Programming exercise 1

(i) Install Python and Visual Code, and then the packages *numpy* and *matplotlib.pyplot* as it is explained on the website/ in Ilias.

(ii) Let $N = 1000$ denote your discretization number per time unit, and $T = 2$ your time horizon. Use the function *random.normal* from *numpy* to generate $N \cdot T$ normal distributed random numbers.

(iii) Generate an approximation of a realization of a Brownian motion $(W_t)_{t\in[0,T]}$ by starting at 0 and then summing up your normally distributed random variables. Make sure to choose the correct standard deviation in (ii).

(iv) Use your approximated Brownian motion from (iii) to approximate the solutions of the following explicitly solvable SDEs:

a)
$$\mathrm{d}X_t = \mu X_t \,\mathrm{d}t + \sigma X_t \,\mathrm{d}W_t, \quad t \in [0,T],$$
with $X_0 = x_0 = 1$, $\mu = 1$, $\sigma = 1$, which is solved by the *geometric Brownian motion*
$$X_t = x_0 e^{(\mu - \frac{\sigma^2}{2})t + \sigma W_t}, \quad t \in [0,T],$$
with drift μ, volatility σ and starting value x_0.

b)
$$\mathrm{d}Y_t = tY_t \,\mathrm{d}t + e^{\frac{t^2}{2}} \,\mathrm{d}W_t, \quad t \in [0,T],$$
with $Y_0 = y_0 = 1$, solved by
$$Y_t = e^{\frac{t^2}{2}}(y_0 + W_t), \quad t \in [0,T].$$

c)
$$\mathrm{d}Z_t = (2\mu Z_t + \sigma^2)\,\mathrm{d}t + 2\sigma\sqrt{Z_t}\,\mathrm{d}W_t, \quad t \in [0,T],$$
with $Z_0 = z_0 = \frac{1}{4}$, $\mu = 1$ and $\sigma = 1$, solved by
$$Z_t = \left(e^{\mu t}\sqrt{z_0} + \sigma \int_0^t e^{\mu(t-s)}\,\mathrm{d}W_s\right)^2, \quad t \in [0,T].$$
Approximate the stochastic integral appropriately by using your approximation of W.

Plot your approximations by using the functions *plot* and *show* from *matplotlib.pyplot*.

Programming exercise 2

Consider the Call option from Exercise 2.3. Note that, analogue to $t = 0$, the price formula for any $t \in [0,T]$ is
$$v(t,S_t) = V_t((S_t - K)^+) = (S_t - K)\Phi\left(\frac{S_t - K}{\sigma\sqrt{T-t}}\right) + \sigma\sqrt{T-t}\,\phi\left(\frac{S_t - K}{\sigma\sqrt{T-t}}\right).$$
In this exercise, we will calculate the *hedging strategy* $(\phi_t)_{t\in[0,T]}$ which *replicates* the Call option. The (continuous time) *delta-hedging strategy* looks as follows:

At every time $t \in [0,T]$, invest:

- $\phi_t^1 = \frac{\partial}{\partial x} v(t, S_t)$ units in the risky asset S_t
- $\phi_t^0 = v(t, S_t) - S_t \frac{\partial}{\partial x} v(t, S_t)$ units in the riskless asset B_t.

Let $T = 5$, $S_0 = 10$, $K = 9$ and $\sigma = 2$. Show graphically that the "Law of one price" holds. Therefore, plot $(v(t, S_t))_{t\in[0,T]}$ for one realization of $(S_t)_{t\in[0,T]}$ together with the value $(\phi_t^0 B_t + \phi_t^1 S_t)_{t\in[0,T]}$ of the replicating portfolio. Use $N = 10, 100, 1000$ discretization steps per time unit to graphically show that the values coincide for $N \to \infty$.

Hint: use from scipy.stats import norm *to import the functions* norm.pdf *and* norm.cdf *to calculate the density and distribution function of the standard normal distribution.*

Programming exercise 3

Consider a call option $(S_T^1 - K)^+$ in the one-dimensional Black-Scholes model at $t = 0$, with value function

$$C(T, S_0^1, K, r, \sigma) = S_0^1 \Phi(d_1) - K \exp(-rT)\Phi(d_2),$$

where

$$d_1 = \frac{1}{\sigma\sqrt{T}}\Big(\log\Big(\frac{S_0^1}{K} + (r + \frac{\sigma^2}{2})T\Big)\Big), \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{T},$$

and the parameters are initially given by $T = 5$, $S_0^1 = 2$, $K = 2$, $r = 0.05$ and $\sigma = 0.01$. Plot $C(T, S_0^1, K, r, \sigma)$ as a function in each of its parameters, by keeping the other parameters fixed and varying

(i) $T = 1, \ldots, 100$

(ii) $S_0 = 0, 0.1, 0.2, \ldots, 5$

(iii) $K = 0.1, 0.2, \ldots, 5$

(iv) $r = 0, 0.01, 0.02, \ldots, 1$

(v) $\sigma = 0, 0.01, 0.02, \ldots, 1.$

Programming exercise 4

In this exercise, you are asked to create a **3D plot** of the price of an **Up-and-out call barrier option**.

Creating three-dimensional plots in Python also works with *Matplotlib*. Therefore, we only need to import the *mplot3d* toolkit. The following example plots te function $f(x, y) = \sin(\sqrt{x^2 + y^2})$ and shows how to work with *mplot3d*, you can use it for the purpose of this exercise.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
```

```
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```

Use *mplot3d* and the pricing formula from Lemma 3.11 to plot the price of an Up-and-out call barrier option in the Black-Scholes-model, where you fix the interest rate $r = 0.01$, volatility $\sigma = 0.05$, strike $K = 60$, initial stock price $S_0 = 60$, and let the barrier $B \in \{70, 70.1, \dots, 90\}$ and the maturity $T \in \{0.01, 0.02, \dots, 5\}$ vary.

Programming exercise 5

In this exercise, you are asked to calculate the **historical 180-days volatilities** of the stock prices from *CocaCola, NVIDIA* and *Apple*. Therefore, you need to install the Python package *pandas* and *pandas-datareader* by running

```
pip install pandas
pip install datareader
```

in your Windows console. Then, you can run

```
import pandas_datareader as pdr
# Request data via Yahoo public API
data_CocaCola = pdr.get_data_yahoo('KO')
data_Nvidia = pdr.get_data_yahoo('NVDA')
data_Apple = pdr.get_data_yahoo('AAPL')
```

to load the historical 5-year prices of the three stocks from *Yahoo Finance*. Note that your data is stored in a *pandas-dataframe*. To get an overview, you can run

```
data_CocaCola.info
```

If you want to convert your data into a *numpy* array, you can run

```
np_CocaCola = data_CocaCola.to_numpy()
```

Use the data, to calculate the historical 180d volatilities, in the time period from 2021/06/28 until 2022/03/14, of CocaCola, NVIDIA and Apple. Use the daily opening prices. You can assume a year to have 253 trading days.

For comparison: according to *alphaquery.com*, the *historical 180-d volatilities* are: CocaCola 0.1754, NVIDIA 0.5771, Apple 0.2769.

Programming exercise 6

Consider the portfolio optimization problem from the lecture, with power utility preference function $U(x) = \frac{x^p}{p}$ with $p = \frac{1}{2}$. Further, fix $T = 5$, $S_0^1 = S_0^0 = 1$, $r = 0,03$, $\mu = 0,06$, $\sigma = 0,5$ and $V_0(\pi) = 1$.

(i) 3D-plot the solution of the HJB-equation $w(t, x)$ defined by (5.16) for $t \in [0, T]$, $x \in [0, 10]$.

(ii) Validate numerically that the strategy $\pi^* = \frac{(\mu - r)}{\sigma^2(1-p)}$ is (locally) optimal. Therefore, consider $t \in \{0, 1, \dots, T-1\}$, on each time step t do $N_{MC} = 100.000$ simulations $S^1_{T-t,i}$, $i = 1, \dots, N_{MC}$, and calculate

$$\mathbb{E}_{MC}[U(V_T^{\pi^*,t,V_0})] := \sum_{i=1}^{N_{MC}} U(V_{T-t}(S^1_{T-t,i})).$$

Do the same with $\pi^* + \epsilon$ and $\pi^* - \epsilon$ for $\epsilon = 0.1$. Then, plot your calculated values together with the optimal value function $w(T - t, V_0(\pi^*))$, $t \in \{0, 1, \dots, T-1\}$.

Note: on each simulation step, you do not need the explicit path of the price process $(S_t)_{t\in[0,T]}$, but only the terminal value. Therefore, you also only need to simulate the terminal value of the driving Brownian motion, which reduces the (nevertheless high) total number of simulations that you need.