# Rapport de PFE

---

Melek Ben Amar - Philip MALOU

Sujet : modèle de valorisation des options avec sauts

Tuteur : Yalcin Aktar

IFI 2022

# Remerciements

La réalisation de ce projet de fin d'étude a été possible grâce au concours de plusieurs personnes à qui nous voudrions témoigner toute notre gratitude.

Nous voudrions tout d'abord adresser toute notre reconnaissance à notre tuteur, Monsieur Yalcin Aktar.

Merci aussi à Madame Irina Kortchemski, dont nous avons suivi les cours de Calibration et de Simulation depuis deux ans déjà. Ces enseignements nous ont permis d'aborder ce projet avec sérénité et confiance.

Nous souhaitons aussi remercier les professeurs de CYTech qui nous ont fourni les outils nécessaires à la réussite de nos études universitaires et à notre épanouissement professionnel.

# Description du sujet:

Option Pricing Models with Jumps. Regularized Calibration of Merton model.

An important issue in finance is model calibration. The calibration problem is the inverse of the option pricing problem.

It can be shown that the usual formulation of the inverse problem via Non-linear Least Squares is an ill posed problem. To achieve well-posedness of the problem, some regularization is needed. This PFE consists of two parts.

The first one is the study of the Merton model, the simulation of the evolution of underlying asset with jumps. The analytical calculation of the price of the options in the Merton model and its evaluation by Monte-Carlo method is required.

The second part is the study a regularization method based on Levenberg - Marquardt regularization and on relative entropy introduced in the article of Rama Cont and Peter Tankov where the authors reformulate the calibration problem into a problem of finding a risk-neutral exponential Levy model that reproduces the observed option prices and has the smallest possible relative entropy with respect to a chosen prior model.

# Table des matières

# Partie 1

L’objectif de cette partie est d'étudier le modèle de Merton de valorisation des options avec sauts.

Après avoir présenté le modèle, nous avons simulé des trajectoires grâce à Python.

Ensuite nous avons résolu numériquement le modèle et comparé différentes méthodes de valorisation des options avec sauts (en utilisant les méthodes de Monte-Carlo et la résolution de l’équation).

# Historique du modèle de Merton

Les travaux de Fischer Black et de Myron Scholes publiés dans les années 1970 ont révolutionné la finance de marché. Ils définissent ce qui demeure aujourd'hui le modèle de référence en valorisation des produits financiers (notamment les options). Sa particularité est qu'il met en place une relation entre le prix d'une option et l'évolution du prix de son actif sous-jacent.

Robert Merton a été un acteur important dans l'institution et la démocratisation de ce modèle en finance quantitative appelé le modèle de Black-Scholes. Il a repris leurs travaux, les a développés mais on a aussi décelé des limites. Ainsi le modèle de Merton de valorisation des options avec sauts est un exemple de la recherche de perfectionnement du modèle de Black-Scholes et nous allons l'étudier dans la suite de ce rapport.

En 1997, Robert Merton et Myron Scholes remportent le Prix Nobel d'économie pour ces travaux. Fischer Black est décédé en 1995 et n'a donc pu être que cité comme contributeur étant inéligible pour le prix officiel.

# Description générale du modèle

Le modèle de Merton ou de valorisation des options avec sauts est un prolongement du modèle de Black-Scholes. Son objectif est de se détacher de ses limites.

En effet, étant un modèle simple d'utilisation, le modèle de Black-Scholes posent de nombreuses hypothèses comme entre autres le fait que le prix du sous-jacent suit un mouvement brownien. Bien que nécessaires, ces postulats posent parfois problèmes en créant des écarts entre les résultats obtenus par le modèle et la réalité des marchés financiers.

On peut notamment citer la skewness négative et l'excès de kurtosis de la densité du prix du sous-jacent observé sur les marchés qui n'est pas retranscrite dans le modèle de Black-Scholes.

La skewness appelé également coefficient d'asymétrie est le moment d'ordre 3 et mesure l'asymétrie de la fonction de densité pour une variable aléatoire. Cela représente la position de la distribution par rapport au centre.

Elle est donnée par la formule :

$$Sk_c = \frac{n}{(n-1)(n-2)} \sum \left( \frac{x_i - \bar{x}}{s_c} \right)^3$$

Le kurtosis appelé également coefficient d'aplatissement est le moment d'ordre 4 et caractérise la queue de distribution. Il permet de mesurer la dispersion des valeurs extrêmes par rapport à la loi normale.

Il est donné par la formule :

$$K_c = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum \left( \frac{x_i - \bar{x}}{s_c} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

Après avoir décrit mathématiquement le modèle, nous expliciterons en quoi le modèle de Merton permet de se rapprocher de la réalité du marché (notamment en termes de skewness et kurtosis).

# Description mathématique du modèle

Comme indiqué dans son nom, le modèle de Merton ajoute la notion de saut à l'équation du prix du sous-jacent du modèle de Black-Scholes. Ces sauts sont mathématiquement interprétés comme un processus de Poisson composé.

L'hypothèses du modèle sur le prix de l'actif sous-jacent se présente donc de la manière suivante :

$$S_t = S_0 * exp\left(\left(r - \frac{\sigma^2}{2} - \lambda k\right)t + \sigma B_t + \sum_{i=1}^{N_t} Y_i\right)$$

avec r, le taux d'intérêt

σ, la volatilité

Bt, un mouvement brownien

et $X_t = \sum_{i=1}^{N_t} Y_i$ , un processus de poisson composé de paramètres ( λ, μ, δ)

Par rapport au modèle de Black-Scholes, trois nouveaux paramètres sont introduits :

- la fréquences des sauts noté λ
- la moyenne de la taille des sauts μ
- l'écart type de la taille des sauts δ

En effet $X_t = \sum_{i=1}^{N_t} Y_i$ étant un processus de poisson composé, on a N qui suit une loi de poisson et et les Yi sont des variables aléatoires indépendantes et identiquement distribuées qui suivent une loi log normale.

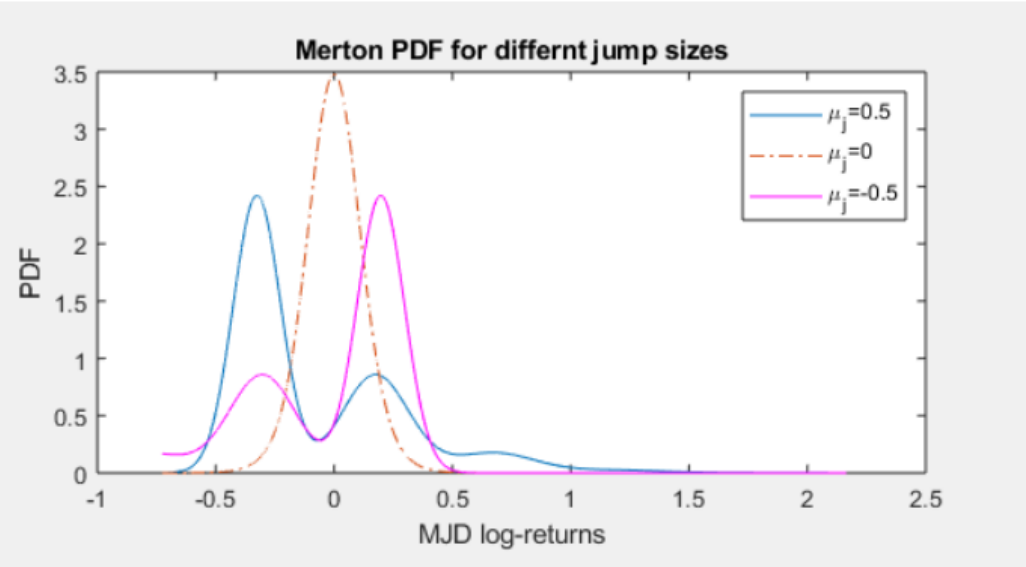
$$N_t \to P(\lambda)$$

$$\forall i \quad \ln(Y_i) \to \aleph(\mu, \delta^2)$$

Pour se rapprocher de la réalité des marches, le modèle de Merton prend en compte la discontinuité du cours du sous-jacent en donnant la possibilité au prix de ‘sauter’ à tout moment.

Pour reprendre le sujet de la skewness et du kurtosis définis précédemment, ce nouveau modèle nous permet bien d’avoir un coefficient d’asymétrie négatif et une dispersion importante.

En effet pour ce qui est de la kurtosis, elle dépend maintenant essentiellement de la moyenne des sauts μ.



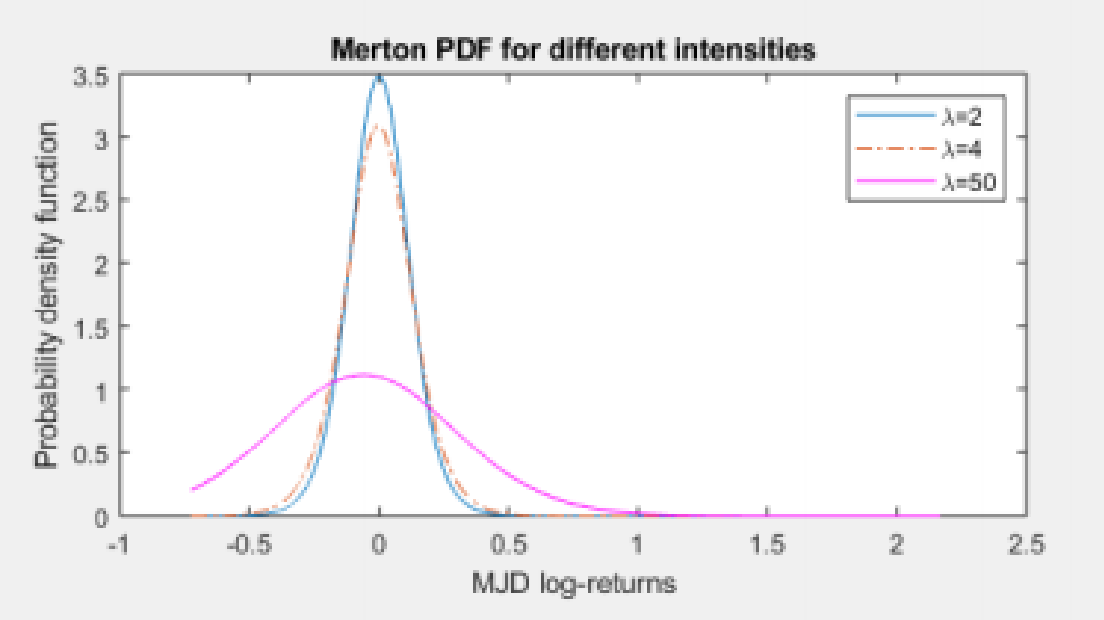

Comme on le voit dans le graphe ci-dessus :

- Si μ = 0 : La distribution est symétrique.

- Si μ < 0 : la distribution est plus étalée à gauche, on aura dans ce cas un coefficient d’asymétrie négatif.

- Si μ > 0 : la distribution est plus étalée à droite, on aura dans ce cas un coefficient d’asymétrie positif.

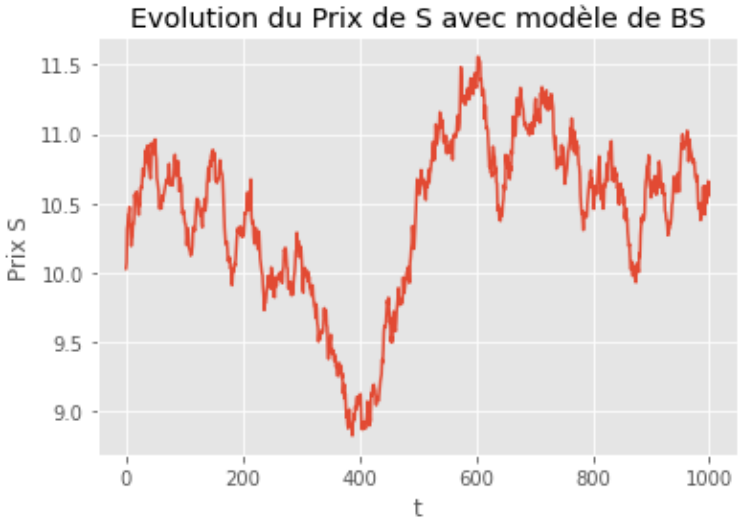
Pour ce qui est du kurtosis, il dépend essentiellement de la fréquence des sauts λ.

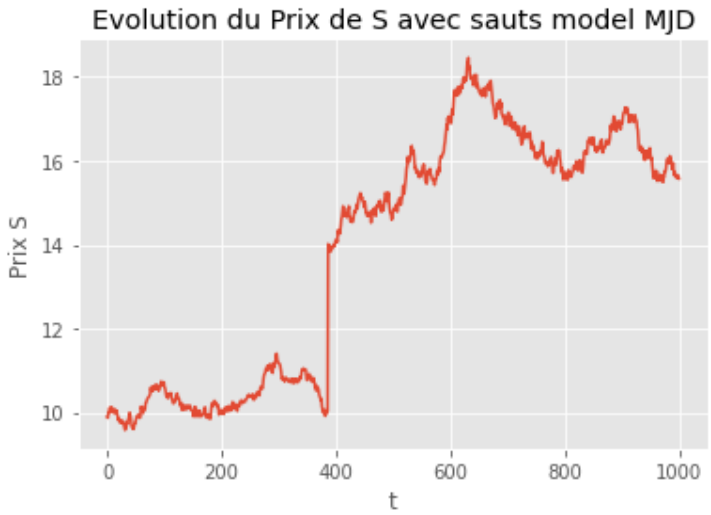



Si λ est élevé on a plus tendance à atteindre des valeurs extrêmes puisqu' on effectue plus de sauts ce qui explique que la courbe de densité du log normal de Merton a une allure aplatie. Dans ce cas, il s'agit d'un excès de Kurtosis.

Par contre si les valeurs de λ sont moins élevées, la densité dans ce cas sera moins aplatie comme le cas de λ =2 et λ = 4.

# Simulation d’une trajectoire avec Python

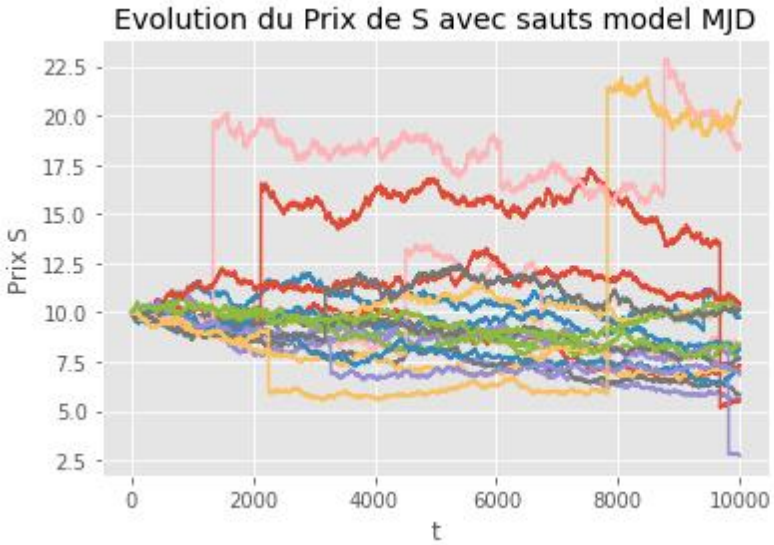



Ces graphiques présentes des simulations de trajectoires du prix de sous-jacent dans les modèles de Black-Scholes et de Merton.

On remarque bien dans le deuxième cas la présence de sauts.

Le graphe ci-dessous montre que ces sauts entrainent une plus forte variance autour de la moyenne ce qui fit écho à ce qui a été dit précédemment à propos de l’excès de kurtosis.




Les codes Python réalisés pour obtenir ces graphes sont présentés ci-après

Code pour la simulation de 1 trajectoire avec le modèle de Black-Scholes

```python
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import numpy as np

def BS(S0, T, r, sigma, N, N_actifs):
    size=(N,N_actifs)
    dt = T/N

    BS = np.cumsum(((r -  sigma**2/2)*dt + sigma*np.sqrt(dt) *
                    np.random.normal(size=size)), axis=0)

    return np.exp(BS)*S0


S0 = 10
T = 1
r = 0.01
sigma = 0.2
N =10000
N_actifs = 1

S = BS(S0, T, r, sigma, N, N_actifs)

plt.plot(S)
plt.xlabel('t')
plt.ylabel('Prix S')
plt.title('Evolution du Prix de S avec modèle de BS')
```

Code pour la simulation de 1 trajectoire avec le modèle de Merton

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import numpy as np

def merton_jump_paths(S0, T, r, sigma,  lam, mu, sigma_saut, N, N_actifs):
    size=(N,N_actifs)
    dt = T/N
    Partie_avec_saut = np.multiply(np.random.poisson( lam*dt, size=size),
                          np.random.normal(mu,sigma_saut, size=size)).cumsum(axis=0)
    Partie_sans_saut = np.cumsum(((r -  sigma**2/2 -lam*(mu  + sigma_saut**2*0.5))*dt +
                              sigma*np.sqrt(dt) *
                              np.random.normal(size=size)), axis=0)

    return np.exp(Partie_sans_saut+Partie_avec_saut)*S0


S0 = 10
T = 1
r = 0.01
mu = 0.1 # moyenne des sauts
sigma_saut = 0.5 # Ecart_type des sauts
lam = 0.5 ## Frequence des sauts
N =10000
N_actifs = 1
sigma = 0.2

S = merton_jump_paths(S0, T, r, sigma, lam, mu, sigma_saut, N, N_actifs)

plt.plot(S)
plt.xlabel('t')
plt.ylabel('Prix S')
plt.title('Evolution du Prix de S avec sauts model MJD')
```

Code pour la simulation de 20 trajectoires avec le modèle de Merton

```
S0 = 10
T = 1
r = 0.01
mu = 0.1 # moyenne des sauts
sigma_saut = 0.5 # Ecart_type des sauts
lam = 0.5 ## Frequence des sauts
N =10000
N_actifs = 20
sigma = 0.2

S = merton_jump_paths(S0, T, r, sigma, lam, mu, sigma_saut, N, N_actifs)

plt.plot(S)
plt.xlabel('t')
plt.ylabel('Prix S')
plt.title('Simulation de 20 actifs avec sauts avec sauts model MJD')
```

# Solution analytique du prix de l'option avec saut

L'objectif de cette partie est de résoudre l'équation numériquement. Comme vu précédemment, le modèle de Merton se caractérise de la manière suivante

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t + saut$$

La partie de la variation causée par un saut est :

$$dS_t = Y_t S_{t^-} - S_t \Rightarrow \frac{dS_t}{S_t} = Y_t - 1$$

On s'intéresse ici au processus de Poisson $N_t$ qui est nul à l'instant 0 : on est à 0 saut presque sûrement.

$$P[N_0 = 0] = 1$$

La probabilité de réaliser un saut dans un intervalle de temps infinitesimal dt est

$$\lambda * dt$$

Par contre comme l'intervalle de temps est très faible, la probabilité de réaliser plus qu'un saut dans cet même intervalle est négligeable.
Donc mathématiquement on obtient:

$$P[N(t + \Delta t) - N(t) = 1] = \lambda\Delta + o(\Delta t)$$

$$P[N(t + \Delta t) - N(t) > 1] = o(\Delta t)$$

On obtient alors:

$$P(dN_t = 1) = \lambda d_t$$

$$P(dN_t = 0) = 1 - \lambda d_t$$

parsuite

$$E[dN_t] = 1 * \lambda d_t + 0 * (1 - \lambda d_t) = \lambda d_t$$

Au final la variation causée par le saut est celle du produit de la taille et de la fréquence en d'autres termes

$$\frac{dS_t}{S_t} = (Y_t - 1)dN_t$$

Il faut retrancher l'espérance de cette quantité de la dynamique pour considérer que la variation pure des sauts. or

car la taille et l'intensité des sauts sont indépendants. On obtient alors

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t + (Y_t - 1)dN_t - \lambda k dt$$

Et pour plusieurs sauts cela donne:

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t + \prod_{j=1}^{dN_t}(Y_t - 1) - \lambda k dt$$

alors

$$S_t = (\mu - \lambda k)Sdt + \sigma S dW_t + S\prod_{j=1}^{dN_t}(Y_t - 1)$$

Maintenant si on applique la lemme d'Itô pour une fonction donnée en tenant compte de la variation du saut on obtient cette égalité.

$$df = \frac{\partial f}{\partial S}dS + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}dS^2 + f(S\prod_{j=1}^{dN_t} Y_j) - f(S)$$

Si on prend f = lnS cela implique:

$$d(lnS) = \frac{1}{S}dS - \frac{1}{2}\frac{1}{S^2}dS^2 + ln(S\prod_{j=1}^{dN_t} Y_j) - ln(S)$$

or

$$ln(S\prod_{j=1}^{dN_t} Y_j) - ln(S) = ln(\frac{S\prod_{j=1}^{dN_t} Y_j}{S}) = ln(\prod_{j=1}^{dN_t} Y_j) = \sum_{j=1}^{dN_t} lnY_j$$

alors

$$d(lnS) = \frac{1}{S}dS - \frac{1}{2}\frac{1}{S^2}dS^2 + \sum_{j=1}^{dN_t} lnY_j \qquad (1)$$

or on a

$$\frac{dS}{S} = (\mu - \lambda k)dt + \sigma dW \quad et \quad \frac{dS^2}{S^2} = \sigma^2 dt$$

En remplaçant dans (1)

$$d(lnS) = (\mu - \lambda k)dt + \sigma dW - \frac{1}{2}\sigma^2 dt + \sum_{j=1}^{dN_t} lnY_j$$

alors

$$d(lnS) = (\mu - \lambda k - \frac{1}{2}\sigma^2)dt + \sigma dW + \sum_{j=1}^{dN_t} lnY_j$$

On intègre entre 0 et t:

$$lnS_t - lnS_0 = (\mu - \lambda k - \frac{1}{2}\sigma^2)t + \sigma(W_t - W_0) + \sum_{j=1}^{N_t - N0} lnY_j$$

$$or \quad W_0 = 0 \quad et \quad P(N_0 = 0) = 1$$

*alors*

$$\ln(S_t) - \ln(S0) = (\mu - \lambda k - \frac{\sigma^2}{2})t + \sigma W_t + \sum_{j=1}^{Nt} \ln(Y_j) \tag{2}$$

et pour un j donné

$$Y_j \sim \mathcal{LN}(\mu_y, \sigma_y^2)$$

et donc

$$ln(Y_j) \sim \mathcal{N}(\mu_y, \sigma_y^2)\,.$$

Comme les sauts sont indépendants alors la somme des log des sauts est également gaussienne

$$\sum_{j=1}^{n} ln(Y_j) \sim \mathcal{N}(n\mu_y, n\sigma_y^2)$$

On sait que:

$$W_t \sim \mathcal{N}(0, t)$$

donc

$$\sigma W_t \sim \mathcal{N}(0, \sigma^2 t)$$

par suite et par indépendance des deux membres, on obtient:

$$\sigma W_t + \sum_{j=1}^{n} ln(Y_j) \sim \mathcal{N}(n\mu_y, n\sigma_y^2 + \sigma^2 t)$$

$$\sigma W_t + \sum_{j=1}^{n} ln(Y_j) \sim \mathcal{N}(n\mu_y, (\frac{n\sigma_y^2}{t} + \sigma^2)t) \sim n\mu_y + \sqrt{\frac{n\sigma_y^2}{t} + \sigma^2} * \sqrt{t}Z$$

$$avec \quad Z \sim \mathcal{N}(0, 1)$$

donc

$$\sigma W_t + \sum_{j=1}^{n} ln(Y_j) \sim n\mu_y + \sqrt{\frac{n\sigma_y^2}{t} + \sigma^2} * W_t$$

On remplace dans (2) et on obtient:

$$lnS_t - lnS_0 = (\mu - \lambda k - \frac{1}{2}\sigma^2)t + n\mu_y + \sqrt{\frac{n\sigma_y^2}{t} + \sigma^2} * W_t \tag{3}$$

La forme obtenue ressemble à la forme ci-dessous de Black & Scholes

$$lnS_t - lnS_0 = (\mu - \frac{1}{2}\sigma^2)t + \sigma W_t$$

Si on pose:

$$\sigma_n = \sqrt{\sigma^2 + \frac{n\sigma_y^2}{t}} \quad -> \quad -\frac{1}{2}\sigma_n^2 = -\frac{1}{2}(\sigma^2 + \frac{n\sigma_y^2}{t}) \tag{4}$$

On rajoute et on retranche $\frac{n\sigma_y^2}{2t}$:

$$lnS_t - lnS_0 = (\mu - \lambda k - \frac{1}{2}\sigma^2 - \frac{n\sigma_y^2}{2t} + \frac{n\sigma_y^2}{2t})\, t + n\mu_y + \sqrt{\frac{n\sigma_y^2}{t} + \sigma^2} * W_t$$

En utilisant (3) et (4):

$$lnS_t - lnS_0 = (\mu - \lambda k - \frac{1}{2}\sigma_n^2 + \frac{n\sigma_y^2}{2t})\, t + n\mu_y + \sigma_n W_t$$

On rassemble les termes

$$lnS_t - lnS_0 = (-\lambda k + n\mu_y + \frac{n\sigma_y^2}{2}) + (\mu - \frac{1}{2}\sigma_n^2)\ t + \sigma_n W_t$$

On passe à l'écriture exponentielle de l'actif.

$$S_t = S_0\, e^{(-\lambda k + n\mu_y + \frac{n\sigma_y^2}{2}) + (\mu - \frac{1}{2}\sigma_n^2)\ t + \sigma_n W_t}$$

ce qui donne finalement

$$S_t = S_0^{(n)} e^{(\mu - \frac{\sigma_n^2}{2})t + \sigma_n W_t}$$

$$\text{avec} \quad \sigma_n = \sqrt{\sigma^2 + \frac{n\sigma_y^2}{T}} \quad \text{et} \quad S_0^n = S_0 \exp(-\lambda k + n\mu_y + \frac{n\sigma_y^2}{2})$$

Sous une mesure martingale équivalente $Q_m$

$$S_t = S_0^{(n)} \exp^{(r-\frac{\sigma_n^2}{2})t+\sigma_n W_t^Q m}$$

Comme les deux dynamiques sont équivalentes à des constantes près, il est de même pour les prix des options. Pour BS on a

$$C(S_0, T) = S_0 N[d_1] - Ke^{-rT} N[d_2]$$

donc pour le prix de Merton sachant nombre de sauts $N_T = n$ se calcule de la même façon que BS. Et par suite le prix de Merton en général est donné par cette formule qui pondère le prix par la probabilités de chaque saut.

$$C(S_0^n, T) = \sum_{n=0}^{infini} C(S_0^n, T|N_t = n)(\lambda T)^n \frac{\exp(-\lambda T)}{n!} \tag{5}$$

# Simulation du Prix par Monte Carlo

```python
import numpy as np
from scipy.stats import norm
from scipy.optimize import minimize_scalar
N = norm.cdf

def BS_CALL(S0, K, T, r, sigma):
    d1 = (np.log(S0/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S0 * N(d1) - K * np.exp(-r*T)* N(d2)

def BS_PUT(S0, K, T, r, sigma):
    d1 = (np.log(S0/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma* np.sqrt(T)
    return K*np.exp(-r*T)*N(-d2) - S0*N(-d1)


def merton_jump_call(S0, K, T, r, sigma, m , v, lam):
    call = 0
    for n in range(40):
        r_n = r - lam*(m-1) + (n*np.log(m) ) / T
        sigma_n = np.sqrt( sigma**2 + (n* v** 2) / T)
        n_fact = np.math.factorial(n)
        call += (np.exp(-m*lam*T) * (m*lam*T)**n / (n_fact)) *
        BS_CALL(S0, K, T, r_n, sigma_n)

    return call


def merton_jump_put(S0, K, T, r, sigma, m , v, lam):
    put = 0
    for n in range(40):
        r_n = r - lam*(m-1) + (n*np.log(m) ) / T
        sigma_n = np.sqrt( sigma**2 + (n* v** 2) / T)
        n_fact = np.math.factorial(n)
        put += (np.exp(-m*lam*T) * (m*lam*T)**n / (n_fact)) \
                    * BS_PUT(S0, K, T, r_n, sigma_n)
    return put
```

```
S0 = 10
T = 1
r = 0.01
m = 0.1
v = 0.5
lam = 0.5
steps =255
Npaths =200000
sigma = 0.2
K =12
np.random.seed(3)

### Génération des sauts
S = merton_jump_paths(S0, T, r, sigma, lam, m, v, steps, Npaths)
### Calcul de la valeur du Call
mcprice = np.maximum(S[-1]-K,0).mean() * np.exp(-r*T)

cf_price =  merton_jump_call(S0, K, T, r, sigma, np.exp(m+v**2*0.5) , v, lam)

print('Merton Price =', cf_price)
print('Monte Carlo Merton Price =', mcprice)
print('Black Scholes Price =', BS_CALL(S0,K,T,r, sigma))
```

```
Merton Price = 1.137910683882444
Monte Carlo Merton Price = 1.1751902160997327
Black Scholes Price = 0.2340649396637784
```

Ce code ci-dessus calcule le prix d'une option CALL européenne avec le modèle de Merton avec les méthodes analytique et de Monte Carlo et avec le modèle de Black-Scholes

On remarque que le prix de l'option Call simulé avec le modèle de Merton à l'aide de la formule précédemment démontrée et celui simulé à l'aide de 200000 trajectoires de l'actif S à l'aide de Monte Carlo sont très proches.

On remarque également que le prix de Black-Scholes est beaucoup plus faible que ces deux valeurs ce qui s'explique bien par le fait que ce modèle ne tient pas compte des sauts.

# Partie 2

Dans cette seconde partie, nous allons travailler sur la calibration de ce modèle. Nos recherche et réflexions, codes et résultats sont notamment basés sur les travaux de Rama Cont and Peter Tankov et notamment leur livre « Financial Modelling with Jump Processes ».

# Définition du problème de calibration

Cette partie est un problème de calibration, on cherche donc les paramètres d'un modèle pour que ce dernier soit en adéquation avec la réalité du marché.

Il s'agit de l'inverse d'un problème de valorisation dans lequel on cherche le prix d'un actif évoluant dans un modèle dont les paramètres sont définis. L'objectif de la calibration est de trouver le modèle approprié à partir de données.

Dans notre cas, les données sont un set d'options CALL européennes que nous simulons nous même artificiellement avec le modèle de merton. Pour se rapprocher de la réalité et ajouter de l'aléas dans les données que nous considérons du marché, nous avons ajouté du bruit.

En reprenant le modèle de Merton définie dans la partie 1, l'objectif est ici de retrouver les paramètres (σ, λ, m, δ) tel que les prix obtenus avec le modèle correspondent à la réalité du marché.

L'actif sous-jacent respecte donc l'équation suivante

$$S_t = S_0 * exp\left(\left(r - \frac{\sigma^2}{2} - \lambda k\right)t + \sigma B_t + \sum_{i=1}^{N_t} Y_i\right)$$

Ce modèle peut être défini comme un modèle de Levy exponentiel. En effet, le terme $\left(r - \frac{\sigma^2}{2} - \lambda k\right)t + \sigma B_t + \sum_{i=1}^{N_t} Y_i$ est un processus de Levy étant la somme d'un mouvement brownien et d'un processus de poisson composé (les mouvement browniens et les processus de poisson composés sont des processus de lévy et cette caractéristique est stable par la somme) .

Par soucis de clarté et pour simplifier la présentation de la calibration du modèle de Merton, nous allons uniquement nous intéresser aux paramètres μ et λ et nous allons prendre des valeurs fixes pour r, σ et δ. Cependant, en utilisant la même méthode, il est aussi possible de déterminer ces paramètres.

Pour réaliser cette calibration, nous avons utilisé deux méthodes. Dans un premier temps nous allons présenter la résolution de ce problème avec la méthode des moindres carrés non linéaire et l'algorithme de Levenberg Marquardt. Ensuite nous allons réaliser une régularisation avec l'entropie relative pour avoir de meilleurs résultats. Les diverses méthodes sont expliquées mathématiquement puis appliquées dans un code Matlab présenté dans ce rapport.

# Présentation de la méthode utilisant l'algorithme de Levenberg Marquardt

Pour réaliser cette calibration, nous allons rechercher les paramètres du modèle de Merton qui ajustent au mieux les données au sens des moindre de carrés, c'est à dire qui réduisent au maximum le résidu, la somme des carrés des différence entre les données observés et les données modéliser.

$$Res = \sum_{i=1}^{n} (V_{marché} - V_{merton})^2$$

Ce problème devient alors un problème de minimisation, plus facile à résoudre mathématiquement, dans lequel nous recherchons les arguments qui réalise le minimum.  Pour le résoudre, nous allons utiliser l'algorithme de Levenberg Marquardt.

Cette méthode est itérative. Tant qu'un résultat satisfaisant n'est pas obtenu (tant que le résidu n'est pas suffisamment petit), nous appliquons aux paramètres utilisé précédemment des coefficients de descente.

La direction de la descente est obtenue grâce au jacobien et à la matrice hessienne de la manière suivante :

$$d = -M^{-1}J^{T}r$$

avec M défini par la relation : $$M = J^{T}J + \lambda * I$$

J la matrice jacobienne de la fonction du résidu par rapports aux paramètres à trouver

et λ donnant la vitesse de descente

Le coefficient λ est à définir avant de réaliser l'algorithme de Levenberg Marquardt. Plus il est grand, moins il y aura d'itérations mais cela peut entrainer des erreurs en fonction du point initial.

Pour cette méthode itérative, il faut aussi définir des paramètres d'entrées qui sont utilisées dans la première itération. Leur choix a de l'importance car plus ils sont proches du résultat final, plus l'algorithme sera rapide. De plus, si la surface n'est pas convexe, il peut y avoir des erreurs avec l'obtention de minimum locaux.

Dans notre cas, nous avons utilisé une simplification de cette méthode qui revient à trouver le minimum de d'une version pondérée du résidu :

$$Resp = \sum_{i=1}^{n} W * (V_{marché} - V_{merton})^2$$

# Application de cette première méthode

Dans le code Matlab, nous avons donc recherche à minimiser la fonction

$$Resp(\lambda, \mu) = \sum_{i=1}^{n} W * (V_{marché} - V_{merton}(\lambda, \mu))^2$$

avec W étant Vega, la dérivé du prix de Black-Scholes en fonction de la volatilité (qui capture la sensibilité du prix à la volatilité)

Vmarché qui correspond aux prix observés sur le marché

Vmerton qui correspond aux prix calculés avec le modèle de Merton

Dans le cas de cette formulation du résidu pondéré, on a supposé qu'il y avait n observations faites sur le marché.

Pour écrire cette fonction sur Matlab, j'ai utilisé l'opérateur de fonction 'handle' @ puis la fonction '`fminsearch`' nous a permis de trouver les paramètres réalisé le minimum du résidu pondéré.

$$arg \min_{\lambda,\mu} \sum_{i=1}^{n} W * (V_{marché} - V_{merton}(\lambda, \mu))^2$$

Cette méthode permet de résoudre facilement la calibration mais a aussi des limites et n'est pas fiable sous certains aspects. En effet, elle ne donne pas des solutions uniques (la surface obtenue pour le résidu pondéré n'est pas convexe), présente une instabilité par rapport aux valeurs initiales choisies. Le problème posé de cette manière est mal posé au sens de Hadamard

D'après Hadamard, un problème est bien posé si la solution existe, est unique et est stable (elle dépend de manière continue des données d'entrée). Pour obtenir une meilleure version, nous avons donc procédé à une régularisation comme présenté ci-après.

# Présentation de la régularisation utilisant l'entropie relative

L'objectif de la régularisation est de mieux poser le problème et ainsi d'avoir de meilleurs résultats. Pour ce faire nous réalisons une régularisation de Tikhonov c'est-à-dire que nous recherchons un problème bien posé proche de celui présenté précédemment (avec les mêmes caractéristiques qui font que le problème précédent donne des résultats corrects) en utilisant l'entropie relative.

L'entropie relative ou divergence de Kullback-Leibler distance mesure la proximité entre 2 mesures de probabilités équivalentes. Elle se présente de la manière suivante :

$$\varepsilon(\mathbb{Q}, \mathbb{P}) = E^{\mathbb{Q}}\left(ln\frac{d\mathbb{Q}}{d\mathbb{P}}\right) = E^{\mathbb{P}}\left(\frac{d\mathbb{Q}}{d\mathbb{P}} ln\frac{d\mathbb{Q}}{d\mathbb{P}}\right)$$

Ainsi cette régularisation nous permet de trouver un modèle exponentiel de Lévy neutre en risque qui donne les prix du marché avec la plus petite entropie relative possibles par rapport à un modèle choisi auparavant.

En effet, la mesure de probabilité Q trouvé par rapport à celle P du problème précédent existe et est de risque neutre d'après Girsanov.

Grâce à cette régularisation, on obtient donc un problème bien posé qui résout la calibration du modèle de Merton.

En effet, la surface du résidu pondéré est bien plus convexe donc on a l'existence et l'unicité de la solution (qui constitue les arguments du minimum observé sur la surface). De plus, la méthode reste simple de réalisation (notamment avec la formule simplifiée de l'entropie relative dans le cas de modèle de Lévy exponentiel).

# Réalisation de la régularisation

Pour mettre en place la régularisation, nous n'avons pas utilisé la formule générale

$$\varepsilon(\mathbb{Q},\mathbb{P}) = E^{\mathbb{Q}}\left(ln\frac{d\mathbb{Q}}{d\mathbb{P}}\right) = E^{\mathbb{P}}\left(\frac{d\mathbb{Q}}{d\mathbb{P}}ln\frac{d\mathbb{Q}}{d\mathbb{P}}\right)$$

mais nous avons utilisé une version simplifié appliqué aux modèle de Levy

$$\varepsilon(\mathbb{Q},\mathbb{P}) = \frac{T}{2\sigma^2} * \left[\lambda^{\mathbb{Q}} * \left(e^{\mu^{\mathbb{Q}}+\frac{\delta^{\mathbb{Q}2}}{2}} - 1\right) - \lambda^{\mathbb{P}} * \left(e^{\mu^{\mathbb{P}}+\frac{\delta^{\mathbb{P}2}}{2}} - 1\right)\right]^2 +$$
$$T * \lambda^{\mathbb{Q}} * ln\frac{\lambda^{\mathbb{Q}}*\delta^{\mathbb{P}}}{\lambda^{\mathbb{P}}*\delta^{\mathbb{Q}}} + T * \lambda^{\mathbb{P}} + T * \lambda^{\mathbb{Q}} * \left(-\frac{3}{2} + \frac{\left(\delta^{\mathbb{Q}2}+\left(\mu^{\mathbb{Q}}+\mu^{\mathbb{P}}\right)^2\right)}{2*\delta^{\mathbb{P}2}}\right)$$

avec pour chaque mesure les paramètres ( λ, μ, δ) du modèle de Levy associé.

Donc en reprenant ce qui est fait précédemment, nous avons recherche le minimum de la fonction

$$Reg(\lambda,\mu) = \sum_{i=1}^{n} W * \left(V_{marché} - V_{merton}(\lambda,\mu)\right)^2 + \alpha * \varepsilon(\mathbb{Q}(\lambda,\mu),\mathbb{P})$$

qui est la somme du résidus pondéré et de l'entropie relative. Le coefficient alpha apposé à l'entropie sert à moduler sa prise en compte dans la régularisation. En effet, s'il doit ne pas être négligeable pour que l'entropie ai un impact, il ne doit pas non plus être trop grand au risque d'écraser le modèle.

Dans le premier membre, on conserve les paramètre λ et μ du 'prior' modèle associé à la mesure P puis on ajoute l'entropie entre Q et P.

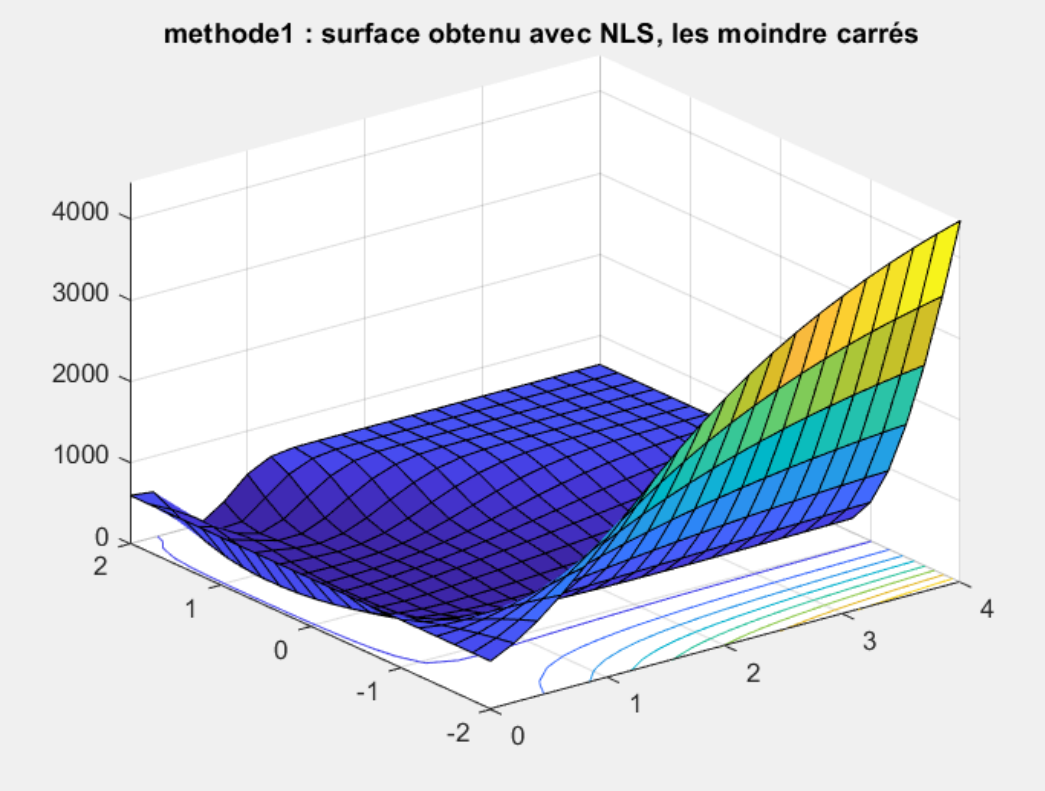
# Résultats obtenus

Le code (présent à la fin de la partie) donne pour chaque méthode une surface de la fonction étudiée (le résidu pondéré pour la première méthode et ce même résidu avec la régularisation pour la seconde méthode) et une approximation des paramètres du modèle de Merton.

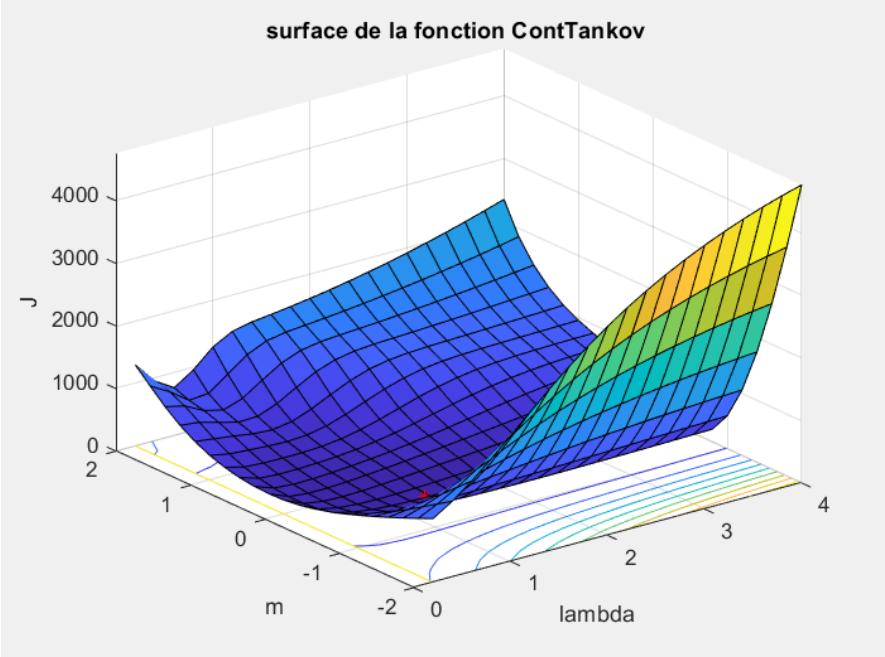
Pour ce qui est de la première méthode, les résultats de la calibration sont satisfaisants.

En effet le code donne des valeurs de μ proche de -0.3 et λ proche de 1.2. Cela correspond bien aux données entrées pour la création des données soi-disant observé sur le marché.

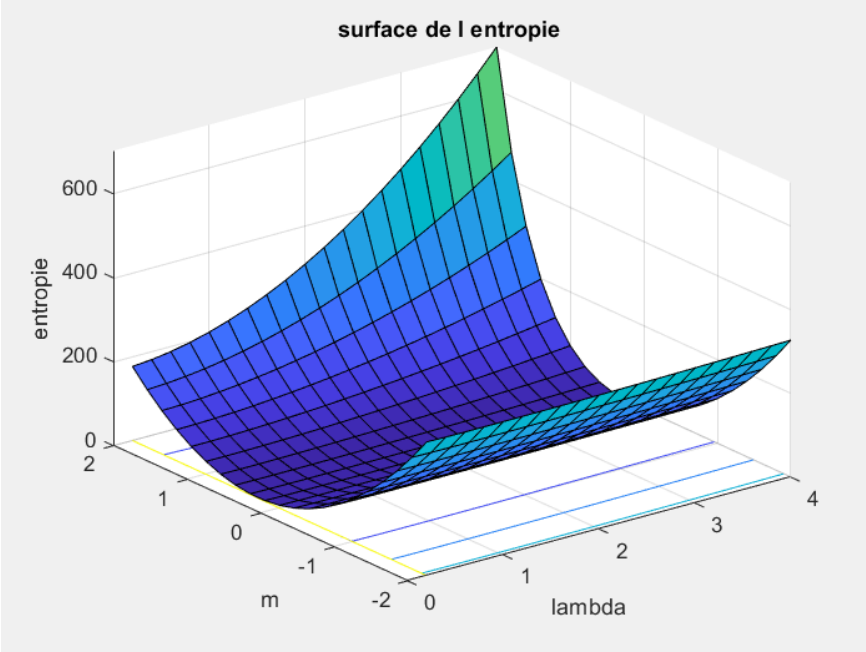
On remarque cependant que la surface n'est pas convexe ce qui indique que le problème n'est pas bien posé.



Avec la seconde méthode, on obtient les mêmes résultats en ce qui concerne λ et μ. La différence se fait sentir au niveau de la surface qui a une forme plus convexe, effet de la régularisation.



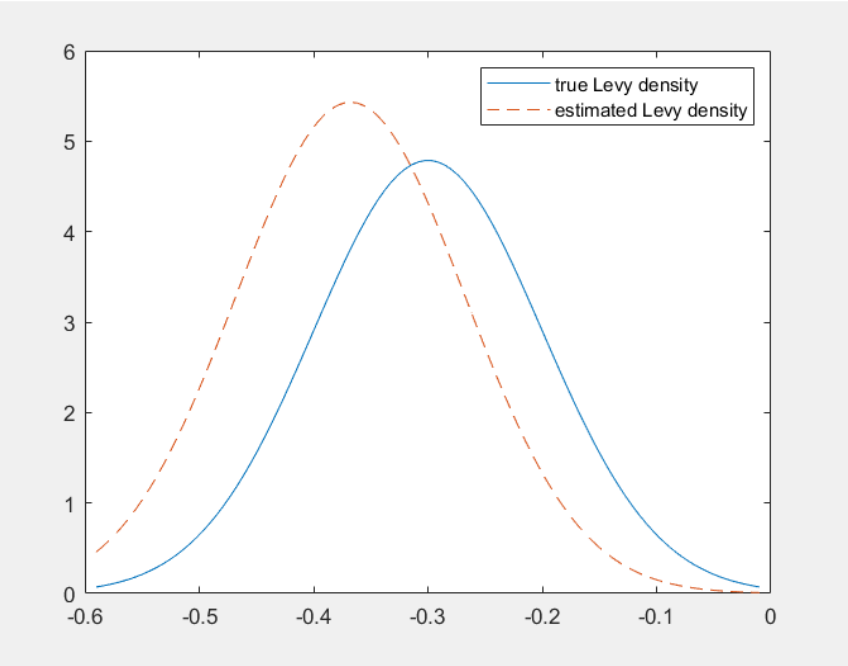

Cependant, la surface pourrait être encore plus convexe, notamment au niveau des extrémités. Cela vient surement du fait que l'entropie relative que nous avons calculée et dont la surface est présentée ci-dessous a des extremums en deçà de ce que nous attendions.



# Vérification de la qualité de la calibration en simulant le processus de Levy

Pour terminer le code et réaliser une vérification sur ce la calibration réalisée, nous avons calculé la fonction densité du process de Levy des données du marchés et celle obtenu avec la calibration.

Ce calcul de la densité a été réalisé grâce à l'approximation du processus de Levy avec la loi normale.




Graphiquement, nous pouvons donc voir une grande proximité entre les données du marché et les données obtenues avec la calibration. La différence peut s'expliquer par l'influence décevante de l'entropie dans notre cas comme expliqué dans la page précédente.

# Code Matlab

```
% PFE Partie 2 : problème de calibration
% Philip MALOU et Melek Ben Amar

%%%%%%%%%% définition des paramètres  %%%%%%%%%%%%%%%
T=rand(1,100);
k=100*ones(1,100);

s0=100;
r=0.001;
sigma=0.8;

delta0=0.1;
lambda0=1.2;
mu0= -0.3

%pricing de 100 trajectoires par Black-Scholes et Merton
cBS0=BSpricing(s0,k,sigma,T,r);
cM=mertonpricing (s0,T,k,r,mu0,delta0,lambda0,sigma);
theta=0.04;
Vmarche=cM+theta*randn(1,length(T));% ajout du bruit

% méthode 1:
Residu=@(lambda,mu,delta)
(1./vega(s0,k,sigma,T,r))*((mertonpricing
(s0,T,k,r,mu,delta,lambda,sigma)-Vmarche).^2)';

LM=fminsearch(@(x)Residu(x(1),x(2),delta0),[1.2 ;-1]);

lambda1 =LM(1);
mu1=LM(2);
disp('paramètres qui minimise le résidu')
disp(lambda1)
disp(mu1)

[lambdav,muv] = meshgrid(0:.2:4, -2:.2:2);
for i=1:length (lambdav)
for j=1:length (lambdav)
z(i,j)=erreur(lambdav(i,j),muv(i,j),delta0);
end
end

figure
surfc(lambdav,muv,z)
title 'methode1 : surface obtenue avec NLS, les moindre
carrés'
```

```
%méthode 2 :
[lambdav,muv] = meshgrid(0:.2:4, -2:.2:2);
for i=1:length (lambdav)
for j=1:length (lambdav)
z(i,j)=entropy(max(T), muv(i,j), delta0, lambdav(i,j),sigma,
mu1, delta0, lambda1);
end
end

figure
surfc(lambdav,muv,z)
xlabel('lambda')
ylabel('m')
zlabel('entropie')
title 'surface de l entropie '

alpha=0.04;
ContTankov=@(lambda,mu,delta)((1./vega(s0,k,sigma0,T,r))*((mer
tonpricing (s0,T,k,r,mu,delta,lambda,sigma)-
cM0).^2)'+alpha*(ones(1,length(T))*entropy(T,mu,delta,lambda,s
igma,mu0,delta0,lambda0)'));

reg=fminsearch(@(x)ContTankov(x(1),x(2),delta0),[1 ;-1]);
lambda2=reg(1);
mu2=reg(2);

[lambdav,muv] = meshgrid(0:.2:4, -2:.2:2);
for i=1:length (lambdav)
for j=1:length (lambdav)
a(i,j)=ContTankov(lambdav(i,j), muv(i,j), delta0);
end
end
reg=ContTankov(lambda2,mu2,delta0);

disp('min pour la fonction ContTankov')
disp(lambda2)
disp(mu2)

figure
surfc(lambdav,muv,a)
hold on
plot3 (lambdareg,mreg,reg,'r*')
xlabel('lambda')
ylabel('m')
zlabel('J')
title 'surface de la fonction ContTankov '
```

```
%vérification avec la densité du processus de Levy
x=m0+delta0*linspace(-2.9,2.9);
h0=lambda0*fonction_normpdf((x-m0)/delta0)/delta0;
h=lambdareg*fonction_normpdf((x-mreg)/delta0)/delta0;
figure
plot(x,h0,x,h,'--')
legend('true Levy density','estimated Levy density')

%%%%%%%% definition des fonctions %%%%%%%%%%%%%%%%%%
function cBS=BSpricing(s0,k,sigma,T,r)
dplus=(log (s0./k)+(r+1/2*sigma.^2).*T)./(sigma.*sqrt(T));
dminus=(log (s0./k)+(r-1/2*sigma.^2).*T)./(sigma.*sqrt(T));
%cBS=s0.*normcdf(dplus)-k.*exp(-r*T).*normcdf(dminus);
cBS=s0.*Nerf(dplus)-k.*exp(-r*T).*Nerf(dminus);
end

function [f]=Nerf(x)
f=1/2 * (1 + erf(x/sqrt(2)));
end

function cM=mertonpricing (s0,T,k,r,m,delta,lambda,sigma)
cM=0;
for n=0:4
sn=s0.*exp(n*m+ n*delta^2/2 -
lambda*exp(m+delta^2/2)+lambda*T);
sigman=sqrt(sigma.^2+n*delta^2./T);
cM=cM+exp(-r*T).*(exp(-
lambda*T).*(lambda*T).^n)/prod(1:n).*BSpricing(sn,k,sigman,T,r
);
end
end

function v = vega(s0,k,sigma,T,r)
dminus=(log (s0./k)+(r-1/2*sigma.^2).*T)./(sigma.*sqrt(T));
v=k.*exp(-r*T).*Nerf(dminus).*sqrt(T);
end

%calcul de l'entropy relative
function e =
entropy(T,mP,deltaP,lambdaP,sigma,mQ,deltaQ,lambdaQ)
e = (T/ (2*sigma^2))*[lambdaQ*(exp(mQ+0.5*(deltaQ^2))-1)-
lambdaP*(exp(mP+0.5*(deltaP^2))-
1)]^2+T*lambdaQ*log(lambdaQ*deltaP/(lambdaP*deltaQ))+T*lambdaP
+T*lambdaQ*(-3/2+(deltaQ^2+(mQ+mP)^2)/(2*deltaP^2));
end

function y = fonction_normpdf(x)
mu= 0 ;
sigma =1 ;
y=(1./(sqrt(2*pi)*sigma)).*exp(-(x-mu).^2./(2*sigma.^2));
end
```

# Conclusion et Sources

# Conclusion :

Dans ce projet de fin d'étude, nous avons étudié les différentes caractéristiques du modèle de de valorisation des options avec sauts de Merton en se basant sur nos connaissances du modèle de Black-Scholes et sur les travaux de Robert Merton mais aussi de Rama Cont et Peter Tankov.

Dans un premier temps, nous avons simulé des trajectoires des actifs en modèle de Merton en ajoutant la composante du processus de Poisson composé à la dynamique de Black-Scholes.Nous avons réalisé le pricing d'une option CALL en comparant les modèles de Merton et Black-Scholes.

Par la suite, nous avons fait la calibration du modèle de Merton avec la méthode des moindres carrés ainsi qu'une régularisation grâce à l'entropie. Ainsi nous avons étudié le modèle dans les deux sens, directement avec le calcul du prix d'une option puis à l'opposé avec la détermination des paramètres du modèle grâce à des données du marché.

Pour ouvrir ce sujet, nous souhaitons apporter quelques éléments supplémentaires de comparaison entre les modèles de Black-Scholes et le modèle de Merton.

Que ce soit en termes de simulation ou de calibration, le modèle de Merton est, comme nous l'avons vu, plus performant que Black-Scholes. En effet l'introduction des sauts, qui représente la discontinuité des prix, permet d'obtenir des résultats plus proches de la réalité des marchés et plus fiables quand il s'agit du pricing d'une option.

On peut affirmer que le modèle de Black-Scholes est complet, ce qui n'est pas le cas pour Merton. Cela est dû au fait qu'il est impossible de complètement hedger le risque apporté par les sauts imprévisibles de Merton causé par le processus de poisson composé.

Mais cependant, en ce qui concerne les courbes de smile de volatilité, il apparaît que Merton est plus performant que BS surtout lorsqu'il s'agit du court terme.

# Sources

Cont, R. and Tankov, P., Calibration of jump-diffusion option pricing mod- els: A robust non-parametric approach, Rapport Interne 490, CMAP, Ecole Polytechnique, 2002. Forthcoming in: Journal of Computational Finance.

Cont, R. and Tankov, P., Financial Modeling With Jump Processes, CHAPMAN and HALL/CRC Financial Mathematics Series, CHAPMAN and HALL/CRC, 2004.

Cont, R. and Tankov, P., Non-parametric calibration of jump diffusion option pricing models,

Nicole El Karoui - Emmanuel Godet, Les outils stochastiques des marchés financiers. Les Editions de l'Ecole Polytechnique.

C. He, J.S. Kennedy, T. Coleman, P.A. Forsyth, Y. Li, K. Vetzal Calibration and Hedging under Jump Diffusion. Review of Derivative Research(2006)9:1-3