Programming exercise 1

Install Visual Studio Code (or any other Editor, if you are an experienced coder and have other preferences) and Python. Therefore, use this useful beginner's Python guide:

`https://github.com/aferdina/Thesis_Onboarding`

Note that it is originally written as an introduction for students writing their Bachelor thesis. For now, you only need the subsection "Installing Python".

Programming exercise 2

(i) Install the package *numpy*, by running (after installing python)

```
pip install numpy pandas matplotlib      (MacOS)
py -m pip install numpy pandas matplotlib    (Windows).
```

in the console of your computer. Then, create a Python file and import the package *numpy* using the expression `import numpy as np`.

(ii) Use `rng = np.random.default_rng()` to create a random generator. Generate $N = 100$ realizations each of a random variable that is discrete uniformly distributed on $[1, 10]$, standard normally distributed and exponentially distributed with parameter $\lambda = 3$.

(iii) Consider an online shop for COVID-19 tests which currently has N tests in stock. Each consumer's demand is discrete uniformly distributed on $[1, 10]$.
Write a function `shop_simulator` in Python that simulates, for a given number N of tests in stock, after how many consumers the tests are sold out (if the demand of the last consumer can not be fully supplied, he still counts), and prints out this number.
For example, the output of `shop_simulator(1000)` could be

```
"The 1000 tests were sold out after 176 consumers."
```

If you are not comfortable with functions in Python yet, solve this exercise for the fixed value $N = 1000$.

Programming exercise 3

Let $B_1 \sim \mathcal{N}(\mu_1, \sigma_1)$, $B_2 \sim \mathcal{N}(\mu_2, \sigma_2)$ be independent, where $\mu_1, \mu_2, \sigma_1, \sigma_2 \in \mathbb{R}$ with $\sigma_1, \sigma_2 > 0$, and define

$$X_1 := B_1 \qquad \text{and} \qquad X_2 := \rho B_1 + (1 - \rho) B_2$$

for some $\rho \in [0, 1]$.

(i) Using your knowledge from Stochastik 2, determine the distribution of $\mathbb{E}[X_1 + X_2 | X_1]$.

(ii) Validate your result from (i) numerically. Therefore, for $\mu_1 = 5, \mu_2 = 3, \sigma_1 = 2, \sigma_2 = 1, \rho = 0.2$, do $N = 10^5$ simulations of X_1, and for each given realization, simulate $M = 10^4$ realizations of X_2. Then, you get N realizations of $\mathbb{E}[X_1 + X_2 | X_1]$. Plot them in a histogram with a suitable number of boxes, and draw the density function of your distribution from (i) in that plot.

To plot, use the package `matplotlib`, and to draw the density function, the package `scipy`. Install both packages via your console, and import them by

```
import matplotlib.pyplot as plt
from scipy.stats import norm
```

Use the functions `plt.hist(...,bins = ?, density = TRUE)` for the histogram, and `norm.pdf(..., loc = ?, scale = ?)` for the density function.

Programming exercise 4

Consider the *Random Walk martingale* from Example 2.28.(i), with

$$\mathbb{P}(X_i = 1) = \mathbb{P}(X_i = -1) = \frac{1}{2}$$

for $i = 1, \ldots, N$ with $N = 10^4$. Simulate $M = 10^5$ realizations of Z_N, and show visually that Z_N converges (more precisely: would converge, if we let the stepsize go to zero, which is fixed at 1 in our setup) against a $\mathcal{N}(0, \sqrt{N})$-distributed random variable by plotting your realizations of Z_N together with the corresponding normal density function.

Programming exercise 5

Consider the *one-period binomial model* from Subsection 3.2 in the lecture notes, with given parameters

$$S_0 = 5,\ u = 3/2,\ d = 2/3,\ R = 1.05,$$

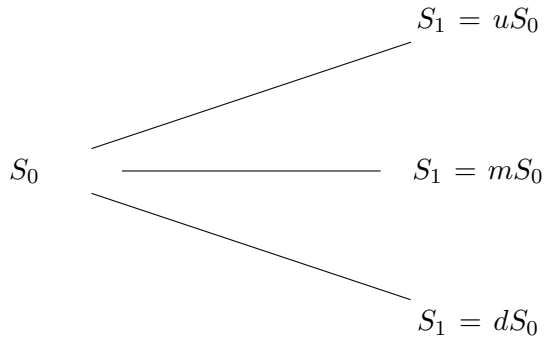
and consider a European Call option (hence, $f(S_1) = (S_1 - K)^+$).

Use the function `linalg.solve` from the package `numpy`, to write a function that uses S_0, u, d, R, f, K as input variables, solves the system of linear equations that we solved to find line (3.3) in the lecture notes, and then returns V_0, α and β of the replicating portfolio.

Use your function, to calculate and then plot the initial values needed to replicate European Call options given the above parameters and for $K = 0, 0.1, 0.2, \ldots, 9.9, 10$. How can you explain the two edges in the graph?

Programming exercise 6

Consider the *one-step-trinomial model*, that is given by the following tree:



Here, $u > m > d > 0$ are chosen such that $u > R > d$ holds. Then, the market is arbitrage-free. But, in general, not every claim is replicable, since one needs to solve a system of 3 equations with only 2 unknowns to replicate a claim with payoff $f(S_1)$:

$$\begin{aligned} &(i) \quad f(uS_0) = \beta + \alpha u S_0, \\ &(ii) \quad f(mS_0) = \beta + \alpha m S_0, \\ &(iii) \quad f(dS_0) = \beta + \alpha d S_0. \end{aligned}$$

Therefore, a new concept of pricing is needed, which is provided by the so called **superhedging price**

$$\Pi_{\sup}(f) := \inf\{V_0(\phi)\colon \exists \phi \text{ s.t. } V_1(\phi) \geq f(S_1) \text{ a.s.}\},$$

that gives the smallest amount of initial capital which is needed to have a payout at least as high as $f(S_1)$. To find $\Pi_{\sup}(f)$, one thus needs to find

$$\min_{\phi} V_0(\phi),$$

under the constraints $(i) - (iii)$ on $\phi = (\beta, \alpha)$, with "$\leq$" instead of "$=$".

Use the function `minimize` from the package `scipy.optimize` (`from scipy.optimize import minimize`) that solves a minimization problem under constraints (see `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html` for the user guide), to write a function which takes S_0, u, m, d, R, f, K as input and returns $\Pi_{\sup}(f)$ in the *one-step-trinomial model*, where f is the payoff of some option with strike K. You might need to work with so-called Python Lambda functions (see `https://realpython.com/python-lambda/`).

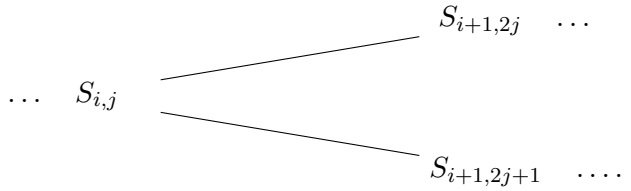
Apply your function to $S_0 = 5$, $u = \frac{3}{2}$, $m = \frac{7}{6}$, $d = \frac{2}{3}$, $R = 1.05$, $f = (S_1 - K)^+$, $K = 3$.

Programming exercise 7

Consider the *n-step-binomial model* $(B_t, S_t)_{t=0}^n$, where $B_t = R^t$ for some $R > 0$ and S_t is given by the price matrix

$$S = \begin{bmatrix} S_{0,0} & 0 & & & & \dots & 0 \\ S_{1,0} & S_{1,1} & 0 & & & \dots & 0 \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ S_{n,0} & S_{n,1} & S_{n,2} & S_{n,3} & S_{n,4} & \dots & S_{n,2^n-1} \end{bmatrix},$$

where row $i \in \{0, \dots, n\}$ represents the time steps, and $S_{i,j}$ with $j \in \{0, \dots, 2^i - 1\}$ are the possible prices at time i, such that $S_{i,j}$ can lead to either $S_{i+1,2j}$ or $S_{i+1,2j+1}$ in time $i+1$:



Write a function, which only needs R and S as input, first checks if the market $(B_t, S_t)_{t=0}^n$ is arbitrage-free, and if it is, calculates the EMM $\mathbb{Q}$, i.e. returns a (quadratic) matrix of the form

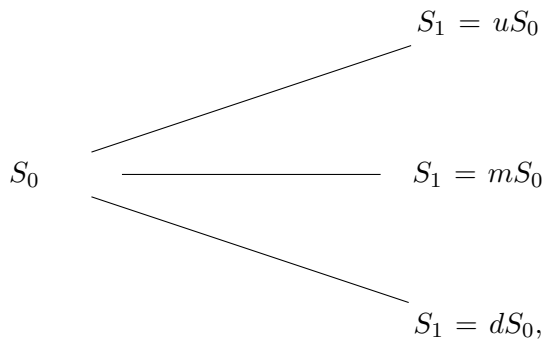
$$Q = \begin{bmatrix} q_{0,0} & 0 & & & \dots & 0 \\ q_{1,0} & q_{1,1} & 0 & & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ q_{n-1,0} & q_{n-1,1} & & & \dots & q_{n-1,n-1} \end{bmatrix},$$

Your function should work for general $n \in \mathbb{N}$. Test it for appropriate price matrices S.

Note: to be consistent with Python, we numerate matrix column/rows starting with 0 *here.*

Programming exercise 8

In this exercise, we are going to determine the set of equivalent martingale measures in the *one-step-trinomial model*, where the risky asset is given by the following tree:



where $u > m > d > 0$ are chosen such that $u > R > d$ holds. To find a $q = (q_1, q_2, q_3) \in \mathcal{M}^*$, one has to solve

$$\begin{aligned} &(i) \quad q_1 + q_2 + q_3 = 1 \\ &(ii) \quad q_1 u S_0 + q_2 m S_0 + q_3 d S_0 = R S_0, \end{aligned}$$

under the constraint $q_1, q_2, q_3 \in (0, 1)$, which is a system of 2 inequalities for 3 unknowns, and hence has infinitely many solutions. To find the set of solutions, use the following algorithm to solve an under determined system of equations:

(i) write the equatios in matrix form $A \cdot q = b$,

(ii) find the QR-decomposition $A^T = Q \cdot R$, where $R = \binom{R_1}{0}$ for some quadratic, upper-triangle matrix R_1, and Q is an orthogonal matrix,

(iii) compute $\hat{b} = (R_1^T)^{-1} \cdot b$,

(iv) get one solution $\hat{q}$ by setting $\hat{q} = Q \cdot \binom{\hat{b}}{0}$,

(v) find a non-zero solution x to $A \cdot x = 0$. Then, every $q = \hat{q} + \lambda x$, $\lambda \in \mathbb{R}$, which fulfills the non-negativity constraint, is also a solution. This describes our set of EMMs.

Implement the above algorithm, and $3d$-plot the set of EMMs for the *one-step-trinomial model* for $S_0 = 5$,$u = 3/2$, $d = 2/3$, $R = 1.05$.

The following might be helpful:

- `https://numpy.org/doc/stable/reference/generated/numpy.matmul.html`
- `https://numpy.org/doc/stable/reference/generated/numpy.linalg.qr.html`
- `https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html`
- `https://numpy.org/doc/stable/reference/generated/numpy.column_stack.html`
- Look at the following example, to learn how to $3d$-plot in Python using the package `mpl_toolkits`.

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

fig = plt.figure()
ax = plt.axes(projection='3d')

ax = plt.axes(projection='3d')

# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')
plt.show()
```

Programming exercise 9

Advanced programming difficulty

Use the package `networkx` (see `https://networkx.org/documentation/stable/tutorial.html`) to plot nice graphs of all possible paths of

(i) a stock price,

(ii) a price of a call option on (i),

in an N-period binomial model, where in each time step, the stock can go either up by a multiplicative factor u or down by d. On each node of your plots, you should write the actual price of the financial derivative. If you e.g. plot the stock with $S_0 = 7$, $u = 1.1$, $d = 0.9$, $N = 2$, your graph should look like Figure 1.

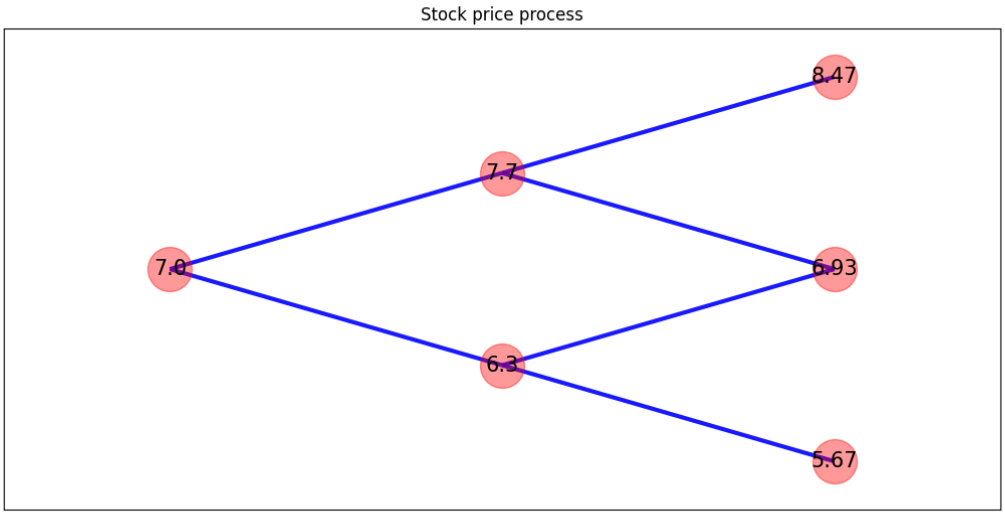


Figure 1: Stock price process

Moreover, calculate the prices of a European call option on each node, and plot the same picture for the option price process.

Programming exercise 10

Advanced programming difficulty

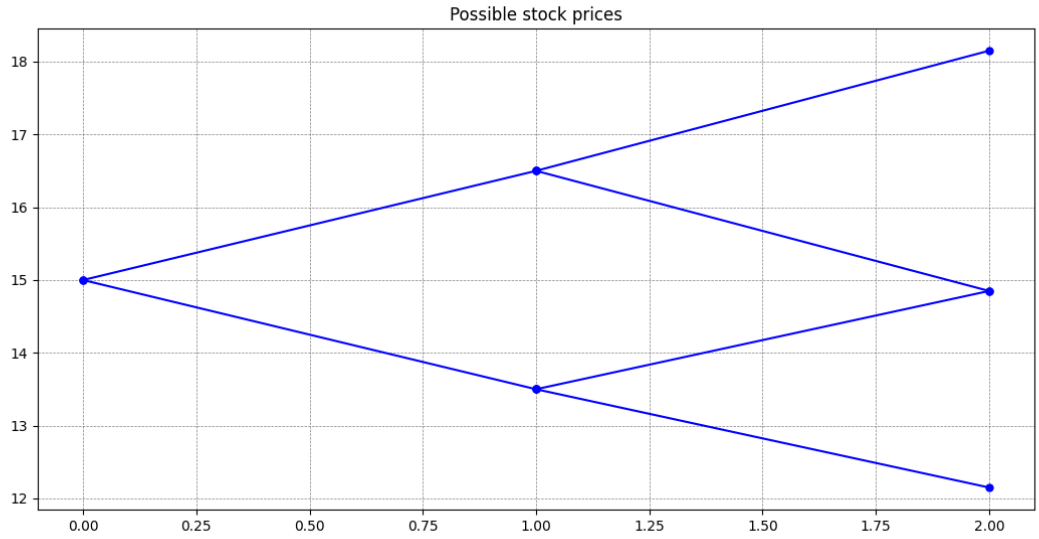
Use the package `networkx` (see `https://networkx.org/documentation/stable/tutorial.html`) to plot nice grids of all possible paths of

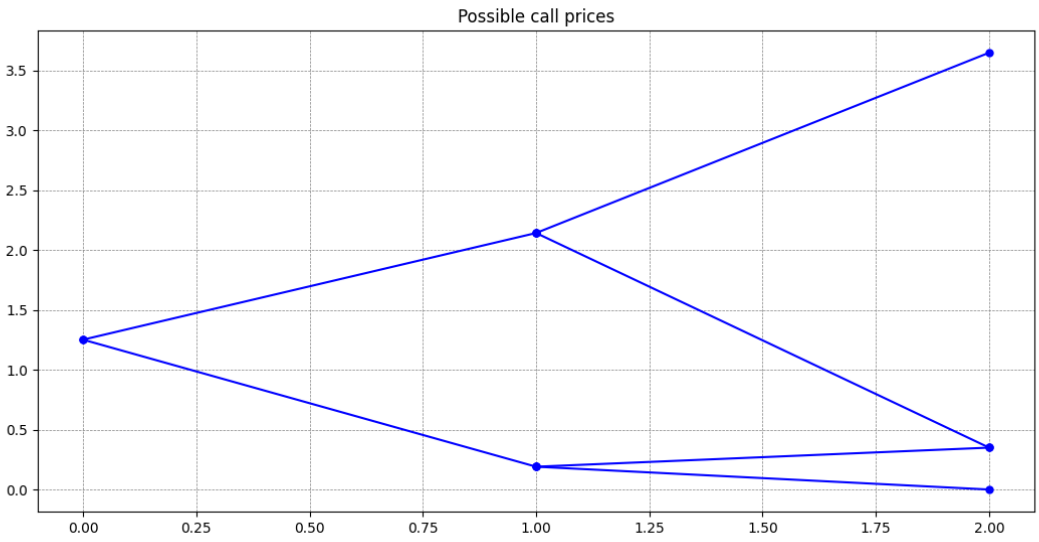
(i) a stock price,

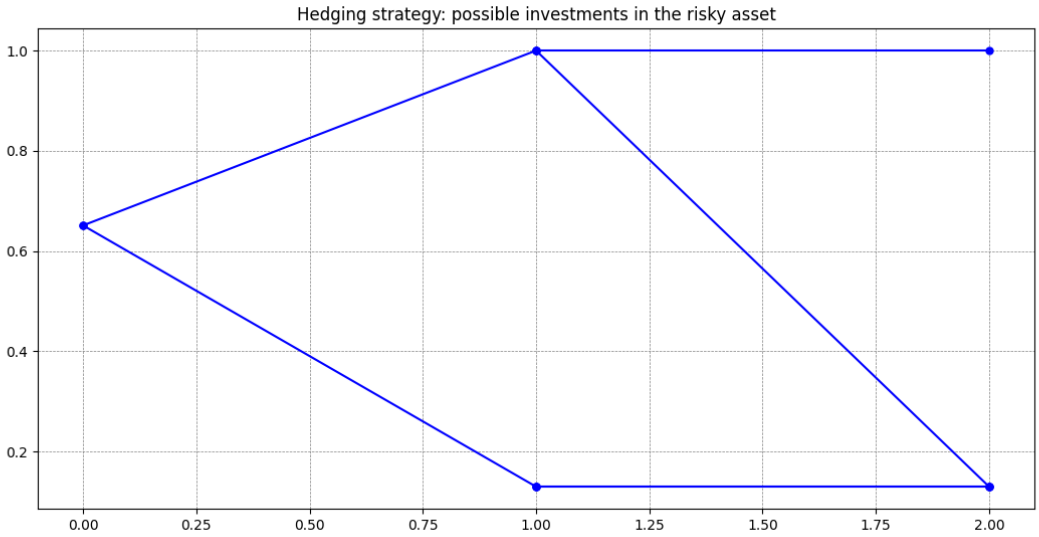
(ii) a price of a call option on (i),

(iii) a replicating strategy of (ii),

in an N-period binomial model, where in each time step, the stock can go either up by a multiplicative factor u or down by d. If you e.g. use $S_0 = 15$, $K = 14.5$, $u = 1.1$, $d = 0.9$, $N = 2$, your graphs should look like that:







Programming exercise 11

Use the formula from Exercise 11.2 (a) to write a recursive function (a function that references itself) to calculate the arbitrage-free price $\pi_0^A(P)$ and $\pi_0^A(C)$ of American-styled put and call options in an N-step binomial model. Adapt the formula to also work with interest rate $r > 0$. Note that the running time of recursive functions can get very large quickly due to the large computational effort.

Use your function to calculate the arbitrage-free prices of American put and call options for $N = 10$, $S_0 = 5$, $K = 5$, $u = 1.2$, $d = 1/u$ and $R := 1 + r = 1.02$.

Programming exercise 12

Consider the one-step binomial model from Exercise 12.1, and validate numerically that a^* given in (**??**) is optimal. Therefore, choose a meaningful (discretized) neighborhood A of a^*, do for every $a \in A$ a Monte–Carlo simulation to estimate $\mathbb{E}[U(V_1(\varphi_a))]$, where $\varphi_a = (b, a)$ and b is chosen such that $b + a = x_0$ and plot these values, to see that its maximum is attained at a^*.

Hint: In a Monte–Carlo simulation, you simulate N realizations x_i of a random variable X, to get the Monte–Carlo estimator $\hat{X}_{MC} := \sum_{i=1}^{N} x_i$ for $\mathbb{E}[X]$.

Programming exercise 13

Show the statement of Theorem 8.2 numerically. Therefore, set $S_0 = 200$, $T = 10$, $r = 0.05$, $\sigma = 0.5$, calculate for $K \in \{5, 10, \ldots, 250\}$

(i) for $n \in \{2, 5, 10, 100\}$ the n-step binomial model prices $C_0^{(n)}$ of a standard European call option with parameters r_n, u_n, d_n chosen as in Section 8.1 of the lecture notes,

(ii) the Black–Scholes call price C_0^{BS} given by the formula in Theorem 8.2,

and plot the 5 curves from (i) and (ii) to obtain that $\lim_{n\to\infty} C_0^{(n)} = C_0^{BS}$.