MONIN Louis

FinTech

Simulation de l'évolution de l'actif

Mme. Kortchemski

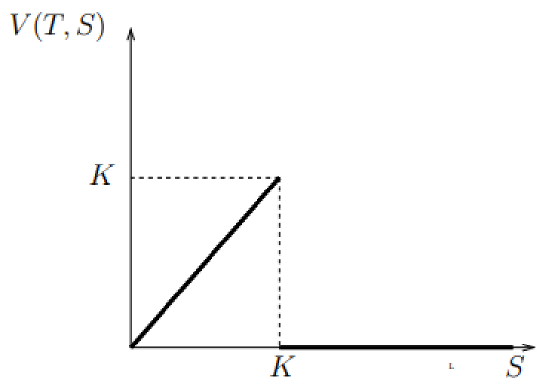
2024-2025

Sommaire

Problème 1 : Résolution par la méthode aux Différences Finies de l'équation de Black et Scholes pour l'option ASSET or NOTHING

La fonction Pay-off de l' option ASSET or NOTHING est montré sur la figure, elle est aussi donnée par la formule suivante :

$$\text{Pay-off-Asset}(S, K) = \begin{cases} S, & S < K \\ 0, & S \geq K \end{cases}$$

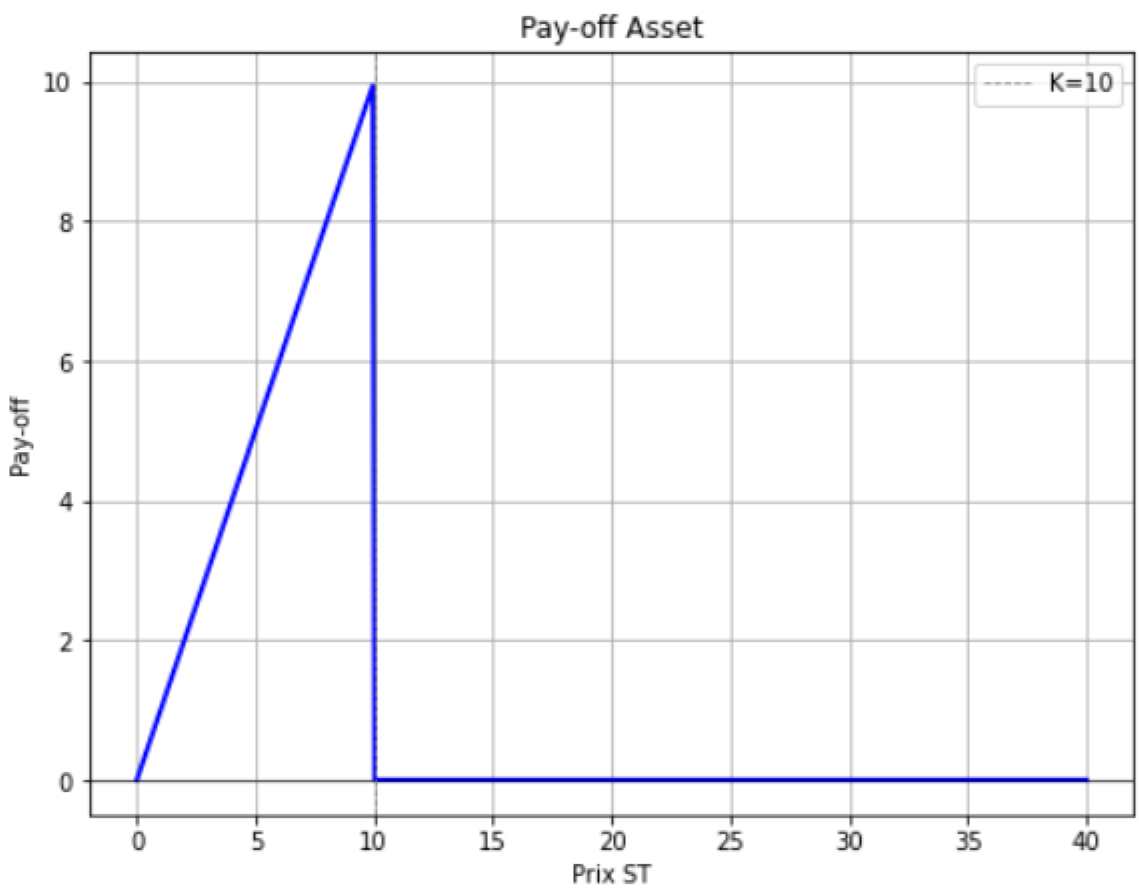


Code pour afficher la fonction Pay-off de l'option ASSET or NOTHING

```
# La fonction Pay-Off de l'option ASSET or NOTHING

import numpy as np
import matplotlib.pyplot as plt

def pay_off_asset(S, K):
    payoff = np.piecewise(S,
                    [S < K, S >= K],
                    [lambda S: S, 0])
    return payoff

# Paramètres
K = 10  # Strike price, je fixe 10 comme on a l'habitude de le faire en classe
S = np.linspace(0, 40, 500)  # Prix du sous-jacent ST entre 0 et 40, pareil ce sont des valeurs que je choisis

# Calcul du Pay-off
payoff = pay_off_asset(S, K)

# Tracé du graphique
plt.figure(figsize=(8, 6))
```

```
plt.plot(S, payoff, color='blue', linewidth=2)
plt.title('Pay-off Asset')
plt.xlabel('Prix ST')
plt.ylabel('Pay-off')
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(K, color='gray', linestyle='--', linewidth=0.8, label=f'K={K}')
plt.legend()
plt.grid()
plt.show()
```

Graphe de la function de Pay-Off



Partie I : Conditions aux limites de Dirichlet

L'équation de Black-Scholes avec la condition finale et les conditions aux limites de Dirichlet s'écrit sous la forme :

$$\begin{cases} \frac{\partial V}{\partial t} - rV + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = 0 \\ V(t = T, S) = \text{ Pay-off-Asset(S,K)} \\ V(t, S = 0) = 0 \\ V(t, S = L) = 0 \end{cases}$$

- Discrétiser l'équation par la méthode d'Euler explicite et utilisez les paramétres suivants :

$$\begin{cases} N = 99 \quad (discrtisation\, de\, l'interval,\ [0, L]) \\ M = 4999 \quad (discrtisation\, de\, l'interval,\ [0, T]) \end{cases}$$

Pour la suite utiliser les valeurs suivantes :

$$\begin{cases} L = 30 \\ T = 0.5 \\ r = 0.1 \\ \sigma = 0.5 \\ K = 10 \end{cases}$$

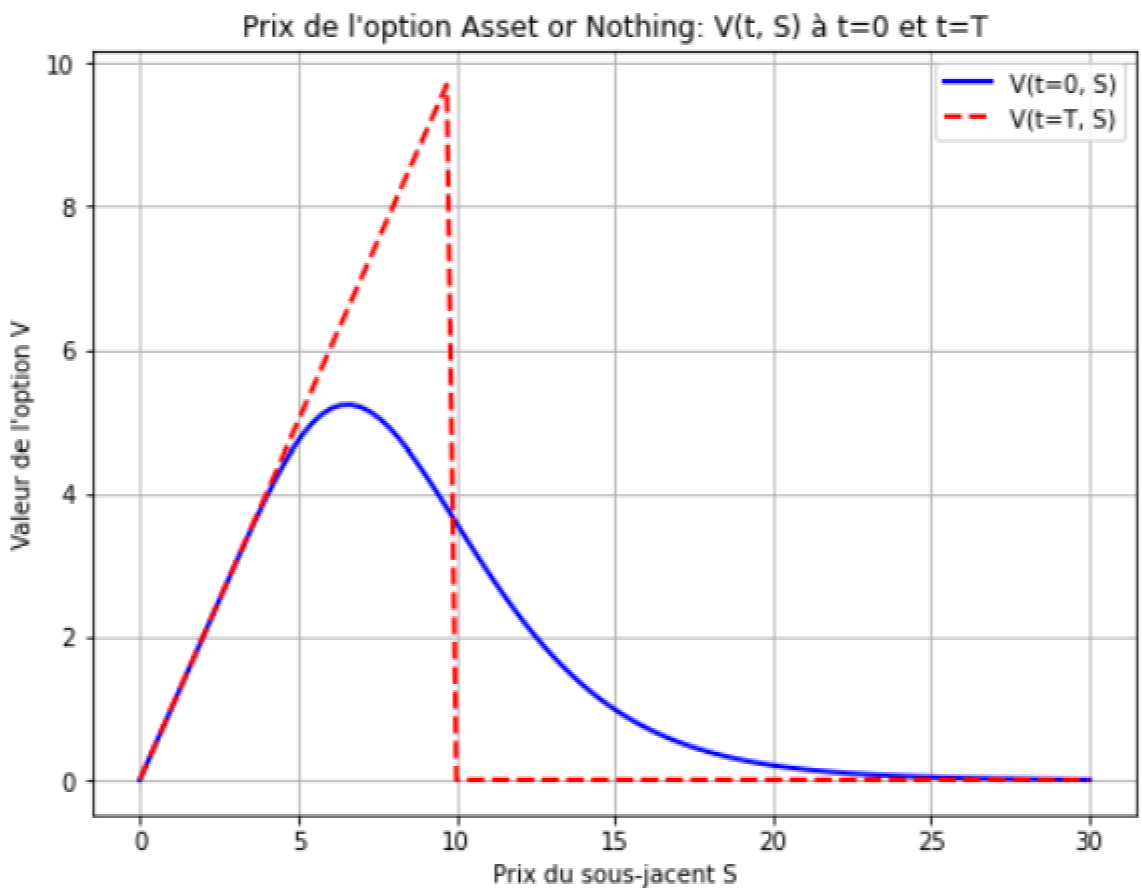
Question 1

Code

```
# Question 1)

import numpy as np
import matplotlib.pyplot as plt

# Paramètres
L = 30
T = 0.5
r = 0.1
```

```
sigma = 0.5
K = 10

N = 99     # Discrétisation de l'intervalle, [O,L]
M = 4999   # Discrétisation de l'intervalle, [0,T]

dS = L / N
dt = T / M

S = np.linspace(0, L, N+1)
t = np.linspace(0, T, M+1)

# Fonction de Pay-off Asset or Nothing
def pay_off_asset(S, K):
   return np.where(S < K, S, 0)

# Résolution de l'équation de Black-Scholes avec Euler explicite
def solve_black_scholes():
   V = np.zeros((M+1, N+1))
   V[-1, :] = pay_off_asset(S, K)  # Condition finale
   V[:, 0] = 0                     # Condition aux limites pour S=0
   V[:, -1] = 0                    # Condition aux limites pour S=L

   # Calcul par méthode d'Euler explicite
   for n in range(M-1, -1, -1):
      for i in range(1, N):
         dV_dS = (V[n+1, i+1] - V[n+1, i-1]) / (2 * dS)
         d2V_dS2 = (V[n+1, i+1] - 2 * V[n+1, i] + V[n+1, i-1]) / (dS**2)
         V[n, i] = V[n+1, i] + dt * (
            r * S[i] * dV_dS
            + 0.5 * sigma**2 * S[i]**2 * d2V_dS2
            - r * V[n+1, i]
         )
   return V

#Résolution
V = solve_black_scholes()

#Code pour affcher le graphe
plt.figure(figsize=(8, 6))
plt.plot(S, V[0, :], label='V(t=0, S)', color='blue', linewidth=2)
plt.plot(S, V[-1, :], label='V(t=T, S)', color='red', linestyle='--', linewidth=2)
plt.title('Prix de l\'option Asset or Nothing: V(t, S) à t=0 et t=T')
plt.xlabel('Prix du sous-jacent S')
plt.ylabel('Valeur de l\'option V')
plt.legend()
plt.grid()
plt.show()
```

Graphe



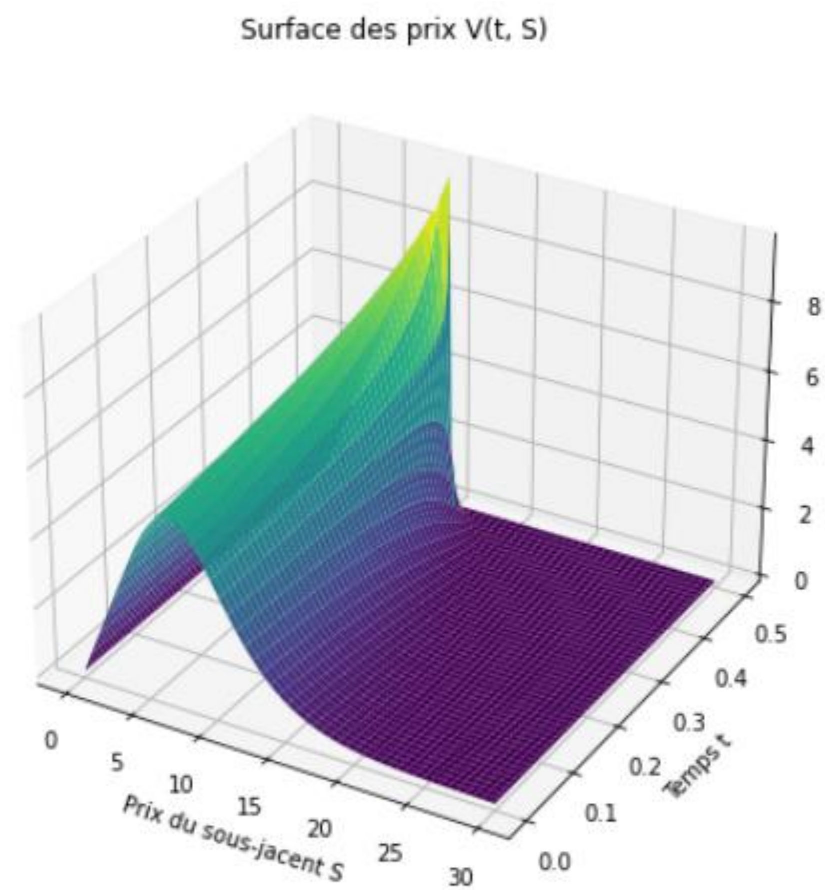
Question 2

Code

```
#Question 2 : Tracer la surface des prix V(t,S) en 3 dimensions

#On importe les bibliothèques nécessaires
from mpl_toolkits.mplot3d import Axes3D

T_grid, S_grid = np.meshgrid(t, S, indexing='ij')

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(S_grid, T_grid, V, cmap='viridis', edgecolor='none')
```

```
ax.set_title('Surface des prix V(t, S)')
ax.set_xlabel('Prix du sous-jacent S')
ax.set_ylabel('Temps t')
ax.set_zlabel('Valeur de l\'option V')
plt.show()
```

Graphe



Question 3

Code

```
# Question 3 : Calculer de V(t = T/3, S = 6)

t_index = int(M * (1/3))
s_index = np.argmin(np.abs(S - 6))
```

```
V_T3_S6 = V[t_index, s_index]
print(f"Valeur de l'option à t = T/3 et S = 6 : {V_T3_S6:.4f}")
```

Output

```
Valeur de l'option à t = T/3 et S = 6 : 5.5919
```

Partie II : Volatilité est locale

Partie II   Volatilité est locale.

La volatilité locale depend de la valeur de l'actif S et du temps de facon suivante :

$$\sigma_{locale}(t) = \begin{cases} \sigma(1+\sin(\pi\frac{T-t}{2T})) & si \quad t < T/2 \\ \sigma \quad si \quad t \geq T/2 \end{cases}$$

Question 6

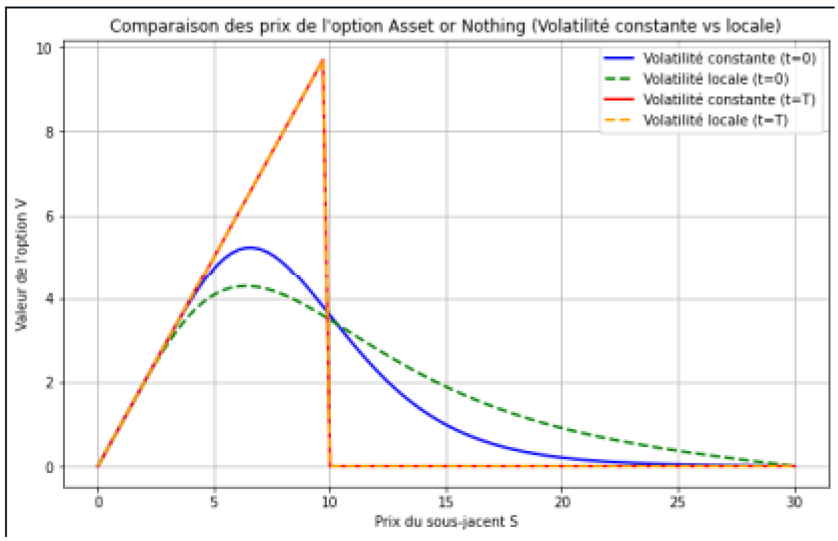
Code

```
#Partie II : Volatilité est locale

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Paramètres
L = 30
T = 0.5
r = 0.1
sigma = 0.5
K = 10

N = 99      # Discrétisation de l'intervalle, [O,L]
M = 4999    # Discrétisation de l'intervalle, [0,T]
```

```
dS = L / N
dt = T / M

S = np.linspace(0, L, N+1)
t = np.linspace(0, T, M+1)

# Fonction de volatilité locale
def sigma_locale(t):
    if t < T/2:
        return sigma * (1 + np.sin(np.pi * (T - t) / (2 * T)))
    else:
        return sigma

# Fonction de Pay-off Asset or Nothing
def pay_off_asset(S, K):
    return np.where(S < K, S, 0)

# Résolution de l'équation de Black-Scholes avec volatilité locale
def solve_black_scholes_local_volatility():
    V = np.zeros((M+1, N+1))
    V[-1, :] = pay_off_asset(S, K)
    V[:, 0] = 0
    V[:, -1] = 0

    # Calcul par méthode d'Euler explicite
    for n in range(M-1, -1, -1):
        for i in range(1, N):
            sigma_t = sigma_locale(t[n])
            dV_dS = (V[n+1, i+1] - V[n+1, i-1]) / (2 * dS)
            d2V_dS2 = (V[n+1, i+1] - 2 * V[n+1, i] + V[n+1, i-1]) / (dS**2)
            V[n, i] = V[n+1, i] + dt * (
                r * S[i] * dV_dS
                + 0.5 * sigma_t**2 * S[i]**2 * d2V_dS2
                - r * V[n+1, i]
            )
    return V

# Résolution avec volatilité locale
V_local = solve_black_scholes_local_volatility()

# Comparaison avec la volatilité constante (Partie I)
V_constant = solve_black_scholes()

# Question 6 : Comparaison des surfaces 2D
plt.figure(figsize=(10, 6))
plt.plot(S, V_constant[0, :], label='Volatilité constante (t=0)', color='blue', linewidth=2)
plt.plot(S, V_local[0, :], label='Volatilité locale (t=0)', color='green', linestyle='--', linewidth=2)
plt.plot(S, V_constant[-1, :], label='Volatilité constante (t=T)', color='red', linewidth=2)
```

```
plt.plot(S, V_local[-1, :], label='Volatilité locale (t=T)', color='orange', linestyle='--', linewidth=2)
plt.title('Comparaison des prix de l\'option Asset or Nothing (Volatilité constante vs locale)')
plt.xlabel('Prix du sous-jacent S')
plt.ylabel('Valeur de l\'option V')
plt.legend()
plt.grid()
plt.show()
```

Graphe



Problème 2 : Prix de l'option ASSET or NOTHING via Monte-Carlo

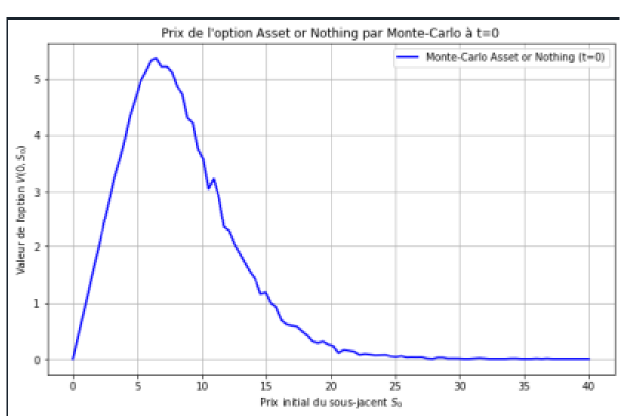
Question 1

Code

```
#Problème 2 : Prix de l'option ASSET or NOTHING
#via Monte-Carlo

#Question 1

import numpy as np
```

```
import matplotlib.pyplot as plt

# Paramètres
K = 10
T = 0.5
r = 0.1
sigma = 0.5
M = 1000
S0_range = np.linspace(0, 40, 100)

# Fonction pour calculer le prix de l'option Asset or Nothing par Monte-Carlo
def monte_carlo_asset_or_nothing(S0, K, T, r, sigma, M):
   Z = np.random.standard_normal(M)
   ST = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)
   payoff = np.where(ST < K, ST, 0)
   return np.exp(-r * T) * np.mean(payoff)

# Calcul des prix pour différents S0
V_mc = [monte_carlo_asset_or_nothing(S0, K, T, r, sigma, M) for S0 in S0_range]

# Tracer le graphique
plt.figure(figsize=(10, 6))
plt.plot(S0_range, V_mc, label="Monte-Carlo Asset or Nothing (t=0)", color='blue', linewidth=2)

plt.title("Prix de l'option Asset or Nothing par Monte-Carlo à t=0")
plt.xlabel("Prix initial du sous-jacent $S_0$")
plt.ylabel("Valeur de l'option $V(0, S_0)$")
plt.legend()
plt.grid()
plt.show()
```

Graphe



Par analyse de la courbe de la partie I, on trouve une courbe semblable ce qui semble cohérent.

Question 2

Code

```
import numpy as np

# Paramètres
K = 10
T = 0.5
r = 0.1
sigma = 0.5
M = 1000
t_partial = T / 3
S_t = 6

# Fonction pour Monte-Carlo avec horizon réduit
def monte_carlo_partial_asset_or_nothing(S, K, t, T, r, sigma, M):
    Z = np.random.standard_normal(M)
    remaining_time = T - t
    ST = S * np.exp((r - 0.5 * sigma**2) * remaining_time + sigma * np.sqrt(remaining_time) * Z)
    payoff = np.where(ST < K, ST, 0)
    return np.exp(-r * remaining_time) * np.mean(payoff)

# Calcul du prix de l'option à t=T/3 pour S=6
V_partial = monte_carlo_partial_asset_or_nothing(S_t, K, t_partial, T, r, sigma, M)
print(f"Le prix de l'option ASSET OR NOTHING à t=T/3 pour S=6 est : {V_partial:.4f}")
```

Output

```
Le prix de l'option ASSET OR NOTHING à t=T/3 pour
S=6 est : 5.6627
```

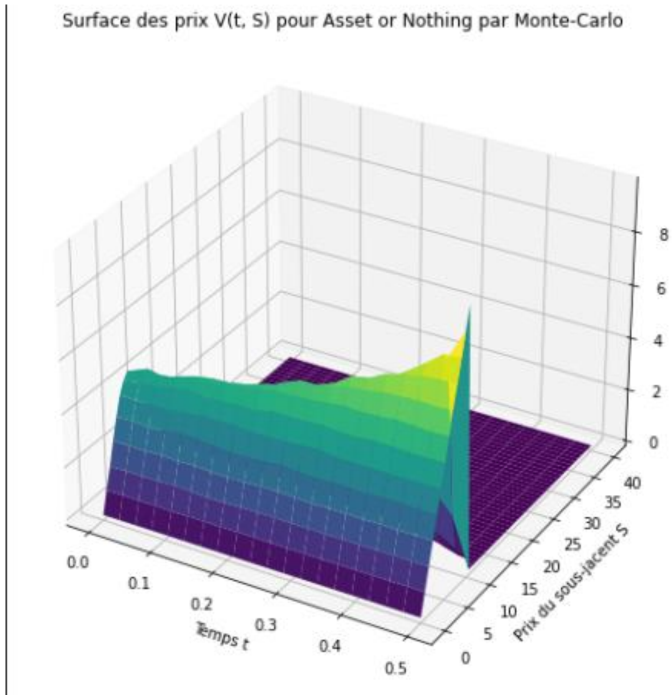
Question 3

Code

```
#Question 3

from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

# Paramètres
K = 10
T = 0.5
r = 0.4
sigma = 0.5
M = 1000

# Fonction Monte-Carlo pour Asset or Nothing
def monte_carlo_partial_asset_or_nothing(S, K, t, T, r, sigma, M):
  Z = np.random.standard_normal(M)
  remaining_time = T - t
  ST = S * np.exp((r - 0.5 * sigma**2) * remaining_time + sigma * np.sqrt(remaining_time) * Z)
  payoff = np.where(ST < K, ST, 0)
  return np.exp(-r * remaining_time) * np.mean(payoff)

# Discrétisation du temps et des prix S
time_steps = np.linspace(0, T, 20)  # Discrétisation de l'intervalle [0, T]
S_values = np.linspace(0, 40, 50)
V_surface = np.zeros((len(time_steps), len(S_values)))

# Calcul de la surface des prix
for i, t in enumerate(time_steps):
  for j, S in enumerate(S_values):
    V_surface[i, j] = monte_carlo_partial_asset_or_nothing(S, K, t, T, r, sigma, M)

# Tracer la surface
T_grid, S_grid = np.meshgrid(time_steps, S_values)

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(T_grid, S_grid, V_surface.T, cmap='viridis')

ax.set_title('Surface des prix V(t, S) pour Asset or Nothing par Monte-Carlo')
ax.set_xlabel('Temps t')
ax.set_ylabel('Prix du sous-jacent S')
ax.set_zlabel('Valeur de l\'option V')
```

plt.show()

Graphe



Problème III : Prix de l'option « Asset or Nothing » dans le modèle de Ho-Lee

Supposons que le taux d'intérêt est régi par l'équation différentielle stochastique de la forme

$$dr_t = a(t)dt + \omega dW_t, \quad a(t) = e^{\gamma \frac{(T-t)}{T}}, \quad r(t=0) = r_0 \tag{1}$$

C'est ce qu'on appelle le modèle Ho-Lee.

Dans le cas où le processus de taux d'intérêt $r_t = r_0$ est une constante le prix de l'option à $t = 0$ est donné par l'expression :

$$V(0, S_0) = e^{-r_0 T}\mathbb{E}[\text{Pay-off-Asset}(S_T)\,/S(0) = S_0]$$

Si r_t est un processus stochastique on remplace la formule du prix par la formule suivante :

$$V(0, S_0) = \mathbb{E}[e^{-\int_0^T r_t dt}\,\text{Pay-off-Asset}(S_T)\,/S(0) = S_0,\, r(0) = r_0]$$

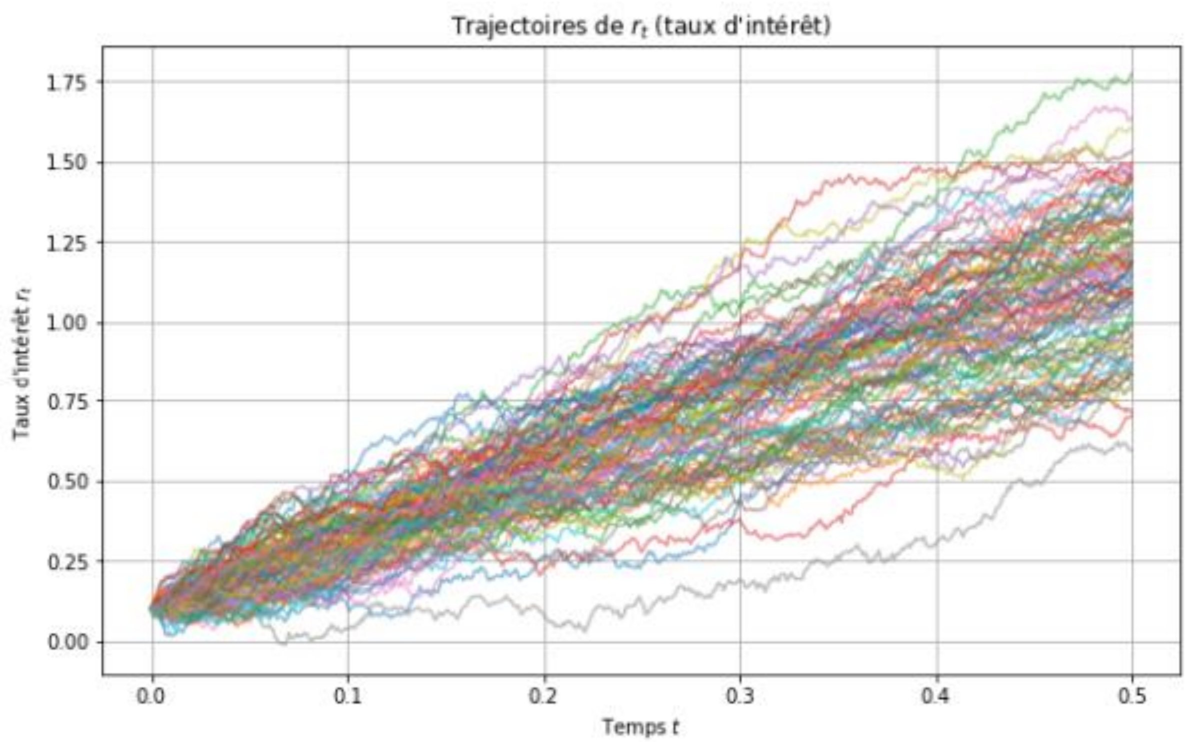
Question 1

Code

```
#Problème 3 : Prix de l'option "Asset or Nothing"
#dans le modèle de Ho-Lee

#Question 1

import numpy as np
import matplotlib.pyplot as plt

# Paramètres
r0 = 0.1
omega = 0.3
T = 0.5
sigma = 0.5
gamma = 0.2
K = 10
Nmc = 1000
dt = 0.001
N = int(T / dt)


# Temps discrétisé
t = np.linspace(0, T, N+1)

# Calcul de a(t)
def a(t):
    return np.exp(gamma * (T - t)) / T

# Simulation de r_t
def simulate_r(Nmc, N, dt, r0, omega, t):
    r_trajectories = np.zeros((Nmc, N+1))
    r_trajectories[:, 0] = r0
    for i in range(1, N+1):
        dW = np.random.normal(0, np.sqrt(dt), Nmc)
        r_trajectories[:, i] = r_trajectories[:, i-1] + a(t[i-1]) * dt + omega * dW
    return r_trajectories

# Générer les trajectoires
r_trajectories = simulate_r(Nmc, N, dt, r0, omega, t)

# Tracer les trajectoires de r_t
plt.figure(figsize=(10, 6))
plt.plot(t, r_trajectories[:100].T, alpha=0.5)
```

```
plt.title('Trajectoires de $r_t$ (taux d\'intérêt)')
plt.xlabel('Temps $t$')
plt.ylabel('Taux d\'intérêt $r_t$')
plt.grid()
plt.show()
```

Graphe



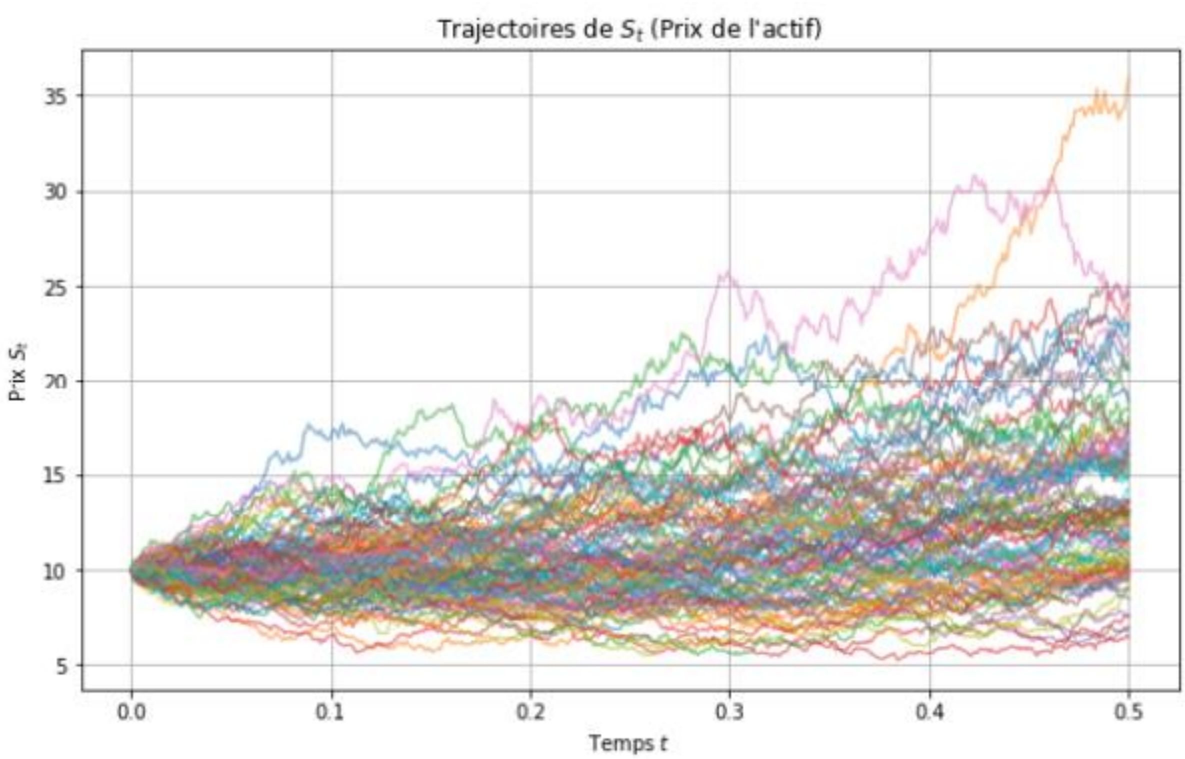
Question 2

Code

```
#Question 2

# Paramètres pour S_t
sigma = 0.5
S0 = 10

# Simulation de S_t
def simulate_S(Nmc, N, dt, S0, r_trajectories, sigma):
    S_trajectories = np.zeros((Nmc, N+1))
    S_trajectories[:, 0] = S0
```

```
    for i in range(1, N+1):
        dW = np.random.normal(0, np.sqrt(dt), Nmc)
        S_trajectories[:, i] = S_trajectories[:, i-1] * (
            1 + r_trajectories[:, i-1] * dt + sigma * dW
        )
    return S_trajectories

# Générer les trajectoires
S_trajectories = simulate_S(Nmc, N, dt, S0, r_trajectories, sigma)

# Tracer les trajectoires de S_t
plt.figure(figsize=(10, 6))
plt.plot(t, S_trajectories[:100].T, alpha=0.5)
plt.title('Trajectoires de $S_t$ (Prix de l\'actif)')
plt.xlabel('Temps $t$')
plt.ylabel('Prix $S_t$')
plt.grid()
plt.show()
```

Graphe

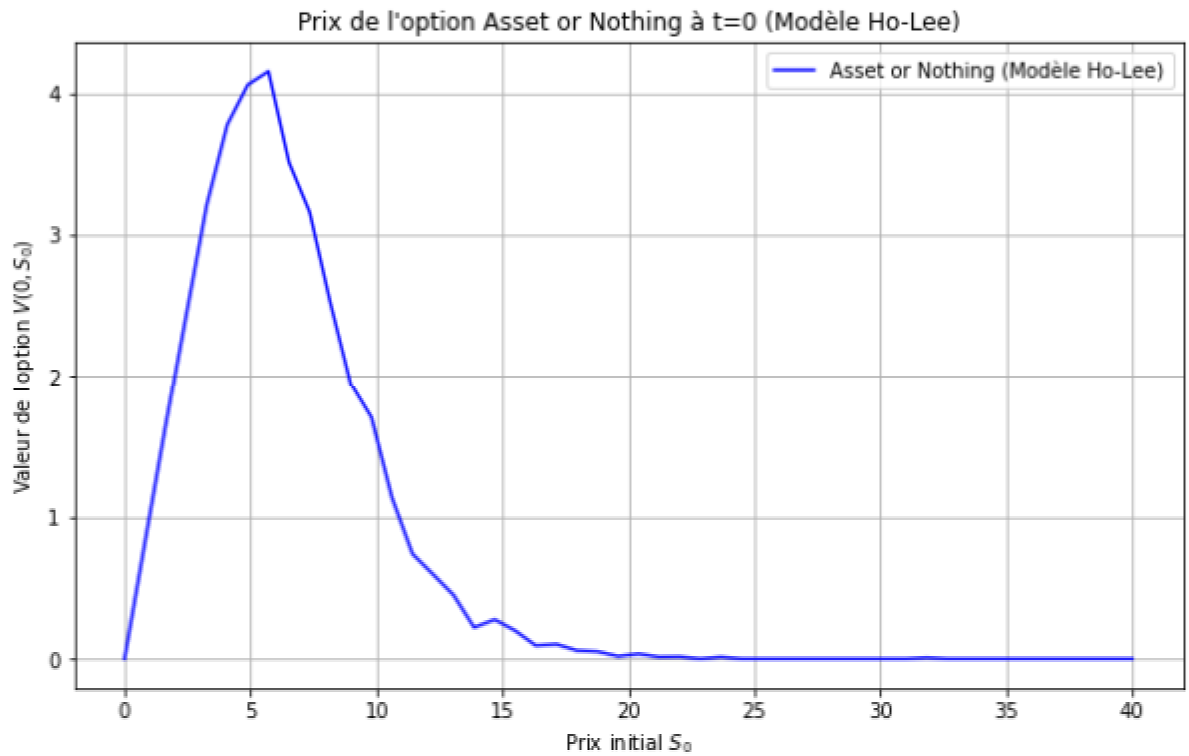


Question 3

Code

```
# Payoff pour Asset or Nothing
def payoff_asset_or_nothing(ST, K):
    return np.where(ST < K, ST, 0)

# Monte Carlo pour V(0, S0)
def monte_carlo_asset_or_nothing(S0, K, T, r_trajectories, S_trajectories, dt):
    discount_factors = np.exp(-np.sum(r_trajectories[:, :-1], axis=1) * dt)
    payoff = payoff_asset_or_nothing(S_trajectories[:, -1], K)
    return np.mean(discount_factors * payoff)

# Calcul du prix pour une gamme de S0
S0_range = np.linspace(0, 40, 50)
V_mc = []

for S0 in S0_range:
    S_trajectories = simulate_S(Nmc, N, dt, S0, r_trajectories, sigma)
    V_mc.append(monte_carlo_asset_or_nothing(S0, K, T, r_trajectories, S_trajectories, dt))

# Tracer le prix de l'option
plt.figure(figsize=(10, 6))
plt.plot(S0_range, V_mc, label='Asset or Nothing (Modèle Ho-Lee)', color='blue')
plt.title('Prix de l\'option Asset or Nothing à t=0 (Modèle Ho-Lee)')
plt.xlabel('Prix initial $S_0$')
plt.ylabel('Valeur de l\'option $V(0, S_0)$')
plt.legend()
plt.grid()
plt.show()
```

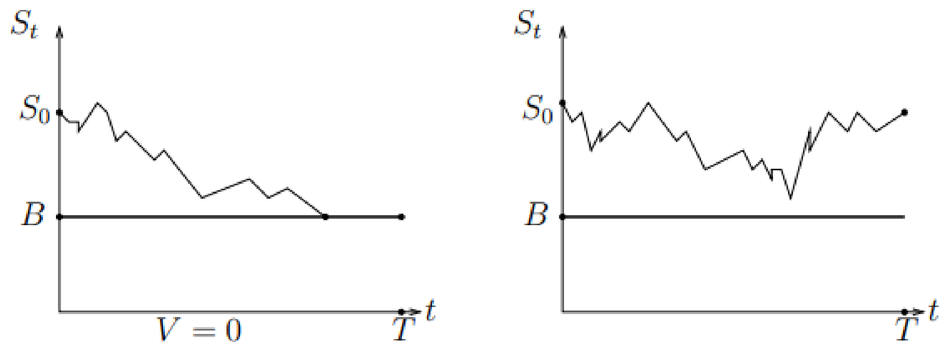
Graphe



On remarque que la comparaison avec le graphe du problème 2, on obtient des valeurs similaires.

Problème 4 : Prix de l'option barrière via Monte-Carlo. Down and Out Asset or Nothing option

Le prix d'une option Barrière $V(t, S)$ dépend du schéma d'evolution de l'actif. Si la barrière B_1 est inférieure à la valeur initiale de l'actif, nous avons une option **"down"**. Si l'actif sous-jacent atteint la barrière alors le contrat devient sans valeur. Si la barrière est atteinte, on dit que l'option est "knocked out".



Le prix de l'option échéance T si la barrière n'est pas atteinte :

$$\text{Pay-off-Asset}(S, K) = \begin{cases} S, & S < K \\ 0, & S \geq K \end{cases}$$

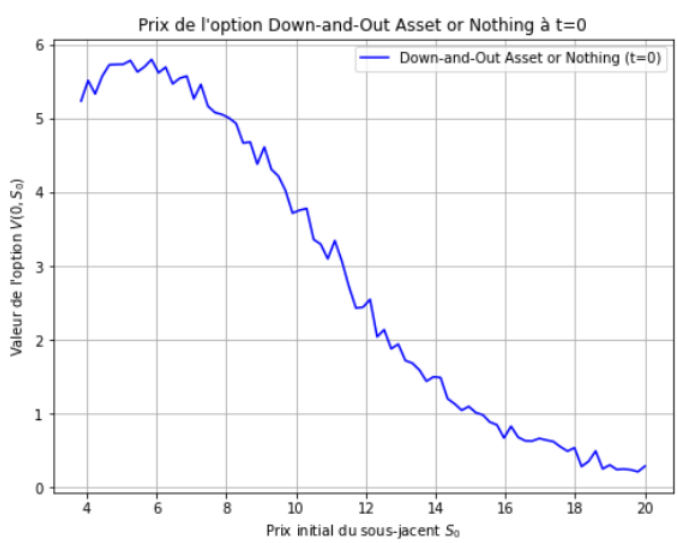
Question 1

Code

```
#Problème 4 : Prix de l'option barrière via
#Monte-Carlo. Down and Out Asset or Nothing option

#Question 1

import numpy as np
import matplotlib.pyplot as plt

# Paramètres
Nmc = 1000
L = 20
T = 0.5
r = 0.1
sigma = 0.5
K = 10
B1 = 4
N = 100
dt = T / N

# Fonction pour simuler le prix d'une option Down and Out Asset or Nothing
def monte_carlo_down_and_out(S0, K, B1, T, r, sigma, Nmc):
```

```
    dt = T / N
    payoff = []
    for _ in range(Nmc):
        S = S0
        knocked_out = False
        for _ in range(N):
            Z = np.random.normal()
            S *= np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z)
            if S <= B1:
                knocked_out = True
                break
        if not knocked_out:
            payoff.append(S if S < K else 0)
    return np.exp(-r * T) * np.mean(payoff)

# Gamme de valeurs pour S0
S0_range = np.linspace(0, L, 100)
V_down_and_out = [monte_carlo_down_and_out(S0, K, B1, T, r, sigma, Nmc) for S0 in S0_range]

# Code pour tracer le graphique
plt.figure(figsize=(8, 6))
plt.plot(S0_range, V_down_and_out, label='Down-and-Out Asset or Nothing (t=0)', color='blue')
plt.title('Prix de l\'option Down-and-Out Asset or Nothing à t=0')
plt.xlabel('Prix initial du sous-jacent $S_0$')
plt.ylabel('Valeur de l\'option $V(0, S_0)$')
plt.legend()
plt.grid()
plt.show()
```

Graphe



Question 2

Code

```
#Question 2

from mpl_toolkits.mplot3d import Axes3D

# Fonction pour calculer la surface des prix
def monte_carlo_surface_down_and_out(S_values, t_values, K, B1, T, r, sigma, Nmc):
    dt = T / N
    V_surface = np.zeros((len(t_values), len(S_values)))

    for i, t in enumerate(t_values):
        remaining_time = T - t
        for j, S in enumerate(S_values):
            payoff = []
            for _ in range(Nmc):
                S_path = S
                knocked_out = False
                for _ in range(int(remaining_time / dt)):
                    Z = np.random.normal()
                    S_path *= np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z)
                    if S_path <= B1:
                        knocked_out = True
                        break
                if not knocked_out:
                    payoff.append(S_path if S_path < K else 0)
            V_surface[i, j] = np.exp(-r * remaining_time) * np.mean(payoff)
    return V_surface

# Discrétisation pour le graphique 3D
S_values = np.linspace(0, L, 50)
t_values = np.linspace(0, T, 20)
V_surface = monte_carlo_surface_down_and_out(S_values, t_values, K, B1, T, r, sigma, Nmc)

# Grille pour le tracé
T_grid, S_grid = np.meshgrid(t_values, S_values)

# Tracé de la surface
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(T_grid, S_grid, V_surface.T, cmap='viridis')

ax.set_title('Surface des prix Down-and-Out Asset or Nothing')
ax.set_xlabel('Temps $t$')
```

```
ax.set_ylabel('Prix du sous-jacent $S$')
ax.set_zlabel('Valeur de l\'option $V(t, S)$')
plt.show()
```

Courbe