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

TP n°1 : Calibration de volatilité implicite dans le modèle de Black et Scholes & Smile de volatilité

Enoncés, codes, réponses et conclusions

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

Question 1 : Tracer le smile à partir de la base de données de couples (K, prix d'option) (LIFFE).

La volatilité implicite est une estimation de la volatilité future du sous-jacent. Elle est comprise dans la prime de l'option. Ce paramètre est présent dans la formule de Black-Scholes, il est le seul que nous ne pouvons pas observer, d'où l'intérêt de sa calibration.

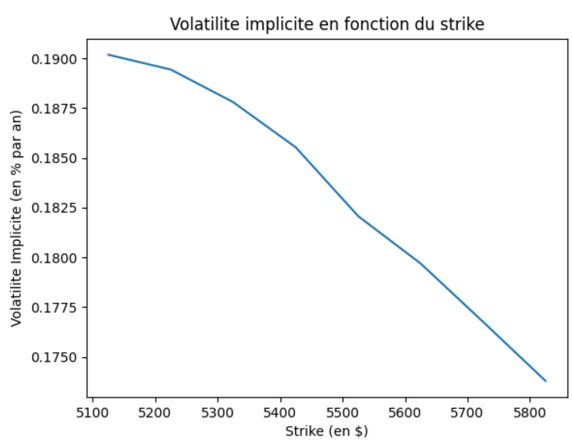
Dans ce contexte, la calibration de ce paramètre signifie que nous devons trouver la volatilité implicite qui satisfait l'égalité : Valeur du modèle = Valeur du marché (ou bien s'en rapproche le plus possible).

Par le cours, nous savons que le modèle de Black-Scholes implique que la volatilité implicite de toutes les options sur le même produit sous-jacent doit être la même, et égale à la volatilité historique du sous-jacent. Nous en déduisons alors que la volatilité implicite dépend de 3 principaux facteurs : le strike de l'option (K), la maturité (T) et le prix de l'option.

Dans cette partie, nous avons fixé T, notre étude concernera donc l'impact de l'évolution des deux autres facteurs sur la volatilité implicite.

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


def repartition(x):
    f = 1 / 2 * (1 + math.erf(x / math.sqrt(2)))
    return f


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


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


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


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

def F_call(marche, t, S, K, T, r, sigma):
    f = call_black_scholes(t, S, K, T, r, sigma) - marche
    return f
```

```python
S = []
Call_test = []
vega_test = []


for i in range(1, 101):
    S.append(0.2 * i)
    Call_test.append(call_black_scholes(0, S[i - 1], 10, 1, 0.1, 0.5))
    vega_test.append(vega_black_scholes(0, S[i - 1], 10, 1, 0.1, 0.5))


t, r, epsilon, T, S0 = 0, 0.05, 0.0001, 1 / 3, 5430.3
K = [5125, 5225, 5325, 5425, 5525, 5625, 5725, 5825]
M = [475, 405, 340, 280.5, 226, 179.5, 139, 105]

sigma=(math.sqrt(2*np.abs((math.log(S0/K[0])+r*T)/T))) # Point de départ tel que défini par l'énoncé
volatilite_implicite=np.zeros(8)


for i in range(0, 8):
    # Vérifions que le prix tombe bien dans l'intervalle défini par les contraintes d'arbitrages
    if M[i] < S0 and M[i] >= max(S0 - K[i] * math.exp(-r * T), 0):
        while (np.abs(F_call(M[i], t, S0, K[i], T, r, sigma)) > epsilon):
            # Selon la récurrence de l'algorithme de Newton
            sigma = sigma - F_call(M[i], t, S0, K[i], T, r, sigma) / vega_black_scholes(t, S0, K[i], T, r, sigma)
            volatilite_implicite[i] = (sigma)
    else:
        volatilite_implicite[i] = (0)
```



La figure 1 nous montre que la volatilité implicite d'un Call est correlé négativement avec le strike : lorsque le strike augmente alors la volatilité implicite du Call baisse et lorsque le strike baisse alors la volatilité implicite du Call augmente.

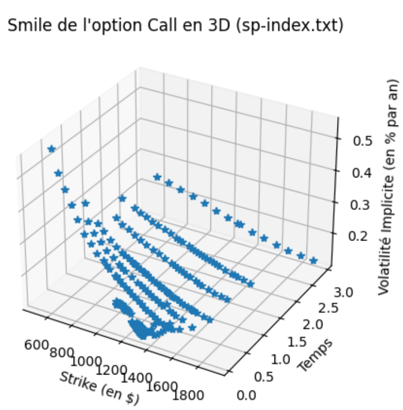
Question 2 : Tracer les smiles pour le Call et pour le Put séparément des bases de données à 3 dimensions (sp-index avec des dividendes). Superposer les graphes.

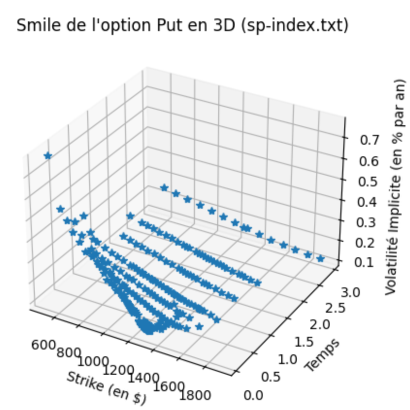
```
for i in range(N):
    Ca = fichier[i][4]
    Cb = fichier[i][5]
    M[i] = (Ca + Cb) / 2
    K[i] = fichier[i][1]
    T[i] = fichier[i][0]
    r[i] = fichier[i][6] * 0.01
    S0[i] = S_0 * math.exp(-q * T[i])
    sigma[i] = (math.sqrt(2 * np.abs((math.log(S0[i] / K[0]) + r[i] * T[i]) / T[i])))

    if (M[i] < K[i] * math.exp(-r[i] * (T[i] - t))) and (M[i] >= max(S0[i] - K[i] * math.exp(-r[i] * T[i]), 0) - S0[i] + K[i] * math.exp(-r[i] * (T[i] - t))):
        while (np.abs(F_Put(M[i], t, S0[i], K[i], T[i], r[i], sigma[i])) > eps):
            sigma[i] = sigma[i] - F_Put(M[i], t, S0[i], K[i], T[i], r[i], sigma[i]) / vega_black_scholes(t, S0[i], K[i], T[i], r[i], sigma[i])
            volatilite_implicite[i] = sigma[i]
    else:
        volatilite_implicite[i] = (0)

volat_nules = np.where(volatilite_implicite == 0)
K = np.delete(K, volat_nules)
T = np.delete(T, volat_nules)
volatilite_implicite = np.delete(volatilite_implicite, volat_nules)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
plt.plot(K, T, volatilite_implicite, '*')
ax.set_xlabel('Strike (en $)')
ax.set_ylabel('Temps')
ax.set_zlabel('Volatilité Implicite (en % par an)')
plt.title("Smile de l'option Put en 3D (sp-index.txt)")
plt.show()
```

```
        if M[i] < S0[i] and M[i] >= max(S0[i] - K[i] * math.exp(-r[i] * T[i]), 0):
            while (np.abs(F_call(M[i], t, S0[i], K[i], T[i], r[i], sigma[i])) > eps):
                sigma[i] = sigma[i] - F_call(M[i], t, S0[i], K[i], T[i], r[i], sigma[i]) / vega_black_scholes(t, S0[i], K[i], T[i], r[i], sigma[i])
                volatilite_implicite[i] = sigma[i]
        else:
            volatilite_implicite[i] = (0)
    volat_nules = np.where(volatilite_implicite == 0)
    K = np.delete(K, volat_nules)
    T = np.delete(T, volat_nules)
    volatilite_implicite = np.delete(volatilite_implicite, volat_nules)
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')  # Affichage de points en 3d
    plt.plot(K, T, volatilite_implicite, '*')
    ax.set_xlabel('Strike (en $)')
    ax.set_ylabel('Temps')
    ax.set_zlabel('Volatilité Implicite (en % par an)')
    plt.title("Smile de l'option Call en 3D (sp-index.txt)")
    plt.show()


f tracer_volatilite_implicite_put_3D_fichiertxt():
    link = "sp-index.txt"
    fichier = np.loadtxt(link)
    q = 0.0217
    eps = 0.0001
    t = 0
    S_0 = 1260.36
    N = len(fichier)
    print(N)
    K = np.zeros(N)
    r = np.zeros(N)
    T = np.zeros(N)
    S0 = np.zeros(N)
    M = np.zeros(N)
    sigma = np.zeros(N)
    volatilite_implicite = np.zeros(N)
```

```python
def put_black_scholes(t, S, K, T, r, sigma):
    if t == T:
        f = max(K - S, 0)
    else:
        f = -S * repartition(-d1(t, S, K, T, r, sigma)) + K * math.exp(-r * (T - t)) * repartition(-d2(t, S, K, T, r, sigma))
    return f

def F_Put(marche, t, S, K, T, r, sigma):
    f = put_black_scholes(t, S, K, T, r, sigma) - marche
    return f


def tracer_volatilite_implicite_call_3D_fichiertxt():
    link = "sp-index.txt"
    fichier = np.loadtxt(link)
    q = 0.0217
    eps = 0.0001
    t = 0
    S_0 = 1260.36
    N = len(fichier)
    print(N)
    K = np.zeros(N)
    r = np.zeros(N)
    T = np.zeros(N)
    S0 = np.zeros(N)
    M = np.zeros(N)
    sigma = np.zeros(N)
    volatilite_implicite = np.zeros(N)

    for i in range(N):
        Ca = fichier[i][3]
        Cb = fichier[i][2]
        M[i] = (Ca + Cb) / 2
        K[i] = fichier[i][1]
        T[i] = fichier[i][0]
        r[i] = fichier[i][6] * 0.01
        S0[i] = S_0 * math.exp(-q * T[i])
        sigma[i] = (math.sqrt(2 * np.abs((math.log(S0[i] / K[0]) + r[i] * T[i]) / T[i])))
```

```python
def tracer_call_black_scholes():
    S = np.zeros(100)
    call_test = np.zeros(100)
    for i in range(1, 100):
        S[i] = 0.2 * i;
        call_test[i] = call_black_scholes(0, S[i], 10, 1, 0.1, 0.5)
    plt.plot(S, call_test, '*')
    plt.xlabel('XXX')
    plt.ylabel('YYY')
    plt.title("Courbe Call")
    plt.show()


def tracer_put_black_scholes():
    S = np.zeros(100)
    put_test = np.zeros(100)
    for i in range(1, 100):
        S[i] = 0.2 * i;
        put_test[i] = put_black_scholes(0, S[i], 10, 1, 0.1, 0.5)
    plt.plot(S, put_test, '*')
    plt.xlabel('XXX')
    plt.ylabel('YYY')
    plt.title("Courbe Put")
    plt.show()

def tracer_vega_black_scholes():
    S = np.zeros(100)
    vega_test = np.zeros(100)
    for i in range(1, 100):
        S[i] = 0.2 * i;
        vega_test[i] = vega_black_scholes(0, S[i], 10, 1, 0.1, 0.5)
    plt.plot(S, vega_test, '*')
    plt.xlabel('XXX')
    plt.ylabel('YYY')
    plt.title("Courbe Vega")
    plt.show()

tracer_volatilite_implicite_call_3D_fichiertxt();
tracer_volatilite_implicite_put_3D_fichiertxt();
```





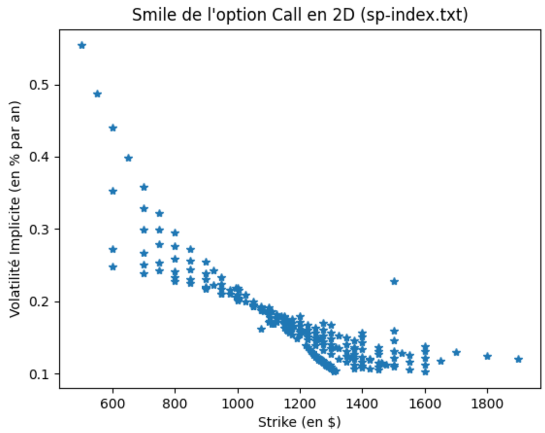
Graphiquement on peut remarquer que plus le temps passe, moins la volatilité implicite varie. On peut aussi remarquer que les smiles de Call et de Put sont très similaires.

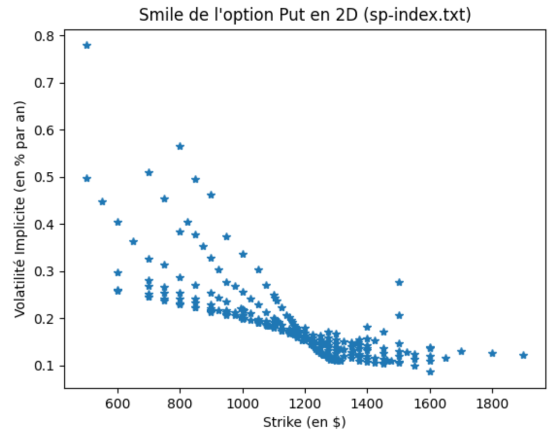
Aussi, la conclusion trouvé dans la question 1 est conservé.

Question 3 : Tracer les smiles pour Call et pour Put de base (sp-index avec des dividendes) de données à 2 dimensions.

```python
def tracer_volatilite_implicite_call_2D_fichiertxt():
    link = "sp-index.txt"
    fichier = np.loadtxt(link)

    q = 0.0217
    eps = 0.0001
    t = 0
    S_0 = 1260.36
    N = len(fichier)
    print(N)
    K = np.zeros(N)
    r = np.zeros(N)
    T = np.zeros(N)
    S0 = np.zeros(N)
    M = np.zeros(N)
    sigma = np.zeros(N)
    volatilite_implicite = np.zeros(N)

    for i in range(N):
        Ca = fichier[i][3]
        Cb = fichier[i][2]
        M[i] = (Ca + Cb) / 2 # Prix mid-quote (bid+ask)/2
        K[i] = fichier[i][1]
        T[i] = fichier[i][0]
        r[i] = fichier[i][6] * 0.01
        S0[i] = S_0 * math.exp(-q * T[i]) #Approximation taux de dividende continu
        sigma[i] = (math.sqrt(2 * np.abs((math.log(S0[i] / K[0]) + r[i] * T[i]) / T[i])))

        if M[i] < S0[i] and M[i] >= max(S0[i] - K[i] * math.exp(-r[i] * T[i]), 0):
            while (np.abs(F_call(M[i], t, S0[i], K[i], T[i], r[i], sigma[i])) > eps):
                sigma[i] = sigma[i] - F_call(M[i], t, S0[i], K[i], T[i], r[i], sigma[i]) / vega_black_scholes(t, S0[i], K[i], T[i], r[i], sigma[i])
                volatilite_implicite[i] = sigma[i]
        else:
            volatilite_implicite[i] = (0)
    volat_nules = np.where(volatilite_implicite == 0)
    K = np.delete(K, volat_nules)
    T = np.delete(T, volat_nules)
    volatilite_implicite = np.delete(volatilite_implicite, volat_nules)
    plt.plot(K, volatilite_implicite, '*')
```

```python
    plt.xlabel('Strike (en $)')
    plt.ylabel('Volatilité Implicite (en % par an)')
    plt.title("Smile de l'option Call en 2D (sp-index.txt)")
    plt.show()


def tracer_volatilite_implicite_put_2D_fichiertxt():
    link = "sp-index.txt"
    fichier = np.loadtxt(link)
    q = 0.0217
    eps = 0.0001
    t = 0
    S_0 = 1260.36
    N = len(fichier)
    print(N)
    K = np.zeros(N)
    r = np.zeros(N)
    T = np.zeros(N)
    S0 = np.zeros(N)
    M = np.zeros(N)
    sigma = np.zeros(N)
    volatilite_implicite = np.zeros(N)
    for i in range(N):
        Ca = fichier[i][4]
        Cb = fichier[i][5]
        M[i] = (Ca + Cb) / 2
        K[i] = fichier[i][1]
        T[i] = fichier[i][0]
        r[i] = fichier[i][6] * 0.01
        S0[i] = S_0 * math.exp(-q * T[i])
        sigma[i] = (math.sqrt(2 * np.abs((math.log(S0[i] / K[0]) + r[i] * T[i]) / T[i])))

        if (M[i] < K[i] * math.exp(-r[i] * (T[i] - t))) and (M[i] >= max(S0[i] - K[i] * math.exp(-r[i] * T[i]), 0) - S0[i] + K[i] * math.exp(-r[i] * (T[i] - t))):
            while (np.abs(F_Put(M[i], t, S0[i], K[i], T[i], r[i], sigma[i])) > eps):
                sigma[i] = sigma[i] - F_Put(M[i], t, S0[i], K[i], T[i], r[i], sigma[i]) / vega_black_scholes(t, S0[i], K[i], T[i], r[i], sigma[i])
                volatilite_implicite[i] = sigma[i]
        else:
            volatilite_implicite[i] = (0)
    volat_nules = np.where(volatilite_implicite == 0)
    K = np.delete(K, volat_nules)
    T = np.delete(T, volat_nules)
    volatilite_implicite = np.delete(volatilite_implicite, volat_nules)
    plt.plot(K, volatilite_implicite, '*')
    plt.xlabel('Strike (en $)')
    plt.ylabel('Volatilité Implicite (en % par an)')
    plt.title("Smile de l'option Put en 2D (sp-index.txt)")
    plt.show()


tracer_volatilite_implicite_call_2D_fichiertxt();
tracer_volatilite_implicite_put_2D_fichiertxt();
```



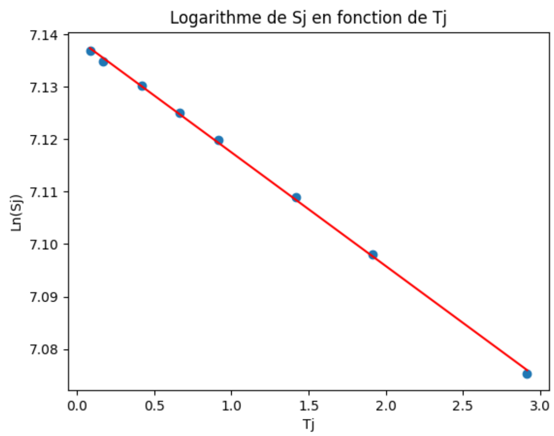


Le smile des option Call et Put en 2D sont eux aussi très similaires (à quelques valeurs extrêmes près).

Question 4 : Calculer S_0 et le taux de dividende q. Tracer une droite de régression linéaire.

```python
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

def regression(T,K,C,P,r):
    c=0
    S_=[]
    T_=[]
    S=C[0]-P[0]+math.exp(-r[0]*T[0])*K[0]

    for i in range(0, len(T)-1):
        if T[i]==T[i+1]:
            c+=1
            S+=C[i+1]-P[i+1]+math.exp(-r[i+1]*T[i+1])*K[i+1]
        else :
            S_.append(S/(c+1))
            T_.append(T[i])
            c = 0
            S = C[i+1]-P[i+1]+math.exp(-r[i+1]*T[i+1])*K[i+1]
    S_.append(S/(c+1))
    T_.append(T[-1])

    for i in range(len(S_)) :
        S_[i] = math.log(S_[i])
    X = np.array(T_).reshape(-1,1)
    Y = np.array(S_)
    model= LinearRegression()
    model.fit(X, Y)
    Beta1 = model.coef_[0]
    Beta2 = model.intercept_
    plt.scatter(X, Y)
    plt.plot(X, Beta1*X+Beta2, color='red')
    plt.xlabel("Tj")
    plt.ylabel("Ln(Sj)")
    plt.title("Logarithme de Sj en fonction de Tj")
    plt.show()
    return -Beta1, math.exp(Beta2)
```

```python
link = "sp-index.txt"
fichier = np.loadtxt(link)

n=len(fichier)
P=np.zeros(n)
C=np.zeros(n)
K=np.zeros(n)
T=np.zeros(n)
r=np.zeros(n)

for i in range(n):
    Pa=fichier[i][4]
    Pb=fichier[i][5]
    P[i]=(Pa+Pb)/2
    Ca = fichier[i][2]
    Cb = fichier[i][3]
    C[i] = (Ca + Cb) / 2
    K[i] = fichier[i][1]
    T[i] = fichier[i][0]
    r[i] = fichier[i][6] / 100

print(regression(T,K,C,P,r));
```




```
(0.021664669661284948, 1260.3666787091668)
```

Voici les valeurs trouvés pour q et S_0 respectivement.

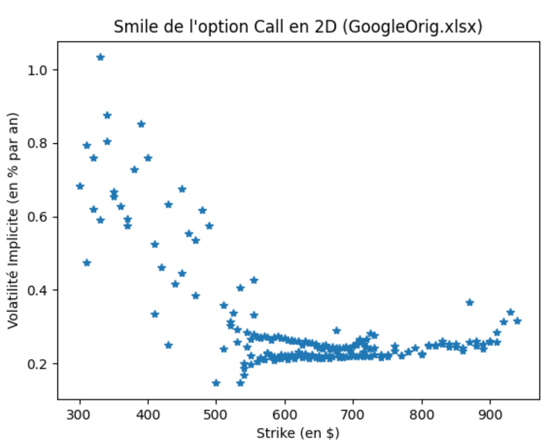
Question 5 : Tracer les smiles de la base de données GoogleOrig.xlsx. Essayez de choisir un taux de dividende q afin d’obtenir de jolies smiles.

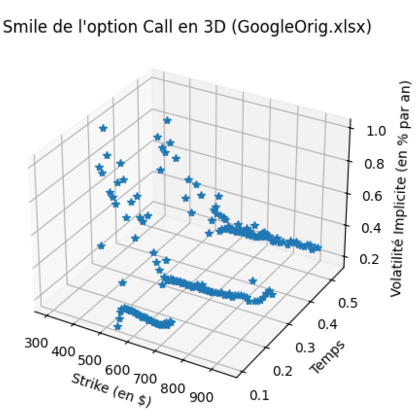
```python
def tracer_volatilite_implicite_call_3D_fichier_Google():
    document = xlrd.open_workbook('GoogleOrig2.xlsx')
    feuille_1 = document.sheet_by_index(0)
    feuille_1 = document.sheet_by_name('Orig')
    cols = feuille_1.ncols
    N = feuille_1.nrows
    q = 0.0217
    eps = 0.0001
    t = 0
    K = []
    r = np.zeros(N)
    T = []
    S0 = np.zeros(N)
    M = [];
    sigma = np.zeros(N)
    volatilite_implicite = np.zeros(N)
    for i in range(0, N):
        r[i] = 0.06
        S0[i] = 591.66
        T += [feuille_1.cell_value(rowx=i, colx=0)]
        K += [feuille_1.cell_value(rowx=i, colx=1)]
        M += [feuille_1.cell_value(rowx=i, colx=2)]
        sigma[i] = (math.sqrt(2 * np.abs((math.log(S0[i] / K[0]) + r[i] * T[i]) / T[i])))
        if M[i] < S0[i] and M[i] >= max(S0[i] - K[i] * math.exp(-r[i] * T[i]), 0):
            while (np.abs(F(M[i], t, S0[i], K[i], T[i], r[i], sigma[i])) > eps):
                sigma[i] = sigma[i] - F(M[i], t, S0[i], K[i], T[i], r[i], sigma[i]) / vega_black_scholes(t, S0[i], K[i], T[i], r[i], sigma[i])
                volatilite_implicite[i] = sigma[i]
        else:
            volatilite_implicite[i] = (0)
    volat_nules = np.where(volatilite_implicite == 0)
    K = np.delete(K, volat_nules)
    T = np.delete(T, volat_nules)
    volatilite_implicite = np.delete(volatilite_implicite, volat_nules)
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')  # Affichage de points en 3d
    ax.set_xlabel('Strike (en $)')
    ax.set_ylabel('Temps')
    ax.set_zlabel('Volatilité Implicite (en % par an)')
    plt.plot(K, T, volatilite_implicite, '*')
    plt.title("Smile de l'option Call en 3D (GoogleOrig.xlsx)")
    plt.show()
```

```python
def tracer_volatilite_implicite_call_2D_fichier_Google():
    document = xlrd.open_workbook('GoogleOrig2.xlsx')
    feuille_1 = document.sheet_by_index(0)
    feuille_1 = document.sheet_by_name('Orig')
    cols = feuille_1.ncols
    N = feuille_1.nrows
    q = 0.0217
    eps = 0.0001
    t = 0
    K = []
    r = np.zeros(N)
    T = []
    S0 = np.zeros(N)
    M = [];
    sigma = np.zeros(N)
    volatilite_implicite = np.zeros(N)
    for i in range(0, N):
        r[i] = 0.06
        S0[i] = 591.66
        T += [feuille_1.cell_value(rowx=i, colx=0)]
        K += [feuille_1.cell_value(rowx=i, colx=1)]
        M += [feuille_1.cell_value(rowx=i, colx=2)]
        sigma[i] = (math.sqrt(2 * np.abs((math.log(S0[i] / K[0]) + r[i] * T[i]) / T[i])))
        if M[i] < S0[i] and M[i] >= max(S0[i] - K[i] * math.exp(-r[i] * T[i]), 0):
            while (np.abs(F(M[i], t, S0[i], K[i], T[i], r[i], sigma[i])) > eps):
                sigma[i] = sigma[i] - F(M[i], t, S0[i], K[i], T[i], r[i], sigma[i]) / vega_black_scholes(t, S0[i], K[i], T[i], r[i], sigma[i])
                volatilite_implicite[i] = sigma[i]
        else:
            volatilite_implicite[i] = (0)
    volat_nules = np.where(volatilite_implicite == 0)
    K = np.delete(K, volat_nules)
    T = np.delete(T, volat_nules)
    volatilite_implicite = np.delete(volatilite_implicite, volat_nules)
    plt.plot(K, volatilite_implicite, '*')
    plt.xlabel('Strike (en $)')
    plt.ylabel('Volatilité Implicite (en % par an)')
    plt.title("Smile de l'option Call en 2D (GoogleOrig.xlsx)")
    plt.show()
```

```python
tracer_volatilite_implicite_call_2D_fichier_Google();
tracer_volatilite_implicite_call_3D_fichier_Google();
```

Pour réaliser ces graphs, j'ai pris q = 0,0217 (comme indiqué dans l'énoncé du TP)



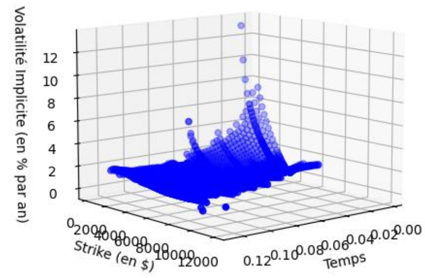


Question 6 : Tracer les smiles de la base spx_quotedata.csv. Tracer séparément les 7 premiers smiles. Expliquer clairement comment vous avez calculé les maturités à partir du traitement de la base de données. Calculer le taux de dividende q via la régression linéaire.

```
    # Suppression des valeurs d'arbitrage et visualisation des "smiles"
    document = document.dropna(subset=["Sigma"])

    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.scatter(document["Time"], document["Strike"], document["Sigma"], c='blue', marker='o')
    ax.view_init(10, 50)
    ax.set_xlabel('Temps')
    ax.set_ylabel('Strike (en $)')
    ax.set_zlabel('Volatilité Implicite (en % par an)')
    ax.set_title("Smile de l'option Call en 3D (spx_quotedata.csv)")
    plt.show()

tracer_volatilite_implicite_call_3D_fichier_CAC40();
```

```
def tracer_volatilite_implicite_call_3D_fichier_CAC40():
    # Chargement et préparation des données CAC40
    document = pd.read_csv("spx_quotedata.csv", delimiter=",")
    q = 0.0217
    S_0 = 3932.69
    document["S0"] = S_0
    document["Marche"] = (document.iloc[:, 4] + document.iloc[:, 5]) / 2
    document["Time"] = 0.0

    # Configuration du temps jusqu'à l'échéance pour chaque option
    TIndex = document.drop_duplicates(subset=['Expiration Date']).index.tolist()
    TIndex.append(9677)
    for i in range(1, len(TIndex)):
        for j in range(TIndex[i-1], TIndex[i]):
            document.at[j, "Time"] = i / 365


    # Nouvelles colonnes df et algorithme de Newton
    document["r"] = 0.0255
    document["Sigma"] = 0.0
    document["Strike"] = document.iloc[:, 11]
    t = 0.0

    initial_sigma = np.sqrt(2 * np.abs(np.log(document["S0"] / document["Strike"]) + document["r"] * document["Time"]) / document["Time"])
    document["Sigma"] = initial_sigma

    for i in range(len(document)):
        if np.max(document.at[i, "S0"] - document.at[i, "Strike"] * np.exp(-document.at[i, "r"] * document.at[i, "Time"]), 0) < document.at[i, "Marche"] < documen
            while np.abs(F_call(document.at[i, "Marche"], t, document.at[i, "S0"], document.at[i, "Strike"],
                                document.at[i, "Time"], document.at[i, "r"], document.at[i, "Sigma"])) > 0.0001:
                document.at[i, "Sigma"] -= F_call(document.at[i, "Marche"], t, document.at[i, "S0"],
                                                  document.at[i, "Strike"], document.at[i, "Time"], document.at[i, "r"],
                                                  document.at[i, "Sigma"]) / vega_black_scholes(
                    t, document.at[i, "S0"], document.at[i, "Strike"], document.at[i, "Time"], document.at[i, "r"], document.at[i, "Sigma"])
        else:
            document.at[i, "Sigma"] = np.nan
```

Smile de l'option Call en 3D (spx_quotedata.csv)




```
(0.021664669661284948, 1260.3666787091668)
```

Question 7 : Analysez les smiles. Conclure.

Les trois conclusions majeures que nous pouvons tirer des précédents smiles sont les suivantes :

1) La volatilité implicite est correlé négativement avec le prix d'exercice d'une option.
2) Qu'il s'agisse d'un call ou d'un put, cela n'affecte peu (voire pas du tout) la forme du smile.
3) Plus nous nous rapprochons de la maturité d'une option moins la volatilité implicite est susceptible de varier en fonction du strike (c'est-à-dire que l'intervalle [volatilité_implicite minimal ; volatilité_implicite maximal] rappetit à mesure que T augmente).