Stochastic Data Processing and Simulation, Assignment A6

Sarah Al-khateeb

1 Introduction

In this lab, we were introduced to the stochastic process in both discrete and continuous time. A stochastic process is simply stochastic variables with time as a parameter. An example of a stochastic process in a discrete-time is white noise and ARMA, and in a continuous-time, we have the Brownian motion. A Brownian motion is a stochastic process in a continuous-time that has independently distributed increments. In this lab, we will try to simulate an approximation of a stochastic process based on a Brownian motion. The approximation will be made using a cheaper approximation scheme since computing the exponential function is very expensive, and we want an easier way to achieve good simulation. Furthermore, we will try to quantify the error using two types of error; strong and weak error, and measure the convergence in each case.

2 Assignment 6(i)

2.1 Problem

We will compute a sample path of a Brownian motion at different resolutions or different time grids using the same noise η generated using normally distributed random variables $\sim N(0, h)$.

2.2 Theory and implementation

As mentioned above, a Brownian motion is a stochastic process in a continuous-time, and it has independently distributed increments. Since it is hard to compute in a continuous-time and we want an easier way to perform an approximation or simulation, we use a time grid and do a partition of time with a finite discretization from 0 to t. To make this easy we use equidistant time discretization $t_n = n.h$ and all have same distance $h = \frac{1}{N}$, N is the number of discrete steps. If we partition using $N = 2^i$, we will get a nested sequence of grids or partitions, and doing this, it becomes easier to implement a simulation, and we simulate the value of the Brownian motion at these points.

$$w(1) = w(1) - w(t_1) + w(t_1) - w(0)$$

$w(1)$ is the Brownian motion at time 1 and according to the equation above, Brownian motion at rime 1 is the sum of two increments of Brownian motion at different times. So we can write Brownian motion at time 1 as a sum of small increments:

$$w(1) = \sum_{n=1}^{N} w(t_n) - w(t_{n-1})$$

All these increments are normally distributed with mean = 0 and variance = $h \sim N(0, h)$. Using the numpy library, we simulate η the noise using random.normal and we use standard deviation

$\sqrt{h}$ to obtain increments as we want them. Then we add the noise η (using cumsum) to the previous value of Brownian motion:

$$w(t_n) := w(t_{n-1}) + \eta$$

In order to see convergence, we simulate a Brownian path on different resolutions or time grids. Moreover, we use a fine grid to show how Brownian motion fluctuating. What we want here is to observe that at common grid points, all Brownian motions have the same value and so we generate different η, and we add this to get the path of Brownian motion. This allows us to construct out of the small increments the increments of the large Brownian motion.

2.3 Results and discussion

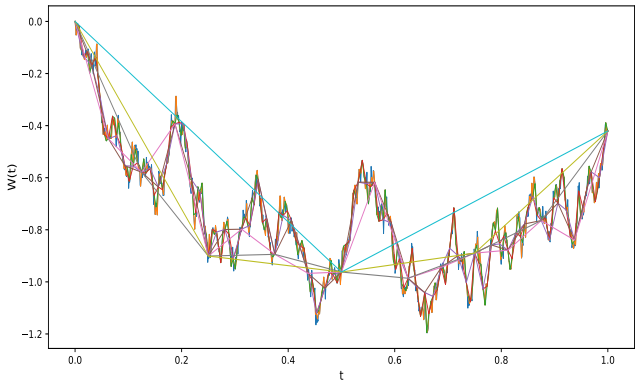

Figure 1: Sample path of a Brownian motion at different resolutions.

Figure 1 shows the sample path of a Brownian motion at different resolutions. Here we used 10 resolutions, and we can say that on common points, indeed, all Brownian motions have the same value (blue line, green line, ...). This is an exact simulation at all time points, which is good. Furthermore, We can notice that as we use finer resolutions, the paths become more jittery because the number of increments is increasing each will get an even chance of being negative or positive. Also, what shows more detail is that between every two neighboring endpoints of a path, there is one point of finer resolution.

3 Assignment 6(ii)

3.1 Problem

In this problem, we compute a sample path of X (a stochastic process) and sample paths of X_{hi} using the same noise η and then check if the approximation is similar to the stochastic process X.

3.2 Theory and implementation

Based on a Brownian motion $w(t)$ we want to simulate approximation of stochastic process X:

$$X(t) = exp((\mu - \frac{\sigma^2}{2})t + \sigma W(t))$$

Here, we are interested in X, and we want to simulate it without using the exponential function; the reason is that computing the exponential function is very expensive, and we would like to find an easier way to do the simulation. In this case, we use an approximation scheme :

$$X_h(t_n) = (1 + h\mu)X_h(t_{n-1}) + \sigma X_h(t_{n-1})(W(t_n) - W(t_{n-1}))$$

Where h is the time discretization parameter, in this problem, we want to see if this scheme looks similar to the stochastic process X (exponential function). If we implement the scheme for different time steps h, we will notice that the true solution and the approximation solution are not the same but not very different or far from each other, and these will be farther away in case of using a bad approximation scheme with large step size, but with smaller step size the performance will be better. When we simulate the sample path several times, we can check if there is a difference, which is the error between the two values at some time point t.

3.3 Results and discussion

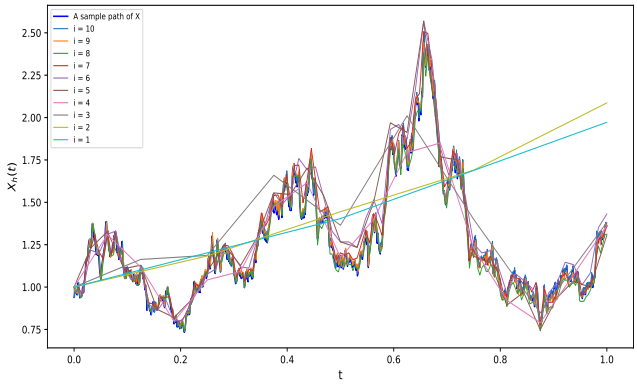

Figure 2: Sample path X along with Sample paths of X_h. $\mu = \sigma = 1, i = 10, N = 2^i, h = \frac{1}{N}$.

Figure 2 display plotting Brownian motion on a lot of discretization schemes (using i = 10 resolutions), and we can notice different quality comparing to **Figure 1** where we saw exact or similarity on all values. Here we can no longer see this exactness because we have different values, especially at the end we can see different results at the same time grid point, and this is the error we make making the approximation because our approximation of the exponential function is not exact, but it is just an approximation of the stochastic process X.
Moreover, we can say that as i (resolutions) increases, h_i gets smaller, and the approximations X_{h_i} apparently converges toward X. The plot is a bit similar to the Brownian motion in **Figure 1**, but we can notice enlarge for both negative and positive y, and this is expected from the exponential function used in the definition of the stochastic process X.

4 Assignment 6(iii)

4.1 Problem

We will estimate the strong error based on $M = 1000$ Monte Carlo simulations for all time parameters $(h_i, i = 1, 2, ..., 10)$.

4.2 Theory and implementation

The strong error or L^2 root mean squared $\mathbb{E}[(X(1)-X_h(1)^2]^{1/2}$ looks at the difference of the true solution and the approximated solution at time 1, we basically take the difference and squares it and then the sqrt to obtain a function space. We say that we have a convergence if we obtain smaller ans smaller step sizes, this error than converges to zero:

$$\lim_{h\to 0} \mathbb{E}[(X(1) - X_h(1)^2]^{1/2} = 0$$

But here, we are interested in how fast it converges to zero because we have to fix step size, and also we want to be sure that the approximation error is small.
We compute the strong error in a simulation using the known exact solution $X(1)$, and then we approximate using the approximation scheme $X_h(1)$; we do that for a lot of omegas (random variable). Since we don't know the expectation, in theory, we use the Monte Carlo method, and we do many samples ($M = 1000$), and then we average the results.

4.3 Results and discussion

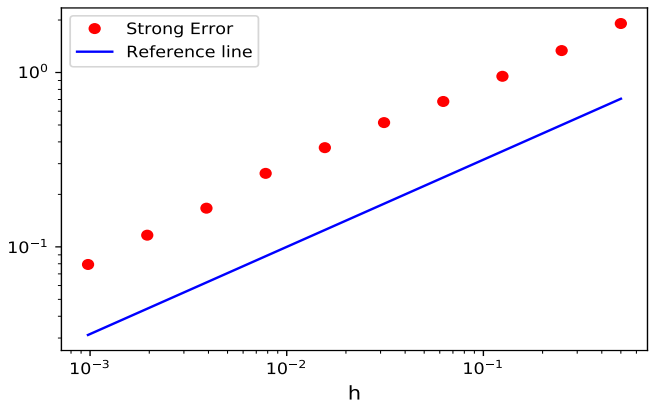

Figure 3: Strong Error Simulation using $M = 1000$, $\mu = \sigma = 1, i = 10, N = 2^i, h = \frac{1}{N}$.

Figure 3 shows the strong error simulation where the x-axis is the step size h, and the y-axis is the L^2 error. The plot is on the log-log scale, and we also have a reference slope to show how the convergence should look like or be. We can see that the errors (dots) for different h or step sizes are approximately parallel to the reference slope, which means that we can write the strong error as $Ch^{1/2}$ for some C and $1/2$ the order of convergence. If we make an assumption that X_h converges strongly, then it will converge at a rate of $1/2$.

5 Assignment 6(iv)

5.1 Problem

In this problem, we will simulate the weak error based on $M = 1000$ Monte Carlo simulations for all $(h_i, i = 1, 2, ..., 10)$ using a test function $\phi = Id$ to not get many errors that might disturb us.

5.2 Theory and implementation

In weak error, we use a class of test functions (in this problem, we will use the test function $\phi = \text{Id}$), and we compute the difference of the expectations of these test functions. In the weak

error, we use absolute value instead of the square (used in the strong error), and so instead of looking at the difference between the true solution and the approximation solution, first we compute the expectation, and then we average over the solution, and then we take the absolute value. We compute the error outside the expectation. For a suitable test function $\phi : \mathbb{R} \to \mathbb{R}$, the weak error is given by:

$$|\mathbb{E}[\phi(X(1))] - \mathbb{E}[\phi(X_h(1))]|$$

We can approximate the weak error using a Monte Carlo simulation by

$$|\mathbb{E}[\phi(X(1))] - \mathbb{E}[\phi(X_h(1))]| \approx \left| \mathbb{E}[\phi(X(1))] - \frac{1}{M} \sum_{m=1}^{M} \phi(X_h(1))^{(m)} \right|$$

for large M, where we know $\mathbb{E}[(X(1))] = \exp(\mu)$. Monte Carlo simulation of the error is sensitive to the value of M. The overall error is additive in the sense that the total error is $\frac{1}{\sqrt{M}} + h^{\gamma}$, γ is the discretization error.

5.3 Results and discussion

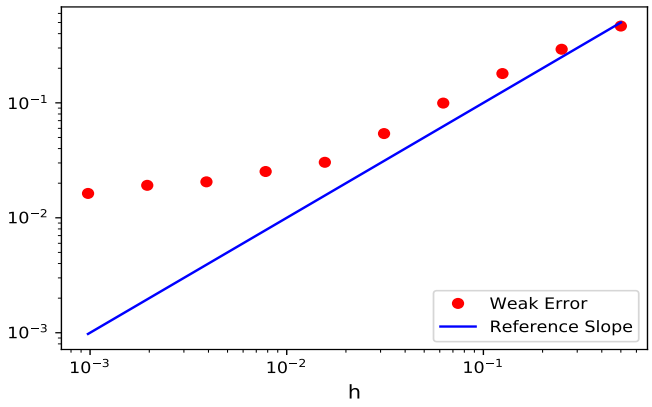

Figure 4: Weak Error Simulation using $M = 1000$, $\mu = \sigma = 1, i = 10, N = 2^i, h = \frac{1}{N}$.

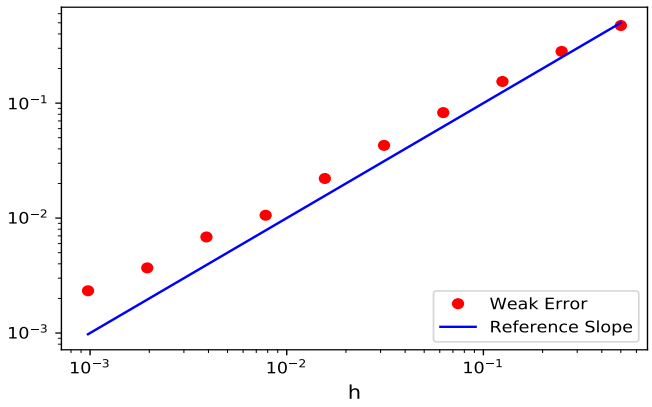

Figure 5: Weak Error Simulation using $M = 100000$, $\mu = \sigma = 1, i = 10, N = 2^i, h = \frac{1}{N}$.

In **Figure 4**, we can see the weak error using $M = 1000$, we can notice that when h increases, Monte Carlo estimates seems to follow the reference slope h, but when h is small, the weak error is larger. We can say about this is that when h is small, the total error is dominated by the Monte Carlo error. When M is small, we have big errors, and it is hard to see the rate, and when h is small, the rate of convergence less and no longer doing what it is supposed to do.

In **Figure 5**, we can see the weak error using $M = 100000$; we can say that when M increasing, the total error is decreasing. So, X_h converges weakly to X with a rate of 1 with enough Monte Carlo samples. This is in consonance with the general rule of thumb that the weak rate of convergence is twice the strong one.

6 Assignment 6(v)

6.1 Problem

In this problem, we will try different test functions that is a nonlinear transformation (ϕ) and observe the convergence comparing to the previous problem where we used $\phi = Id$.

6.2 Theory and implementation

With a suitable test function $\phi : \mathbb{R} \to \mathbb{R}$, the weak error is given by:

$$|\mathbb{E}[\phi(X(1))] - \mathbb{E}[\phi(X_h(1))]|$$

In this case, we will use a non linear transformation (polynomial). We will use the same transformation on $\mathbb{E}[\phi(X(1))]$ and $\mathbb{E}[\phi(X_h(1))]$. To do the transformation on $E[\phi(X(1))]$ we need to consider the distribution of $W(1)$ and $X(1)$.

As we know, the distribution of a Brownian motion is Gaussian $\sim N(0, 1)$ and since the $\mathbb{E}[(X(1))] = \exp(\mu)$ we have the $exp(Gaussian)$ that result in a Lognormal $\sim Lognormal(\mu, \sigma^2)$ distribution for $X(1)$.

The moment for the $Lognormal(\mu, \sigma^2)$ is:

$$\mathbb{E}X^n = e^{n\mu + \frac{n^2\sigma^2}{2}}$$

An important part is to keep track of parameters when we change or transform a distribution and so: In the stochastic process

$$X(t) = exp((\mu - \frac{\sigma^2}{2})t + \sigma W(t)) \longrightarrow \mu = \mu - \frac{\sigma^2}{2}$$

When we transform $\mathbb{E}[\phi(X(1))]$ which is a known quantity we use the formula:

$$\mathbb{E}X^n = e^{n\mu + \frac{n^2\sigma^2}{2}}, \;\; \mu = \mu - \frac{\sigma^2}{2} \;\; where \;\; \mu = 1, \sigma = 1$$

$$\longrightarrow \mathbb{E}X^n = e^{n(\mu - \frac{\sigma^2}{2}) + \frac{n^2\sigma^2}{2}} \;\; and \; the \; transformation \; of \; \mathbb{E}[X_h] \; is \; X_h^{(n)}$$

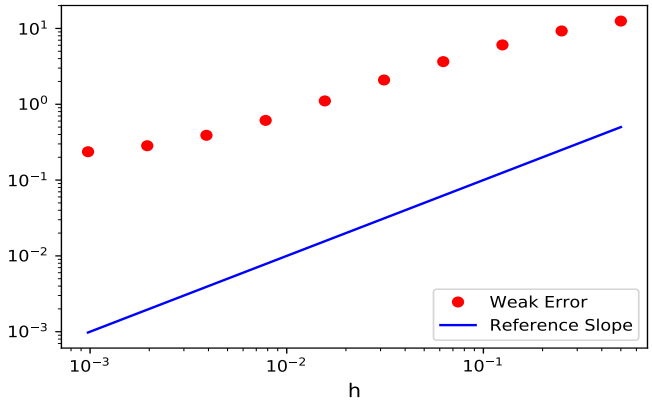

Figure 6: Weak Error Simulation with polynomial transformation of degree 2 using $M = 100000$.

6.3 Results and discussion

Figure 6 shows how the weak error converges similar to the strong error using a nonlinear transformation ϕ instead of the Lipschitz continuous. This test function allows us to compute exactly the expectations, but in our case, we use Monte Carlo simulation to approximate the expectation. In the previous section, we saw how using ϕ Lipschitz continuous, the weak error is bounded by the strong error. But now, using other test functions shows that this is not optimal. We can say that the strong convergence is a weak convergence but not the opposite.

Appendix - code

```
import numpy as np
import matplotlib.pyplot as plt
import random
from math import exp, sqrt


#Q1

i = 10 #all resolutions
N = 2**i #the number of discrete steps
h = 1/N #the variance of the increments

# the epsilon values
noise = np.random.normal(0, np.sqrt(h), N)

plt.figure(figsize=(12,6))
#calculate a sample path of a Brownian motion at all resolutions
for i in range(i, 0, -1):
    bm = np.cumsum(noise)  # calculate the brownian motion
    bm = np.insert(bm, 0, 0) # insert the initial condition W_0 = 0. for
        brownian motion

    # plot the Brownian motion
    xx = np.linspace(0, 1, 2**i+1)
    noise = noise[1::2] + noise[0::2]
    plt.plot(xx, bm, linewidth=0.8)

plt.xlabel('t', fontsize=12)
plt.ylabel('W(t)', fontsize=12)
plt.savefig('bm.pdf');


#Q2

mu = sigma = 1
i = 10 #all resolutions
N = 2**i #the number of discrete steps
h = 1/N #the variance of the increments

# the epsilon values
noise = np.random.normal(0, np.sqrt(h), N)

f = plt.figure(figsize=(12,6))

#a sample path of X
t = np.linspace(0, 1, N)
bm = np.cumsum(noise)
x = np.exp(((mu-(sigma**2/2))*t) + sigma*bm)
#plot the sample path
plt.plot(t, x, linewidth=1.5, c='b', label = 'A sample path of X')

#sample paths of X_hi
for i in range(i, 0, -1):
    bm = np.cumsum(noise)  # calculate the brownian motion; bm
```

```
    bm = np.insert(bm, 0, 0) # insert the initial condition W_0 = 0 for
        brownian motion
    x_h = np.zeros(len(bm))
    x_h[0] = 1

    # calculate the different motions for X_hi
    for t in range(1, len(bm)):
        x_h[t] = (1 + mu*2**(-i))*x_h[t-1] + sigma*x_h[t-1]*(bm[t]-bm[t-1])
    # plot each motion
    label = 'i = ' + str(i)
    noise = noise[1::2] + noise[0::2]
    t = np.linspace(0, 1, 2**i+1)
    plt.plot(t, x_h, linewidth=0.6, label = label)


plt.xlabel('t', fontsize=12)
plt.ylabel('$X_{h}(t)$', fontsize=12)
plt.legend(loc = 2, prop={'size': 8})
plt.savefig('sbm.pdf');


#Q3
M = 1000
mu = sigma = 1
i = 10 #all resolutions
N = 2**i #the number of discrete steps
h = 1/N #the variance of the increments
s = np.zeros(i) #strong error simulation

#Estimate the strong error with a Monte Carlo simulation based on M = 1000
for m in range(M):
    noise = np.random.normal(0, np.sqrt(h), N)
    for j in range(i, 0, -1):
        bm = np.cumsum(noise)  # calculate the brownian motion; bm
        bm = np.insert(bm, 0, 0) # insert the initial condition W_0 = 0 for
            brownian motion
        x_h = np.zeros(len(bm))
        x_h[0] = 1

        # calculate x_1 for the noise
        if j==10:
            x1 = np.exp((mu-sigma**2/2) + sigma*bm[-1])

        # calculate the different motions for X_hi
        for t in range(1, len(bm)):
            x_h[t] = (1 + mu*2**(-j))*x_h[t-1] + sigma*x_h[t-1]*(bm[t]-bm[t
                -1])
        s[j-1] += (x1 - x_h[-1])**2
        noise = noise[1::2] + noise[0::2]


# calculate strong error
h = [2**-(i+1) for i in range(i)]
strong_error = np.sqrt(s/M)

plt.loglog(h, strong_error, 'ro',label='Strong Error')
plt.loglog(h, np.sqrt(h),'b', label= 'Reference line')
plt.xlabel('h', fontsize=12)
```

```
#plt.suptitle('Strong Error Simulation')
plt.legend()
plt.savefig('strong_error.pdf');


#Q4

M = 1000
mu = sigma = 1
i = 10 #all resolutions
N = 2**i #the number of discrete steps
h = 1/N #the variance of the increments
s = np.zeros(i) #weak error simulation
exp_1 = np.exp(mu)

#Estimate the weak error with a Monte Carlo simulation based on M = 1000
for m in range(M):
    noise = np.random.normal(0, np.sqrt(h), N)

    for j in range(i, 0, -1):
        bm = np.cumsum(noise)  # calculate the brownian motion; bm
        bm = np.insert(bm, 0, 0) # insert the initial condition W_0 = 0 for
             brownian motion
        x_h = np.zeros(len(bm))
        x_h[0] = 1

        # calculate the different motions for X_hi
        for t in range(1, len(bm)):
            x_h[t] = (1 + mu*2**(-j))*x_h[t-1] + sigma*x_h[t-1]*(bm[t]-bm[t
                -1])

        # storing the sum of all points at time 1
        s[j-1] += x_h[-1]
        noise = noise[1::2] + noise[0::2]

# computing weak error
h = [2**-(i+1) for i in range(i)]
weak_error = np.abs(exp_1 - s/M)

plt.loglog(h, weak_error, 'ro', label = 'Weak Error')
plt.loglog(h, h, 'b', label = 'Reference Slope')
#plt.suptitle('Weak Error Simulation')
plt.xlabel('h', fontsize=12)
plt.legend(loc = 4)
plt.savefig('weak_error.pdf');




#Q4
#here we try larger M
M = 100000
mu = sigma = 1
i = 10 #all resolutions
N = 2**i #the number of discrete steps
h = 1/N #the variance of the increments
s = np.zeros(i) #weak error simulation
exp_1 = np.exp(mu)
```

```
#Estimate the weak error with a Monte Carlo simulation based on M = 1000
for m in range(M):
    noise = np.random.normal(0, np.sqrt(h), N)

    for j in range(i, 0, -1):
        bm = np.cumsum(noise)  # calculate the brownian motion; bm
        bm = np.insert(bm, 0, 0) # insert the initial condition W_0 = 0 for
            brownian motion
        x_h = np.zeros(len(bm))
        x_h[0] = 1

        # calculate the different motions for X_hi
        for t in range(1, len(bm)):
            x_h[t] = (1 + mu*2**(-j))*x_h[t-1] + sigma*x_h[t-1]*(bm[t]-bm[t
                -1])

        # storing the sum of all points at time 1
        s[j-1] += x_h[-1]
        noise = noise[1::2] + noise[0::2]

# computing weak error
h = [2**-(i+1) for i in range(i)]
weak_error = np.abs(exp_1 - s/M)

plt.loglog(h, weak_error, 'ro', label = 'Weak Error')
plt.loglog(h, h, 'b', label = 'Reference Slope')
#plt.suptitle('Weak Error Simulation')
plt.xlabel('h', fontsize=12)
plt.legend(loc = 4)
plt.savefig('weak_errorm.pdf');



#Q5

M = 100000
mu = sigma = 1
i = 10 #all resolutions
N = 2**i #the number of discrete steps
h = 1/N #the variance of the increments
s = np.zeros(i) #weak error simulation
#exp_1 = (np.exp(mu))

n = 2 # polynomial degree

#Estimate the weak error with a Monte Carlo simulation based on M = 1000
for m in range(M):
    noise = np.random.normal(0, np.sqrt(h), N)

    for j in range(i, 0, -1):
        bm = np.cumsum(noise)  # calculate the brownian motion; bm
        bm = np.insert(bm, 0, 0) # insert the initial condition W_0 = 0 for
            brownian motion
        x_h = np.zeros(len(bm))
        x_h[0] = 1

        # calculate the different motions for X_hi
```

```
        for t in range(1, len(bm)):
            x_h[t] = ((1 + mu*2**(-j))*x_h[t-1] + sigma*x_h[t-1]*(bm[t]-bm[
                t-1]))

        #applying a non linear transformation for x_h
        phi_x_h = (x_h**n)

        # storing the sum of all points at time 1
        s[j-1] += phi_x_h[-1]
        noise = noise[1::2] + noise[0::2]

# computing weak error
h = [2**-(i+1) for i in range(i)]
#non linear transformation for X_1
phi_X_1 = np.exp((n*(mu-sigma**2/2)) + ((n**2)*(sigma**2))/2)
weak_error = np.abs(phi_X_1 - s/M)

plt.loglog(h, weak_error, 'ro', label = 'Weak Error')
plt.loglog(h, h, 'b', label = 'Reference Slope')
#plt.suptitle('Weak Error Simulation')
plt.xlabel('h', fontsize=12)
plt.legend(loc = 4)
plt.savefig('phi_error.pdf');
```