C1 Example Class 2

Boris Bolliet

1st November 2024

We start with a warm-up. Then, we build a Python package that simulates Brownian motion.

1) Python warm-up

You are allowed to use any resources and packages you want to answer the following questions. You must provide an explanation for your code.

1.1 Calculate sin(0.1) using its taylor expansion to order 5.

1.2 Print the result as a descriptive string stating the order expanded to and value to 5 decimal places.

1.3 Construct a function which returns a list of prime numbers less than a given integer, N.

1.4 Construct a function which returns a list of the first N terms in the Recaman's sequence (see also here).

1.5 Compute a list of the numbers which appear in both lists when they are both N items long.

1.6 Create a list of all pairs of factors (as tuples) of 362880 using list comprehension.

1.7 Write a generator function for a random walk, step size 1, which is equally likely to go up or down. End the generator when you have total displacement of 10 steps (you will need a random number generator like random.randint(a,b) which gives a random integer between a and b inclusive, you will need to add the line import random at the top in order to use it).

2) Geometric Brownian motion simulations

Geometric Brownian motion is a stochastic process that grows multiplicatively. It follows the stochastic differential equation (SDE):

$$dY(t) = \mu Y(t)\,dt + \sigma Y(t)\,dB(t) \tag{2.1}$$

where $B(t)$ is called a *Brownian motion*, μ is the *drift* and σ is the *diffusion* coefficient. The solution to this equation is given by (derive it at home):

$$Y(t) = Y_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma B(t)\right) \tag{2.2}$$

where $Y_0 = Y(0)$. Create a Python package that contains a function to simulate the Geometric Brownian motion. Call the package `pygbm`. It must contain classes (base class and daughter classes) and methods and a command-line interface must be implemented. For instance, one could be able to run the following Python code and show a plot of a simulated path:

```python
from pygbm.gbm_simulator import GBMSimulator
import matplotlib.pyplot as plt

# Parameters for GBM
y0 = 1.0
mu = 0.05
sigma = 0.2
T = 1.0
N = 100

# Initialize simulator
simulator = GBMSimulator(y0, mu, sigma)

# Simulate path
t_values, y_values = simulator.simulate_path(T, N)

# Plot the simulated path
plt.plot(t_values, y_values, label="GBM Path")
plt.xlabel("Time")
plt.ylabel("Y(t)")
plt.title("Simulated Geometric Brownian Motion Path")
plt.legend()
plt.show()
```

For the command-line interface, one could be able to run something like:

```
pygbm simulate --y0 1.0 --mu 0.05 --sigma 0.2 --T 1.0 --N 100 --output gbm_plot.png
```

and get a plot of the simulated path as an output.

Optional extension: Assuming you based your code on the analytical solution, extend your package so it solves the SDE numerically using (i) the Euler-Maruyama method and (ii) the Milstein method. Compare the results with the analytical solution and discuss.