

Projet calcul stochastique.

Choix des paramètres :

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt

#Parametres
T = 3 # Temps final
n = 1000 # Nombre de pas de temps
N = 10 # Nombre de trajectoires
pas = T / n

a = 0.1
r0 = 0.015
b = 1.5 * r0
gamma = 0.015
alpha = 0.1
```

Q1. D'après l'équation (1), on a:

$$dr_t = a(b - r_t) dt + \gamma dW_t \quad \text{où } a, b, \gamma > 0. \quad W \text{ représente}$$

un brownian.

$$dr_t = a(b - r_t) dt + \gamma dW_t$$

$$\Leftrightarrow r_{t,i} - r_{t-1,i} = a(b - r_{t-1,i}) dt + \gamma (w_{t,i} - w_{t-1,i})$$

$$\Leftrightarrow r_{t,i} = r_{t-1,i} + a(b - r_{t-1,i}) \frac{T}{n} + \gamma \sqrt{\frac{T}{n}} \xi_i \quad \text{où } \xi_i \sim \mathcal{U}(0,1) \quad \text{et}$$

$r_0 = r_0$ donne

#Initialisation des matrices pour r, S0 et S_R

```
r = r0 * np.ones((n+1, N))
```

```
S0 = np.ones((n+1, N))
```

```
S_R = np.ones((n+1, N))
```

```
S = np.ones((n+1, N))
```

#Simulation

```
dates = np.linspace(0, T, n+1)
```

```
for j in range(N):
```

```
    for i in range(1, n+1):
```

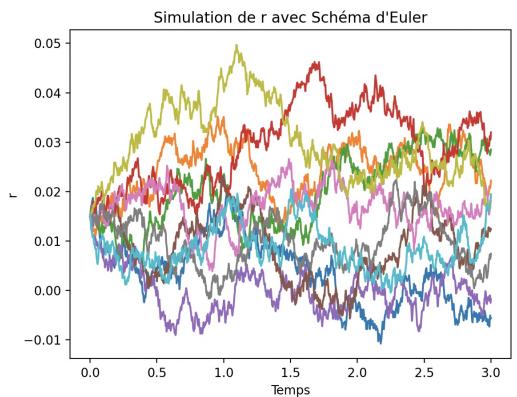
```
        #Calcul de r, S0 et S_R
```

```
        delta_W = np.sqrt(pas) * npr.randn()
```

```
        r[i, j] = r[i-1, j] + a * (b - r[i-1, j]) * pas + gamma * delta_W
```

$$G_i = \frac{w_{t,i} - w_{t-1,i}}{\sqrt{\frac{T}{n}}}$$

```
#1 Graphe des résultats pour r
plt.plot(dates, r)
plt.title('Simulation de r avec Schéma d'Euler')
plt.xlabel('Temps')
plt.ylabel('r')
plt.show()
```



Q.2) D'après l'équation (2),

$$\Delta S_e^o = r_e S_e^o \Delta t$$

$$S_0^o = 1$$

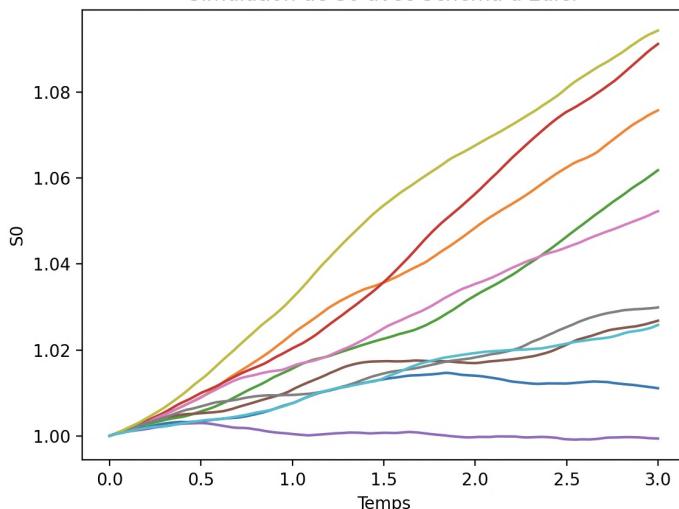
On a $S_{e_i}^o - S_{e_{i-1}}^o = r_{e_{i-1}} S_{e_{i-1}}^o \frac{T}{n}$

$$\Leftrightarrow \begin{cases} S_{e_i}^o = S_{e_{i-1}}^o + r_{e_{i-1}} S_{e_{i-1}}^o \frac{T}{n} \\ S_0^o = 1 \text{ donné} \end{cases}$$

#Simulation

```
dates = np.linspace(0, T, n+1)
for j in range(N):
    for i in range(1, n+1):
        #Calcul de r, S0 et S_R
        delta_W = np.sqrt(pas) * np.random()
        r[i, j] = r[i-1, j] + a * (b - r[i-1, j]) * pas + gamma * delta_W
        S0[i, j] = S0[i-1, j] + r[i-1, j] * S0[i-1, j] * pas
```

Simulation de S0 avec Schéma d'Euler



Q.3) Soit τ une fonction donnée par $\tau(t, x) = \alpha(1 + f(t) + g(x))$

- $\alpha \in [0,05; 0,2]$

- f et g sont des fonctions différentiables et bornées.

$$d\tilde{S}_t = \tau(t, \tilde{S}_t) \tilde{S}_t dt$$

$$\tilde{S}_{t_i} - \tilde{S}_{t_{i-1}} = \tau(t_{i-1}, \tilde{S}_{t_{i-1}}) \tilde{S}_{t_{i-1}} \sqrt{\frac{T}{n}} G_i$$

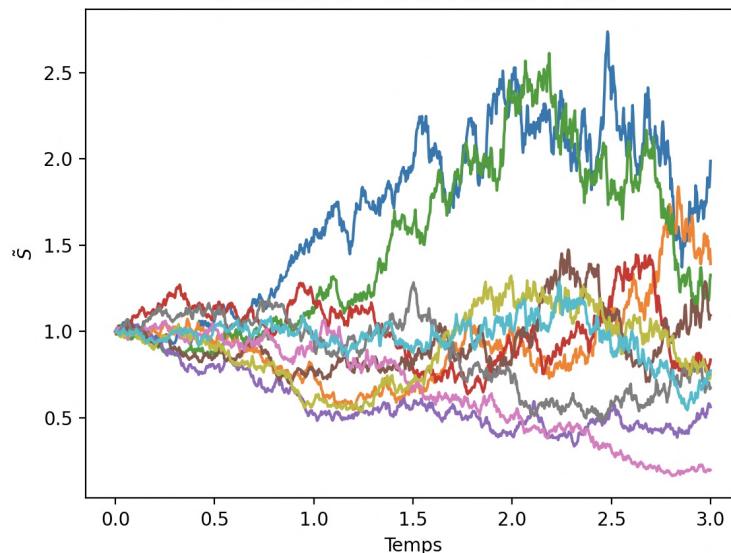
$$\Leftrightarrow \tilde{S}_{t_i} = \tilde{S}_{t_{i-1}} + \tau(t_{i-1}, \tilde{S}_{t_{i-1}}) \tilde{S}_{t_{i-1}} \sqrt{\frac{T}{n}} G_i$$

$$\left\{ \begin{array}{l} \tilde{S}_0 = \tilde{S}_0 \text{ donné} \\ \text{avec } (G_i)_{i=1}^n \text{ i.i.d u}(0,1) \end{array} \right.$$

#Simulation

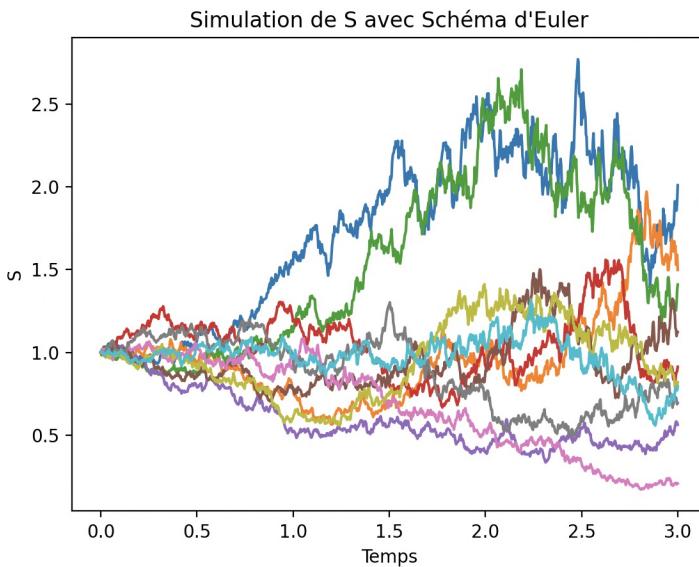
```
dates = np.linspace(0, T, n+1)
for j in range(N):
    for i in range(1, n+1):
        #Calcul de r, S0 et S_R
        delta_W = np.sqrt(pas) * np.random()
        r[i, j] = r[i-1, j] + a * (b - r[i-1, j]) * pas + gamma * delta_W
        S0[i, j] = S0[i-1, j] + r[i-1, j] * S0[i-1, j] * pas
        delta_B = np.sqrt(pas) * np.random()
        S_R[i, j] = S_R[i-1, j] + sigma(dates[i-1], S_R[i-1, j]) * S_R[i-1, j] * delta_B
```

Simulation de \tilde{S} avec Schéma d'Euler



Q.4) Pour déterminer les trajectoires de S_t , nous avons $S_t = \tilde{S}_t \times S_0^t$.

```
#Simulation
dates = np.linspace(0, T, n+1)
for j in range(N):
    for i in range(1, n+1):
        #Calcul de r, S0 et S_R
        delta_W = np.sqrt(pas) * np.random()
        r[i, j] = r[i-1, j] + a * (b - r[i-1, j]) * pas + gamma * delta_W
        S0[i, j] = S0[i-1, j] + r[i-1, j] * S0[i-1, j] * pas
        delta_B = np.sqrt(pas) * np.random()
        S_R[i, j] = S_R[i-1, j] + sigma(dates[i-1], S_R[i-1, j]) * S_R[i-1, j] * delta_B
        S[i, j] = S_R[i, j] * S0[i, j]
```



Q.5) Soit g_T la variable aléatoire associée au payoff d'une option telle que g_T est \mathcal{F}_T mesurable et \mathbb{Q} -intégrable avec \mathbb{Q} une mesure de probabilité sur la base stochastique $(\Omega, (\mathcal{F}_t)_{t \in [0, T]}, \mathbb{P})$.

Dans un marché complet, deux actifs de prix identiques à maturité T ont le même prix initialement ($t=0$).

De ce fait, le prix d'une option, en $t=0$, est la valeur du portefeuille de réplique de l'option. De plus, il est unique.

Remarque :

Soit g_T le prix du payoff, on a $g_T = (S_T - u)^+$ et $\tilde{g}_T^j = \frac{(S_T^j - u)^+}{S_0^j}$
 $j = 1, \dots, N$.

On note u le strike price de S_T le prix du sous-jacent à maturité et
 S_t le prix du sous-jacent au temps $t \in [0, T]$.

Q.6) $g_T^1 = (u^1 - S_T)^+$ avec $u^1 \in [\frac{1}{2}S_0, 2S_0]$

$g_T^2 = (u^2 - \frac{1}{T} \int_0^T S_{t,T} dt)^+$ avec $u^2 \in [S_0, 3S_0]$

Pour cette partie du code, nous avons créé une fonction payoff
dans le but de calculer g_T^1 et g_T^2 .

Pour la fonction g_T^2 , on peut approximer $\frac{1}{T} \int_0^T S_{t,T} dt$ comme $\frac{1}{T} \times$ moyenne
(somme de Riemann).

```
#Question 6
#Fonction payoff pour option put
def payoff(K, S):
    return max(K - S, 0)

K1 = S[0,0]
gT1 = np.array([payoff(K1, S[-1, i])*np.exp(r0*pas*n) for i in range(N)])
print("Payoff de la fonction gT1:", gT1)

K2 = S[0,0]*2
#Calcul de la moyenne intégrale de S sur T
mean_S = np.mean(S, axis=0) #np.mean(S, axis=0) * T * (1/T)
gT2 = np.array([payoff(K2, mean_S[i])*np.exp(r0*pas*n) for i in range(N)])
print("Payoff de la fonction gT2 :", gT2)
```

```

1 import numpy as np
2 import numpy.random as npr
3 import matplotlib.pyplot as plt
4
5 #Parametres
6 T = 3 # Temps final
7 n = 1000 # Nombre de pas de temps
8 N = 10 # Nombre de trajectoires
9 pas = T / n
10
11 a = 0.1
12 r0 = 0.015
13 b = 1.5 * r0
14 gamma = 0.015
15 alpha = 0.1
16
17
18 #Fonctions f et g
19 def f(x):
20     return (x/2*T)
21
22 def g(x):
23     return 1/(1+x)
24
25 #Fonction sigma
26 def sigma(t, x):
27     return alpha * (1 + f(t) + g(x))
28
29 #Initialisation des matrices pour r, S0 et S_R
30 r = r0 * np.ones((n+1, N))
31 S0 = np.ones((n+1, N))
32 S_R = np.ones((n+1, N))
33 S = np.ones((n+1, N))
34
35 #Simulation
36 dates = np.linspace(0, T, n+1)
37 for j in range(N):
38     for i in range(1, n+1):
39         #Calcul de r, S0 et S_R
40         delta_W = np.sqrt(pas) * npr.randn()
41         r[i, j] = r[i-1, j] + a * (b - r[i-1, j]) * pas + gamma * delta_W
42         S0[i, j] = S0[i-1, j] + r[i-1, j] * S0[i-1, j] * pas
43         delta_B = np.sqrt(pas) * npr.randn()
44         S_R[i, j] = S_R[i-1, j] + sigma(dates[i-1], S_R[i-1, j]) * S_R[i-1, j] * delta_B
45         S[i, j] = S_R[i, j] * S0[i, j]
46
47
48 #1 Graphe des résultats pour r
49 plt.plot(dates, r)
50 plt.title('Simulation de r avec Schéma d\'Euler')
51 plt.xlabel('Temps')
52 plt.ylabel('r')
53 plt.show()
54
55 #2 Graphe des résultats pour S0
56 plt.plot(dates, S0)
57 plt.title('Simulation de S0 avec Schéma d\'Euler')
58 plt.xlabel('Temps')
59 plt.ylabel('S0')
60 plt.show()
61
62 #3 Graphe des résultats pour S_R
63 plt.plot(dates, S_R)
64 plt.title('Simulation de $\tilde{S}$ avec Schéma d\'Euler')
65 plt.xlabel('Temps')
66 plt.ylabel(r'$\tilde{S}$')
67 plt.show()
68
69 #4 Graphe des résultats pour S
70 plt.plot(dates, S)
71 plt.title('Simulation de S avec Schéma d\'Euler')
72 plt.xlabel('Temps')
73 plt.ylabel('S')
74 plt.show()
75
76 #Question 6
77 #Fonction payoff pour option put
78 def payoff(K, S):
79     return max(K - S, 0)
80
81 K1 = S[0,0]
82 gT1 = np.array([payoff(K1, S[-1, i]) / np.exp(r0*pas*n) for i in range(N)])
83 print("Payoff de la fonction gT1:", gT1)
84
85 K2 = S[0,0]*2
86 #Calcul de la moyenne intégrale de S sur T
87 mean_S = np.mean(S, axis=0) #np.mean(S, axis=0) * T * (1/T)
88 gT2 = np.array([payoff(K2, mean_S[i]) / np.exp(r0*pas*n) for i in range(N)])
89 print("Payoff de la fonction gT2 :", gT2)
90

```

Sorbie :

Payoff de la fonction gT1: [0.38553776 0.58276087 0.69033098 0.]	0.	0.	0.	0.24826163]	0.57079408
Payoff de la fonction gT2 : [0.92791045 1.03420396 1.26309366 0.82166473 0.73430359 1.24802962	0.83463456	0.74520965	0.84387196	0.90331652]	