Rapport rédigé par Massinissa SMAÏL, à l'attention de Mme. KORTCHEMSKI

TP n°2 : Simulation du Delta-Hedging dans le marché complet et incomplet
&
TP n°2 bis : Simulation du Delta-Hedging avec un coût de transaction constant

Enoncés, codes, réponses et conclusions

Année scolaire 2023-2024 *ING3, Génie Mathématique, Option MMF*

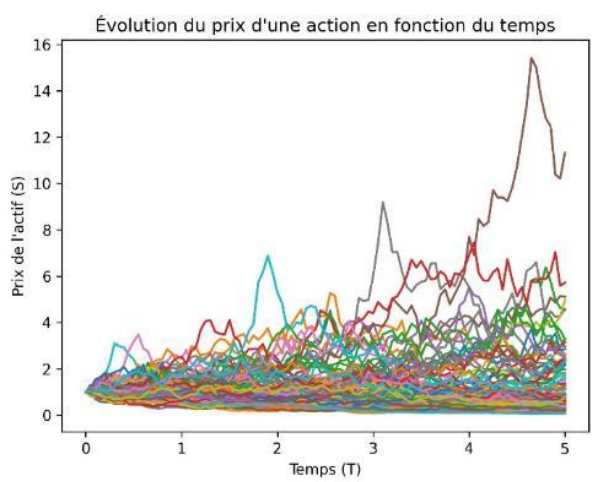
TP n°2 : Simulation du Delta-Hedging dans le marché complet et incomplet

Partie 1 – Construction d'un portefeuille pour éliminer le risque

Question 1 : Simulation d'évolution du prix d'une action **(TP2_Partie_1_Q1_Q2)**

```python
def repartition(x):
    f = (1 + erf(x / sqrt(2))) / 2
    return f


def d1(t, S, K, T, r, sigma):
    d1 = (log(S / K) + (r + ((sigma ** 2) / 2)) * (T - t)) / (sigma * sqrt(T - t))
    return d1


def d2(t, S, K, T, r, sigma):
    d2 = (log(S / K) + (r - ((sigma ** 2) / 2)) * (T - t)) / (sigma * sqrt(T - t))
    return d2


def call_black_scholes(t, S, K, T, r, sigma):
    if t == T:
        return max(S - K, 0)
    else:
        return S * repartition(d1(t, S, K, T, r, sigma)) - K * exp(-r * (T - t)) * repartition(d2(t, S, K, T, r, sigma))


def vega_black_scholes(t, S, K, T, r, sigma):
    f = (S * sqrt(T - t) * exp(-((d1(t, S, K, T, r, sigma)) ** 2) / 2)) / sqrt(2 * 3.14)
    return f


def delta(t, S, K, T, sigma, r):
    if t == T:
        return 1
    else:
        return repartition(d1(t, S, K, T, sigma, r))
```

```python
def evolution_prix():
    s0 = 1
    N = 100
    r = 0.05
    T = 5
    sigma = 0.5
    somme = 0
    Nmc = 100

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

    dt = T / N
    S = np.zeros(N + 1)
    S[0] = s0
    ST = []
    for j in range(Nmc):
        for i in range(N):
            g = np.random.randn()
            S[i + 1] = S[i] * np.exp((r - sigma ** 2 / 2) * dt + sigma * np.sqrt(dt) * g)
        plt.plot(t, S)
        plt.xlabel("Temps (T)")
        plt.ylabel("Prix de l'actif (S)")
        plt.title("Évolution du prix d'une action en fonction du temps")
        ST.append(S[Nmc])
    moyenne = statistics.mean(ST)
    for i in range(Nmc):
        somme = somme + (ST[i] - moyenne) ** 2
    var = somme / Nmc
    print('Moyenne =', moyenne, "; Variance = ", var)
```

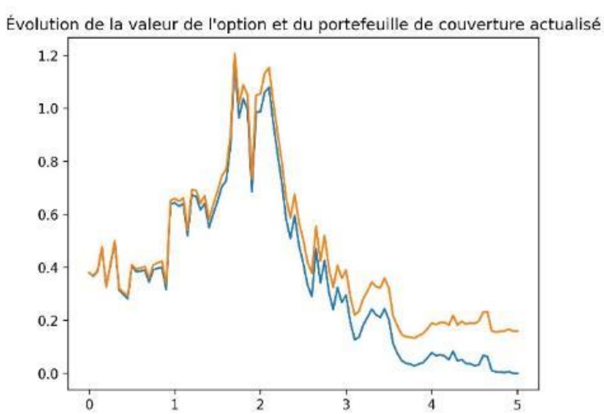


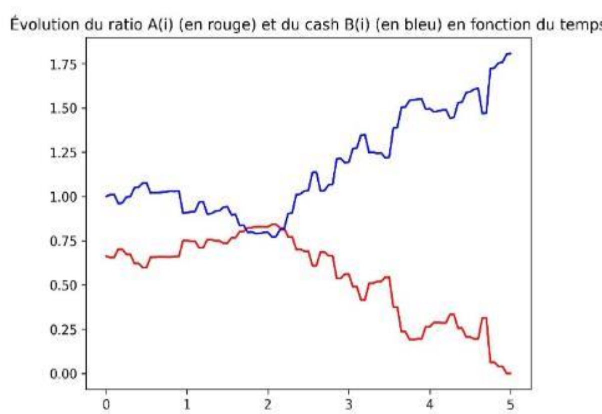

```
Moyenne = 1.3796967710888755 ; Variance =  2.47797412601031
```

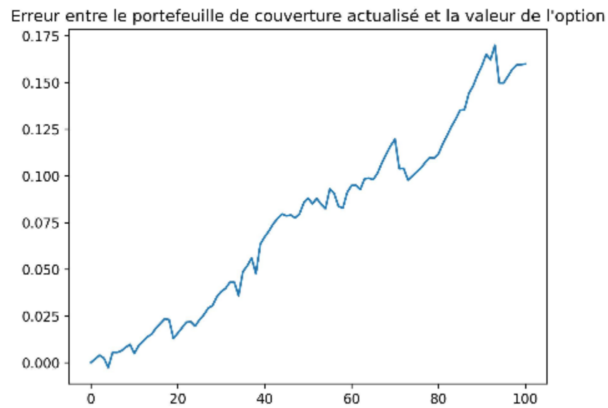
Logiqument nous pouvons remarquer que les N_{mc} chemins d'évolution du prix de l'action ont tendance à converger vers la moyenne empirique trouvé. Cette moyenne s'approche de la valeur théorique $S_0 = 1$.

Question 2 : Simulation d’un portefeuille de couverture **(TP2_Partie_1_Q1_Q2.py)**

```python
def profit_and_loss(Nmc):
    S0, r, K, T, N, sigma = 1, 0.05, 1.5, 5, 100, 0.5
    B0 = 1
    delta_t = T / (N + 1)
    A0 = delta(0, S0, K, T, r, sigma)
    V0 = call_black_scholes(0, S0, K, T, r, sigma)
    P0 = A0 * S0 + B0
    P0_actu = V0
    t = np.linspace(0, T, N + 1)
    A = np.zeros(N + 1)
    V = np.zeros(N + 1)
    S = np.zeros(N + 1)
    B = np.zeros(N + 1)
    P = np.zeros(N + 1)
    P_actu = np.zeros(N + 1)

    PL = np.zeros(Nmc)
    esp = 0
    A[0], B[0], S[0], V[0], P[0], P_actu[0] = A0, B0, S0, V0, P0, P0_actu
    for j in range(Nmc):
        for i in range(0, N):
            S[i + 1] = S[i] * np.exp((r - (sigma ** 2) / 2) * delta_t + sigma * np.sqrt(delta_t) * np.random.randn())
            if i % 2 == 0: # Nous rebalançons le portefeuille une fois sur deux
                A[i + 1] = delta(t[i + 1], S[i + 1], K, T, r, sigma)
            else:
                A[i + 1] = A[i]
            B[i + 1] = (A[i] - A[i + 1]) * S[i + 1] + B[i] * (1 + r * delta_t)
            P[i + 1] = A[i + 1] * S[i + 1] + B[i + 1]
            V[i + 1] = call_black_scholes(t[i + 1], S[i + 1], K, T, r, sigma)
            P_actu[i + 1] = P[i + 1] - (P0 - V0) * np.exp(r * t[i + 1])
        PL[j] = V[N] - P_actu[N]
        esp = esp + PL[j]
    plt.figure()
    plt.plot(t, V, t, P_actu)
    plt.title("Évolution de la valeur de l'option et du portefeuille de couverture actualisé")
    plt.figure()
    plt.plot(t, A, c='red')
    plt.plot(t, B, c='blue')
    plt.title("Évolution du ratio A(i) (en rouge) et du cash B(i) (en bleu) en fonction du temps")
    plt.show()
    plt.figure()
    plt.plot(P_actu - V)
    plt.title("Erreur entre le portefeuille de couverture actualisé et la valeur de l'option")
    plt.show()
    print("L'éspérance vaut :", esp / Nmc)
```







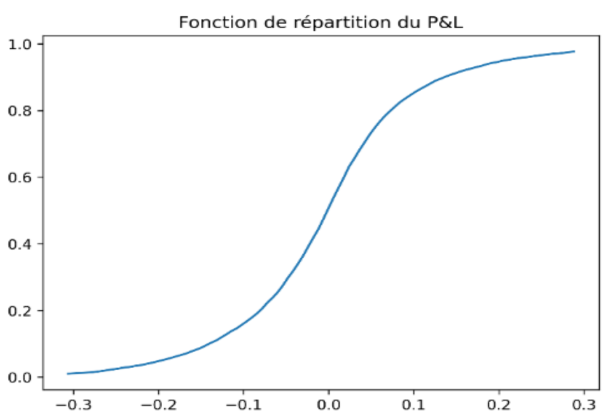

```
L'éspérance vaut : -0.0038800986724179197
```

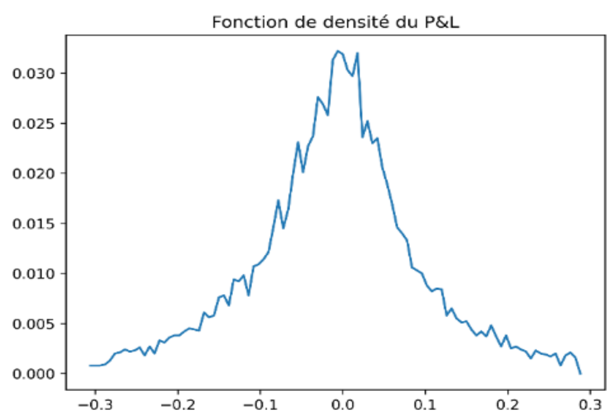
En croisant le premier et le troisième graphique, nous pouvons faire deux conclusions : la première est que le portefeuille est bien couvert mais aussi que cette bonne couverture a tendance à se dégrader au fur et à mesure qu'on avance dans le temps (cependant cela est relatif puisque l'erreur est de l'ordre de 10^{-3}).

En traçant les courbes d'évolution du ratio A (défini par l'énoncé) et du cash B, on remarque que celles-ci sont corrélés négativement (c'est-à-dire que lorsque l'une des deux courbes augmentent, l'autre baisse nécéssairement). Cette conclusion est intuitive, en effet elle correspond à la perte de cash lors de l'augmentation du ratio A et de l'investissement.

Question 3 : Calcul de la moyenne et de la variance du P&L final **(TP2_Partie_1_Q3.py)**

```python
def repartition_densite_pnl():
    N_x = 100
    x = np.zeros(N_x)
    repartition = np.zeros(N_x)
    a = 0.3
    Nmc = 10000
    ProfitAndLoss = np.zeros(Nmc)
    ProfitAndLoss = profit_and_loss(Nmc)
    densite = np.zeros(N_x)
    for i in range(N_x):
        x[i] = -a + (2 * a) / (N_x) * (i - 1) #?????
        compteur = 0
        for n in range(Nmc):
            if ProfitAndLoss[n] <= x[i]:
                compteur = compteur + 1
        repartition[i] = compteur / Nmc
    for j in range(N_x - 1):
        densite[j] = repartition[j + 1] - repartition[j]
    plt.figure()
    plt.plot(x, repartition)
    plt.title("Fonction de répartition du P&L")
    plt.show()
    plt.figure()
    plt.plot(x, densite)
    plt.title("Fonction de densité du P&L")
    plt.show()
    esperance = sum(ProfitAndLoss) / len(ProfitAndLoss)
    somme = 0;
    for k in ProfitAndLoss:
        somme = somme + (k - esperance) ** 2
    variance = somme / len(ProfitAndLoss)
    print("Esperance =", esperance, "; Variance =", variance)
    return ProfitAndLoss
```






```
Esperance = 0.0009131022024605432 ; Variance = 0.015463894837894284
```

Question 4 : L’influence de la fréquence de trading **(TP2_Partie_1_Q4_Q5.py)**

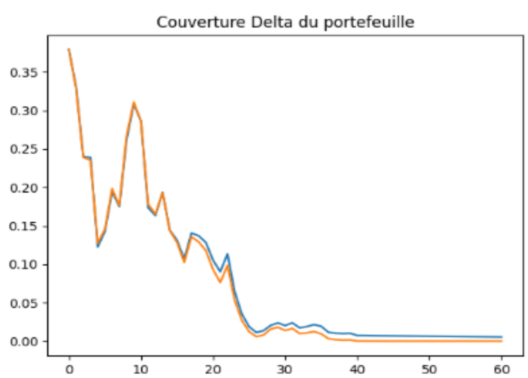
Dans cette question, nous cherchons à démontrer que si le rebalancement du portefeuille de couverture a lieu à des instants discrets, l’erreur de couverture est proportionnelle au Gamma de l’option. Par exemple, si nous souhaitons diviser l’erreur de la couverture par 2, on doit rebalancer le portefeuille 4 fois plus souvent.

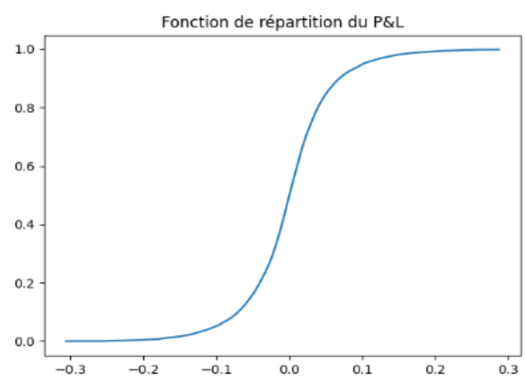
Pour agir sur la fréquence de rebalancement du portefeuille, nous faisons varier la valeur de N_trading.

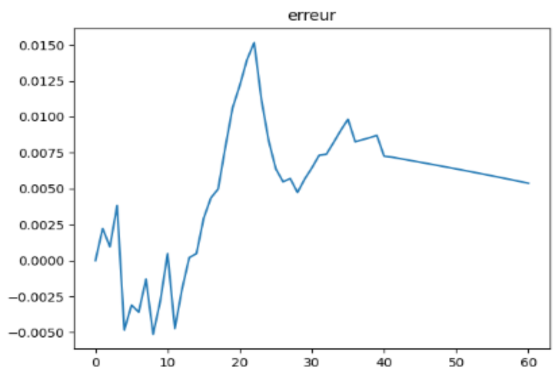
```python
def profit_and_loss(Nmc):
    S0, r, K, T, N, sigma = 1, 0.05, 1.5, 5, 60, 0.5
    B0 = 1
    delta_t = T / (N + 1)
    A0 = delta(0, S0, K, T, r, sigma)
    V0 = call_black_scholes(0, S0, K, T, r, sigma)
    P0 = A0 * S0 + B0
    P0_actu = V0
    t = np.linspace(0, T, N + 1)
    A = np.zeros(N + 1)
    V = np.zeros(N + 1)
    S = np.zeros(N + 1)
    B = np.zeros(N + 1)
    P = np.zeros(N + 1)
    P_actu = np.zeros(N + 1)
    PL = np.zeros(Nmc)
    N_trading = 10 # ou 100 ou 50 ou 25 ou 20 ou 5 ou 2 ou 1
    rebalancement = 100 / N_trading
    A[0], B[0], S[0], V[0], P[0], P_actu[0] = A0, B0, S0, V0, P0, P0_actu
    for j in range(Nmc):
        for i in range(0, N):
            S[i + 1] = S[i] * exp((r - (sigma ** 2) / 2) * delta_t + sigma * sqrt(delta_t) * np.random.randn())
            if i % rebalancement == 0:
                A[i + 1] = delta(t[i + 1], S[i + 1], K, T, r, sigma)
            else:
                A[i + 1] = A[i]
            B[i + 1] = (A[i] - A[i + 1]) * S[i + 1] + B[i] * (1 + r * delta_t)
            P[i + 1] = A[i + 1] * S[i + 1] + B[i + 1]
            V[i + 1] = call_black_scholes(t[i + 1], S[i + 1], K, T, r, sigma)
            P_actu[i + 1] = P[i + 1] - (P0 - V0) * exp(r * t[i + 1])
        PL[j] = V[N] - P_actu[N]
    plt.plot(P_actu)
    plt.plot(V)
    plt.title("Couverture Delta du portefeuille")
    plt.show()
    plt.plot(A)
    plt.plot(B)
    plt.title("Affichage du ratio A et du cash B")
    plt.legend()
    plt.show()
    plt.plot(P_actu - V)
    plt.title("Erreur")
    plt.show()
    return (PL)
```

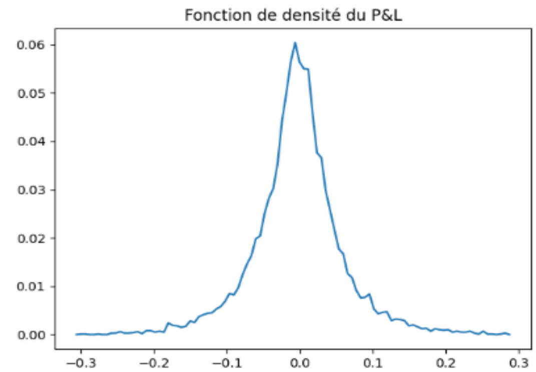
```python
def repartition_densite_pnl():
    N_x = 100
    x = np.zeros(N_x)
    repartition = np.zeros(N_x)
    a = 0.3
    Nmc = 10000
    ProfitAndLoss = np.zeros(Nmc)
    ProfitAndLoss = profit_and_loss(10_000)
    densite = np.zeros(N_x)
    for i in range(N_x):
        x[i] = -a + (2 * a) / (N_x) * (i - 1)
        compteur = 0
        for n in range(Nmc):
            if ProfitAndLoss[n] <= x[i]:
                compteur = compteur + 1
        repartition[i] = compteur / Nmc
    for j in range(N_x - 1):
        densite[j] = repartition[j + 1] - repartition[j]
    plt.figure()
    plt.plot(x, repartition)
    plt.title("Fonction de répartition du P&L")
    plt.show()
    plt.figure()
    plt.plot(x, densite)
    plt.title("Fonction de densité du P&L")
    plt.show()
    esperance = sum(ProfitAndLoss) / len(ProfitAndLoss)
    somme = 0
    for k in ProfitAndLoss:
        somme = somme + (k - esperance) ** 2
    variance = somme / len(ProfitAndLoss)
    print("Espérance =", esperance, "; Variance =", variance)
    return ProfitAndLoss
```

Lorsque N_trading = 100 :



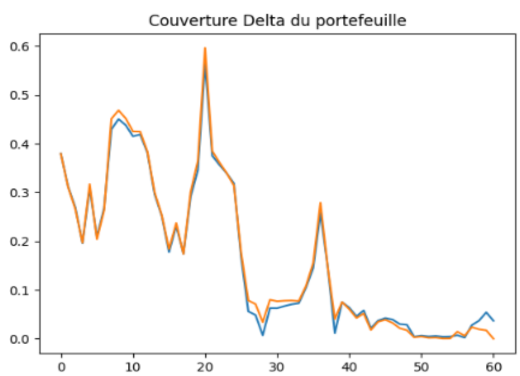


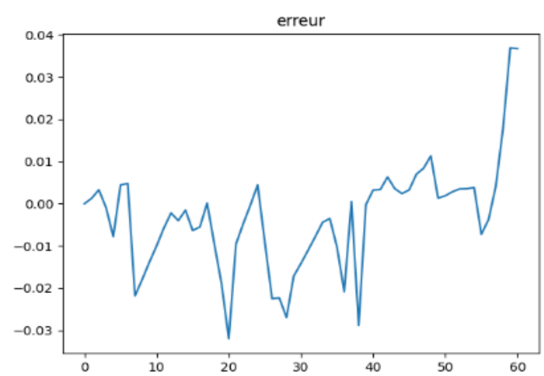


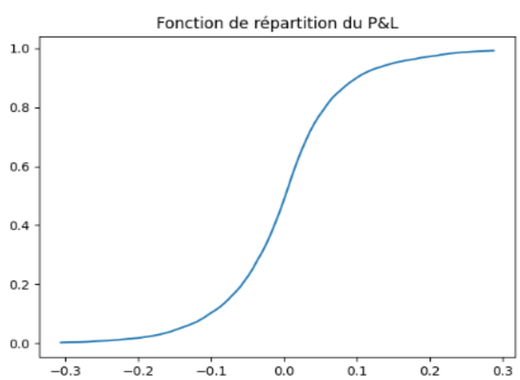


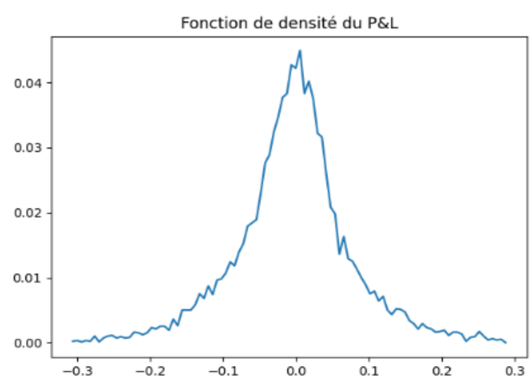

```
Esperance = 0.0007381456714248486 Variance = 0.003909969532986053
```

Lorsque N_trading = 50 :



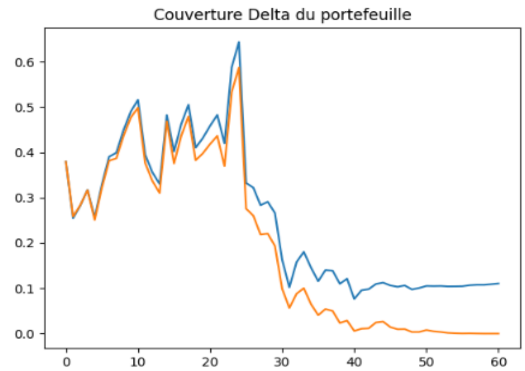


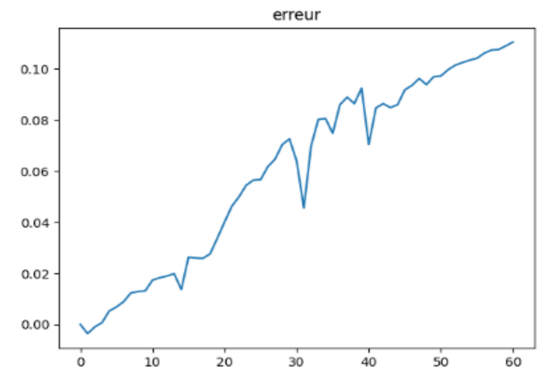


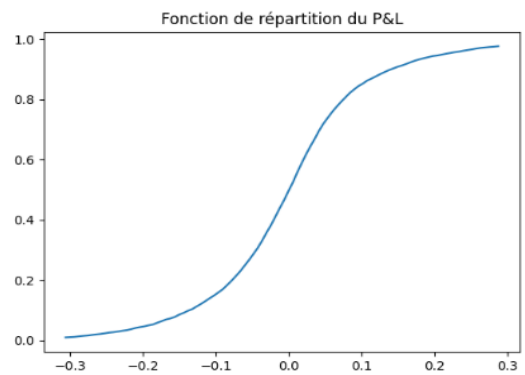


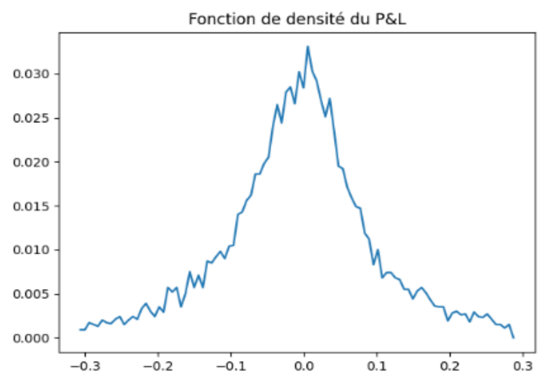

```
Esperance = 0.00181500270863693 Variance = 0.0073961208646755455
```

Lorsque N_trading = 25 :



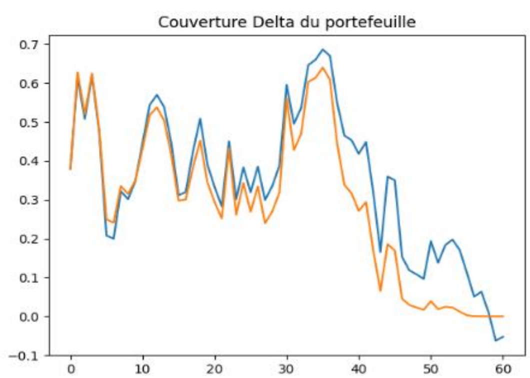


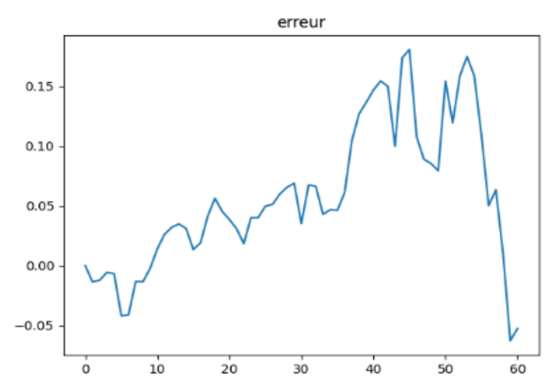


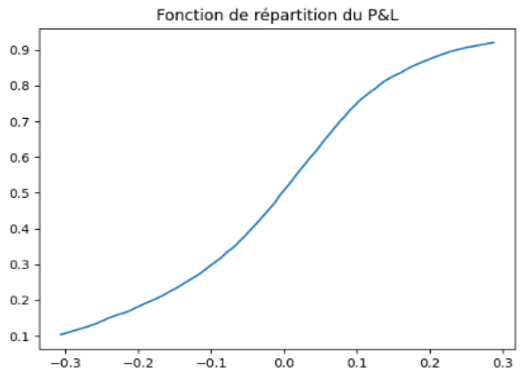


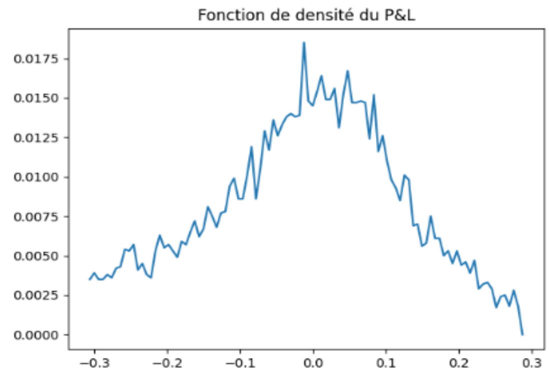

```
Esperance = 0.002862449794343103 Variance = 0.014877684074439299
```

Lorsque N_trading = 5 :



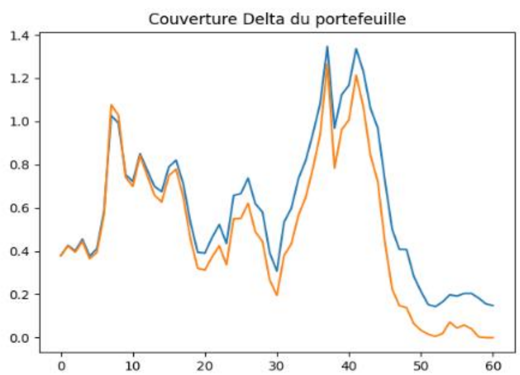


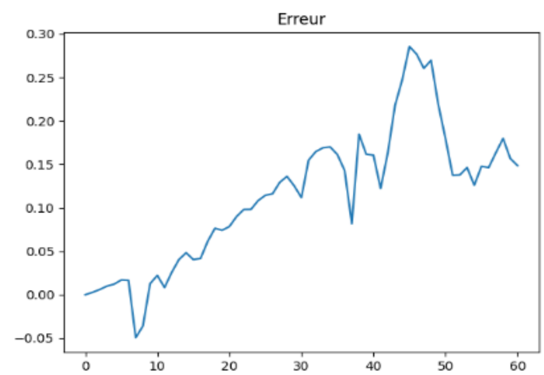


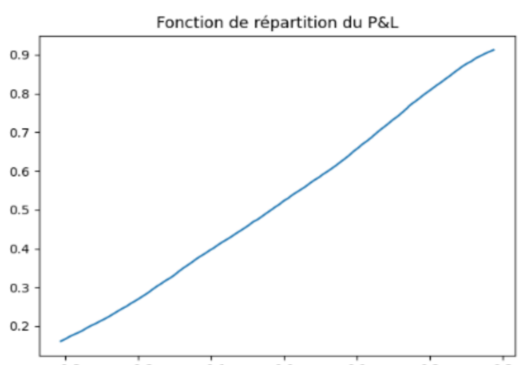


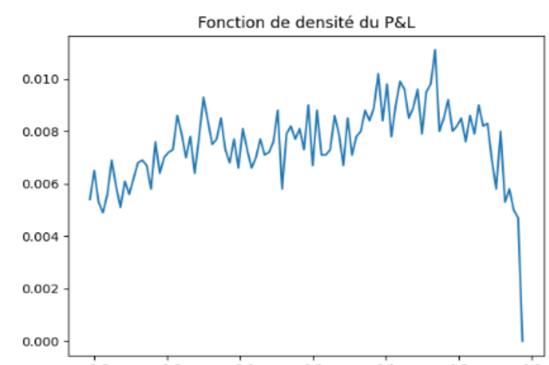

```
Esperance = 0.004705615458435151 Variance = 0.08247803861761167
```

Lorsque N_trading = 2 :



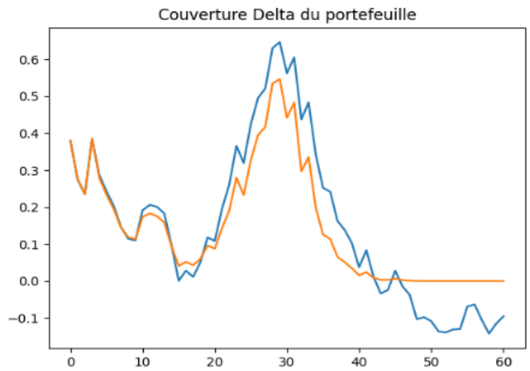


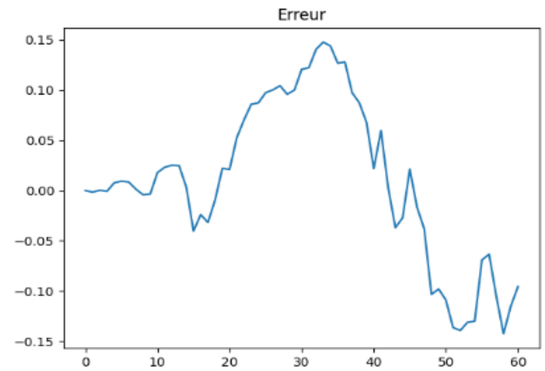


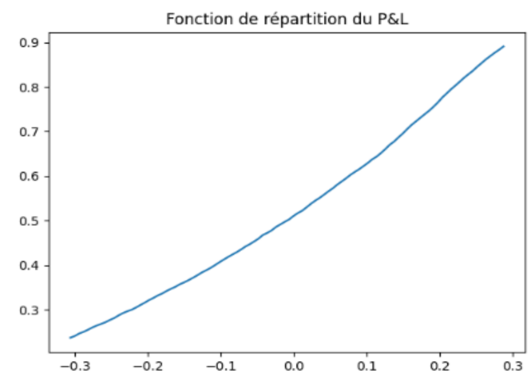


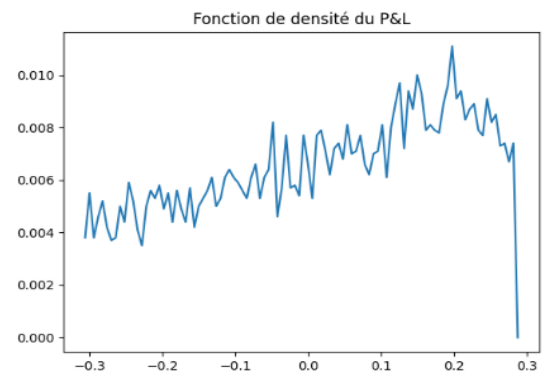

```
Esperance = 0.004613963025914282 Variance = 0.21869725951924437
```

Lorsque N_trading = 1 :










```
Esperance = 0.002321413929876715 Variance = 0.7224225467859092
```

A partir de ces observations nous remarquons que moins le portefeuille est rebalancé, moins la couverture est efficace (en d'autres termes, l'erreur devient de plus en plus grande). Aussi, moins le portefeuille est rebalancé, moins les fonctions de répartitions et de densité sont précises. Dernièrement, un portefeuille moins souvent balancé implique une variance plus grande.

Question 5 : VaR (Value at Risk) **(TP2_Partie_1_Q4_Q5.py)**

La Value-At-Risk représente la perte potentielle maximale d’un investisseur sur la valeur d’un actif ou d’un portefeuille d’actifs financiers qui ne devrait être atteinte qu’avec une probabilité donnée sur un horizon donné. Elle est, en d'autres termes, la pire perte attendue sur un horizon de temps donné pour un certain niveau de confiance.

```
def VaR(Nmc, alpha):
    K = alpha * Nmc
    ProfitAndLoss = np.zeros(Nmc)
    ProfitAndLoss = profit_and_loss(Nmc)
    ProfitAndLoss = sorted(ProfitAndLoss)
    print("VaR =", ProfitAndLoss[int(K)])


profit_and_loss(10000);
repartition_densite_pnl();
VaR(10000, 0.1);
```

Si on hedge à tous les delta_t on a :

```
VaR = -0.07228392345179402
```

Si on hedge une fois sur 10 alors :

```
VaR = -0.22342127747321494
```

La VaR est plus élevée lorsque hedging est moins fréquent, ce qui est cohérent avec les résultats constatés dans la question précédente.

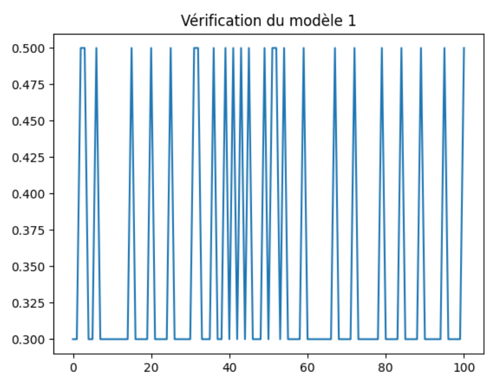
Question 6 : Modèle à volatilité stochastique **(TP2_Partie_1_Q6.py)**

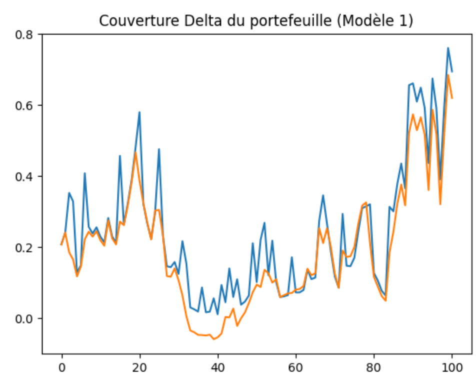
Modèle 1 :

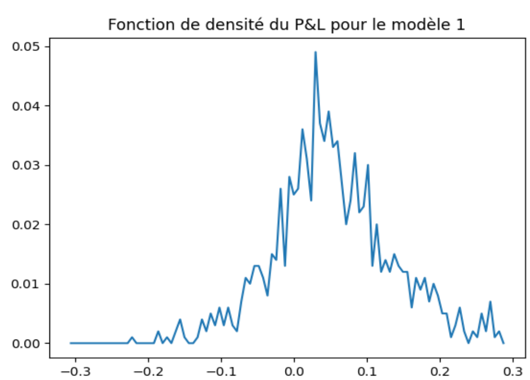
```python
def pnl_modele_1(Nmc):
    S0, r, K, T, N = 1, 0.05, 1.5, 5, 100
    B0 = 1
    t = np.linspace(0, T, N + 1)
    A = np.zeros(N + 1)
    V = np.zeros(N + 1)
    S = np.zeros(N + 1)
    B = np.zeros(N + 1)
    P = np.zeros(N + 1)
    sigma = np.zeros(N + 1)
    delta_t = T / (N + 1)
    sigma1 = 0.3;
    sigma2 = 0.5;
    if np.random.rand() < 0.8:
        sigma[0] = sigma1
    else:
        sigma[0] = sigma2
    P_actu = np.zeros(N + 1)
    PL = np.zeros(Nmc)
    P_actu = np.zeros(N + 1)
    A0 = delta(0, S0, K, T, r, sigma[0])
    V0 = call_black_scholes(0, S0, K, T, r, sigma[0])
    P0 = A0 * S0 + B0
    P0_actu = V0
    A[0], B[0], S[0], V[0], P[0], P_actu[0] = A0, B0, S0, V0, P0, P0_actu

    for j in range(Nmc):
        for i in range(0, N):
            U = np.random.rand()
            if U < 0.8:
                sigma[i + 1] = sigma1
            else:
                sigma[i + 1] = sigma2
            S[i + 1] = S[i] * math.exp(
                (r - (sigma[i + 1] ** 2) / 2) * delta_t + sigma[i + 1] * math.sqrt(delta_t) * np.random.randn())
            # si on veut rebalancer une fois sur deux
            if i % 4 == 0:
                A[i + 1] = delta(t[i + 1], S[i + 1], K, T, r, sigma[i + 1])
            else:
                A[i + 1] = A[i]
            B[i + 1] = (A[i] - A[i + 1]) * S[i + 1] + B[i] * (1 + r * delta_t)
            P[i + 1] = A[i + 1] * S[i + 1] + B[i + 1]
            V[i + 1] = call_black_scholes(t[i + 1], S[i + 1], K, T, r, sigma[i + 1])
            P_actu[i + 1] = P[i + 1] - (P0 - V0) * math.exp(r * t[i + 1])
        PL[j] = V[N] - P_actu[N]
```

```python
    plt.plot(sigma)
    plt.title("Vérification du modèle 1")
    plt.show()
    plt.plot(V)
    plt.plot(P_actu)
    plt.title("Couverture Delta du portefeuille (Modèle 1)")
    plt.show()
    #plt.plot(A)
    #plt.plot(B)
    #plt.title("Affichage de A et B")
    #plt.legend()
    #plt.show()
    plt.plot(P_actu)
    plt.title("Portefeuille actualisé")
    plt.show()
    plt.plot(P_actu - V)
    plt.title("Erreur du modèle 1")
    plt.show()
    return (PL)
```

```python
def repartition_densite_pnl_sigma_variable_modele_1():
    N_x = 100
    x = np.zeros(N_x)
    repartition = np.zeros(N_x)
    a = 0.3
    Nmc = 1000
    ProfitAndLoss = pnl_modele_1(Nmc)
    densite = np.zeros(N_x)
    for i in range(N_x):
        x[i] = -a + (2 * a) / (N_x) * (i - 1)
        compteur = 0
        for n in range(Nmc):
            if ProfitAndLoss[n] <= x[i]:
                compteur = compteur + 1
        repartition[i] = compteur / Nmc
    for j in range(N_x - 1):
        densite[j] = repartition[j + 1] - repartition[j]
    plt.figure()
    plt.plot(x, repartition)
    plt.title("Fonction de répartition du P&L pour le modèle 1")
    plt.show()
    plt.figure()
    plt.plot(x, densite)
    plt.title("Fonction de densité du P&L pour le modèle 1")
    plt.show()
```



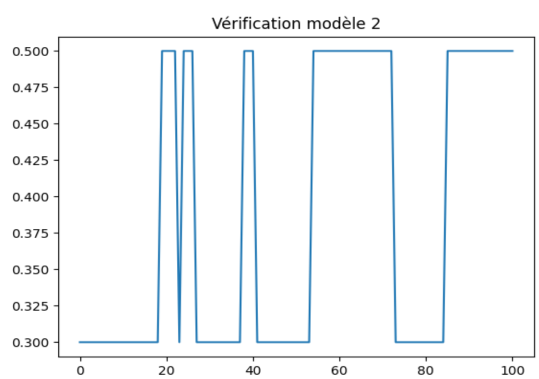


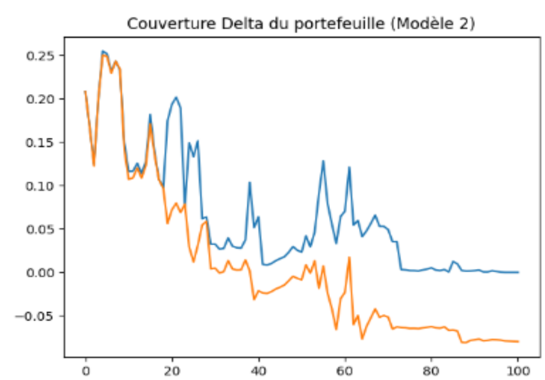


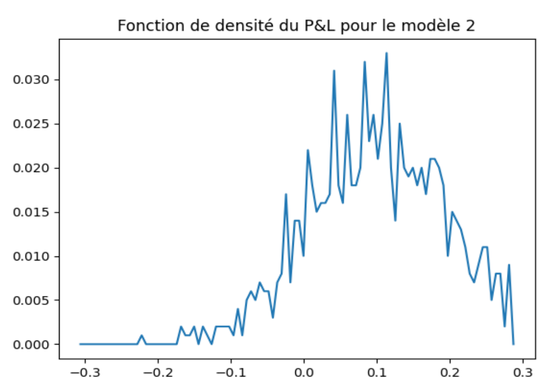
Modèle 2 :

```
def pnl_modele_2(Nmc):
    S0, r, K, T, N = 1, 0.05, 1.5, 5, 100
    B0 = 1
    t = np.linspace(0, T, N + 1)
    A = np.zeros(N + 1)
    V = np.zeros(N + 1)
    S = np.zeros(N + 1)
    B = np.zeros(N + 1)
    P = np.zeros(N + 1)
    sigma = np.zeros(N + 1)
    delta_t = T / (N + 1)
    sigma1 = 0.3;
    sigma2 = 0.5;
    sigma[0] = sigma1;
    P_actu = np.zeros(N + 1)
    PL = np.zeros(Nmc)
    P_actu = np.zeros(N + 1)
    A0 = delta(0, S0, K, T, r, sigma[0])
    V0 = call_black_scholes(0, S0, K, T, r, sigma[0])
    P0 = A0 * S0 + B0
    P0_actu = V0
    A[0], B[0], S[0], V[0], P[0], P_actu[0] = A0, B0, S0, V0, P0, P0_actu

    for j in range(Nmc):
        for i in range(0, N):
            U = np.random.rand()
            if U < 0.05:
                if sigma[i] == sigma1:
                    sigma[i + 1] = sigma2
                else:
                    sigma[i + 1] = sigma1
            else:
                sigma[i + 1] = sigma[i]
            S[i + 1] = S[i] * math.exp(
                (r - (sigma[i + 1] ** 2) / 2) * delta_t + sigma[i + 1] * math.sqrt(delta_t) * np.random.randn())
            if i % 2 == 0: # si on veut rebalancer une fois sur deux
                A[i + 1] = delta(t[i + 1], S[i + 1], K, T, r, sigma[i + 1])
            else:
                A[i + 1] = A[i]
            # A[i+1]=delta(t[i+1],S[i+1],K,T,r,sigma[i+1])
            B[i + 1] = (A[i] - A[i + 1]) * S[i + 1] + B[i] * (1 + r * delta_t)
            P[i + 1] = A[i + 1] * S[i + 1] + B[i + 1]
            V[i + 1] = call_black_scholes(t[i + 1], S[i + 1], K, T, r, sigma[i + 1])
            P_actu[i + 1] = P[i + 1] - (P0 - V0) * math.exp(r * t[i + 1])
        PL[j] = V[N] - P_actu[N]
    plt.plot(sigma)
    plt.title("Vérification du modèle 2")
    plt.show()
    plt.plot(V)
    plt.plot(P_actu)
    plt.title("Couverture Delta du portefeuille (Modèle 2)")
    plt.show()
    #plt.plot(A)
    #plt.plot(B)
    #plt.title("Affichage de A et B")
    #plt.legend()
    #plt.show()
    plt.plot(P_actu)
    plt.title("Portefeuille actualisé")
    plt.show()
    plt.plot(P_actu - V)
    plt.title("Erreur du modèle 2")
    plt.show()
    return (PL)
```

```python
def repartition_densite_pnl_sigma_variable_modele_2():
    N_x = 100
    x = np.zeros(N_x)
    repartition = np.zeros(N_x)
    a = 0.3
    Nmc = 1000
    ProfitAndLoss = pnl_modele_2(Nmc)
    densite = np.zeros(N_x)
    for i in range(N_x):
        x[i] = -a + (2 * a) / (N_x) * (i - 1)
        compteur = 0
        for n in range(Nmc):
            if ProfitAndLoss[n] <= x[i]:
                compteur = compteur + 1
        repartition[i] = compteur / Nmc
    for j in range(N_x - 1):
        densite[j] = repartition[j + 1] - repartition[j]
    plt.figure()
    plt.plot(x, repartition)
    plt.title("Fonction de répartition du P&L pour le modèle 2")
    plt.show()
    plt.figure()
    plt.plot(x, densite)
    plt.title("Fonction de densité du P&L pour le modèle 2")
    plt.show()


pnl_modele_1(10000);
pnl_modele_2(10000);
repartition_densite_pnl_sigma_variable_modele_1();
repartition_densite_pnl_sigma_variable_modele_2();
```







Nous constatons que le modèle 1 permet une nette meilleure couverture que le modèle 2.

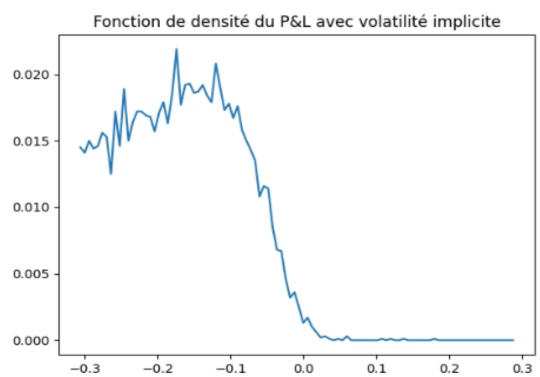
Question 7 : Modèle à volatilité stochastique **(TP2_Partie_1_Q7.py)**

```python
def pnl(Nmc):
    S0, r, K, T, N = 1, 0.05, 1.5, 5, 60
    sigma_h = pnl_modele_1(Nmc)
    sigma_imp = 0.5
    B0 = 1
    delta_t = T / (N + 1)
    A0 = delta(0, S0, K, T, r, sigma_imp)
    V0 = call_black_scholes(0, S0, K, T, r, sigma_imp)
    P0 = A0 * S0 + B0
    P0_actu = V0
    t = np.linspace(0, T, N + 1)
    A = np.zeros(N + 1)
    V = np.zeros(N + 1)
    S = np.zeros(N + 1)
    B = np.zeros(N + 1)
    P = np.zeros(N + 1)
    P_actu = np.zeros(N + 1)
    PL = np.zeros(Nmc)
    A[0], B[0], S[0], V[0], P[0], P_actu[0] = A0, B0, S0, V0, P0, P0_actu

    for j in range(Nmc):
        for i in range(0, N):
            S[i + 1] = S[i] * math.exp(
                (r - (sigma_h[i] ** 2) / 2) * delta_t + sigma_h[i] * math.sqrt(delta_t) * np.random.randn())
            # si on veut rebalancer une fois sur deux
            if i % 4 == 0:
                A[i + 1] = delta(t[i + 1], S[i + 1], K, T, r, sigma_imp)
            else:
                A[i + 1] = A[i]
            B[i + 1] = (A[i] - A[i + 1]) * S[i + 1] + B[i] * (1 + r * delta_t)
            P[i + 1] = A[i + 1] * S[i + 1] + B[i + 1]
            V[i + 1] = call_black_scholes(t[i + 1], S[i + 1], K, T, r, sigma_imp)
            P_actu[i + 1] = P[i + 1] - (P0 - V0) * math.exp(r * t[i + 1])
        PL[j] = V[N] - P_actu[N]
    return (PL)
```

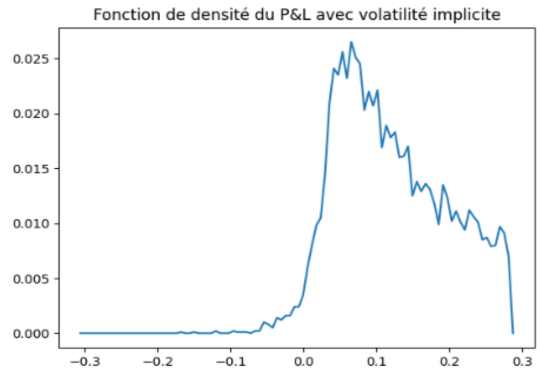
```python
def pnl_modele_1(Nmc):
    sigma = np.zeros(Nmc)
    sigma1 = 0.3
    sigma2 = 0.5
    if np.random.rand() < 0.8:
        sigma[0] = sigma1
    else:
        sigma[0] = sigma2
    for i in range(Nmc):
        U = np.random.rand()
        if U < 0.8:
            sigma[i] = sigma1
        else:
            sigma[i] = sigma2
    return sigma
```

```python
def repartition_densite_pnl():
    N_x = 100
    x = np.zeros(N_x)
    repartition = np.zeros(N_x)
    a = 0.3
    Nmc = 10000
    ProfitAndLoss = np.zeros(Nmc)
    ProfitAndLoss = pnl(Nmc)
    densite = np.zeros(N_x)
    for i in range(N_x):
        x[i] = -a + (2 * a) / (N_x) * (i - 1)
        compteur = 0
        for n in range(Nmc):
            if ProfitAndLoss[n] <= x[i]:
                compteur = compteur + 1
        repartition[i] = compteur / Nmc
    for j in range(N_x - 1):
        densite[j] = repartition[j + 1] - repartition[j]
    plt.figure()
    plt.plot(x, repartition)

    plt.title("Fonction de répartition du P&L avec volatilité implicite")
    plt.show()
    plt.figure()
    plt.plot(x, densite)
    plt.title("Fonction de densité du P&L avec volatilité implicite")
    plt.show()


repartition_densite_pnl()
```

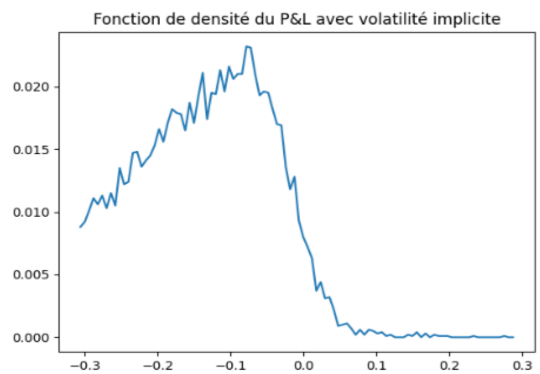
Pour $\sigma_{implicite}$ = 0.5 et $\sigma_{historique}$ = 0.3 :



Pour $\sigma_{implicite}$ = 0.3 et $\sigma_{historique}$ = 0.5 :



Pour $\sigma_{implicite}$ = 0.4 et $\sigma_{historique}$ = Modèle 1 :



Partie 2 – Construction d’un portefeuille qui réplique l’option (TP2_Partie_2.py)

Nous observons que l’égalité $P_0 = V_0$ est vérifiée jusqu’à la maturité de l’option.

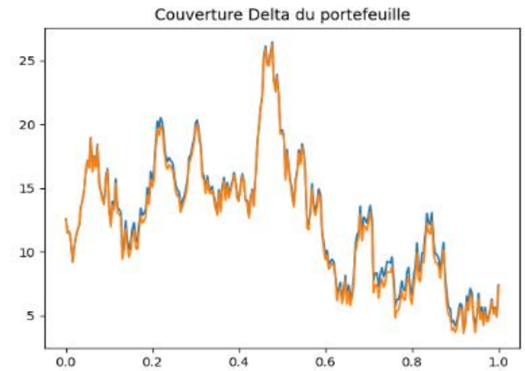
TP n°2 bis : Simulation du Delta-Hedging avec un coût de transaction constant

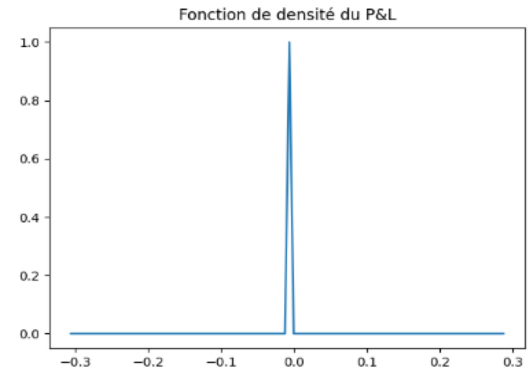
Partie 2 – Simulations

```python
def profit_and_loss(Nmc):
    S0, r, K, T, N, sigma, cost = 100, 0.05, 100, 1, 1040, 0.25, 0.01
    delta_t = T / N
    A0 = delta(0, S0, K, T, r, sigma, cost, N)
    V0 = call_black_scholes(0, S0, K, T, r, sigma, cost, N)
    P0 = V0 - cost * np.abs(A0) * S0
    P0_actu = V0
    B0 = V0 - A0 * S0 - cost * np.abs(A0) * S0
    t = np.linspace(0, T, N + 1)
    A = np.zeros(N + 1)
    V = np.zeros(N + 1)
    S = np.zeros(N + 1)
    B = np.zeros(N + 1)
    P = np.zeros(N + 1)
    P_actu = np.zeros(N + 1)
    PL = np.zeros(Nmc)
    A[0], B[0], S[0], V[0], P[0], P_actu[0] = A0, B0, S0, V0, P0, P0_actu

    for j in range(Nmc):
        for i in range(0, N):
            S[i + 1] = S[i] * math.exp(
                (r - (sigma ** 2) / 2) * delta_t + sigma * math.sqrt(delta_t) * np.random.randn())
            if i % 4 == 0:
                A[i + 1] = delta(t[i + 1], S[i + 1], K, T, r, sigma, cost, N)
            else:
                A[i + 1] = A[i]
            # A[i+1]=delta(t[i+1],S[i+1],K,T,r,sigma,cost,N)
            B[i + 1] = (A[i] - A[i + 1]) * S[i + 1] + B[i] * (1 + r * delta_t)
            P[i + 1] = A[i] * S[i + 1] + B[i] * (1 + r * delta_t) - cost * np.abs(A[i + 1] - A[i]) * S[i + 1]
            V[i + 1] = S[i + 1] * repartition(d1(t[i + 1], S[i + 1], K, T, r, sigma, cost, N)) - K * math.exp(
                -r * (T - t[i + 1])) * repartition(d2(t[i + 1], S[i + 1], K, T, r, sigma, cost, N))
            P_actu[i + 1] = P[i + 1] - (P0 - V0) * math.exp(r * t[i + 1])
        # PL[j]=V[N]-P_actu[N]
    plt.plot(t, V)
    plt.plot(t, P_actu)
    plt.title("Couverture Delta du portefeuille")
    plt.show()
    return (PL)
```

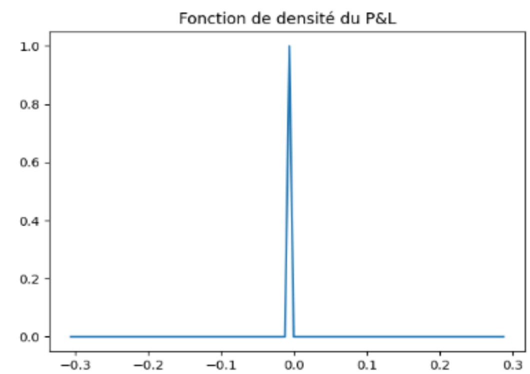
```python
def repartition_densite_pnl():
    N_x = 100
    x = np.zeros(N_x)
    repartition = np.zeros(N_x)
    a = 0.3
    Nmc = 100
    ProfitAndLoss = np.zeros(Nmc)
    ProfitAndLoss = profit_and_loss(Nmc)
    densite = np.zeros(N_x)
    for i in range(N_x):
        x[i] = -a + (2 * a) / (N_x) * (i - 1)
        compteur = 0
        for n in range(Nmc):
            if ProfitAndLoss[n] <= x[i]:
                compteur = compteur + 1
        repartition[i] = compteur / Nmc
    for j in range(N_x - 1):
        densite[j] = repartition[j + 1] - repartition[j]
    plt.figure()
    plt.plot(x, repartition)
    plt.title("Fonction de répartition du P&L")
    plt.show()
    plt.figure()
    plt.plot(x, densite)
    plt.title("Fonction de densité du P&L")
    plt.show()


repartition_densite_pnl();
```

Pour Δt = 1/260 avec un rebalancement chaque jour, K = 100, κ = 0.001 :

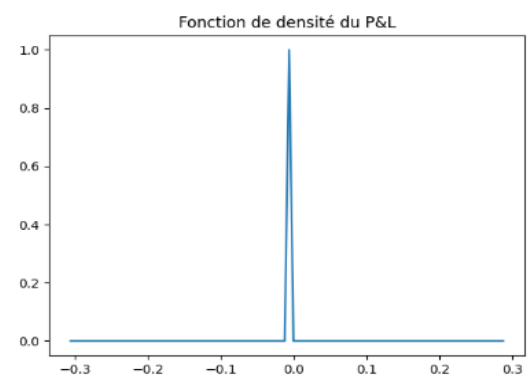




Pour Δt = 1/1040 avec un rebalancement quatre fois par jour, K = 100, κ = 0.001 :



Pour Δt = 1/260 avec un rebalancement chaque jour, K = 100, κ = 0.01 :



Pour Δt = 1/1040 avec un rebalancement quatre fois par jour, K = 100, κ = 0.01 :