Simulation Methods Project 2023

Dylan Riboulet, Anaelle Guennou, Benoît Gras, Dany Mey, Quentin Le Bournot

March 31, 2023

Contents

[utf8]inputenc amsmath amsfonts amssymb graphicx listings xcolor
Exercises

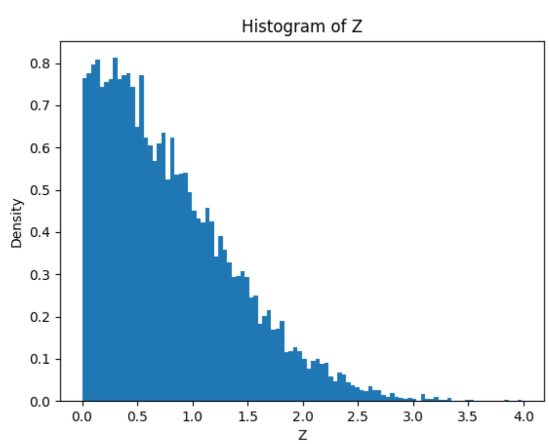
1 Exercice 1 - Ratio of Uniforms

Python Code

1.1 Question 1

Python Code

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

def ratio_of_uniforms():
    while True:
        U1 = np.random.uniform(0, 1)
        U2 = np.random.uniform(0, 1)
        if U1 > np.exp(-1/4 * (U2/U1)**2):
            continue
        else:
            return U2/U1

samples = [ratio_of_uniforms() for _ in range(10000)]

plt.hist(samples, bins=100, density=True)
plt.xlabel('Z')
plt.ylabel('Density')
plt.title('Histogram of Z')
plt.show()
```



1.2 Question 2

Given that U_1 and U_2 are independent and uniformly distributed over $[0, 1]$, we have:

$$U_1 = X$$
$$U_2 = XZ$$

Now, we compute the Jacobian of the transformation:

$$J = \left|\frac{\partial(U_1, U_2)}{\partial(X, Z)}\right| = \left|\frac{\partial(X, XZ)}{\partial(X, Z)}\right|$$

The Jacobian matrix is:

$$\begin{bmatrix} 1 & 0 \\ Z & X \end{bmatrix}$$

The determinant of this matrix is:

$$J = 1 \cdot X - 0 \cdot Z = X$$

We have :

$$0 \leq U_1 \leq exp(\frac{-1}{4} * (\frac{U_2}{U_1})^2)$$

And :

$$0 \leq U_2 \leq 1$$

So we can say that :

$$0 \leq X \leq exp(\frac{-1}{4} * (\frac{U_2}{U_1})^2)$$

And :

$$0 \leq Z \leq exp(\frac{-1}{4} * (\frac{U_2}{U_1})^2)$$

And we obtain the following PDF :

$f_Z(z) = \int_0^{exp(\frac{-1}{4}*(\frac{U_2}{U_1})^2)} \int_0^{exp(\frac{-1}{4}*(\frac{U_2}{U_1})^2)} f_{X,Z}(x,z)dxdz$

So :

$f_Z(z) = \int_0^{exp(\frac{-1}{4}*(\frac{U_2}{U_1})^2)} \int_0^{exp(\frac{-1}{4}*(\frac{U_2}{U_1})^2)} Xdxdz$

And finally :

$f_Z(z) = \frac{1}{2} * exp(\frac{-1}{4} * (\frac{U_2}{U_1})^2)^2$

1.3 Question 3

We have

$$M = \|\frac{f_X(x)}{f_Z(z)}\|_\infty$$

And we can define the infinity norm as :

$$M = sup(\frac{f_X(x)}{f_Z(z)})$$

So :

$$M = 2 * \|\frac{1}{exp(\frac{-x^2}{4})^2}\|_\infty$$

We need to take the derivative of the quotient and the derivative is :

$$2 * exp(\frac{-x^2}{4}) * (\frac{-x}{2}) * exp(\frac{-x^2}{4}) = -x * exp(\frac{-x^2}{4})^2$$

We have to resolve :

$$-x * exp(\frac{-x^2}{4})^2 = 0$$

And the solution is $x = 0$ So we have :

$$M = 2 * (\frac{1}{exp(0)}) = 2$$

So we have an acceptance rate of $\frac{1}{M}$ = 0,5 and we have to generate M = 2 samples.

2 Exercice 2 - Importance sampling

2.1 Question 1

```
import numpy as np
import scipy.stats as stats

S0 = 100
r = 0.01
sigma = 0.2
T = 1
K_far_OTM = 1000000
K_ATM = 100

def black_scholes_call(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * stats.norm.cdf(d1) - K * np.exp(-r * T) * stats.norm.cdf(d2)

def black_scholes_digital_call(S, K, T, r, sigma):
    d2 = (np.log(S / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    return np.exp(-r * T) * stats.norm.cdf(d2)

call_price_far_OTM = black_scholes_call(S0, K_far_OTM, T, r, sigma)
```

```python
digital_call_price_far_OTM = black_scholes_digital_call(S0, K_far_OTM, T, r, sigma)

call_price_ATM = black_scholes_call(S0, K_ATM, T, r, sigma)
digital_call_price_ATM = black_scholes_digital_call(S0, K_ATM, T, r, sigma)



# Compute the ratio for the far OTM strike using the Black-Scholes formula
ratio_far_OTM = call_price_far_OTM / (digital_call_price_far_OTM)
ratio_ATM = call_price_ATM / digital_call_price_ATM

print("Ratio for far OTM strike (K = $1,000,000):", ratio_far_OTM)
print("Ratio for ATM strike (K = $100):", ratio_ATM)
```

```
Ratio for far OTM strike (K = $1,000,000): nan
Ratio for ATM strike (K = $100): 17.743727373955107
```

Figure 1: Comparaison entre K = 1,000,000 avec K = 100 selon le modèle de pricing de Black Scholes calculé en TD par méthode de Monte Carlo

Le prix actuel de l'option est très éloigné de son prix d'exercice ce qui créé une erreur sur Python. Les simulations de Monte Carlo (avec modèle de Black Scholes) ont montré qu'aucune des simulations ne permettait d'atteindre le prix d'exercice (cf plot de la distribution des prix à maturité sous la mesure de probabilité historique ci dessous). Ainsi, nous avons décidé d'utiliser un r drifté pour obtenir la distribution de S_T avec une moyenne égale à K. La fonction de distribution de S_T est noté f, et celle de S^T est noté g. En utilisant l'Importance Sampling (cf. calculs ci-dessous et le Trick de l'énoncé nous avons pu estimer le rapport EC/EDC)

2.2 Question 2

```python
# Create a subplot with 1 row and 2 columns
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# Plot the price distribution under the original measure
ax1.hist(ST, bins=100, alpha=0.75, color='blue', label="Price Distribution (Original Measure)")
ax1.set_xlabel("Price")
ax1.set_ylabel("Frequency")
ax1.legend()

# Plot the price distribution under the new measure
ax2.hist(ST_, bins=100, alpha=0.75, color='green', label="Price Distribution (New Measure)")
ax2.set_xlabel("Price")
ax2.set_ylabel("Frequency")
ax2.legend()

# Adjust the layout to avoid overlapping
plt.tight_layout()

# Display the plots
plt.show()
```

Notons EC : le prix d'un European Call et EDC: le prix d'un European Digital Call

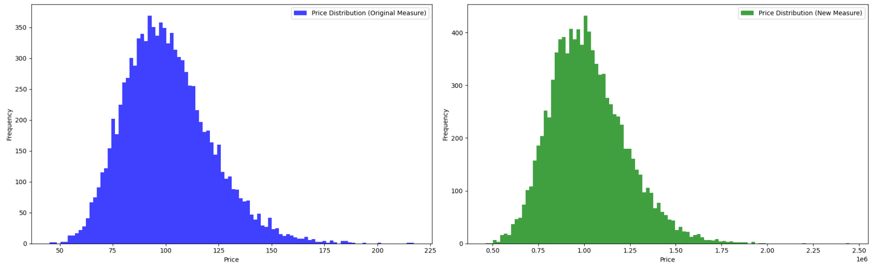


Figure 2: Comparaison des prix possibles sous les deux mesures de probabilités

$$EC = \mathbb{E}((S_T - K)^+) = \mathbb{E}(h(S_T)) = \mathbb{E}\left(\frac{h(S_T^*)f(S_T^*)}{g(S_T^*)}\right)$$

(espérance prise sous la mesure de probabilité risque neutre)

où $\frac{f(S_T^*)}{g(S_T^*)}$ est la dérivée de Radon-Nikodym évaluée à S_T^*.

De même, $EDC = \mathbb{E}(1_{S_T>K}) = \mathbb{E}(\phi(S_T)) = \mathbb{E}\left(\frac{\phi(S_T^*)f(S_T^*)}{g(S_T^*)}\right)$.
Ainsi,

$$\hat{R} = \frac{\mathbb{E}\left(\frac{h(S_T^*)f(S_T^*)}{g(S_T^*)}\right)}{\mathbb{E}\left(\frac{\phi(S_T^*)f(S_T^*)}{g(S_T^*)}\right)} = \frac{\sum_{i=1}^{n} \frac{h(S_{T}^*{}_i)f(S_{T}^*{}_i)}{g(S_{T}^*{}_i)}}{\sum_{i=1}^{n} \frac{\phi(S_{T}^*{}_i)f(S_{T}^*{}_i)}{g(S_{T}^*{}_i)}}$$

(En utilisant la méthode d'estimation des moments)
avec

$$\frac{f(x)}{g(x)} = \frac{\frac{1}{\sqrt{2\pi T}\sigma X}\exp\left(-\frac{(\ln(X/S_0)+(r-\frac{\sigma^2}{2})T)^2}{2\sigma^2 T}\right)}{\frac{1}{\sqrt{2\pi T}\sigma X}\exp\left(-\frac{(\ln(X/S_0)+(\mu-\frac{\sigma^2}{2})T)^2}{2\sigma^2 T}\right)}$$

En simplifiant

$$\frac{f(x)}{g(x)} = \exp\left(-\frac{(\ln(X/S_0)+(r-\frac{\sigma^2}{2})T)^2}{2\sigma^2 T} + \frac{(\ln(X/S_0)+(\mu-\frac{\sigma^2}{2})T)^2}{2\sigma^2 T}\right)]$$

En posant $e^{l_i} = \frac{f(S_T^*)}{g(S_T^*)}$, on a
$\hat{R} = \frac{\sum_{i=1}^{n} h(S_{T}^*{}_i)\exp(l_i)}{\sum_{i=1}^{n} \phi(S_{T}^*{}_i)\exp(l_i)}$

Par implémentation sur Python cette estimation est quasiment nul pour tout i

Donc le Trick utilisé à partir de l'énoncé est le suivant:
$\hat{R} = \frac{\sum_{i=1}^{n} h(S_{T}^*{}_i)\exp(l_i+C)}{\sum_{i=1}^{n} \phi(S_{T}^*{}_i)\exp(l_i+C)}$

En prenant $C = 500$ (après une étude empirique, celle-ci semble efficace)

```
mu = (np.log(K) - np.log(S0))/T + (sigma**2)/2

# Log-normal distribution
mean = np.log(S0) + (r - (sigma**2)/2)*T
mean_ = np.log(S0) + (mu - (sigma**2)/2)*T
```

```
std = sigma*np.sqrt(T)
ST = np.random.lognormal(mean, std, 10000)
ST_ = np.random.lognormal(mean_, std, 10000)

li = -((np.log(ST_/S0) - (r - (sigma**2)/2)*T)**2/(2*(sigma**2)*T)) + ((np.log(ST_
    ) - np.log(K))/(2*(sigma**2)*T))

C = 500
exp_li = np.exp(li + C)

h = np.maximum(ST_ - K, 0)
phi = np.where(ST_ > K, 1, 0)

# Estimation of the ratio
R = sum(h*exp_li) / sum(phi * exp_li)
print("Estimated ratio:", R)
```


Estimated ratio: 4650.354883340368

Figure 3: Python Output

2.3 Question 3

On remarque que le ratio n'est plus nan (cf. Question 1) et il est estimé à 4650 pour le strike $>> OTM$. L'estimation du ratio semble cohérent, puisque le gain d'un call européen peut etre bien supérieur à celui d'un call digital. Donc Le changement de mesure permet alors de donner un prix, aussi faible soit-il, aux options qui ont une probabilité quasi nulle d'exécution (>> OTM).

3 Exercice 3

3.1 Question 1

Notons EC : le prix d'un European Call et EDC: le prix d'un European Digital Call
$R = \frac{e^{-rT}\mathbb{E}((S_T-K)^+)}{e^{-rT}\mathbb{E}(1_{(S_T-K>0)})}$ (par définition du d'un EC et EDC, et linéarité de l'espérance)
$R = \frac{\mathbb{E}((S_T-K)1_{(S_T>K)})}{\mathbb{E}(1_{(S_T>K)})}$ (en simplifiant)
$R = \frac{\mathbb{E}(S_T 1_{(S_T>K)})-K\mathbb{E}(1_{(S_T>K)})}{\mathbb{E}(1_{(S_T>K)})}$ (linéarité espérance)
$R = \mathbb{E}(S_T|S_T > K) - K$

Donc $R = \mathbb{E}(S_T - K|S_T > K)$ (linéarité de l'espérance)

En posant $\varphi(X) = X - K$
On a $R = \mathbb{E}(\varphi(S_T)|S_T > K)$.
Donc R peut s'écrire comme une espérance conditionnelle sous la forme $\mathrm{E}(\varphi(S_T)|S_T > K)$ avec $\varphi(X) = X-K$ et A = K $\in \mathbb{R}$

3.2 Question 2

Soient $A > 0$ et $y \in \mathbb{R}$
Soit $Y \sim E(\lambda)$ en notant F_Y la fonction de répartition de Y.

$$\begin{aligned}
F_{Y\geq A|Y-A}(y) &= \frac{P(Y-A \leq y, Y \geq A)}{P(Y \geq A)} \\
F_{Y\geq A|Y-A}(y) &= \frac{P(Y \leq y+A) - P(Y \leq A)}{P(Y \geq A)} \\
F_{Y\geq A|Y-A}(y) &= \frac{(1-e^{-\lambda(y+A)}) - (1-e^{-\lambda A})}{e^{-\lambda A}} \\
(car \text{Y} \sim E(\lambda)) \text{ Donc } \mathrm{F}_{Y\geq A|Y-A}(y) &= \begin{cases} 1-e^{-\lambda y} & \text{si } y \in [A, +\infty[\\ 0 & \text{sinon} \end{cases}
\end{aligned}$$

Donc en dérivant par rapport à y

$$\boxed{f_{Y\geq A|Y-A}(y) = \begin{cases} \lambda e^{-\lambda y} & \text{si } y \in [A, +\infty[\\ 0 & \text{sinon} \end{cases}}$$

3.3 Question 3

Soient g la densité de $Y|Y > A$ et $y \in \mathbb{R}$

$$\begin{aligned}
G(y) &= \frac{P(Y \leq y, Y \geq A)}{P(Y \geq A)} \\
G(y) &= \frac{P(A \leq Y \leq y)}{P(Y \geq A)} \\
G(y) &= \frac{P(Y \leq y) - P(Y \leq A)}{P(Y \geq A)} \\
G(y) &= \frac{(1-e^{-\lambda y}) - (1-e^{-\lambda A})}{e^{-\lambda A}} \\
G(y) &= \begin{cases} 1-e^{-\lambda(y-A)} & \text{si } y \in [A, +\infty[\\ 0 & \text{sinon} \end{cases}
\end{aligned}$$

Donc en dérivant par rapport à y

$$\boxed{g(y) = \begin{cases} \lambda e^{-\lambda(y-A)} & \text{si } y \in [A, +\infty[\\ 0 & \text{sinon} \end{cases}}$$

3.4 Question 4

Proposition d'algorithme pour générer $Y|Y > A$:
Soit $y \in \mathbb{R}$ tel que y $\geq A$

$$\sup \left| \frac{f(x)}{g(x)} \right| = \sup \left| \frac{\lambda \cdot e^{-\lambda x}}{\lambda \cdot e^{-\lambda(x-A)}} \right| = \sup |e^{-\lambda A}| \text{ (indépendant de x)}$$

Donc $M = e^{-\lambda A}$.

D'après les Questions 2 et 3,

Si y $\geq A \quad \frac{g(y)}{e^{\lambda A}} = f_{Y \geq A|Y-A}(y)$
$Si y \leq A \quad \frac{g(y)}{e^{\lambda A}} >= f_{Y \geq A|Y-A}(y)$

Donc $\forall y \in \mathbb{R}, \;\; g(y) \cdot M \geq f_{Y \geq A|Y-A}(y)$

$$\boxed{g(y) \cdot M \geq f_{Y \geq A|Y-A}(y)}$$

Comme g est "proche" de $f_{Y \geq A|Y-A}$, les conditions sont respectés pour utiliser la méthode d'acceptance rejection

Méthode d'Acceptance-Rejection:

1. Générer Z de densité g (i.e., $(Y|Y \geq A)$)
2. Générer $u \sim U([0,1])$
3. Si $u \leq \frac{f(Z)}{M*g(Z)}$, Alors posons $X = Z$
4. Sinon, revenir à l'étape 1

3.5 Question 5

Soient $X \sim N(0,1)$ et $x > A$.

$$F_{X|X>A}(x) = \frac{P(X \leq A, X > A)}{P(X > A)} = \frac{F_X(x) - F_X(A)}{1 - F_X(A)}$$

(définition fonction de répartition) Donc $F_{X|X>A}(x) = \begin{cases} \frac{F_X(x) - F_X(A)}{1 - F_X(A)} & \text{si } x > A \\ 0 & \text{sinon} \end{cases}$

Donc $f(x) = \begin{cases} \frac{f_X(x)}{1 - F_X(A)} & \text{si } x > A \\ 0 & \text{sinon} \end{cases}$

Donc $f(x) = K \cdot f_X(x) \cdot \mathbb{I}_{\{X>A\}}(x)$

Donc

$$\boxed{f(x) = K \cdot e^{-\frac{x^2}{2}} \cdot \mathbb{I}_{\{X>A\}}(x), \quad \text{avec } K = \frac{1}{(1 - F_X(A))\sqrt{2\pi}}}$$

3.6 Question 6

Soit $x \in \mathbb{R}$

On cherche $M_{A,\lambda}$ tel que $f(x) \leq M_{A,\lambda} * g(x)$

Posons la fonction $h(x) = \frac{f}{g}(x) = \begin{cases} \frac{K * e^{-\frac{x^2}{2}}}{\lambda e^{-\lambda(x-A)}} & \text{si } x > A \\ 0 & \text{sinon} \end{cases}$

h est de classe C^1 sur $\mathbb{R}$ par composition de fonctions qui le sont et

$$h'(x) = \frac{K}{\lambda}(-x + \lambda)e^{-\frac{x^2}{2} + \lambda(x-A)}$$

On remarque que $h'(\lambda) = 0$

$h(\lambda) = \frac{C}{\lambda} e^{-\frac{\lambda^2}{2} + \lambda(\lambda - A)} = \frac{C}{\lambda} e^{\frac{\lambda^2}{2} - \lambda A}$

Donc $h(x) \leq \frac{1}{\lambda} \frac{e^{\frac{\lambda^2}{2} - \lambda A}}{\int_A^\infty e^{-\frac{u^2}{2}} du} e^{\frac{\lambda^2}{2} - \lambda A + \frac{A^2}{2}}$ (avec $\int_A^\infty e^{-\frac{u^2}{2}} du = (1 - F_X(A))\sqrt{2\pi}$)

Donc $f(x) \leq \frac{1}{\lambda} e^{\frac{1}{2}(\lambda - A)^2} g(x)$, $M_{A,\lambda} = \frac{e^{\frac{1}{2}(\lambda - A)^2}}{\lambda}$

3.7 Question 7

On cherche λ en fonction de A afin d'avoir la distribution conditionnelle $X|X > A$ connaisant la distribution $Y|Y > A$.

Nous avons donc $f(x) \leq M_{A,\lambda} g(x)$ avec $M_{A,\lambda} = \frac{e^{\frac{1}{2}(\lambda - A)^2}}{\lambda}$.

Nous cherchons $min(M_{A,\lambda})$ Pour λ fixé, $min(M_{A,\lambda})$ est atteint pour $\lambda = 1$ tel que $M_{A,1} = \frac{1}{A}$ et $f(x) \leq \frac{1}{A} g(x)$

Donc $f(x) \leq M g(x)$

Ainsi, les conditions sont respéctés pour utiliser la méthode d'Acceptance-Rejection

3.8 Question 8

```python
import numpy as np

def g(y, A, lambda_):
    return lambda_ * np.exp(-lambda_ * (y - A)) if y >= A else 0

def f(y, A):
    # Define the f distribution here, assuming f is the target distribution
    pass

def acceptance_constant(A, lambda_):
    return np.exp(0.5 * (lambda_ - A)**2) / lambda_

def rejection_sampling(A, lambda_):
    M = acceptance_constant(A, lambda_)
    while True:
        Y = np.random.exponential(scale=1/lambda_) + A  # Sample Y from g, assuming g is a shifted exponential distribution
        U = np.random.uniform()  # Generate U from uniform distribution [0, 1]

        if U <= f(Y, A) / (M * g(Y, A, lambda_)):  # Check if the acceptance condition is met
            X = Y
            break

    return X

A = 2
lambda_ = 1

X = rejection_sampling(A, lambda_)
print("Sampled value X:", X)
```

4 Exercice 4

4.1 Question 1

```python
def basis_expansion(k, b):
    a = []
    while k > 0:
        a.append(k % b)
        k = (k - a[-1]) // b
    return a

```

```
print(basis_expansion(5043, 10)) # Output [3, 4, 0, 5]


def van_der_corput_sequence(k, b):
    a = basis_expansion(k, b)
    phi = a[-1]
    for i in range(len(a)-2, -1, -1):
        phi = phi/b + a[i]
    return phi/b

print(van_der_corput_sequence(5043, 10)) # Output 0.3405


def increment_b_ary_expansion(expansion, b):
    carry = 1
    i = 0

    while carry:
        if i < len(expansion):
            expansion[i] += carry
            carry = expansion[i] // b
            expansion[i] %= b
        else:
            expansion.append(carry)
            carry = 0
        i += 1

    return expansion

# Example usage:
k_expansion = [8, 9]  # k = 98 in base 10
b = 10
k_plus_1_expansion = increment_b_ary_expansion(k_expansion, b)
print(k_plus_1_expansion)  # Output: [9, 9] (k+1 = 99 in base 10)

k_expansion = [9, 9]  # k = 99 in base 10
b = 10
k_plus_1_expansion = increment_b_ary_expansion(k_expansion, b)
print(k_plus_1_expansion)  # Output: [0, 0, 1] (k+1 = 100 in base 10)

k_plus_1_expansion = increment_b_ary_expansion(basis_expansion(5048, 10),10)
print("5048 en base 10, k+1 expansion", k_plus_1_expansion) # Output [9, 4, 0, 5]
```

4.2 Question 2

```
def van_der_corput_sequence(b_ary_expansion, b, precision=4):
    phi = b_ary_expansion[-1]
    for i in range(len(b_ary_expansion) - 2, -1, -1):
        phi = phi / b + b_ary_expansion[i]
    return round(phi / b, precision)


def first_k_terms_of_van_der_corput_sequence(k, b):
    b_ary_expansion = [0]
    sequence = []

    for _ in range(k):
```

```python
        sequence.append(van_der_corput_sequence(b_ary_expansion, b))
        b_ary_expansion = increment_b_ary_expansion(b_ary_expansion, b)

    return sequence

# Example
k = 100
b = 10
van_der_corput_sequence_terms = first_k_terms_of_van_der_corput_sequence(k, b)
print(van_der_corput_sequence_terms)
```

Output : Les 100 premiers termes de la suite de Van der Corput ψ^{10}

```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.01, 0.11, 0.21, 0.31, 0.41, 0.51,
0.61, 0.71, 0.81, 0.91, 0.02, 0.12, 0.22, 0.32, 0.42, 0.52, 0.62, 0.72, 0.82, 0.92,
0.03, 0.13, 0.23, 0.33, 0.43, 0.53, 0.63, 0.73, 0.83, 0.93, 0.04, 0.14, 0.24, 0.34,
0.44, 0.54, 0.64, 0.74, 0.84, 0.94, 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75,
0.85, 0.95, 0.06, 0.16, 0.26, 0.36, 0.46, 0.56, 0.66, 0.76, 0.86, 0.96, 0.07, 0.17,
0.27, 0.37, 0.47, 0.57, 0.67, 0.77, 0.87, 0.97, 0.08, 0.18, 0.28, 0.38, 0.48, 0.58,
0.68, 0.78, 0.88, 0.98, 0.09, 0.19, 0.29, 0.39, 0.49, 0.59, 0.69, 0.79, 0.89, 0.99]
```

Figure 4: Output

En utilisant la méthode de Horner

```python
def increment_b_ary_expansion(expansion, b):
    carry = 1
    i = 0

    while carry:
        if i < len(expansion):
            expansion[i] += carry
            carry = expansion[i] // b
            expansion[i] %= b
        else:
            expansion.append(carry)
            carry = 0
        i += 1

    return expansion

def van_der_corput_sequence(b_ary_expansion, b, precision=10):
    phi = 0
    for a in reversed(b_ary_expansion):
        phi = round((phi + a) / b, precision)
    return phi

def first_k_terms_of_van_der_corput_sequence(k, b):
    b_ary_expansion = [0]
    sequence = []

    for _ in range(k):
        sequence.append(van_der_corput_sequence(b_ary_expansion, b))
        b_ary_expansion = increment_b_ary_expansion(b_ary_expansion, b)

    return sequence

# Example usage
```

```python
k = 10
b = 2
van_der_corput_sequence_terms = first_k_terms_of_van_der_corput_sequence(k, b)
print(van_der_corput_sequence_terms)
```

On obtient les même résultats pour les 100 premiers termes de la suite de Van der Corput ψ^{10}

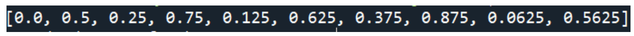

[0.0, 0.5, 0.25, 0.75, 0.125, 0.625, 0.375, 0.875, 0.0625, 0.5625]

Figure 5: résultats obtenues pour les 10 premiers termes de la suite de Van der Corput ψ^2

4.3 Question 3

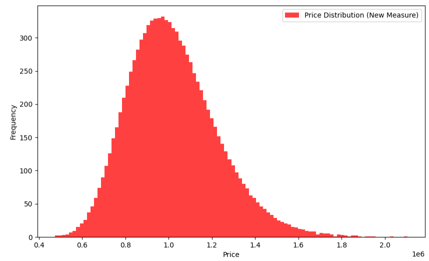
```python
from scipy.stats import norm

def inverse_cdf(p):
    return norm.ppf(p)

def monte_carlo_call_price(S0, K, T, r, sigma, n, phi_sequence):
    total = 0

    for phi in phi_sequence:
        Z = inverse_cdf(phi)
        S_i_Vdp = S0 * np.exp(sigma * np.sqrt(T) * Z + (mu- sigma ** 2 / 2) * T)
        total += max(S_i_Vdp - K, 0)

    return (np.exp(-r * T) / n) * total


def increment_b_ary_expansion(expansion, b):
    carry = 1
    i = 0

    while carry:
        if i < len(expansion):
            expansion[i] += carry
            carry = expansion[i] // b
            expansion[i] %= b
        else:
            expansion.append(carry)
            carry = 0
        i += 1

    return expansion

def van_der_corput_sequence(b_ary_expansion, b, precision=4):
    phi = 0
    for a in reversed(b_ary_expansion):
        phi = round((phi + a) / b, precision)
    return phi

def first_k_terms_of_van_der_corput_sequence(k, b):
    b_ary_expansion = [0]
    sequence = []

    for _ in range(k):
        sequence.append(van_der_corput_sequence(b_ary_expansion, b))
```

```
        b_ary_expansion = increment_b_ary_expansion(b_ary_expansion, b)

    return sequence



# Using Van der Corput sequence from previous exercises
n = 10001
b = 2
phi_sequence = first_k_terms_of_van_der_corput_sequence(n, b)
Z_VDP = []
ST_VDP = []
for i in range(1,len(phi_sequence)):
    Z_VDP.append(norm.ppf(phi_sequence[i]))

for i in range(len(Z_VDP)):
    ST_VDP.append(S0 * math.exp(sigma * np.sqrt(T) * Z_VDP[i] + (mu - sigma**2/2)
        * T))

ST_VDP = np.array(ST_VDP)
h = np.maximum(ST_VDP - K, 0)

li = -((np.log(ST_VDP/S0) - (r - (sigma**2)/2)*T)**2/(2*(sigma**2)*T)) + ((np.log(
    ST_VDP) - np.log(K))/(2*(sigma**2)*T))

h = np.maximum(ST_VDP - K, 0)
C = 400
exp_li = np.exp(li + C)

phi = np.where(ST_VDP > K, 1, 0)

# Estimation of the ratio
R = sum(h * exp_li) / sum(phi * exp_li)
print("Estimated ratio:", R)

# Plot the price distribution under the new measure
plt.figure(figsize=(10, 6))
plt.hist(ST_VDP, bins=100, alpha=0.75, label="Price Distribution (New Measure)",
    color = "red")
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

```
Estimated ratio: 4613.226604808557
```

```
# Plot the price distribution under the new measure
plt.figure(figsize=(10, 6))
plt.hist(ST_VDP, bins=100, alpha=0.75, label="Price Distribution (New Measure)",
    color = "red")
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

4.4 Question 4




```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import math

# Parameters
S0 = 100  # initial stock price
r = 0.01  # risk-free rate
sigma = 0.2   # volatility
T = 1     # time horizon
K = 1000000
mu = (np.log(K) - np.log(S0))/T + (sigma**2)/2

def first_k_terms_of_van_der_corput_sequence(k, b):
    b_ary_expansion = [0]
    sequence = []

    for _ in range(k):
        sequence.append(van_der_corput_sequence(b_ary_expansion, b))
        b_ary_expansion = increment_b_ary_expansion(b_ary_expansion, b)

    return sequence

# Simulate the ST_VDP
n = 10001
b = 2
phi_sequence = first_k_terms_of_van_der_corput_sequence(n, b)
Z_VDP = []
ST_VDP = []

for i in range(1, len(phi_sequence)):
    Z_VDP.append(norm.ppf(phi_sequence[i]))

for i in range(len(Z_VDP)):
    ST_VDP.append(S0 * math.exp(sigma * np.sqrt(T) * Z_VDP[i] + (mu - sigma**2/2) * T))

ST_VDP = np.array(ST_VDP)

# Randomized Quasi Monte Carlo
m = 100  # number of random shifts
n = len(ST_VDP)  # number of simulations

shifts = np.random.uniform(0, 1, size=(m, n))
shifted_phi_sequences = np.array([(phi_sequence[1:] + shift) % 1 for shift in shifts])
```

```

Z_VDP_shifted = np.array([norm.ppf(shifted_phi) for shifted_phi in shifted_phi_sequences])
ST_VDP_shifted = np.array([S0 * np.exp(sigma * np.sqrt(T) * Z_shifted + (mu - sigma**2/2) * T) for Z_shifted in Z_VDP_shifted])

# Calculate the li values and Radon-Nikodym derivative
li = -((np.log(ST_VDP_shifted / S0) - (r - (sigma**2) / 2) * T)**2 / (2 * (sigma**2) * T)) + ((np.log(ST_VDP_shifted) - np.log(K)) / (2 * (sigma**2) * T))
h = np.maximum(ST_VDP_shifted - K, 0)
phi = np.where(ST_VDP_shifted > K, 1, 0)

# Use a larger constant, C = 500
C = 500
exp_li = np.exp(li + C)

# Estimate the ratio for each shifted sequence
R_values = np.array([np.sum(h_i * exp_li_i) / np.sum(phi_i * exp_li_i) for h_i, exp_li_i, phi_i in zip(h, exp_li, phi)])

# Calculate the mean ratio and its standard error
mean_ratio = np.mean(R_values)
std_error = np.std(R_values, ddof=1) / np.sqrt(m)

print("Mean ratio:", mean_ratio)
print("Standard error:", std_error)
```

```
Mean ratio: 4620.902482728402
Standard error: 23.267031938808614
```

Figure 6: Output

5 Exercice 5

5.1 Question 1

Python Code

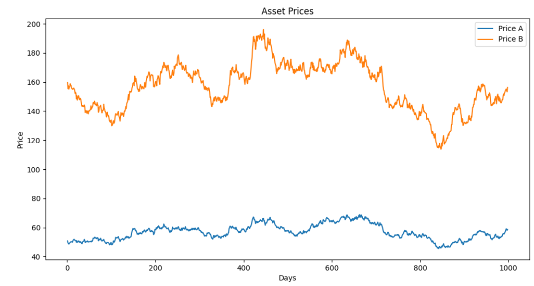
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy.optimize import brentq

# Exercice 5
# Question 1
# Question 1
data = pd.read_csv("data_simulation_methods.csv")


# Nommer les colonnes
data.columns = ["Price A", "Price B"]

```

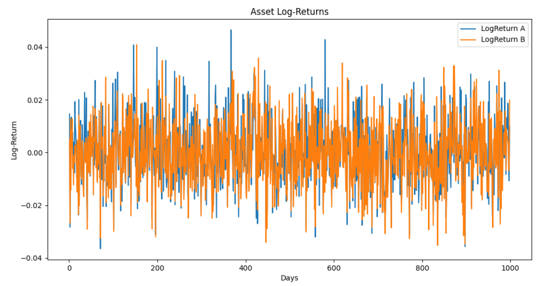
```python
# Calculate daily log-returns for each asset:
data['LogReturnA'] = np.log(data['Price A'] / data['Price A'].shift(1))
data['LogReturnB'] = np.log(data['Price B'] / data['Price B'].shift(1))
data = data.dropna()  # Remove missing values generated from the first log-return calculation

# Calculate drift (mean log-return), volatility (standard deviation of log-returns), and log-return correlation:
drift_A = data['LogReturnA'].mean()
drift_B = data['LogReturnB'].mean()

volatility_A = data['LogReturnA'].std()
volatility_B = data['LogReturnB'].std()

correlation_matrix = data[['LogReturnA', 'LogReturnB']].corr()
correlation = correlation_matrix.iloc[0, 1]


print(f"Drift (Asset A): {drift_A:.6f}")
print(f"Drift (Asset B): {drift_B:.6f}")

print(f"Volatility (Asset A): {volatility_A:.6f}")
print(f"Volatility (Asset B): {volatility_B:.6f}")

print(f"Log-return Correlation: {correlation:.6f}")


# Asset prices plot
plt.figure(figsize=(12, 6))
plt.plot(data['Price A'], label='Price A')
plt.plot(data['Price B'], label='Price B')
plt.xlabel('Days')
plt.ylabel('Price')
plt.title('Asset Prices')
plt.legend()
plt.show()

# Log-returns plot
plt.figure(figsize=(12, 6))
plt.plot(data['LogReturnA'], label='LogReturn A')
plt.plot(data['LogReturnB'], label='LogReturn B')
plt.xlabel('Days')
plt.ylabel('Log-Return')
plt.title('Asset Log-Returns')
plt.legend()
plt.show()
```

```
Drift (Asset A): 0.000159
Drift (Asset B): -0.000016
Volatility (Asset A): 0.012705
Volatility (Asset B): 0.012663
Log-return Correlation: 0.719829
```

Figure 7: Summary statistics of stock A and B

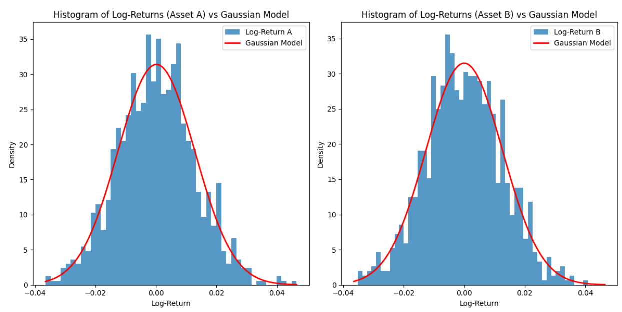
5.2 Question 2






```python
# Calculate mean and standard deviation for each asset's log-returns
mean_A = data['LogReturnA'].mean()
std_A = data['LogReturnA'].std()

mean_B = data['LogReturnB'].mean()
std_B = data['LogReturnB'].std()

# Create a range of x values for the Gaussian model
x_values = np.linspace(data['LogReturnA'].min(), data['LogReturnA'].max(), 100)

# Gaussian model for log-returns
gaussian_A = norm.pdf(x_values, mean_A, std_A)
gaussian_B = norm.pdf(x_values, mean_B, std_B)

# Comparison plots
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(data['LogReturnA'], bins=50, alpha=0.75, density=True, label='Log-Return A')
plt.plot(x_values, gaussian_A, 'r-', lw=2, label='Gaussian Model')
plt.xlabel('Log-Return')
plt.ylabel('Density')
plt.title('Histogram of Log-Returns (Asset A) vs Gaussian Model')
plt.legend()

plt.subplot(1, 2, 2)
plt.hist(data['LogReturnB'], bins=50, alpha=0.75, density=True, label='Log-Return B')
plt.plot(x_values, gaussian_B, 'r-', lw=2, label='Gaussian Model')
plt.xlabel('Log-Return')
plt.ylabel('Density')
plt.title('Histogram of Log-Returns (Asset B) vs Gaussian Model')
plt.legend()
```

```python

plt.tight_layout()
plt.show()
```

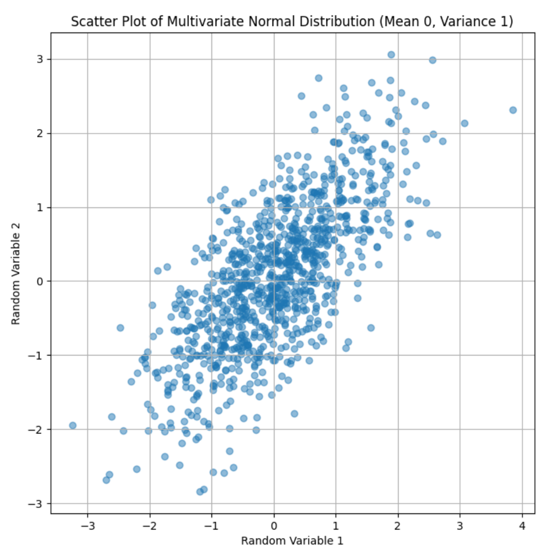


On a vu que les actifs A et B, la moyenne des log-rendements est environ égale à 0 et que la volatilité est environ égale à 1. De plus, on remarque visuellement que l'histogramme des log-returns fit la distribution du modèle Gaussien. Donc choisir un modèle Gaussien pour modéliser la distribution univarié des log-retruns semble etre une bonne idée.

5.3 Question 3

```python
# Question 3

# scatter plot of the log-returns of the two assets:

plt.figure(figsize=(8, 8))
plt.scatter(data['LogReturnA'], data['LogReturnB'], alpha=0.5)
plt.xlabel('Log-Return Asset A')
plt.ylabel('Log-Return Asset B')
plt.title('Scatter Plot of Log-Returns')
plt.grid(True)
plt.show()

# Now, let's create a scatter plot of two multivariate normal random variables
    with mean 0 and variance 1 without using Python packages:

# Number of samples
num_samples = len(data)

# Calculate the correlation coefficient
correlation_coeff = np.corrcoef(data['LogReturnA'], data['LogReturnB'])[0, 1]

# Cholesky decomposition to generate correlated random variables
L = np.array([[1, 0], [correlation_coeff, np.sqrt(1 - correlation_coeff**2)]])

# Generate two uncorrelated standard normal random variables
np.random.seed(42)  # For reproducibility
uncorrelated_normals = np.random.randn(2, num_samples)

# Generate correlated standard normal random variables
correlated_normals = L @ uncorrelated_normals

# Scatter plot
plt.figure(figsize=(8, 8))
plt.scatter(correlated_normals[0], correlated_normals[1], alpha=0.5)
```

```
plt.xlabel('Random Variable 1')
plt.ylabel('Random Variable 2')
plt.title('Scatter Plot of Multivariate Normal Distribution (Mean 0, Variance 1)')
plt.grid(True)
plt.show()

# On voit que les deux scatterplots sont diff rents. Donc on rejette l'hypoth se
    du mod le gaussien bivari  pour d crire la distribution jointe des log-
    returns
```



On observe que les deux scatter plots sont différents donc un modèle Gaussien bivarié n'est pas un modèle réaliste pour représenter la distribution jointe des log-returns des stocks A et B.

5.4 Question 4

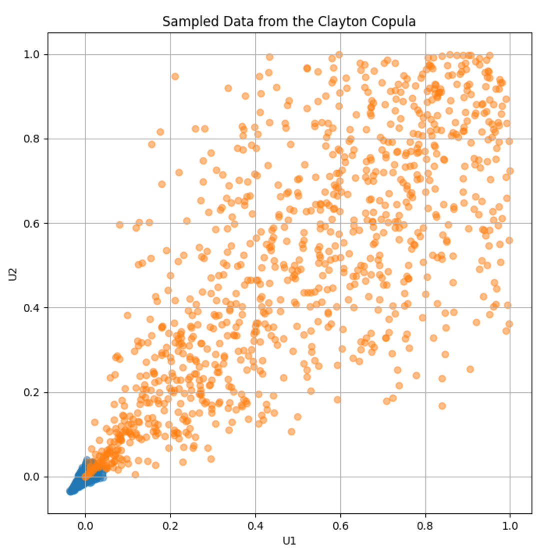
```python
# Question 4

def kendalls_tau(x, y):
    n = len(x)
    num_concordant, num_discordant = 0, 0

    for i in range(n):
        for j in range(i + 1, n):
            sgn_x = np.sign(x[i] - x[j])
            sgn_y = np.sign(y[i] - y[j])

            if sgn_x == sgn_y:
                num_concordant += 1
            else:
                num_discordant += 1

    R_tau = (num_concordant - num_discordant) / (0.5 * n * (n - 1))
    return R_tau

log_returns_A = data['LogReturnA'].values
log_returns_B = data['LogReturnB'].values

tau = kendalls_tau(log_returns_A, log_returns_B)
print(f"Kendall's rank correlation coefficient: {tau:.6f}")
```

```
Kendall's rank correlation coefficient: 0.551854
```

Figure 8:

```python
def clayton_copula(u, v, theta, eps=1e-8):
    return np.maximum((np.power(u + eps, -theta) + np.power(v + eps, -theta) - 1)
        **(-1 / theta), 0)

# Find theta using Kendall's tau
theta = 2*tau/(1 - tau) # Output 2.462830897853369

# Define the partial derivatives of the Clayton copula
def partial_derivative_u(u, v, theta, eps=1e-8):
    if clayton_copula(u, v, theta, eps) != 0:
        return (u + eps)**(-theta - 1) * ((u + eps)**(-theta) + (v + eps)**(-theta
            ) - 1)**((-1 - theta) / theta)
    else:
        return 0

# Define the conditional CDF for the Clayton copula
def conditional_cdf(u, v, theta):
    return partial_derivative_u(u, v, theta)

def conditional_cdf_v_given_u(u, v, theta):
    # print(partial_derivative_u(u, 1, theta)) => equal to 1 OK
    return partial_derivative_u(u, v, theta)

def inverse_conditional_cdf_v_given_u(u, y, theta):
    equation = lambda v: conditional_cdf_v_given_u(u, v, theta) - y
    return brentq(equation, 0, 1)
```

```

def sample_clayton_copula(theta, n_samples=1):
    samples = []
    for _ in range(n_samples):
        # Step 1: Simulate U1 following a uniform distribution in [0, 1]
        u = np.random.uniform()

        # Step 2: Simulate U2 by finding the inverse of the conditional CDF given
            U1 = u
        y = np.random.uniform()
        v = inverse_conditional_cdf_v_given_u(u, y, theta)

        # Step 3: Return the sample (u, v)
        samples.append((u, v))
    return samples


n_samples = len(data)
# Fit a Clayton copula on the data
clayton_samples = sample_clayton_copula(theta, n_samples)


# Plot the sampled data
copula_samples_array = np.array(clayton_samples)
plt.scatter(copula_samples_array[:, 0], copula_samples_array[:, 1], alpha=0.5)

plt.xlabel('U1')
plt.ylabel('U2')
plt.title('Sampled Data from the Clayton Copula')
plt.show()
```



5.5 Question 5

```
# Define the number of future days to simulate
```

```
n_simulations = 1000

# Generate Clayton copula samples
clayton_samples = sample_clayton_copula(theta, n_simulations)

from scipy.interpolate import interp1d

# Calculate the empirical quantile function (inverse CDF) for log-returns of each
    asset
quantile_A = interp1d(np.linspace(0, 1, len(log_returns_A)), np.sort(log_returns_A
    ))
quantile_B = interp1d(np.linspace(0, 1, len(log_returns_B)), np.sort(log_returns_B
    ))

# Generate 1000 Clayton copula samples
n_future_samples = 1000
future_clayton_samples = sample_clayton_copula(theta, n_future_samples)

# Transform the uniform samples from the Clayton copula back to log-returns using
    the empirical quantile function
future_log_returns_A = quantile_A([u for u, v in future_clayton_samples])
future_log_returns_B = quantile_B([v for u, v in future_clayton_samples])

# Compute the simulated prices for the two assets based on the simulated log-
    returns
last_price_A = data['Price A'].iloc[-1]
last_price_B = data['Price B'].iloc[-1]

future_prices_A = np.concatenate(([last_price_A], last_price_A * np.exp(np.cumsum(
    future_log_returns_A))))
future_prices_B = np.concatenate(([last_price_B], last_price_B * np.exp(np.cumsum(
    future_log_returns_B))))

# Plot the simulated prices
future_dates = pd.date_range(start=pd.Timestamp(data.index[-1]) + pd.Timedelta(
    days=1), periods=n_future_samples + 1, freq='D')

plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Price A'], label='Price A - Historical')
plt.plot(data.index, data['Price B'], label='Price B - Historical')
plt.plot(future_dates, future_prices_A, label='Price A - Simulated')
plt.plot(future_dates, future_prices_B, label='Price B - Simulated')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Simulated Asset Prices')
plt.legend()
plt.show()
```

```
# Calculate drift (mean log-return), volatility (standard deviation of log-returns
    ), and log-return correlation for the simulated data
sim_drift_A = future_log_returns_A.mean()
sim_drift_B = future_log_returns_B.mean()

sim_volatility_A = future_log_returns_A.std()
sim_volatility_B = future_log_returns_B.std()

sim_corr_matrix = np.corrcoef(future_log_returns_A, future_log_returns_B)
sim_correlation = sim_corr_matrix[0, 1]

```

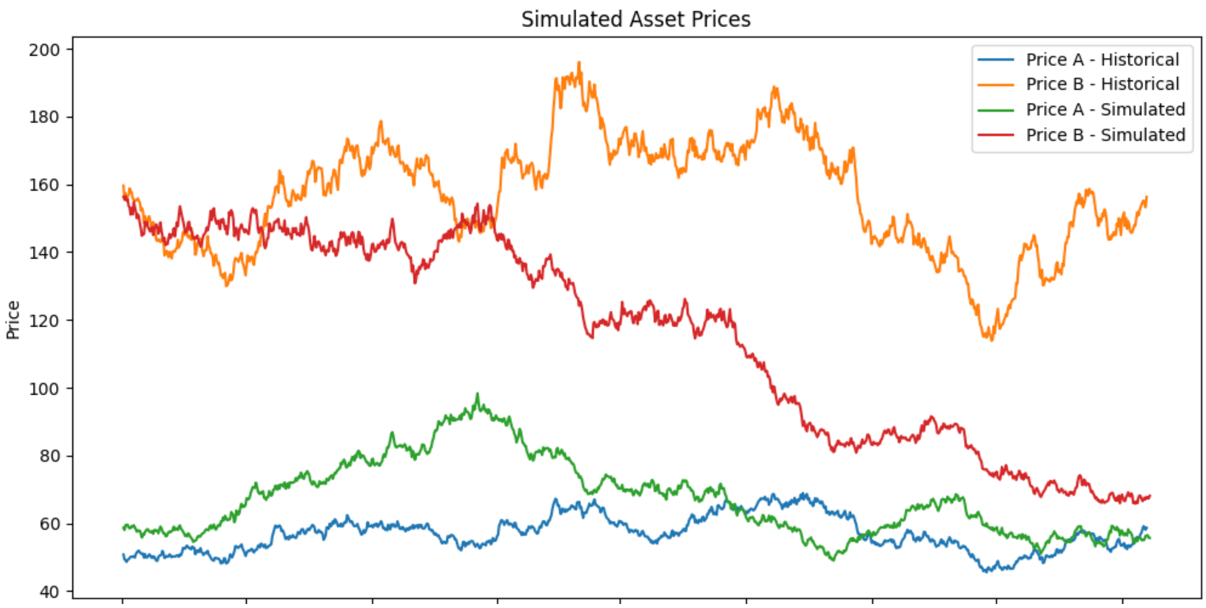


Figure 9:

```
print("Historical Data:")
print(f"Drift (Asset A): {drift_A:.6f}")
print(f"Drift (Asset B): {drift_B:.6f}")

print(f"Volatility (Asset A): {volatility_A:.6f}")
print(f"Volatility (Asset B): {volatility_B:.6f}")

print(f"Log-return Correlation: {correlation:.6f}")

print("\nSimulated Data:")
print(f"Drift (Asset A): {sim_drift_A:.6f}")
print(f"Drift (Asset B): {sim_drift_B:.6f}")

print(f"Volatility (Asset A): {sim_volatility_A:.6f}")
print(f"Volatility (Asset B): {sim_volatility_B:.6f}")

print(f"Log-return Correlation: {sim_correlation:.6f}")
```

```
Simulated Data:
Drift (Asset A): -0.000
Drift (Asset B): -0.001
Volatility (Asset A): 0.012300
Volatility (Asset B): 0.011809
Log-return Correlation: 0.714883
```

Figure 10: Summary statistics Simulated 1000 next days data for the 2 price trajectories

Comparé aux résultats obtenues à la question 1 sur les données de base, les données simulées sont cohérentes.

5.6 Question 6

```python
from scipy.stats import rankdata

def spearmans_rho(x, y):
    n = len(x)
    rank_x = rankdata(x)
    rank_y = rankdata(y)

    sum_diff_sq = np.sum((rank_x - rank_y)**2)
    rho = 1 - (6 * sum_diff_sq) / (n * (n**2 - 1))
    return rho


def bootstrap_correlation(x, y, correlation_func, n_bootstrap_samples=10):
    n = len(x)
    bootstrap_correlations = []

    for _ in range(n_bootstrap_samples):
        indices = np.random.choice(n, size=n, replace=True)
        x_sample = x[indices]
        y_sample = y[indices]

        bootstrap_correlation = correlation_func(x_sample, y_sample)
        bootstrap_correlations.append(bootstrap_correlation)

    return bootstrap_correlations


# Assuming log_returns_A and log_returns_B are the original data, and
# simulated_log_returns_A and simulated_log_returns_B are the simulated data

# Bootstrap Kendall's rank correlation
bootstrap_kendall_original = bootstrap_correlation(log_returns_A, log_returns_B,
    kendalls_tau)
bootstrap_kendall_simulated = bootstrap_correlation(future_log_returns_A,
    future_log_returns_B, kendalls_tau)

# Bootstrap Spearman's rank correlation
bootstrap_spearman_original = bootstrap_correlation(log_returns_A, log_returns_B,
    spearmans_rho)
bootstrap_spearman_simulated = bootstrap_correlation(future_log_returns_A,
    future_log_returns_B, spearmans_rho)

# Compare the bootstrap distributions using histograms
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(bootstrap_kendall_original, bins=30, alpha=0.75, density=True, label='
    Original Data')
plt.hist(bootstrap_kendall_simulated, bins=30, alpha=0.75, density=True, label='
    Simulated Data')
plt.xlabel('Kendall\'s Rank Correlation')
plt.ylabel('Density')
plt.title('Bootstrap Distribution of Kendall\'s Rank Correlation')
plt.legend()

plt.subplot(1, 2, 2)
```

```python
plt.hist(bootstrap_spearman_original, bins=30, alpha=0.75, density=True, label='Original Data')
plt.hist(bootstrap_spearman_simulated, bins=30, alpha=0.75, density=True, label='Simulated Data')
plt.xlabel('Spearman\'s Rank Correlation')
plt.ylabel('Density')
plt.title('Bootstrap Distribution of Spearman\'s Rank Correlation')
plt.legend()

plt.tight_layout()
plt.show()
```

```
[0.54755247707049,
 0.5244330184943609,
 0.5653533747535191,
 0.5382580607554125,
 0.5521353640078552,
 0.5486700582710055,
 0.565461916812562,
 0.5358741555327304,
 0.5439384285119889,
 0.5577715109255622]
```

Figure 11: Kendall's rank correlation using statistical boostrap n sample = 10 on Original Data

```
[0.5052892892892893,
 0.5106306306306306,
 0.5325885885885886,
 0.5358558558558558,
 0.514078078078078,
 0.5355755755755756,
 0.510990990990991,
 0.5228748748748748,
 0.5103103103103103,
 0.5285005005005005]
```

Figure 12: Kendall's rank correlation using statistical boostrap $n_{s}ample = 10onSimulatedData$

```
[0.7216235658657111,
 0.7479703247628265,
 0.7222209757571354,
 0.7682869451240697,
 0.715304138778972,
 0.76143298068995,
 0.7313091488805001,
 0.7297146272810742,
 0.7391875808568773,
 0.7432179547905483]
```

Figure 13: Spearman's rank correlation using statistical boostrap n = 10 on Original Data

```
[0.7271530751530751,
 0.7282872202872203,
 0.7361473841473841,
 0.7094785934785934,
 0.7280434400434401,
 0.7261405501405501,
 0.7313472113472114,
 0.7121658041658041,
 0.7324229524229524,
 0.7459162699162699]
```

Figure 14: Spearman's rank correlation using statistical boostrap n = 10 on Simulated Data

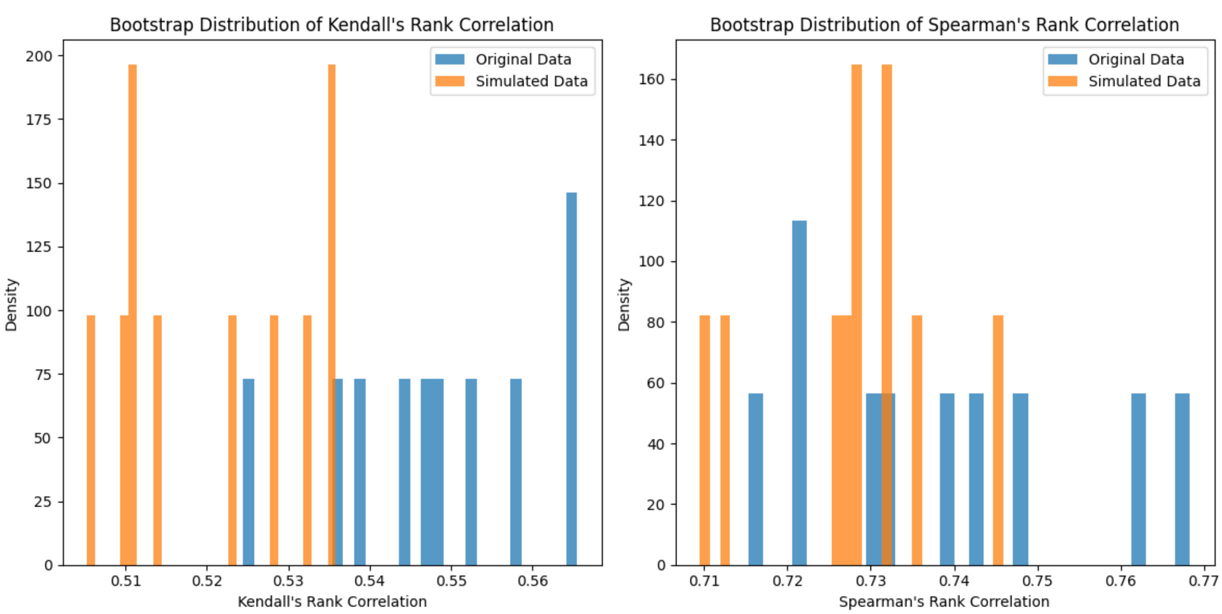


Figure 15: Comparison of Kendall's rank and Spearman's rank using a statistical boostrap