 1.9982 respectively. This seems to happen when adjusting the parameters of the processes as well as the number of trajectories used, etc. While it still satisfies the definition of weak convergence of order 1 it was unexpected.

We can now use our numerical solutions to generate trajectories of a Geometric Brownian motion. In particular, if for each trajectory we output the values of ΔW_t at each time point, the Brownian motion W_t can be computer. This means we can compare our computed results at time T to the exact solution at time T.

Figure 3 shows the results of this for the Euler-Maruayma and Runge-Kutat methods. Note that the best fit lines have slopes of 0.5321 and 1.0204 respectively. These are the order of strong convergence expected for both methods.

10 Conclusion

We introduce stochastic differential equations. In order to solve these numerically we introduced methods of generating Brownian motion on a discrete time mesh. We then introduced numerical SDE methods by approximating the solution using a (stochastic) Taylor expansion. Finally some numerical results were tested. For the most part the figures matched our expectations. However, the plot of Figure 2b shows that $\left|\mathbb{E}[(X_T - \mathbb{E}[X_T])^2] - \mathbb{E}[(\hat{X}_T - \mathbb{E}[X_T])^2]\right| / \left|\mathbb{E}[(X_T - \mathbb{E}[X_T])^2]\right|$ goes to zero like k^2 rather than k. At the moment it is not clear what affect the polynomial we chose, the processes, and the method have on this result. A more detailed look at this would be warranted.

References

[1] M. Lorig, "Introduction to probability and stochastic processes." PDF course notes for University of Washington AMATH 561/562, 2018.

[2] M. Holmes-Cerfon, "Numerically solving sdes," 2015.

[3] T. Sauer, "Numerical solution of stochastic differential equations in finance," 2006.

[4] P. E. Kloeden, *Numerical Solutions of Stochastic Differential Equations*. New York: Springer-Verlag Berlin Heidelberg, 1995.

[5] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Philadelphia, Pennsylvania: SIAM, 2007.

11 Code for Numerical Tests

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 23 08:07:06 2018

@author: tyler
"""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

'''
  solve SDE dX_t = a(t,X_t,t)dt + b(t,X_t)dW_t

    Parameters
    ----------
    mu : callable mu(t,X_t),
        t is scalar time and X_t is vector position
    sigma : callable sigma(t,X_t),
        where t is scalar time and X_t is vector position
    x0 : ndarray
        the initial position
    t : ndarray
        list of times at which to evaluate trajectory

    Returns
    -------
    x : ndarray
        positions of trajectory at each time in t
    dWt : incriments of Brownian Motion used
'''
#<start:em>
def euler_maruyama(a,b,x0,t,dWt):
    N = len(t)
    x = np.zeros((N,len(x0)))
    x[0] = x0
    for i in range(N-1):
        dt = t[i+1]-t[i]
        x[i+1] = x[i] + a(t[i],x[i])*dt + b(t[i],x[i])*dWt[i+1]
    return x
#<end:em>
#<start:rk>
def runge_kutta(a,b,x0,t,dWt):
    N = len(t)
    x = np.zeros((N,len(x0)))
    x[0] = x0
    for i in range(N-1):
        dt = t[i+1]-t[i]
        x_hat = x[i]+b(t[i],x[i])*np.sqrt(dt)
        x[i+1] = x[i] + a(t[i],x[i])*dt + b(t[i],x[i])*dWt[i+1]
                + (b(t[i],x_hat)-b(t[i],x[i]))*(dWt[i+1]**2-dt)/(2*np.sqrt(dt))
    return x
#<end:rk>

#%% OU Process

theta = 10
u = 1
s = 0.5

def mu(t,x):
    return theta*(u-x)

def sigma(t,x):
    return s

#%% Plot some trajectories
N = 251
T = 1
t = np.linspace(0,T,N)

x0 = [0]

num_trajectories = 10

plt.figure()
for i in range(num_trajectories):
    dWt = np.random.normal(0,1,size=(N))*np.sqrt(t[1])
    x_em = euler_maruyama(mu,sigma,x0,t,dWt)
    plt.plot(t,x_em,alpha=1/num_trajectories,color='k')
plt.savefig('img/OU_em_'+str(num_trajectories)+'.pdf')
```

```
plt.figure()
for i in range(num_trajectories):
    dWt = (-1+2*np.random.randint(2,size=(N)))*np.sqrt(t[1])
    x_em2 = euler_maruyama(mu,sigma,x0,t,dWt)
    plt.plot(t,x_em2,alpha=1/num_trajectories,color='k')
plt.savefig('img/OU_em_2_'+str(num_trajectories)+'.pdf')

plt.figure()
for i in range(num_trajectories):
    dWt = np.random.normal(0,1,size=(N))*np.sqrt(t[1])
    x_rk = runge_kutta(mu,sigma,x0,t,dWt)
    plt.plot(t,x_rk,alpha=1/num_trajectories,color='k')
plt.savefig('img/OU_rk_'+str(num_trajectories)+'.pdf')


#%%

theta = 2
u = 1
s = 1/100

def mu(t,x):
    return theta*(u-x)

def sigma(t,x):
    return s

Ns = 2**np.linspace(3,9,7)
#Ns = 2**np.linspace(2,7,6)
#Ns = 2**np.linspace(4,9,6)
#Ns = 2**np.linspace(5,12,4)
#Ns = 2**np.linspace(4,10,7)

expected_error_em = np.zeros(len(Ns))
expected_error_em2 = np.zeros(len(Ns))
expected_error_rk = np.zeros(len(Ns))

num_trajectories= 2000
T = 1

x_T_em = np.zeros((num_trajectories,len(Ns)))
x_T_em2 = np.zeros((num_trajectories,len(Ns)))
x_T_rk = np.zeros((num_trajectories,len(Ns)))

x0 = np.array([2])

for k,N in enumerate(Ns):
    print(N)
    t = np.linspace(0,T,N)



    for i in range(num_trajectories):
        dWt = np.random.normal(0,1,size=(int(N)))*np.sqrt(t[1])
        dWt[0] = 0

        x_em = euler_maruyama(mu,sigma,x0,t,dWt)
        x_T_em[i,k] = x_em[-1]

        x_rk = runge_kutta(mu,sigma,x0,t,dWt)
        x_T_rk[i,k] = x_rk[-1]

        dWt_discrete = (-1+2*np.random.randint(2,size=(int(N))))*np.sqrt(t[1])
        dWt_discrete[0] = 0

        x_em2 = euler_maruyama(mu,sigma,x0,t,dWt_discrete)
        x_T_em2[i,k] = x_em2[-1]

#%% First Moment

mean_x_T_true = (x0 - u)*np.exp(-theta*T)+u

mean_error_em = np.abs(np.mean(x_T_em,axis=0) - mean_x_T_true)/mean_x_T_true
mean_error_em2 = np.abs(np.mean(x_T_em2,axis=0) - mean_x_T_true)/mean_x_T_true
mean_error_rk = np.abs(np.mean(x_T_rk,axis=0) - mean_x_T_true)/mean_x_T_true


linear_fit_em = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(mean_error_em), 1))
linear_fit_em2 = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(mean_error_em2), 1))
linear_fit_rk = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(mean_error_rk), 1))

print(linear_fit_em)
print(linear_fit_em2)
print(linear_fit_rk)

plt.figure()
plt.yscale('log')
plt.xscale('log')
```

```
plt.plot(T/Ns,10**linear_fit_em(np.log10(T/Ns)), color='0.8')
plt.plot(T/Ns,10**linear_fit_em2(np.log10(T/Ns)), color='0.8')
plt.plot(T/Ns,10**linear_fit_rk(np.log10(T/Ns)), color='0.8')

plt.plot(T/Ns,mean_error_em, color='0', marker='o', Linestyle='None')
plt.plot(T/Ns,mean_error_em2, color='0', marker='s', Linestyle='None')
plt.plot(T/Ns,mean_error_rk, color='0', marker='^', Linestyle='None')

#%%
plt.savefig('img/weak_order_'+str(num_trajectories)+'.pdf')

np.savetxt('img/weak_order_em_'+str(num_trajectories)+'.txt',[linear_fit_em[1]],fmt='%.4f')
np.savetxt('img/weak_order_em2_'+str(num_trajectories)+'.txt',[linear_fit_em[1]],fmt='%.4f')
np.savetxt('img/weak_order_rk_'+str(num_trajectories)+'.txt',[linear_fit_rk[1]],fmt='%.4f')


#%% p-th Moment E[(x-E[x])^p]

p = 2
pmom_x_T_true = (s**p/(p*theta))*(1 - np.exp(-p*theta*T))

pmom_error_em = np.abs(np.mean((x_T_em-mean_x_T_true)**p,axis=0) - pmom_x_T_true)/pmom_x_T_true
pmom_error_em2 = np.abs(np.mean((x_T_em2-mean_x_T_true)**p,axis=0) - pmom_x_T_true)/pmom_x_T_true
pmom_error_rk = np.abs(np.mean((x_T_rk-mean_x_T_true)**p,axis=0) - pmom_x_T_true)/pmom_x_T_true


linear_fit_em = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(pmom_error_em), 1))
linear_fit_em2 = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(pmom_error_em2), 1))
linear_fit_rk = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(pmom_error_rk), 1))

print(linear_fit_em)
print(linear_fit_em2)
print(linear_fit_rk)

plt.figure()
plt.yscale('log')
plt.xscale('log')

plt.plot(T/Ns,10**linear_fit_em(np.log10(T/Ns)), color='0.8')
plt.plot(T/Ns,10**linear_fit_em2(np.log10(T/Ns)), color='0.8')
plt.plot(T/Ns,10**linear_fit_rk(np.log10(T/Ns)), color='0.8')

plt.plot(T/Ns,pmom_error_em, color='0', marker='o', Linestyle='None')
plt.plot(T/Ns,pmom_error_em2, color='0', marker='s', Linestyle='None')
plt.plot(T/Ns,pmom_error_rk, color='0', marker='^', Linestyle='None')

#%%
plt.savefig('img/weak_order_'+str(p)+'_mom_'+str(num_trajectories)+'.pdf')

np.savetxt('img/weak_order_em_'+str(p)+'_mom_'+str(num_trajectories)+'.txt',[linear_fit_em[1]],fmt='%.4f')
np.savetxt('img/weak_order_em2_'+str(p)+'_mom_'+str(num_trajectories)+'.txt',[linear_fit_em[1]],fmt='%.4f')
np.savetxt('img/weak_order_rk_'+str(p)+'_mom_'+str(num_trajectories)+'.txt',[linear_fit_rk[1]],fmt='%.4f')


#%% Geometric Brownian Motion
u = 1
s = 0.5

def mu(t,x):
    return u*x

def sigma(t,x):
    return s*x

def x_true_(x0,t,Wt):
    return x0*np.exp((u-s**2/2)*t+s*Wt)

#%% Plot some sample trajectories of Geometric Brownian Motion

N = 201
T = 1
t = np.linspace(0,T,N)

dWt = np.random.normal(0,1,size=(N))*np.sqrt(t[1])
dWt[0] = 0

x0 = [300]

x_true = x_true_(x0,t,np.cumsum(dWt))

x_em = euler_maruyama(mu,sigma,x0,t[::20],np.append([0],np.diff(np.cumsum(dWt)[::20])))

print(x_true[-1]-x_em[-1])

plt.figure()
plt.plot(t,x_true,color='.7')
plt.plot(t[::20],x_em,'o',Linestyle='None',color='k')
plt.savefig('img/GBM_true_vs_10.pdf')
```

```
x_rk = runge_kutta(mu,sigma,x0,t,dWt)

print(x_true[-1]-x_rk[-1])

plt.figure()
plt.plot(t,x_rk)
plt.plot(t,x_true)
plt.show()



#%%

Ns = 2**np.linspace(5,12,8)
expected_error_em = np.zeros(len(Ns))
expected_error_rk = np.zeros(len(Ns))

num_trajectories= 1000
T = 1

for k,N in enumerate(Ns):
    print(N)
    t = np.linspace(0,T,N)
    x0 = [300]

    error_em = np.zeros(num_trajectories)
    error_rk = np.zeros(num_trajectories)

    for i in range(num_trajectories):
        dWt = np.random.normal(0,1,size=(int(N)))*np.sqrt(t[1])
        dWt[0] = 0

        x_true = x_true_(x0,t,np.cumsum(dWt))

        x_em = euler_maruyama(mu,sigma,x0,t,dWt)
        error_em[i] = np.abs(x_true[-1] - x_em[-1])

        x_rk = runge_kutta(mu,sigma,x0,t,dWt)
        error_rk[i] = np.abs(x_true[-1] - x_rk[-1])

    expected_error_em[k] = np.mean(error_em)
    expected_error_rk[k] = np.mean(error_rk)

    plt.figure()
    plt.hist(error_em)
    plt.show()

    plt.figure()
    plt.hist(error_rk)
    plt.show()

#%%

expected_error_em /= x0[0]*np.exp(u*T)
expected_error_rk /= x0[0]*np.exp(u*T)

linear_fit_em = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(expected_error_em), 1))
linear_fit_rk = np.poly1d(np.polyfit(np.log10(T/Ns), np.log10(expected_error_rk), 1))

print(linear_fit_em)
print(linear_fit_rk)

plt.figure()
plt.yscale('log')
plt.xscale('log')

plt.plot(T/Ns,10**linear_fit_em(np.log10(T/Ns)), color='0.8')
plt.plot(T/Ns,10**linear_fit_rk(np.log10(T/Ns)), color='0.8')

plt.plot(T/Ns,expected_error_em, color='0', marker='o', Linestyle='None')
plt.plot(T/Ns,expected_error_rk, color='0', marker='^', Linestyle='None')



plt.savefig('img/strong_order_'+str(num_trajectories)+'.pdf')

np.savetxt('img/strong_order_em_'+str(num_trajectories)+'.txt',[linear_fit_em[1]],fmt='%.4f')
np.savetxt('img/strong_order_rk_'+str(num_trajectories)+'.txt',[linear_fit_rk[1]],fmt='%.4f')
```