# Geometric Brownian Motion

Mero Elmarassy

February 3, 2025

## 1 Introduction

In this numerical project I will study properties of the geometric Brownian motion introduced in the second problem set. My simulation will be described by the equation

$$\dot{y}(t) = fy(t) + \sqrt{2D}\eta(t) \tag{1}$$

where $y(t)$ represents the value of an investment at time $t$, $f$ is a constant describing the growth rate of $y$, $D$ is a constant describing the diffusion, and $\eta(t)$ is a unit variance Gaussian white noise.

## 2 Simulation

In this project I will use the Milstein stochastic integration method to simulate the trajectories of investments. I discretize time into intervals of length $dt$, and define $y_n$ to be the value of the investment at time $ndt$. $y_{n+1}$ is then recursively defined by

$$y_{n+1} = y_n + fy_n dt + \sqrt{2Ddt}y_n Z_n + Dy_n dt(Z_n^2 - 1) \tag{2}$$

where $Z_n$ is a $N(0, 1)$ random variable sampled for each iteration.

For consistency across my simulations, I will simulate the value of an investment over a period of one year, with $dt = 10^{-4}$ years. To obtain large enough samples to use for statistics, I will run simulations with $N = 100,000$.

## 3 Analysis of $< y(t) >$ and $< y^2(t) >$

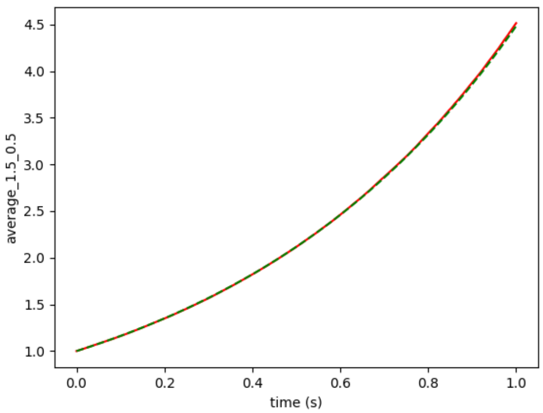
In the second problem set, I showed that for this SDE we have

$$< y(t) >= y_0 e^{ft}, \tag{3}$$

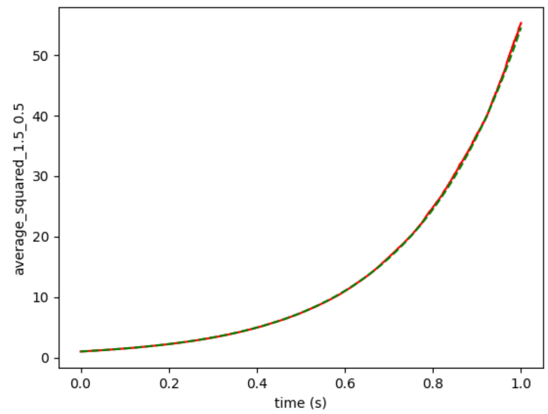
$$< y^2(t) >= y_0^2 e^{2(f+D)t} \tag{4}$$

where $y_0$ is the initial value of the investment. Here are some plots for different values of $f$ and $D$ showing these averages. The theoretical prediction is shown as a dashed green line while the simulated average is shown as a solid red line.

As expected, for positive values of $f$ we get an increasing average and for negative values of $f$ we have a decreasing average. For $f = 0$, we get a roughly constant average of 1, although due to the scaling of the axis the variations have been greatly exaggerated.

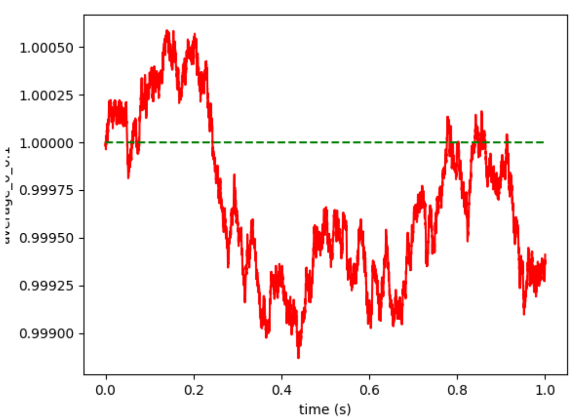



(a) $< y(t) >$

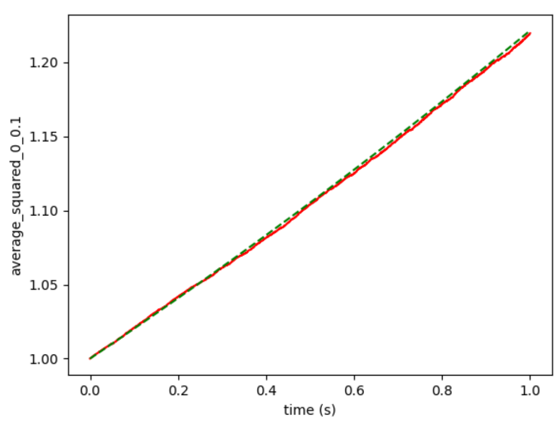



(b) $< y^2(t) >$

Figure 1: Plots for $f = 1.5,\ D = 0.5$

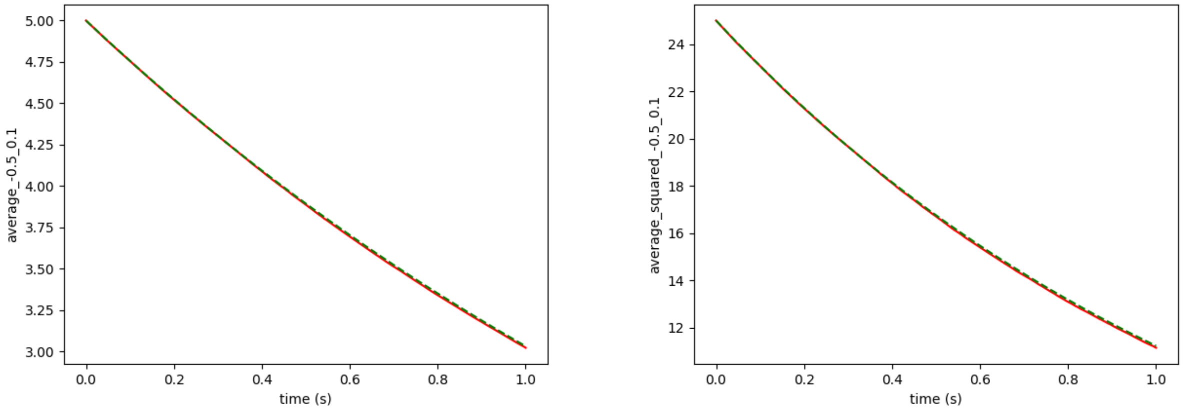



(a) $< y(t) >$



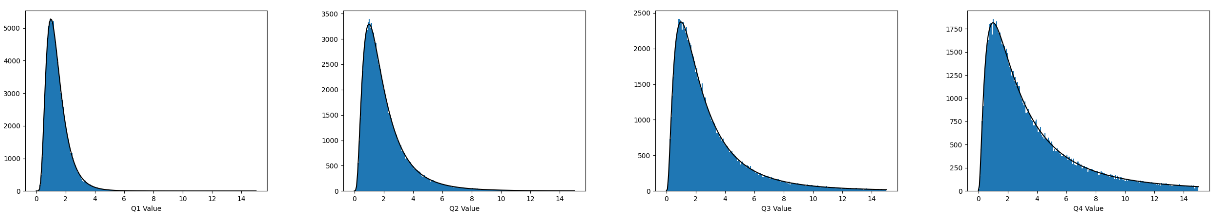

(b) $< y^2(t) >$

Figure 2: Plots for $f = 0,\ D = 1$




(a) $< y(t) >$ (b) $< y^2(t) >$

Figure 3: Plots for $f = -0.5$, $D = 0.1$




(a) Quarter 1 (b) Quarter 2 (c) Quarter 3 (d) Quarter 4

Figure 4: Distributions of $P(y, t)$ for $f = 1.5$, $D = 0.5$

# 4 Fokker-Planck Equation

In the second problem set, we used a change of variable to make our noise additive, solved the Fokker-Planck equation for the new variable, and converted back to $y$ to obtain

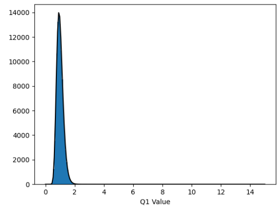
$$P(y,t) = \frac{1}{y\sqrt{4\pi Dt}} \exp\left[\frac{-(\ln y - \ln y_0)^2}{4Dt}\right] \tag{5}$$

which is valid when $f = D$. Following the same steps but taking $f \neq D$ leads to the more general equation

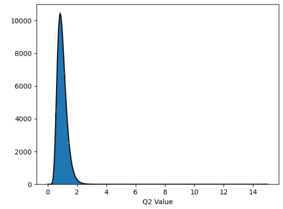
$$P(y,t) = \frac{1}{y\sqrt{4\pi Dt}} \exp\left[\frac{-(\ln y - \ln y_0 + (D-f)t)^2}{4Dt}\right] \tag{6}$$

I implemented a function to graph this function for specific values of $f$, $D$, $t$, and $y_0$. I then used the simulations to create distributions of the values at certain times to compare them with this model.

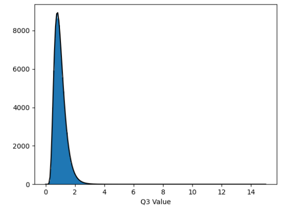
Since I am simulating the value of investments over a period of one year, I took the distributions by the end of each quarter. As seen in the plots, the simulation follows the expected distribution very closely. For positive $f$, the peak moves to the right while for negative $f$ it moves to the left. It remains roughly in the same spot for $f = 0$. Additionally, increasing the value of $D$ causes the distribution to widen more quickly. This is expected since we have more diffision.



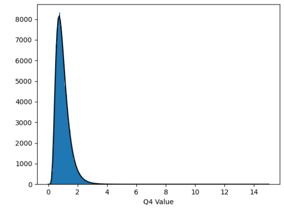
(a) Quarter 1



(b) Quarter 2

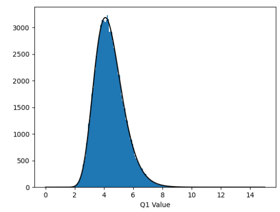


(c) Quarter 3

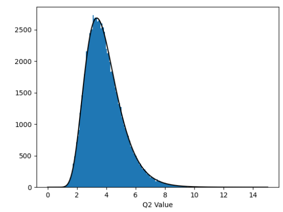


(d) Quarter 4

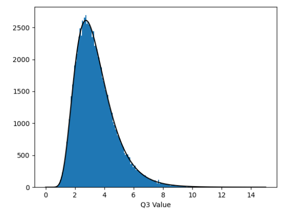
Figure 5: Distributions of $P(y,t)$ for $f = 0$, $D = 0.1$



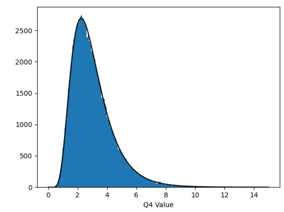
(a) Quarter 1



(b) Quarter 2



(c) Quarter 3



(d) Quarter 4

Figure 6: Distributions of $P(y,t)$ for $f = -0.5$, $D = 0.1$

# 5 Code

Here is the code I used to create this simulation and the plots. It contains two files, main.py and plot.py.

```
// main.py
import pickle

import numpy as np
import matplotlib.pyplot as plt

import plot


def save_object(obj, filename):
    with open(filename, 'wb') as outp: # Overwrites any existing file.
        pickle.dump(obj, outp, pickle.HIGHEST_PROTOCOL)



def simulate(n_simulations, dt, steps, growth_rate, dispersion, initial_value):
    """
    Simulates geometrical Brownian motion for N_simulations processes (ex: value of an
        investment).

    :param n_simulations: The number of processes to simulate
    :param dt: The time step of the simulation
    :param steps: Number of iterations to run each simulation
    :param growth_rate: Value of f in the geometrical Brownian motion SDE (roughly the
        rate of growth)
```

```
    :param dispersion: Value of D in the geometrical Brownian motion SDE (controls the amplitude of the noise term)
    :param initial_value: Starting value for each simulation
    :return result: Numpy array of shape (steps, n_simulations) describing value of each simulation at each time step
    """

    values = np.empty(shape=(int(steps), n_simulations))
    values[0] = initial_value
    for iteration in range(1, int(steps)):
        eta = np.random.normal(0, 1, size=n_simulations) #generate N(0, 1) random numbers
        values[iteration] = values[iteration - 1] #calculate values using Milstein method
        values[iteration] += growth_rate * dt * values[iteration - 1]
        values[iteration] += np.sqrt(2 * dispersion * dt) * values[iteration - 1] * eta
        values[iteration] += dispersion * values[iteration - 1] * (eta**2 - 1) * dt
    return values


def make_plots(result, steps, dt, growth_rate, dispersion, initial_value):

    time_axis = np.linspace(start=0, stop=steps*dt, num=int(steps))

    average = np.average(result, axis=1)
    average2 = np.average(result**2, axis=1)

    #plot average and squared average from simulation, along with their predicted graphs from theory
    plot.plot_data_and_exp_graph(average, time_axis, initial_value, growth_rate, f"average_{growth_rate}_{dispersion}")
    plot.plot_data_and_exp_graph(average2, time_axis, initial_value**2, 2*(growth_rate + dispersion), f"average_squared_{growth_rate}_{dispersion}")

    for i in range(1, 5):
        step = int(i * steps/4) - 1
        plot.plot_with_distribution(dispersion, step*dt, growth_rate, initial_value, f"histogram_{growth_rate}_{dispersion}_{i}", data=result[step], label=f'Q{i} Value')


n_simulations = 100000
dt = 1e-4
steps = 1e4
growth_rate = 0
dispersion = 0.1
initial_value = 1 #parameters to run the simulation
result = simulate(n_simulations, dt, steps, growth_rate, dispersion, initial_value)

make_plots(result, steps, dt, growth_rate, dispersion, initial_value) #make the plots
```

```
// plot.py
import matplotlib.pyplot as plt
import numpy as np
```

```
def plot_data(data, tdata, plot_name):
    plt.plot(tdata, data)
    plt.xlabel("time (s)")
    plt.ylabel(plot_name)
    plt.savefig('plots/' + plot_name + '.png')
    plt.show()

def plot_data_and_exp_graph(data, tdata, initial_value, factor, plot_name):
    plt.plot(tdata, data, color='red')
    y = initial_value * np.exp(factor * tdata)
    plt.plot(tdata, y, color='green', linestyle='dashed')
    plt.xlabel("time (s)")
    plt.ylabel(plot_name)
    plt.savefig('plots/' + plot_name + '.png')
    plt.show()


def plot_histogram(data, name, bins=100, range=(0, 15)):
    plt.hist(data, bins=bins, range=range)
    plt.xlabel(name)
    plt.savefig('plots/' + name)
    plt.show()


def plot_with_distribution(D, t, f, initial_condition, name, range=(1e-5, 15),
     data=None, bins=200, label=None):
    if range[0] <= 0:
        raise ValueError('lower bound on range must be positive')
    normalization = 1
    if data is not None:
        plt.hist(data, bins=bins, range=range)
        normalization = len(data) / bins * (range[1] - range[0])
    if label is not None:
        plt.xlabel(label)

    y = np.linspace(*range, num=bins)
    p = normalization / (np.sqrt(4 * np.pi * D * t) * y) * np.exp(-(np.log(y) -
        np.log(initial_condition) + (D-f)*t)**2 / (4 * D * t))

    plt.plot(y, p, color='k')
    plt.savefig('plots/' + name + '.png')
    plt.show()
```