# Note Call-Spread Dupire EDP

Kerlaouezo Erwan

Janvier 2025

Cette note décrit en détail les étapes nécessaires pour valoriser un call spread sur le CAC 40 en utilisant la méthode des équations aux dérivées partielles, selon le modèle de volatilité locale de Dupire. Elle présente également les objets utilisés ainsi que les différentes fonctions du code indispensables à cette valorisation. Le code couleur est le suivant: **TURQUOISE** pour les variables, **VERT** pour les objets et **JAUNE** pour les fonctions. La première partie "Surface de volatilité implicite" n'est pas quelque chose d'essentiel dans ce projet car les données étaient récupérées via SuperDerivatives et le sujet de ce prejet n'est pas la construction de surface de volatilité implicite.

## 1 Surface de volatilité implicite

Si vous avez SuperDerivatives de disponible sur votre excel, alors la première étape est de récupérer la surface de volatilité implicite disponible sur Ice. Tous les jours, il faut aller dans ce répertoire :

`...PricerCS\Inputs-Outputs\CACSDSurfaces`

1. **Accéder à la feuille "Data"** :
   - Ouvrez le fichier Excel et sélectionnez l'onglet `Data`.
2. **Activer l'add-in** :
   - Cliquez sur l'onglet **Développeur** en haut de l'écran.
   - Sélectionnez **Compléments Excel**.
   - Cochez la case `Sdexceladdineq64`, puis cliquez sur **OK**.
3. **Utiliser l'add-in** :
   - Une nouvelle barre **Compléments** apparaîtra à côté de l'onglet **Développeur**.
   - Cliquez sur cette barre, puis sélectionnez **SD EQ** pour vous connecter avec vos identifiants.
4. **Récupérer les données** :
   - Cliquez sur le bouton **Calculate** pour effectuer le premier calcul.
   - Ensuite, allez à la feuille `Surface` et cliquez à nouveau sur **Calculate** pour obtenir la surface de volatilité implicite.

Le format du fichier Excel n'est pas à changer car la récupération des données sur python dépend de ce format. On peut ajouter des lignes si on le souhaite mais il est impossible de rajouter une colonne. Maintenant que nous avons notre surface de volatilité implicite, on peut s'intéresser à l'adapter au modèle à volatilité locale de Dupire.

## 2 Main.py

Il faut ensuite se rendre une nouvelle fois dans le premier répertoire donné dans la page 1 et ouvrir le fichier **main.py**. On va présenter et expliquer toutes les fonctions créées et utilisées pour arriver à notre prix. Nous allons ici détailler l'ensemble de ce qui se trouve et ce qui est utilisé dans le main pour pouvoir valoriser notre call spread.

### 2.1 DataFetcher.py

L'objet **DataFetcher** permet de récupérer les données de volatilité implicite sous forme de dictionnaire qui sont plus simples à manipuler pour la suite. Il s'initialise avec un **path** et avec la possibilité d'ajouter le nom d'une **feuille** spécifique. On initialise notre objet avec le bon path (mis dans le main) des données de volatilités implicites et on les prend sur la feuille **Surface**:

dataFetcher = **DataFetcher**(path=path, sheet_name="Surface")

#### 2.1.1 getData()

Cette fonction dans la classe **DataFetcher** permet de rassembler les données et de les mettre en forme pour une utilisation simple. Elle remplit deux dictionnaires et une liste:

1. quotesDic : Dictionnaire de toutes les cotations de notre surface avec comme clefs les time to maturity et en values la liste de toutes les cotations associées à cette maturité.
2. forwardsRatesDivs : Dictionnaire qui a également comme clefs les time to maturity et en values la liste avec [forward, rate, dividende] associée à chaque maturité.
3. expiries : la liste des time to maturity.

À savoir que le dictionnaire quotesDic est rempli d'objets **Quote** (défini dans le fichier **Option.py**) qui s'initialisent de la manière suivante:

quote = **Quote**(strike, implied_volatility, optionType)

Si jamais le fichier Excel qui contient les données de surface de volatilité est changé, c'est dans ce fichier python (DataFetcher.py) qu'il faudra modifier le code pour qu'il renvoie les données dans le même type de format qui est utilisé actuellement sous peine d'impossibilité de calculer les prix du call-spread.

### 2.2 SVI.py

La surface de volatilité implicite disponible sur Super Derivatives peut présenter des incohérences à certains endroits, ce qui peut entraîner des complications lors des calculs ultérieurs. Pour remédier à cela, nous allons créer une surface de volatilité implicite à partir des données récupérées sur Super Derivatives, en utilisant des modèles SVI. Ces modèles permettent de calibrer des paramètres pour chaque maturité, afin d'obtenir un smile de volatilité continu et lisse. Nous utilisons le modèle dit SSVI suivant:

$$w(k,t) = \frac{\theta}{2}\left(1 + \rho\phi(\theta)k + \sqrt{(\phi(\theta)k + \rho)^2 + 1 - \rho^2}\right) \tag{1}$$

Où $w(k,t) = \sigma(k,t)^2 \cdot t$ est appelée la variance totale implicite, $k = \ln\left(\frac{K}{F_t}\right)$, $\phi(\theta)$ une fonction que l'on définit plus bas et qui permet d'assurer que notre surface sera sans opportunité d'arbitrage et $\rho$ est la courbure du smile de volatilité pour le time to maturity $t$.

#### 2.2.1 SVIAOA(k, timeToMaturity ($t$), theta, phiTheta ($\phi(\theta)$), rho)

Cette fonction retourne la volatilité implicite calculée avec notre modèle SVI. Elle retourne la valeur suivante:

$$\sigma(k,t) = \sqrt{\frac{\frac{\theta}{2}\left(1 + \rho\phi(\theta)k + \sqrt{(\phi(\theta)k+\rho)^2 + 1 - \rho^2}\right)}{t}} \tag{2}$$

#### 2.2.2 phi(theta, etha, gamma)

Cette fonction retourne la valeur de $\phi$ pour une valeur de $\theta \geq 0$ donnée sachant que l'on a $\gamma \in [0; \frac{1}{2}]$, ainsi que $\eta \leq \frac{2}{\sqrt{1+|\rho|}}$. On a alors que cette fonction retourne la valeur suivante:

$$\phi(\theta;\eta,\gamma) = \frac{\eta}{\theta^{\gamma}\cdot(1+\theta)^{1-\gamma}} \tag{3}$$

À noter que cette fonction permet de satisfaire les conditions de non-arbitrage de notre surface. Pour plus d'informations vous pouvez aller regarder l'article (1).

#### 2.2.3 FirstDerivativeAOA(k, theta, phiTheta ($\phi(\theta)$), rho)

Cette fonction permet de calculer la dérivée première de la variance totale implicite par rapport à $k$. La dérivée première est donnée par:

$$\frac{\partial w}{\partial k}(k) = \frac{\theta\phi(\theta)}{2}\left(\rho + \frac{\phi(\theta)k+\rho}{\sqrt{(\phi(\theta)k+\rho)^2+1-\rho^2}}\right) \tag{4}$$

#### 2.2.4 SecondeDerivativeAOA(k, theta, phiTheta ($\phi(\theta)$), rho)

Cette fonction permet de calculer la dérivée seconde de la variance totale implicite par rapport à $k$. La dérivée seconde est donnée par:

$$\frac{\partial^2 w}{\partial k^2}(k) = \frac{\theta\phi(\theta)^2(1-\rho^2)}{2\left((\phi_\theta k+\rho)^2+1-\rho^2\right)^{3/2}} \tag{5}$$

#### 2.2.5 SVI(quotesDic, forwardsRatesDivs)

Ces 4 fonctions sont utilisées dans l'objet SVI notamment pour la calibration mais également lors de la création de la surface de volatilité locale. Notre objet SVI est défini de la manière suivante:

svi = SVI(quotesDic, forwardsRatesDivs)

Dans cet objet, quotesDic et forwardsRatesDivs sont les données qui sont récupérées de l'objet DataFetcher présenté plus haut.

L'objectif de cet objet est de créer une surface de volatilité implicite la moins arbitrable possible à l'aide d'une calibration de qualité.

Dans **Main.py** on utilise la fonction présentée dans la page suivante qui nous crée notre surface à l'aide des modèles SVI.

### 2.2.6 CreateSurfaceAOA(strikes=None)

Cette fonction crée une surface SVI à partir des équations ci-dessus en calibrant, pour chaque maturité, les paramètres du modèle au smile du CAC 40. Cette fonction crée donc une surface et un dictionnaire qui contient tous les paramètres calibrés pour chaque maturité. On stocke les paramètres pour pouvoir ensuite les utiliser lors de la construction de la surface de volatilité locale.

Plus précisément, on a les étapes suivantes qui se répètent pour chaque maturité de notre surface de volatilité implicite:

1. On récupère toutes les quotes associées à la maturité, quotes_maturity.
2. On fait appel à la fonction GetSVIValuesForMaturityAOA(quotes_maturity, strikes, timeToMaturity) qui, pour une maturité donnée, calibre les paramètres du SVI et ensuite on calcule notre smile SVI avec ces paramètres. Pour la calibration on utilise la fonction SVICalibrationAOA(quotes_maturity, timeToMaturity) qui calibre les paramètres du SVI en respectant le domaine de solutions souhaité. Pour calibrer on décide d'utiliser l'algorithme de Nelder-Mead en cherchant à minimiser l'écart entre le smile de volatilité implicite du CAC 40 et notre smile SVI. On cherche donc à trouver l'ensemble de paramètres $\Psi = (\theta, \eta, \gamma, \rho)$ qui se situe dans le domaine $D$ suivant :

$$D := \begin{cases} \theta \geq 0, \\ \gamma \in \left[0, \frac{1}{2}\right], \\ |\rho| < 1, \\ \eta \leq \frac{2}{\sqrt{1+|\rho|}}. \end{cases}$$

   Pour cela on utilise la fonction de coût suivante :

$$C(\Psi) = \underset{\Psi \in D}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\sigma_{i,svi}(\Psi) - \sigma_{i,market}\right)^2 \tag{6}$$

   Il y a la possibilité d'ajouter comme poids le vega des options pour pouvoir mieux calibrer les options proches de la monnaie (les plus liquides):

$$C(\Psi) = \underset{\Psi \in D}{\operatorname{argmin}} \sum_{i=1}^{n} \frac{\partial V_{i,market}}{\partial \sigma_{i,market}} \left(\sigma_{i,svi}(\Psi) - \sigma_{i,market}\right)^2 \tag{7}$$

   Il se trouve que pour la surface du CAC 40, la calibration est de meilleure qualité si l'on ne considère pas le vega dans notre fonction de coût. On prend donc la fonction de coût définie dans (6) pour calibrer nos modèles SVI. Toujours dans la fonction GetSVIValuesForMaturityAOA(quotes_maturity, strikes, timeToMaturity), une fois les paramètres calibrés, on les stocke dans le dictionnaire parameters en mettant comme clef le timeToMaturity et en valeur la liste des paramètres calibrés. Ensuite on construit notre smile de volatilité implicite et on retourne la liste de Quote associée. À noter que dans la fonction de calibration on ajoute des pénalités si nos solutions pour $\Psi$ ne sont pas dans le domaine souhaité $D$.

Pour finir, on stocke chaque liste de Quote dans le dictionnaire surface avec également timeToMaturity comme clef et en value la liste de Quote avec les bons strikes et volatilités implicites. Ainsi, nous avons maintenant notre surface de volatilité implicite construite par SSVI qui est sans opportunité d'arbitrage. On peut donc maintenant s'intéresser à la construction de notre surface de volatilité locale avec le modèle de Dupire.

### 2.3 LocalSurface.py

Maintenant que nous avons une surface de volatilité implicite de qualité, nous pouvons générer notre surface de volatilité locale.

#### 2.3.1 LocalSurface(quotesDic, forwardsRatesDivs, SurfaceType, parameters)

Étant donné que nous avons une surface de volatilité implicite construite par SVI et que nous avons tous les paramètres associés à toutes les maturités de notre surface, il devient facile de créer notre surface de volatilité locale. En effet, la formule de base de la volatilité locale est donnée par:

$$\sigma_{loc}^2(K,t) = \frac{\frac{\partial C}{\partial t} - (r-q)\left(C - K\frac{\partial C}{\partial K}\right)}{\frac{1}{2}K^2\frac{\partial^2 C}{\partial K^2}} \tag{8}$$

Comme nous avons notre surface SVI, il est facile pour nous d'avoir nos variances totales implicites. L'équation (8) peut être dérivée en une nouvelle formule qui utilise la variance totale implicite pour exprimer notre volatilité locale avec $k = \ln\left(\frac{K}{F}\right)$:

$$\sigma_{loc}^2(k,t) = \frac{\frac{\partial w}{\partial t}}{\left(1 - \frac{k}{w}\frac{\partial w}{\partial k} + \frac{1}{2}\frac{\partial^2 w}{\partial k^2} + \frac{1}{4}\left(-\frac{1}{4} - \frac{1}{w} + \frac{k^2}{w}\right)\left(\frac{\partial w}{\partial k}\right)^2\right)} \tag{9}$$

Cette formule est très pratique car avec nos SVI, nous avons pour une maturité donnée accès à tous les $w$, ainsi qu'à toutes les dérivées première et seconde présentes au dénominateur définies par (4) et (5). Il ne reste plus qu'à s'assurer que $\frac{\partial w}{\partial t} \geq 0$, mais normalement c'est le cas car le modèle SVI choisi calibre les paramètres pour satisfaire cette condition de positivité. On applique cette formule pour toutes les variances totales implicites et nous avons ainsi notre surface de volatilité locale.

#### 2.3.2 CreateSurface(logStrike)

Cette fonction, appelée dans le **Main.py**, permet de construire la surface de volatilité locale. Avec logStrike qui est une variable booléenne pour dire si on laisse les strikes originaux (False) ou si on passe en échelle log par rapport au forward (True). Cette fonction itère sur toutes les maturités de la surface de volatilité implicite, où pour chaque maturité elle prend la liste de Quote associée, récupère les parameters associés calculés lors de la calibration dans SVI, puis pour chaque strike de notre smile on calcule les volatilités locales associées avec la formule (9).

À savoir que lors de la création de notre objet LocalSurface, nous calculons toutes les dérivées de notre variance totale implicite par rapport au temps avec la fonction GetAllDerivative() qui calcule pour chaque couple $(k,t)$ de notre surface la dérivée par rapport au temps de ce point. On utilise l'objet CubicSpline fourni par python qui permet d'interpoler pour ensuite calculer les dérivées empiriques:

$$\frac{\partial w}{\partial t}(k,t) = \begin{cases} \frac{w(k,t+\epsilon)-w(k,t)}{\epsilon} & \text{si} \quad t \leq t_{min}, \\ \frac{w(k,t+\epsilon)-w(k,t-\epsilon)}{2\epsilon} & \text{si} \quad t_{min} < t < t_{max}, \\ \frac{w(k,t)-w(k,t-\epsilon)}{\epsilon} & \text{si} \quad t_{max} \leq t. \end{cases}$$

Où $t_{min}$ et $t_{max}$ sont respectivement le plus petit time to maturity et le plus grand time to maturity de notre surface de volatilité locale. De plus, dans le code, on définit $\epsilon = 1e-6$ et s'appelle EPSILON dans le fichier **LocalSurface.py**.

### 2.4 PDE.py

L'équation aux dérivées partielles de Dupire pour la volatilité locale avec dividendes est donnée par :

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma_{loc}^2(S,t)S^2\frac{\partial^2 V}{\partial S^2} + (r-q)S\frac{\partial V}{\partial S} - rV = 0 \tag{10}$$

Contrairement à l'équation aux dérivées partielles (EDP) de Black-Scholes, qui suppose une volatilité constante pour tous les spots et maturités, nous prenons ici en compte la dynamique de la volatilité locale pour chaque spot et chaque maturité. L'objet PricerPDE doit être défini comme suit :

pricerPDE = PricerPDE(scheme, model, y_max, y_numbers, x_max, x_numbers, CFL, localSurface, forwardsRatesDivs)

- scheme : Le schéma utilisé pour le pricing (par exemple, Explicit, Implicit, Crank-Nicholson).
- model : Le modèle fourni en entrée (dans ce cas, celui de Black-Scholes). Bien que ce modèle n'impacte pas directement le pricing, il est essentiel pour le paramètre CFL.
- CFL : Le nombre de Courant, qui garantit la stabilité de l'algorithme pour un schéma explicite (nous détaillerons ce point plus tard).
- y_max : Le spot le plus éloigné considéré (correspondant à 4S_initial dans le code).
- y_numbers : Le nombre de points souhaités sur l'axe des spots (axe y).
- x_max : La valeur maximale de la variable x considérée.
- x_numbers : Le nombre de points souhaités sur l'axe x.
- localSurface : La surface de volatilité locale construite précédemment avec l'objet LocalSurface en utilisant la fonction CreateSurface().
- forwardsRatesDivs : Un dictionnaire contenant, pour chaque maturité, les informations sur le forward, le taux et le dividende.

#### 2.4.1 Calibration de CFL

À noter que x_numbers est déduit à partir du nombre de points que l'on souhaite sur l'axe des ordonnées y_numbers et du nombre de Courant qui permet d'avoir des calculs stables CFL. Ce nombre CFL est défini comme suit avec $\Delta t = \frac{x_{max}}{N_x}$ et $\Delta y = \frac{y_{max}}{N_y}$:

$$\text{CFL} = \frac{\frac{\Delta t}{(\Delta y)^2}}{\frac{1}{(\sigma S_{max})^2}} \tag{11}$$

Dans la littérature, souvent on impose CFL $\leq 1$ ou bien CFL $\leq \frac{1}{2}$. Lors de la création de l'objet PricerPDE, on choisit un nombre CFL. Nous avons $x_{max}$, $\Delta y_{max}$ et $N_y$ (x_max, y_max, y_numbers). La variable à calculer reste donc x_numbers que l'on calcule à partir du nombre CFL pour garantir la stabilité des calculs:

$$N_x = \frac{x_{max} N_y^2 \sigma^2}{\text{CFL}} \tag{12}$$

Ainsi, si l'on augmente le nombre CFL on diminue $N_x$, donc on diminue le temps de calcul mais on augmente l'incertitude sur la stabilité des calculs. Dans la pratique, pour y_numbers=120 et CFL=1, les calculs sont stables et très rapides.

### 2.4.2 Price(option)

La fonction Price calcule le prix d'une option vanille en utilisant des méthodes de différences finies dans un cadre de volatilité locale. Cette fonction prend en entrée un objet VanillaOption et utilise divers paramètres pour effectuer le calcul. L'équation (10) peut se ramener à une écriture matricielle (ici pour le schéma explicite) après un changement de variable sur l'axe des abscisses (le temps) en posant $\tau = T - t$:

$$\frac{V_{\tau+\Delta t} - V_\tau}{\Delta t} = (r-q)\cdot S\cdot D_1\cdot V_\tau + \frac{1}{2}\cdot\Sigma^2_{loc}(S,t)\cdot S^2\cdot D_2\cdot V_\tau - r\cdot V_\tau \tag{13}$$

La matrice D1 est définie pour calculer la dérivée première. Les coefficients sont calculés comme suit :

$$D1 = \begin{pmatrix} \frac{1}{\Delta y} & -\frac{1}{\Delta y} & 0 & \dots & 0 \\ \frac{1}{2\Delta y} & 0 & -\frac{1}{2\Delta y} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \frac{1}{2\Delta y} & 0 & -\frac{1}{2\Delta y} \\ 0 & \dots & 0 & \frac{1}{\Delta y} & -\frac{1}{\Delta y} \end{pmatrix}$$

La matrice D2 est définie pour calculer la dérivée seconde :

$$D2 = \begin{pmatrix} \frac{1}{(\Delta y)^2} & -\frac{2}{(\Delta y)^2} & \frac{1}{(\Delta y)^2} & \dots & 0 \\ \frac{1}{(\Delta y)^2} & -\frac{2}{(\Delta y)^2} & \frac{1}{(\Delta y)^2} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \frac{1}{(\Delta y)^2} & -\frac{2}{(\Delta y)^2} & \frac{1}{(\Delta y)^2} \\ 0 & \dots & \frac{1}{(\Delta y)^2} & -\frac{2}{(\Delta y)^2} & \frac{1}{(\Delta y)^2} \end{pmatrix}$$

Le calcul du prix de l'option se fait de manière itérative, où pour chaque $\tau \in \{0, \cdots, x_{max}\}$ qui correspondent à toutes les valeurs sur l'axe des abscisses que l'on considère et qui dépendent de la maturité de l'option renseignée dans la variable x_max et du nombre de points que l'on veut considérer x_numbers.

Dans le code, on laisse la possibilité à l'utilisateur de choisir son schéma. On peut montrer qu'à partir de l'équation (10) pour les schémas Explicit, Implicit et de Crank-Nicholson, on a les égalités suivantes en notant $C = -rI_n + (r-q)SD_1 + \frac{1}{2}\Sigma^2_{loc}(S,t)S^2D_2$ :

$$V_{\tau+\Delta t} = [I_n + \Delta t C]\, V_\tau \tag{14}$$

$$V_{\tau+\Delta t} = [I_n - \Delta t C]^{-1}\, V_\tau \tag{15}$$

$$V_{\tau+\Delta t} = \left[I_n - \frac{1}{2}\Delta t C\right]^{-1}\left[I_n + \frac{1}{2}\Delta t C\right] V_\tau \tag{16}$$

Avec $S = \text{diag}(S_n, \cdots, 0)$ la matrice diagonale qui contient tous nos spots choisis à partir du spot max y_max et du nombre de points souhaités y_numbers. La matrice $\Sigma^2_{loc}(S,t) = \text{diag}(\sigma^2_{loc,n}, \cdots, \sigma^2_{loc,0})$ est la matrice des volatilités locales, bien sélectionnée sur la bonne maturité de notre surface de volatilité locale donnée en input de l'objet localSurface.

# 3 Bibliographie


1. "Arbitrage-free SVI volatility surfaces" de Jim Gatheral et Antoine Jacquier, 2024.
2. "Derivation of Local Volatility" de Fabrice Douglas Rouah, 1998.
3. "Surface de volatilité" de Peter Tankov, 2015.