Rapport du projet de calibration

Charlotte Li et Thibault Jaumotte

21 Janvier 2023

Contents

1 Densités risque neutre

1.1 Question 1

Nous avons comme données les prix de calls ayant une maturité d'un an sur une même action de prix actuel 100 :

Strike	95	96	97	98	99	100	101	102	103	104
Prix	12.40	9.59	8.28	7.40	6.86	6.58	6.52	6.49	6.47	6.46

A l'aide de celles-ci, nous calibrerons une densité risque neutre en utilisant la formule de **Breeden-Litzenberger**.

Pour ce faire :

1. A partir du prix des options calculé à l'aide de la formule de Black-Scholes et le prix de ces options sur le marché, on récupère les volatilités implicites propres à chaque option par le biais de l'algorithme de Newton-Raphson. Ainsi, on obtient le dataset suivant (arondi à 10^{-2}) :

Strike	95	96	97	98	99	100	101	102	103	104
Prix	12.40	9.59	8.28	7.40	6.86	6.58	6.52	6.49	6.47	6.46
Volatilité	0.250	0.190	0.170	0.161	0.16	0.165	0.175	0.185	0.195	0.205

2. On interpole ces volatilités implicites avec la méthode d'interpolation de Lagrange, qui permet d'obtenir un polynôme vérifiant :

$$P(X) = \sum_{i=1}^{n} y_i \prod_{\substack{j=i \\ j\neq i}}^{n} \frac{X - x_j}{x_i - x_j} \tag{1}$$

3. A partir de cette interpolation, on calcule de nouveau le prix des options, puis on l'introduit dans la formule de Breeden-Litzenberger pour calibrer la densité risque neutre.

$$q(t) = e^{r(T-t)} \frac{\partial^2 C(t, S_t, K, T)}{\partial K^2}\Big|_{K=S_t} = \frac{C_{i+1} - 2C_i + C_{i-1}}{(\Delta k)^2}$$

Nous comparons la distribution précédente avec une Gaussienne car c'est celle qui se rapproche le plus du modèle de Black-Scholes.
Pour rappel, on a :

$$C(S, K, T) = S_t N(d_1) - e^{-rT} K N(d_2)]$$

$N(d_1)$ et $N(d_2)$ sont les fonctions de répartitions de la loi normale standardisée, où:

$$d_1 = \frac{ln\frac{S}{K} + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \text{ et } d_2 = d_1 - \sigma\sqrt{T} \tag{2}$$

On prend μ = mean(Strike) et σ = std(Strike) comme paramètres initiaux. Puis, on va choisir plus proprement les paramètres μ et σ de la gaussienne de façon à réduire l'écart entre nos deux plots. Pour ce faire, on va utiliser **Kolmogorov-Smirnov** qui est un test statistique utilisé pour comparer une distribution de données à un échantillon de référence. Il mesure l'écart maximal entre les deux distributions de probabilité.

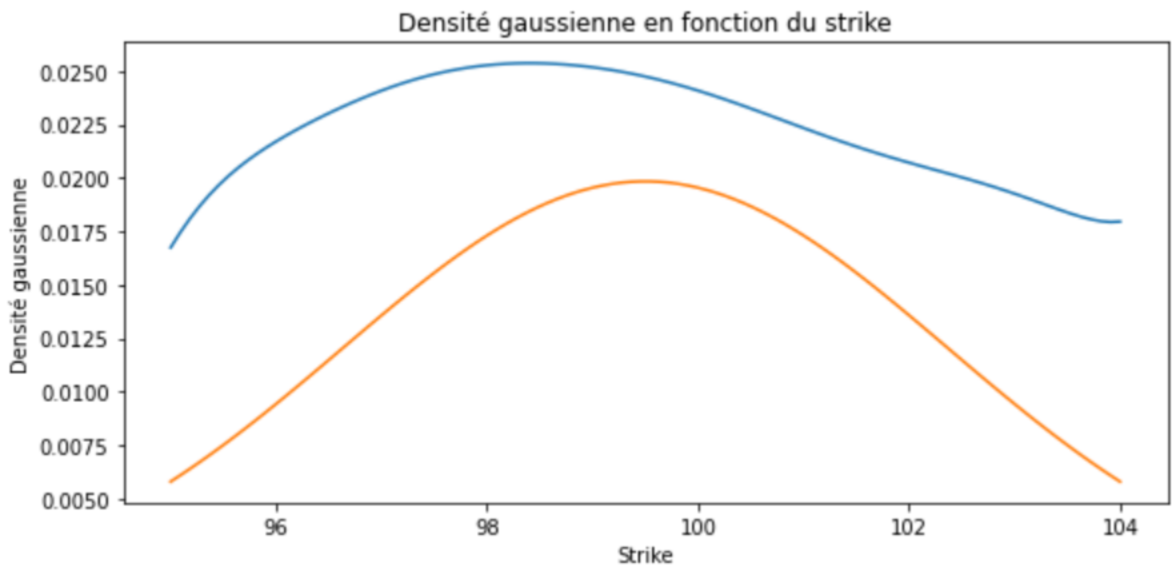
On calcule la distance entre les deux distributions (la densité gaussienne et la densité de risque neutre) en utilisant la formule de Kolmogorov-Smirnov:

$$D = max(|F_{gaussienne}(x) - F_{risque\_neutre}(x)|)$$

où $F_{gaussienne}(x)$ et $F_{risque\_neutre}(x)$ sont respectivement les fonctions de répartition de la densité gaussienne et de la densité risque neutre. On obtient alors les paramètres suivants (arrondis à 10^{-3}:

μ	σ	ratio
99.500	2.872	0.143

Enfin, on multiplie la densité gaussienne par un ratio que l'on a déterminé en essayant différentes valeurs permettant de minimiser les écarts entre la densité risque neutre et la densité gaussienne. On obtient alors les courbes suivantes :



La courbe bleue correspond à la densité risque neutre, et celle en orange représente la courbe de densité gaussienne (avec les paramètres μ et σ et multipliée par le ratio).

1.2 Question 2

Et pour finir, on vérifie si l'on trouve un prix de modèle proche du prix de marché pour les options données au départ. Pour cela, on utilise la méthode de simulation de Monte Carlo dans lequel on tire un échantillon du sous-jacent et on va pricer en utilisant cette formule :

Pour conclure cette question on trouve des prix plutôt proche du marché pour les options précédentes.

```
Found after 12 iterations:
K = 95 and vol = 0.2500 -> Price = 12.39
K = 96 and vol = 0.1901 -> Price = 9.51
K = 97 and vol = 0.1700 -> Price = 8.29
K = 98 and vol = 0.1610 -> Price = 7.37
K = 99 and vol = 0.1601 -> Price = 6.86
K = 100 and vol = 0.1651 -> Price = 6.60
K = 101 and vol = 0.1750 -> Price = 6.52
K = 102 and vol = 0.1851 -> Price = 6.46
K = 103 and vol = 0.1949 -> Price = 6.41
K = 104 and vol = 0.2045 -> Price = 6.40
```

Ces valeurs ont été obtenues à l'aide des instructions suivantes :

1. On fait un nombre noté N de tirages de prix de sous-jacents S à partir de Monte Carlo, à la maturité T de l'option avec le taux sans risque $r = 0$, ce qui donne le vecteur S_T suivant :

$$\begin{gathered} N = 100000 \\ W_T = \sqrt{T} \times \mathcal{N}(0, T, N) \\ S_T = S \times e^{(r - \frac{\sigma^2}{2})T + \sigma W_T} \end{gathered}$$

2. Le prix pour le strike K est alors déterminé grâce à la formule suivante :

$$C_K^n = e^{-rT} \frac{1}{N} \sum_{i=1}^{N} (S_T - K)^+ (3)$$

Aussi, il est bon de préciser qu'une condition a été ajoutée, de sorte à ce que les prix soient cohérents : tant que les prix ne sont pas tous décroissants, on recommence les tirages. Pour des raisons de complexité, on se limite à 30 vérifications de la condition (voir code en annexe).

2 Interpolation et volatilité locale

En plus du tableau donné précédemment, on va utiliser pour cette partie les prix d'options suivants :

- pour des options de maturité 9 mois :

Strike	95	96	97	98	99	100	101	102	103	104
Prix	11.79	8.95	8.07	7.03	6.18	6.04	5.76	5.50	5.50	5.39

- pour des options de maturité 6 mois :

Strike	95	96	97	98	99	100	101	102	103	104
Prix	10.71	8.28	6.91	6.36	5.29	5.07	4.76	4.47	4.35	4.14

- pour des options de maturité 3 mois :

Strike	95	96	97	98	99	100	101	102	103	104
Prix	8.67	7.14	5.98	4.93	4.09	3.99	3.43	3.01	2.72	2.53

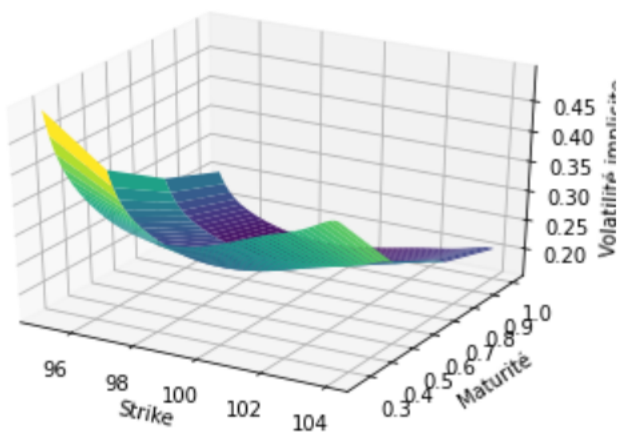
2.1 Question 3

Nous allons afficher la nappe de volatilité correspondant à ces 40 options. Ensuite, nous donnerons un prix, le plus juste possible, pour une option de strike 99.50 et de maturité 8 mois.

Notre première approche était d'utiliser l'algorithme de la question précédente pour déterminer les prix de marché pour les différentes maturités. Pour vérifier notre méthode, nous avons tenté de simuler les prix avec des données (strike et maturité) connues :

```
Found after 0 iterations:
K = 95 and vol = 0.2886 -> Price = 9.24
K = 96 and vol = 0.2195 -> Price = 7.29
K = 97 and vol = 0.1963 -> Price = 6.29
K = 98 and vol = 0.1859 -> Price = 5.51
K = 99 and vol = 0.1848 -> Price = 4.95
K = 100 and vol = 0.1907 -> Price = 4.68
K = 101 and vol = 0.2021 -> Price = 4.37
K = 102 and vol = 0.2137 -> Price = 4.28
K = 103 and vol = 0.2251 -> Price = 4.16
K = 104 and vol = 0.2362 -> Price = 4.05
```

Malheureusement, l'incohérence des résultats obtenus via cette méthode nous a poussé à en opter pour une autre : nous crééons la nappe de volatilités implicites en $3D$ (où l'abscisse représente les strikes, l'ordonnée les maturités et la hauteur correspond aux volatilités implicites). On obtient finalement:



Une fois la volatilité calibrée, on calcule le prix de l'option de strike 99.50 et de maturité 8 mois à partir de la formule de Black Scholes. Pour cela, nous pourrions utiliser la nappe réalisée ci-avant. Nonobstant, cette méthode de lecture graphique est très imprecise, ce qui nous oblige à trouver une méthode d'interpolation, qui nous permettra d'avoir des résultats avec une meilleure précision.
La troisième approche est d'utiliser une interpolation bilinéaire, que l'on peut utiliser en connaissant les coordonnées de 4 points formant un rectangle. Il faut, de plus, que le point dont on cherche la volatilité soit à l'intérieur de ce rectangle.

Supposons que l'on ait le point $E(x;y;z)$ avec x et y connus, et que l'on cherche à déterminer la coordonnée z en sachant que ce point E est encadré par les points $A(x_1;y_1;z_A)$, $B(x_1;y_2;z_B)$, $C(x_2;y_1;z_C)$ et $D(x_2;y_2;z_D)$, formant un rectangle où se trouve le point E (c'est-à-dire que $x_1 \leq x \leq x_2$ et $y_1 \leq y \leq y_2$). Pour trouver z, il suffit de calculer :

$$\frac{\frac{x_2-x}{y_2-y} \times z_A + \frac{x-x_1}{y_2-y} \times z_C + \frac{x_2-x}{y-y_1} \times z_B + \frac{x-x_1}{y-y_1} \times z_D}{(x_2-x_1) \times (y_2-y_1)}$$

PS : On peut aussi utiliser la fonction interp2d de la librairie scipy.interpolate, qui permet de trouver des valeurs sur la nappe. Nous obtenons des résultats similaires à 10^{-2} près.

En faisant des tests, on trouve que (les résultats sont arrondis à 10^{-4}) la valeur de la volatilité implicite interpolée de strike 99 et de maturité 12 mois est de 0.1630. Par comparaison, la vraie valeur de la volatilité devrait être 0.1601.

Ainsi, la valeur de la volatilité implicite interpolée de strike 99.50 et de maturité 8 mois est de 0.2018 (ce résultat a été arrondi à 10^{-4}). On en déduit alors le prix de l'option (aussi arrondi à 10^{-4}), qui est 6.8032.

2.2 Question 4

Pour rappel, un modèle à volatilité locale est de la forme:

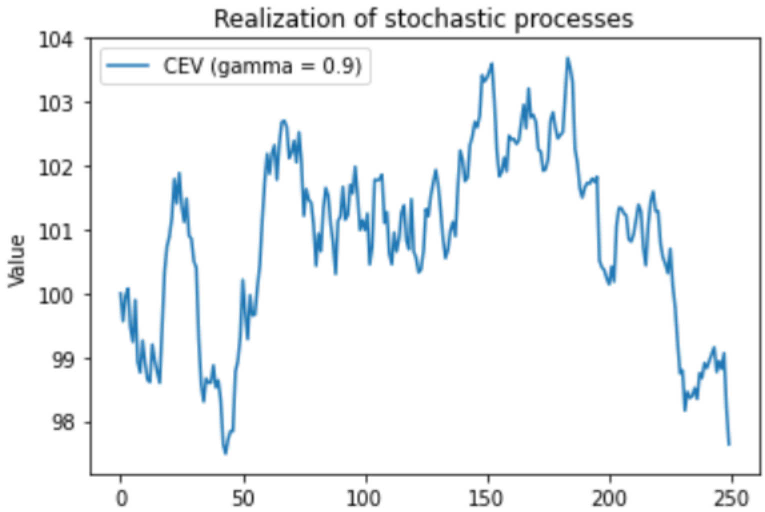
$$dS_t = \mu S_t dt + \sigma(t, S_t) S_t dW_t$$

Dans cette question on désire calibrer un modèle à volatilité locale de type CEV.(constant elasticity of variance, John Cox,1975):

$$dS_t = \mu S_t dt + \sigma_0 S_t^\gamma dW_t$$

avec $\gamma \sec 0$. Si $\gamma < 1$, on observe l'effet de levier propre aux actions (vol augmente quand prix diminue) ; si $\gamma > 1$, effet de levier inverse propre aux commodities.

Dans un premier temps nous diffusons les prix avec le modèle CEV. Voici l'évolution d'un seul prix du sous-jacent au cours du temps:



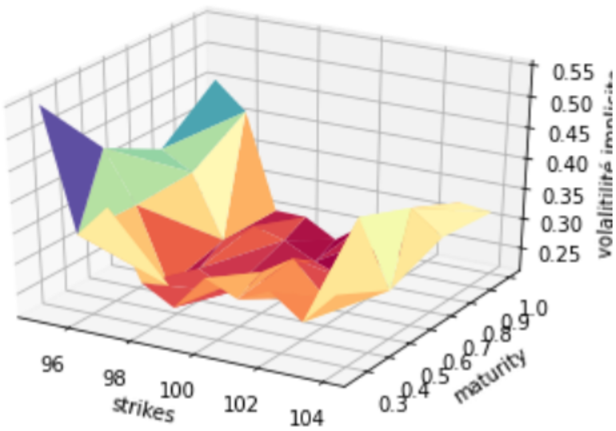
On valorise les options en suivant la méthode de Monte Carlo.
Pour continuer, nous cherchons à calibrer le σ du modèle pour un certain γ fixé.

Le but ici est de minimiser la fonction:

$$f_test = abs(MonteCarloValue - Cmarche)$$

et cela pour chaque strike et chaque maturité.
Nous avons d'abord voulu utiliser l'algorithme de Newton-Raphson pour sa convergence quadratique. Cependant la dérivée de la fonction du prix de l'option montecarlo est beaucoup trop hératique donc l'algorithme se perd dans des **minimums locaux**.

Nous avons alors implémenté l'algorithme de Nelder-Mead qui ne nécessite pas de connaître la dérivée de la fonction minimiser. Ainsi on obtient la liste des volatilités pour calibrer notre nappe de volatilité. On obtient donc la nappe suivante:



Ensuite on utlise de nouveau Nelder-Mead pour calibrer nos sigmas et nos gamma simultanément et on obtient à gauche sigma et à droite gamma pour chaque maturité et chaque strike:

```
[array([0.60446736, 0.80733255])
[array([0.38277523, 0.83083775])
[array([0.59995607, 0.73427156])
[array([0.23868184, 0.92175626])
[array([0.29844973, 0.87892595])
[array([0.2761915 , 0.88498285])
[array([0.32816637, 0.86663848])
[array([0.35809463, 0.86066613])
[array([0.30483218, 0.9014598 ])
[array([0.36546576, 0.91292747])
[array([0.43481716, 0.8593155 ])
[array([0.34297176, 0.78301015])
[array([0.25149762, 0.87395604])
[array([0.30914383, 0.76362082])
[array([0.28900431, 0.80022908])
[array([0.45424511, 0.71062853])
[array([0.2746965 , 0.92614268])
[array([0.31438607, 0.86679317])
[array([0.40554362, 0.91220076])
[array([0.2934671 , 0.87276691])
[array([0.39454603, 0.92489036])
[array([0.37680653, 0.91181646])
[array([0.38637846, 0.83163855])
[array([0.24713909, 0.95335225])
[array([0.29677553, 0.89646134])
```

Ensuite, il suffit à partir de ces données de construire la nappe de volatilité et de gamma.

2.3 Question 6

Dans cette section, nous allons calibrer un unique modèle à volatilité locale (non-paramétrique) pour toutes les options, et ce en utilisant la dormule de Dupire.

Pour chacun des strikes K du dataset, nous allons :

1. Déterminer les dérivées empiriques du prix du call en fonction de la maturité et du strike :
$$\frac{\partial C}{\partial T} = \frac{C(K, T + \Delta T) - C(K, T - \Delta T)}{2\Delta T}$$
$$\frac{\partial C}{\partial K} = \frac{C(K + \Delta K, T) - C(K - \Delta K, T)}{2\Delta K}$$

2. Déterminée la dérivée empirique seconde du prix du call en fonction du strike :
$$\frac{\partial^2 C}{\partial K^2} = \frac{C(K - \Delta K, T) - 2C(K, T) + C(K + \Delta K, T)}{(\Delta K)^2}$$

3. Calculer la volatilté locale grâce à la formule de Dupire :
$$\sigma(K, T) = \sqrt{\frac{2(\frac{\partial C}{\partial T} + rK\frac{\partial C}{\partial K})}{K^2\frac{\partial^2 C}{\partial K^2}}}$$

On suppose toujours que l'on a $S = 100$ et $r = 0$.

2.4 Question 7

Grâce à la formule de Dupire, on trouve la volatilité locale de chaque option, qu'on introduit par la suite dans la formule de Black-Scholes. On obtient en output pour T = 1 an et r = 0 :

```
Calculs pour une maturité T = 1 an et de taux sans risque r = 0
Pour un strike de 95, on a un prix de 12.40.
Pour un strike de 96, on a un prix de 9.59.
Pour un strike de 97, on a un prix de 8.28.
Pour un strike de 98, on a un prix de 7.40.
Pour un strike de 99, on a un prix de 6.86.
Pour un strike de 100, on a un prix de 6.58.
Pour un strike de 101, on a un prix de 6.52.
Pour un strike de 102, on a un prix de 6.49.
Pour un strike de 103, on a un prix de 6.47.
Pour un strike de 104, on a un prix de 6.46.
```

Enfin, nous déterminons le prix d'un call avec un strike de 99.50 et une maturité de 8 mois à l'aide de la formule de Black-Scholes. On a en output:

```
Pour un strike de 99.5, avec une maturité à 8 mois et un taux de risque à 0, on a un call (BS) de 5.51
```

3 Annexes

```python
# -*- coding: utf-8 -*-
"""Projet_calib.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1ab6xPPs0girugsJAE71mQZMMvEBL1IKj
"""

import numpy as np
import scipy
from scipy.stats import norm, ks_2samp
import matplotlib.pyplot as plt
import pandas as pd
import math
from mpl_toolkits.mplot3d import Axes3D
from scipy.optimize import minimize, curve_fit, fmin
from scipy.ndimage.interpolation import shift

N_prime = norm.pdf #fonction de densit  pour la fonction normale
N = norm.cdf #cumulative distribution function pour la loi normale

#Donn es

Strike = [95,96,97,98,99,100,101,102,103,104]
Cmarche = [12.40,9.59,8.28,7.40,6.86,6.58,6.52,6.49,6.47,6.46]
r = 0
S = 100
T = 1
Pas = 0.01

def BS_call(S, K, T, r, vol):

    d1 = (np.log(S / K) + (r + vol ** 2 / 2) * T) / (vol * np.sqrt(T))
    d2 = d1 - vol * np.sqrt(T)

    call = S * N(d1) - N(d2) * K * np.exp(-r * T)
    delta = N(d1)
    gamma =  N_prime(d1)/(S*vol*np.sqrt(T))
    vega = S * np.sqrt(T) * N_prime(d1)

    return call, delta, gamma, vega

def BS_call_delta_gamma_vega_list(Strike, S, maturity, vol_list):


    BS_call_delta_gamma_vega_list = []

    for i in range(0, len(Strike)):

        BS_call_delta_gamma_vega_list.append(BS_call(S, Strike[i], 1, r, np.array(vol_list)))


```

```
    return(BS_call_delta_gamma_vega_list)



def newtonRaphson(f, f_prime, x0,epsilon):

    i = 0

    condition = True

    while condition:

        if abs(f_prime(x0)) < epsilon:

            print("erreur (division par 0)")
            break

        x0 -= f(x0)/f_prime(x0)
        i = i + 1

        condition = abs(f(x0)) > epsilon and i<25 #on impose un nombre d’it rations max

    # print("le nombre d’it rations est de", i, "et l’abscisse pour lequel f s’annule est x1 (vol_list) =", round(x1, 4))
    return(x0)


def Vol_list(Strike, price, Cmarch ,  maturity):



    Vol_list = np.zeros(len(Strike))
    x0, epsilon = 0.1, 0.00001 #x0 a   t   choisi arbitrairement
    r = 0


    for i, K in enumerate(Strike):

        #f_function = call - Cmarch          f_prime = vega
        f_function = lambda vol_list: BS_call(price, K, maturity, r, vol_list)[0] - Cmarche[i]
        f_prime = lambda vol_list: BS_call(price, K, maturity, r, vol_list)[3]
        Vol_list[i] = newtonRaphson(f_function, f_prime, x0, epsilon)

    return list(Vol_list)

vol_list=Vol_list(Strike, 100, Cmarche, 1)
toto = [str(round(zizi, 3)) for zizi in vol_list]
" & ".join(toto)

def BS_continuous_vol(spot, exercise_prices, implied_volatility, r, T):

    replicate = np.zeros(len(exercise_prices))
    for i in range(len(exercise_prices)):
        d1 = (math.log(spot / exercise_prices[i]) + (implied_volatility[i] ** 2 / 2 + r) * (T))
        d1 = d1 / (implied_volatility[i] * pow(T, 0.5))
        d2 = d1-implied_volatility[i] * math.pow(T, 0.5)
```

```
        BS = spot * scipy.stats.norm(0, 1).cdf(d1)
        BS = BS - scipy.stats.norm(0, 1).cdf(d2) * exercise_prices[i] * math.exp(-r * T)
        replicate[i] = BS

    return replicate

#interpolation lin aire des vols
x = np.linspace(min(Strike), max(Strike), 100)
y = np.interp(x, Strike, vol_list)
plt.plot(x,y, label="volatilit  implicite (vol_list) en fonction du strike (K)")

M=np.zeros((27, 27))
s=0
for i in range(1,10):
    M[i-1,s+i-1]=Strike[i]**2
    M[i-1,s+i]=Strike[i]
    M[i-1,s+i+1]=1
    s+=2
s=0
for i in range(10,16):
    M[i-1,s+i-1]=Strike[i-10]**2
    M[i-1,s+i]=Strike[i-10]
    M[i-1,s+i+1]=1
    s+=2

# M

"""# DENSITES RISQUE NEUTRE

## Question 1
"""

## Premi re  tape  : On a vol_list une liste des volatilit s implicites, on fait maintenant leur interpolation ##

# Fonction pour interpoler les volatilit s implicites (interpolation de Lagrange)

def interpolation_lagrange(list_x, list_y, point_d_evaluation):
    polynome_evalue = 0

    for i in range(len(list_x)):
        produit = 1
        for j in range(len(list_x)):
            produit *= 1 if i == j else (point_d_evaluation - list_x[j]) / (list_x[i] - list_x[j])

        produit *= list_y[i]
        polynome_evalue += produit

    return polynome_evalue

# Cr ation d'un tableau de points x
x = np.linspace(min(Strike),max(Strike), 145)


implied_vols=[]

```

```
for i in x:
  resultat = interpolation_lagrange(Strike, vol_list, i) # appel de la fonction
  implied_vols.append(resultat) # affichage du r sultat

## Deuxi me  tape  : calibrer en utilisant la formule de Breeden-Litzenberger et Shimko ##

def calc_density_neutral_risk( exercise_prices, r, S, T, implied_vols):

  # Initialisation de la densit  de risque neutre
  density = np.zeros(len(exercise_prices))

  # Interpolation des volatilit s implicites
  #sigma = interpolation_lagrange(exercise_prices, implied_vols)

  for i, K in enumerate(exercise_prices):

      sigma = interpolation_lagrange(exercise_prices, implied_vols, K)
      # DeltaK
      deltaK = 0.01

      # Calcul des prix d'option avec BS
      C_plus_deltaK = BS_call(S, K + deltaK, T, r, sigma)[0]
      C_minus_deltaK = BS_call(S, K - deltaK, T, r, sigma)[0]
      C_actuel = BS_call(S, K, T, r, sigma)[0]

      # Formule de Breeden-Litzenberger pour calculer la densit  de risque neutre
      density[i] = np.exp(r*T)*((C_plus_deltaK - 2 * C_actuel + C_minus_deltaK) / deltaK ** 2)

  return density

# Calcul de la densit  de risque neutre
density = calc_density_neutral_risk(x, r, S, T, implied_vols)

plt.subplot(1, 2, 1)

# Trac  de la densit  de risque neutre
plt.plot(x, density, label='Densit  risque neutre')
plt.xlabel('Strike')
plt.ylabel('Densit  de risque neutre')
plt.title('Densit  de risque neutre en fonction du strike')


# density2 = norm.pdf(x, np.mean(Strike),np.std(Strike))/7
density2 = norm.pdf(x, 99.28041656531246, 2.5736127922802576)/7

# Trac  de la densit  gaussienne
plt.plot(x, density2)
plt.xlabel('Strike')
plt.ylabel('Densit  gaussienne')
plt.title('Densit  gaussienne en fonction du strike')

plt.subplots_adjust(left=0.1, bottom=0.1, right=3, top=0.9, wspace=0.4, hspace=0.4)
plt.show()

"""J'ai commenc    multiplier la densit  gaussienne par un ratio que j'ai
```

```
    d termin   en essayant diff rentes valeurs permettant de minimiser les
      carts   entre la densit   de risque neutre et la densit   gaussienne
    dont je cherche les param tres $\mu$ et $\sigma$.

Pour commencer on prend $\mu$ = mean(Strike) et $\sigma$ = std(Strike). Puis
    , on va choisir plus proprement les param tres $\mu$ et $\sigma$ de la
    gaussienne de fa on     r duire l' cart   entre nos deux plots. Pour ce
    faire, on va utiliser Kolmogorov-Smirnov qui est un test de statistiques
    utilis   pour comparer une distribution de donn es     un   chantillon   de
     r f rence. Il mesure l' cart   maximal entre les deux distributions de
    probabilit , ce qui permet de d terminer si elles proviennent de la
    m me distribution ou non.

On calcule la distance entre les deux distributions (la densit   gaussienne
    et la densit   de risque neutre) en utilisant la formule de Kolmogorov-
    Smirnov:

D = max(|F_gaussienne(x) - F_risque_neutre(x)|)
o   F_gaussienne(x) et F_risque_neutre(x) sont les fonctions de r partition
     de la densit   gaussienne et de la densit   de risque neutre,
    respectivement.
"""

cumulative_distribution2= norm.cdf(x, np.mean(Strike),np.std(Strike))
plt.plot(x,cumulative_distribution2)
plt.title('Fonction de r partition de la densit   gaussienne')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.show()

from scipy.integrate import cumulative_trapezoid

# int gration de la fonction de densit
Fx = cumulative_trapezoid(density, x)

# normalisation pour avoir une fonction de r partition entre 0 et 1
Fx = Fx/Fx[-1]

plt.plot(x[:-1], Fx)
plt.title('Fonction de r partition de la densit   risque neutre')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.show()

def kolmogorov_smirnov_distance(F1, F2):
  # Calcul de la distance de Kolmogorov-Smirnov entre les deux fonctions de
    r partition F1 et F2
  return np.max(np.abs(F1 - F2))

def iteration_function(params):
  # Calcul de l' cart   entre les fonctions de r partition de la densit   de
     risque neutre et de la densit   gaussienne, en utilisant les param tres
     donn s
  mu, sigma = params
  F_gaussienne = norm.cdf(x, mu, sigma)
  return kolmogorov_smirnov_distance(Fx, F_gaussienne[:-1])

# Initialisation des param tres    optimiser
params = np.array([np.mean(Strike),np.std(Strike)])

```

```
# Optimisation des param tres avec l'algorithme de Melder Mead
result = minimize(iteration_function, params, method='Nelder-Mead')

# Affichage des r sultats de l'optimisation
print(f"sigma={result.x[0]}, mu={result.x[1]}")

plt.subplot(1, 2, 1)

# Trac  de la densit  de risque neutre
plt.plot(x, density)
plt.xlabel('Strike')
plt.ylabel('Densit  de risque neutre')
plt.title('Densit  de risque neutre en fonction du strike')


density2 = norm.pdf(x, 99.28041656531246, 2.5736127922802576)/7
print(np.mean(Strike),np.std(Strike))

# Trac  de la densit  gaussienne
plt.plot(x, density2)
plt.xlabel('Strike')
plt.ylabel('Densit  gaussienne')
plt.title('Densit  gaussienne en fonction du strike')

plt.subplots_adjust(left=0.1, bottom=0.1, right=3, top=0.9, wspace=0.4, hspace=0.4)
plt.show()

"""# Question 2"""

def sample_from_implied_law(exercice_price, r, S, T, sigma, size=100_000):

  price = 0

  # Tirer un  chantillon  du sous-jacent   la maturit  de l'option (Monte Carlo)
  S_T = S * np.exp((r - 0.5 * sigma ** 2) * T + sigma * np.sqrt(T) * np.random.normal(0, T, size))

  for i in range(size):
    # Calculer le prix de mod le de l'option en utilisant la densit  risque neutre
    price += max(S_T[i] - exercice_price, 0)

  return np.exp(-r * T) * (price / size)

nb_iterations = 0

MCPricing = [sample_from_implied_law(K, r, S, T, sigma) for K, sigma in zip(Strike, vol_list)]

while sorted(MCPricing) != MCPricing[::-1] and nb_iterations < 30:
  MCPricing = [sample_from_implied_law(K, r, S, T, sigma) for K, sigma in zip(Strike, vol_list)]
  nb_iterations += 1

print(f"Found after {nb_iterations} iterations:")
for i in range(len(Strike)):
  print(f"K = {Strike[i]} and vol = {vol_list[i]:.4f} -> Price = {MCPricing[i]:.2f}")
```

```
"""# INTERPOLATION ET VOLATILITE LOCALE

# Question 3
"""

Cmarche9 = [11.79,8.95,8.07,7.03,6.18,6.04,5.76,5.50,5.50,5.39]
Cmarche6 = [10.71,8.28,6.91,6.36,5.29,5.07,4.76,4.47,4.35,4.14]
Cmarche3 = [8.67,7.14,5.98,4.93,4.09,3.99,3.43,3.01,2.72,2.53]
r = 0

vol_list3=Vol_list(Strike, 100, Cmarche3, 0.25)
#vol_list3

implied_vols3=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol_list3, i) # appel de la fonction
  implied_vols3.append(resultat) # affichage du r sultat

print(implied_vols3)

vol_list6=Vol_list(Strike, 100, Cmarche6, 0.5)
#vol_list6

implied_vols6=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol_list6, i) # appel de la fonction
  implied_vols6.append(resultat) # affichage du r sultat

print(vol_list6)

vol_list9=Vol_list(Strike, 100, Cmarche9, 0.75)
#vol_list9

implied_vols9=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol_list9, i) # appel de la fonction
  implied_vols9.append(resultat) # affichage du r sultat

print(vol_list9)

"""On d termine la nappe de volatilit  correspondant   ces 40 options: (ins rer tableau)"""

# D finissez les tableaux de volatilit s implicites
interpolate_vol_list_1an = implied_vols
interpolate_vol_list_9mois = implied_vols9
interpolate_vol_list_6mois = implied_vols6
interpolate_vol_list_3mois = implied_vols3

# Cr ez une figure et un axe 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Cr ez les donn es pour tracer la nappe de volatilit
X, Y = np.meshgrid(x, [1, 0.75, 0.5, 0.25])
Z = np.array([interpolate_vol_list_1an, interpolate_vol_list_9mois,
```

```
    interpolate_vol_list_6mois, interpolate_vol_list_3mois])

# Tracez la nappe de volatilit
ax.plot_surface(X, Y, Z,cmap='viridis')

# Ajoutez des  tiquettes  aux axes
ax.set_xlabel('Strike')
ax.set_ylabel('Maturit ')
ax.set_zlabel('Volatilit  implicite')

# Affichez le graphique
plt.show()

"""Avec cette nappe de volatilit s, on utilise Nelder Mead pour calibrer la volatilit  en chaque point de la nappe. D'ailleurs, on va la comparer avec la nappe de volatilit s d termin e   partir de Newton-Raphson.

Une fois la volatilit  calibr e, on calcule le prix de l'option de strike $99.50$ et de maturit  8 mois   partir de la formule de Black Scholes. Pour cela, nous pourrions utiliser la nappe r alis e ci-avant. Nonobstant, cette m thode de lecture graphique est tr s imprecise, ce qui nous oblige   trouver une m thode d'interpolation, qui nous permettra d'avoir des r sultats avec une meilleure pr cision. La premi re approche est d'utiliser une interpolation bilin aire, que l'on peut utiliser en connaissant les coordonn es de $4$ points formant un rectangle. Il faut, de plus, que le point dont on cherche la volatilit  soit   l'int rieur de ce rectangle :

Supposons que l'on ait le point $E(x;y;z)$ avec $x$ et $y$ connus, et que l'on cherche   d terminer $z$. Ce point $E$ est encadr  par les points $A(x_1;y_1;z_A)$, $B(x_1;y_2;z_B)$, $C(x_2;y_1;z_C)$ et $D(x_2;y_2;z_D)$ formant un rectangle o  se trouve le point $E$ (c'est-  -dire que $x_1 \leq x \leq x_2$ et $y_1 \leq y \leq y_2$). Pour trouver $z$, il suffit de calculer :

\begin{equation}
\frac{\frac{x_2-x}{y_2-y} \times z_A + \frac{x-x_1}{y_2-y} \times z_C + \frac{x_2-x}{y-y_1} \times z_B + \frac{x-x_1}{y-y_1} \times z_D}{(x_2-x_1) \times (y_2-y_1)}
\end{equation}
"""

def bilinear_interpolation(x, y, points):

    # Formule trouv e sur https://fr.wikipedia.org/wiki/Interpolation_bilin%C3%A9aire

    points = sorted(points)               # tri des points par x, puis par y
    (x1, y1, q11), (_x1, y2, q12), (x2, _y1, q21), (_x2, _y2, q22) = points

    if x1 != _x1 or x2 != _x2 or y1 != _y1 or y2 != _y2:
        raise ValueError('points do not form a rectangle')
    if not x1 <= x <= x2 or not y1 <= y <= y2:
        raise ValueError('(x, y) not within the rectangle')

    return (q11 * (x2 - x) * (y2 - y) +
            q21 * (x - x1) * (y2 - y) +
            q12 * (x2 - x) * (y - y1) +
            q22 * (x - x1) * (y - y1)
           ) / ((x2 - x1) * (y2 - y1) + 0.0)
```

```


x_val = 99
y_val = 1
z_val = bilinear_interpolation(x_val, y_val, [(98, 9/12, vol_list9[3]),
                                              (98, 1, vol_list[3]),
                                              (100, 9/12, vol_list9[5]),
                                              (100, 1, vol_list[5])])

print(f"La valeur de la volatilit  implicite interpol e de strike {x_val} et de maturit  {int(y_val*12)} mois est de {z_val}")
print(f"Par comparaison, la vraie valeur de la volatilit  devrait  tre  {vol_list[4]}")

"""En utilisant la m thode d'interpolation bilin aire, nous voyons que l'estimation de certaines valeurs connues ne sont pas tr s proches (erreur $10^{-3}$ pr s). C'est pour cette raison que nous utiliserons la librairie `scipy`, qui propose la fonction `interp2d` au sein du module `interpolate` :"""

from scipy.interpolate import interp2d

# Cr ation de la fonction d'interpolation
interp_func = interp2d(x, [1, 0.75, 0.5, 0.25], np.array([interpolate_vol_list_1an, interpolate_vol_list_9mois, interpolate_vol_list_6mois, interpolate_vol_list_3mois])
, kind='linear')

# R cup ration de la valeur de Z   partir de X et Y
x_val = 99.5
y_val = 8/12
#x_val = 99
#y_val = 0.75
z_val = interp_func(x_val, y_val)

print(f"La valeur de la volatilit  implicite interpol e de strike {x_val} et de maturit  {int(y_val*12)} mois est de {z_val[0]}")

## Calcul d'une option de strike 99.50 et de maturit  8 mois ##
x_val = 99.5
y_val = 8/12
z_val = bilinear_interpolation(x_val, y_val, [(99, 6/12, vol_list6[4]),
                                              (99, 9/12, vol_list9[4]),
                                              (100, 6/12, vol_list6[5]),
                                              (100, 9/12, vol_list9[5])])

print(f"La valeur de la volatilit  implicite interpol e de strike {x_val} et de maturit  {int(y_val*12)} mois est de {z_val}.")
price_q4 = BS_call(100, 99.5, 8/12, r, z_val)[0]
print(f"La prix est donc {price_q4}.")

"""# Question 4"""

##Simulation des prix

np.random.seed(123)
def ProcessCEV(sigma,gamma,T,dt,S0):
    n = round(T / dt)
    gaussian_increments = np.random.normal(size=n - 1)
    res = np.zeros(n)
```

```
    res[0] = S0
    S = S0
    sqrt_dt = dt ** 0.5
    for i in range(n - 1):
        S = S + sigma * (S ** (gamma)) * gaussian_increments[i] * sqrt_dt
        res[i + 1] = S
    return res

T = 20
dt = 0.001
a=ProcessCEV(0.2, 0.9, 3/12,dt,100)


plt.plot(a, label="CEV (gamma = 0.9)")
plt.xlabel('Time index')
plt.ylabel('Value')
plt.title("Realization of stochastic processes")
plt.legend()
plt.show()

Cmarche9 = [11.79,8.95,8.07,7.03,6.18,6.04,5.76,5.50,5.50,5.39]
Cmarche6 = [10.71,8.28,6.91,6.36,5.29,5.07,4.76,4.47,4.35,4.14]
Cmarche3 = [8.67,7.14,5.98,4.93,4.09,3.99,3.43,3.01,2.72,2.53]

def ValoMC(n,K,sigma,gamma,T,dt,S0):

    price=0
    r=0
    for i in range(n):
        a=ProcessCEV(sigma,gamma,T,dt,S0)
        S_T=ProcessCEV(sigma,gamma,T,dt,S0)[-1]
        price += (np.maximum((S_T - K), 0))
    return (np.exp(-r * T) * (price/n))
#Monte Carlo

ValoMC(10000,95,0.288, 1, 3/12,0.001,100) #tests

def f_x(x,Xparams):
    return abs(ValoMC(int(Xparams[1]),Xparams[2],x[0],x[1],Xparams[3],
   Xparams[4],Xparams[5])-Xparams[0])

def ftest(x):
  return abs(ValoMC(n,k,x, gamma, T,dt,S0)-cmarche)

def NelderMeadV2(x01, x02,f,epsilon):
    # Initialisation
    if f(x01) > f(x02):
        transitx = x01
        x01 = x02
        x02 = transitx
    else:
        pass

    # It ration
    while f(x01) > epsilon:
        xir = 2*x01 - x02
        if f(xir) < f(x01):
            xie = x01 + 2*(x01 - x02)
            if f(xie) < f(xir):
                x02 = x01
```

```
                x01 = xie
            else:
                x02 = x01
                x01 = xir
        else:
            x01 = x01
            x02 = x02 + (x01 - x02)/2

        if f(x01) > f(x02):
            transitx = x01
            x01 = x02
            x02 = transitx
    return x01

vol6=[]
gamma=0.9
n=100
T=6/12
dt=T/n
S0=100
x01=0.1
x02=0.4
epsilon=0.01
for i in range(len(Strike)):
  print(i)
  cmarche=Cmarche6[i]
  k=Strike[i]
  def ftest(x):
      if (x<0):
        x=0.01
      return abs(ValoMC(n,k,x, gamma, T,dt,S0)-cmarche)
  test = NelderMeadV2(x01, x02,ftest,epsilon)
  vol6.append(test)

vol6

vol3=[]
gamma=0.9
n=100
T=3/12
dt=T/n
S0=100
x01=0.1
x02=0.4
epsilon=0.01
for i in range(len(Strike)):
  print(i)
  cmarche=Cmarche3[i]
  k=Strike[i]
  def ftest(x):
      if (x<0):
        x=0.01
      return abs(ValoMC(n,k,x, gamma, T,dt,S0)-cmarche)
  test = NelderMeadV2(x01, x02,ftest,epsilon)
  vol3.append(test)

vol3

vol9=[]
gamma=0.9
```

```
n=100
T=9/12
dt=T/n
S0=100
x01=0.1
x02=0.4
epsilon=0.01
for i in range(len(Strike)):
  print(i)
  cmarche=Cmarche9[i]
  k=Strike[i]
  def ftest(x):
      if (x<0):
        x=0.01
      return abs(ValoMC(n,k,x, gamma, T,dt,S0)-cmarche)
  test = NelderMeadV2(x01, x02,ftest,epsilon)
  vol9.append(test)

vol9

vol1y=[]
gamma=0.9
n=100
T=12/12
dt=T/n
S0=100
x01=0.1
x02=0.4
epsilon=0.01
for i in range(len(Strike)):
  print(i)
  cmarche=Cmarche[i]
  k=Strike[i]
  def ftest(x):
      if (x<0):
        x=0.01
      return abs(ValoMC(n,k,x, gamma, T,dt,S0)-cmarche)
  test = NelderMeadV2(x01, x02,ftest,epsilon)
  vol1y.append(test)

vol1y

from mpl_toolkits.mplot3d import Axes3D
xaxis=[]
maturity=[3/12,6/12,9/12,1]
vol3.extend(vol6)
vol3.extend(vol9)
vol3.extend(vol1y)
x = np.linspace(95,104, 40)
print(vol3)
# D finissez les tableaux de volatilit s implicites
vol_implicit_interpolated=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol3, i) # appel de la fonction
  vol_implicit_interpolated.append(resultat) # affichage du r sultat


for i in maturity:
    xaxis+=[i]*len(Strike)
yaxis=[]
```

```
yaxis+=list(Strike)*len(maturity)
zaxis=vol_implicit_interpolated

fig=plt.figure()
ax=fig.add_subplot(111,projection='3d')
ax.plot_trisurf(yaxis,xaxis,zaxis,cmap=plt.cm.Spectral)
ax.set
ax.set_xlabel("strike")
ax.set_ylabel("maturity")
ax.set_zlabel("volatilit  implicite")

import copy
import numpy
Ts=[1]*10+[0.75]*10+[0.5]*10+[0.25]*10
Ks=list(Strike)*4
Cs=list(Cmarche)+list(Cmarche3)+list(Cmarche6)+list(Cmarche9)
sigamma=[]
dt=T/n
S0=100
gamma0=0.9
for k,cmarche,T,sigma0 in zip(Ks,Cs,Ts,vol3):
  Xparams = [cmarche, n , k, T, dt,S0]
  x = [sigma0,gamma0]
  ttest = neldermead(f_x,x,0.05,0.05,50, 50,Xparams)
  sigamma.append(ttest)

sigamma

sigamma[2][0][]

Ts=[1]*10+[0.75]*10+[0.5]*10+[0.25]*10
Ks=list(Strike)*4
Cs=list(Cmarche)+list(Cmarche3)+list(Cmarche6)+list(Cmarche9)
r=0
save=[]
(x01, x02,ftest,epsilon)
for k,c,t in zip(Ks,Cs,Ts):
    init=np.ones(2)
    np.random.seed(123)
    save.append(neldermead(obj_fun_Q4,init,tol=1e-6)[0])
save=abs(np.array(save))

NelderMeadV2(x01, x02,ftest,epsilon)

x = np.linspace(95,104, 100)
# D finissez les tableaux de volatilit s implicites
implied_vol1y=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol1y, i) # appel de la fonction
  implied_vol1y.append(resultat) # affichage du r sultat

implied_vol9=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol9, i) # appel de la fonction
  implied_vol9.append(resultat) # affichage du r sultat

implied_vol6=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol6, i) # appel de la fonction
  implied_vol6.append(resultat) # affichage du r sultat
```

```python

implied_vol3=[]
for i in x:
  resultat = interpolation_lagrange(Strike, vol3, i) # appel de la fonction
  implied_vol3.append(resultat) # affichage du r sultat
interpolate_vol_list_1an = implied_vol1y
interpolate_vol_list_9mois = implied_vol9
interpolate_vol_list_6mois = implied_vol6
interpolate_vol_list_3mois = implied_vol3


# Cr ez une figure et un axe 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Cr ez les donn es pour tracer la nappe de volatilit
X, Y = np.meshgrid(x, [12, 9, 6, 3])
Z = np.array([interpolate_vol_list_1an, interpolate_vol_list_9mois,
    interpolate_vol_list_6mois, interpolate_vol_list_3mois])

# Tracez la nappe de volatilit
ax.plot_surface(X, Y, Z,cmap='viridis')

# Ajoutez des  tiquettes  aux axes
ax.set_xlabel('Strike')
ax.set_ylabel('Maturit ')
ax.set_zlabel('Volatilit  implicite')

# Affichez le graphique
plt.show()

import numpy
from scipy.stats import norm
import matplotlib.pyplot as plt
import pandas as pd
import copy
from mpl_toolkits.mplot3d import Axes3D

vol1y=[]
for i in range(len(Strike)):
  cmarche=Cmarche[i]
  n=100
  k=Strike[i]
  gamma=0.9
  T=12/12
  dt=T/n
  S0=100
  Xparams = [cmarche, n , k, gamma, T, dt,S0]
  x = [sigma0]
  ttest = neldermead(f_x,x,0.05,0.05,50, 50,Xparams)
  vol1y.append(ttest[0][0])

def neldermead(funct, X,step,stop_stagne,stop, ittBreak,X_params):
    nb_params = len(X)
    F0 = funct(X,X_params)
    m = 0
    k = 0
    refl = 1
    exp = 2
    contr = -1 / 2
```

```
    red = 0.5
    params = [[X, F0]]
    for i in range(nb_params):
        vect = copy.copy(X)
        vect[i] = vect[i] + step
        params.append([vect,funct(vect,X_params)])
    while 1:
        params.sort(key = lambda x: x[1])
        F = params[0][1]
        if ittBreak<= k :
            print(k)
            return params[0]
        k = k + 1
        if F<F0-stop_stagne:
            m = 0
            F0 = F
        else:
            m = m + 1
        if m >= stop:
            print(k)
            return params[0]

        centroid = [0.] * nb_params
        for cen in params[:-1]:
            for i, c in enumerate(cen[0]):
                centroid[i] += c / (len(params)-1)
        newParam_refl = centroid + refl * (centroid - numpy.array(params[-1][0]))
        refl_r = funct(newParam_refl,X_params)
        if params[0][1] <= refl_r < params[-2][1]:
            del params[-1]
            params.append([newParam_refl, refl_r])
            continue
        if refl_r < params[0][1]:
            newParam_exp = centroid + exp * (centroid - numpy.array(params[-1][0]))
            exp_e = funct(newParam_exp,X_params)
            if exp_e < refl_r:
                del params[-1]
                params.append([newParam_exp,exp_e])
                continue
            else:
                del params[-1]
                params.append([newParam_refl, refl_r])
                continue
        newParam_contr = centroid + contr * (centroid -  numpy.array(params[-1][0]))
        contr_c = funct(newParam_contr,X_params)
        if contr_c < params[-1][1]:
            del params[-1]
            params.append([newParam_contr, contr_c])
            continue
        par = params[0][0]
        params2 = []
        for li in params:
            newParam_red = par + red * (li[0] - numpy.array(par))
            red_r = funct(newParam_red,X_params)
            params2.append([newParam_red, red_r])
        params = params2
```

```
import copy
import numpy
sigma0=0.20
cmarche=8.67
n=100
k=95
gamma=0.9
T=9/12
dt=T/n
S0=100
Xparams = [cmarche , n , k, gamma, T, dt,S0]
x = [sigma0]
ttest = neldermead(f_x,x,0.04,0.04,50, 50,Xparams)
ttest



vol3=[]
for i in range(len(Strike)):
  cmarche=Cmarche3[i]
  n=100
  k=Strike[i]
  gamma=0.9
  T=3/12
  dt=T/n
  S0=100
  Xparams = [cmarche , n , k, gamma, T, dt,S0]
  x = [sigma0]
  ttest = neldermead(f_x,x,0.05,0.05,50, 50,Xparams)
  vol3.append(ttest[0][0])

vol6=[]
for i in range(len(Strike)):
  cmarche=Cmarche6[i]
  n=100
  k=Strike[i]
  gamma=0.9
  T=6/12
  dt=T/n
  S0=100
  Xparams = [cmarche , n , k, gamma, T, dt,S0]
  x = [sigma0]
  ttest = neldermead(f_x,x,0.05,0.05,50, 50,Xparams)
  vol6.append(ttest[0][0])

implied_vol6=[]
for i in np.linspace(min(Strike),max(Strike),100):
  resultat = interpolation_lagrange(Strike, vol6, i) # appel de la fonction
  implied_vol6.append(resultat) # affichage du r sultat

vol9=[]
for i in range(len(Strike)):
  cmarche=Cmarche9[i]
  n=100
  k=Strike[i]
  gamma=0.9
  T=9/12
  dt=T/n
  S0=100
  Xparams = [cmarche , n , k, gamma, T, dt,S0]
```

```
  x = [sigma0]
  ttest = neldermead(f_x,x,0.05,0.05,50, 50,Xparams)
  vol9.append(ttest[0][0])

interpolate_vol_list_1an = vol1y
interpolate_vol_list_9mois = vol9
interpolate_vol_list_6mois = vol6
interpolate_vol_list_3mois = vol3

# Cr ez une figure et un axe 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Cr ez les donn es pour tracer la nappe de volatilit
X, Y = np.meshgrid(x, [1, 0.75, 0.5, 0.25])
Z = np.array([interpolate_vol_list_1an, interpolate_vol_list_9mois, interpolate_vol_list_6mois, interpolate_vol_list_3mois])
len(Z)
# Tracez la nappe de volatilit
ax.plot_surface(X, Y, Z,cmap='viridis')

# Ajoutez des  tiquettes  aux axes
ax.set_xlabel('Strike')
ax.set_ylabel('Maturit ')
ax.set_zlabel('Volatilit  implicite')

# Affichez le graphique
plt.show()

"""# Question 6"""

def Dupire(exercise_prices, r, S, T, implied_vols):

  # DeltaT et DeltaK
  deltaT, deltaK = 0.01, 0.01

  dict_volLoc = {}

  for i, K in enumerate(exercise_prices):

    sigma = implied_vols[i]

    C_plus_deltaT = BS_call(S, K, T + deltaT, r, sigma)[0]
    C_minus_deltaT = BS_call(S, K, T - deltaT, r, sigma)[0]

    C_plus_deltaK = BS_call(S, K + deltaK, T, r, sigma)[0]
    C_minus_deltaK = BS_call(S, K - deltaK, T, r, sigma)[0]

    C_actuel = BS_call(S, K, T, r, sigma)[0]

    dC_over_dT = (C_plus_deltaT - C_minus_deltaT) / (2 * deltaT)
    dC_over_dK = (C_plus_deltaK - C_minus_deltaK) / (2 * deltaK)
    d2C_over_dK2 = (C_minus_deltaK - 2 * C_actuel + C_plus_deltaK) / (deltaK ** 2)

    # Formule de Breeden-Litzenberger pour calculer la densit  de risque neutre
    volLoc = np.sqrt(2 * (dC_over_dT + r * K * dC_over_dK) / ((K ** 2) * d2C_over_dK2))
    dict_volLoc[K] = volLoc
```

```python

  return dict_volLoc

"""# Question 7"""

print(f"Calculs pour une maturit  T={T} an et de risque r={r}")

dict_volLoc = Dupire(x, r, S, T, implied_vols)

for K in Strike:

  call = BS_call(S, K, T, r, dict_volLoc[K])[0]
  print(f"Pour un strike de {K}, on a un prix de {call:.2f}.")

strike_q7 = 99.5
T_q7 = 8/12

dict_volLoc2 = Dupire(x, r, S, T_q7, implied_vols)

price_q7_BS = BS_call(S, strike_q7, T_q7, r, dict_volLoc2[strike_q7])[0]
price_q7_MC = sample_from_implied_law(strike_q7, r, S, T_q7, dict_volLoc2[strike_q7])

print(f"Pour un strike de {strike_q7}, avec une maturit     {int(T_q7 * 12)} mois et un taux de risque    0, on a un call (BS) de {price_q7_BS:.2f}")
print(f"Pour un strike de {strike_q7}, avec une maturit     {int(T_q7 * 12)} mois et un taux de risque    0, on a un call (MC) de {price_q7_MC:.2f}")
```