Pricer Valo

# Objectifs

La mise en place de ce projet « pricer Valo » a pour but de :

1. Créer un environnement de travail, pour les travaux des projets de la practice risque et ceux des sujets de stagiaires
2. Développer une librairie de pricing axis interne
3. Contribuer à la montée en compétences des inter-contrats/stagiaires sur différents langages de programmation (VBA, C#, C++, C#)
4. Contribuer à la montée en compétences des inter-contrats/stagiaires sur les pricing des produits dérivés (taux, equity, crédit etc…)
5. Préparer les consultant à des soutenance sur des thématiques de pricing / validations de produits dérives

# Fonctionnement

Nous vous proposons une première approche pour un environnement de travail *(cf.* **§ *Environnement de travail***) que nous vous recommandions. Après cela, la mise en développement jusqu’à la mise en production se fait de manière suivante :

1. Développement ou migration
   1. Développer votre librairie sur Visual studio en C# ou C++ *(cf*. **§ *Pricing des produits*** …)
   2. Créer le fichier .dll (cf. la section ***Création du fichier .dll***)
2. Intégration : *(cf*. **§ *intégration de C# à VBA***)
3. Déploiement : *(cf*. **§ *Déploiement)***

# Environnement de travail

## Installation des outils

### Étape 1 : Installation de Visual Studio Code (VSC)

1. **Téléchargement :** Téléchargez et installez Visual Studio Code à partir du site officiel.
   1. Téléchargez et installez la dotnet: [Framework](https://dotnet.microsoft.com/en-us/download/dotnet/6.0).
      1. choisir la version installer x-64 au niveau de l’OS Windows SDK 6.0.418 (par exemple).
   2. Téléchargez et installez l'éditeur de code VSC: [Editeur](https://code.visualstudio.com/).
2. **Installation :** Suivez les instructions d'installation spécifiques à votre système d'exploitation.

### Étape 2 : Installation de Visual Studio

1. **Téléchargement :** Téléchargez et installez Visual Studio à partir du site officiel.
   1. Téléchargez et installez la SDK: [Framework2](https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/).
      1. Choisir le download the installer.
   2. Téléchargez et installez l’IDE: [IDE](https://visualstudio.microsoft.com/fr/downloads/).
2. **Installation :** Suivez les instructions d'installation spécifiques à votre système d'exploitation.

### Étape 3 : Installation de Python

1. **Téléchargement :** Téléchargez et installez Python à partir du site officiel
   1. Python: [python](https://www.python.org/downloads/release/python-3116/).
   2. Version: ***3.11.6*** puis tout en bas de page la version ***Windows installer (64-bit).***
2. **Sélection du PATH:** . Assurez vous de cocher la case "Add Python to PATH" pendant l'installation.
3. **Installation**: Suivez les instructions d'installation spécifiques à votre système d'exploitation.
4. **Vérification de l'installation :** Ouvrez une nouvelle fenêtre de terminal (Command Prompt) et tapez python --version ou python -V. Vous devriez voir la version de Python que vous venez d'installer.

### Étape 4: Installation de Git

1. **Téléchargement :** Téléchargez et installez Git à partir du site officiel
   1. Git: [git](https://git-scm.com/download/win).
      1. Version: prendre la standalone Installer ***64-bit Git for windows Setup***.
2. **Installation:** Suivez les instructions d'installation spécifiques à votre système d'exploitation.
3. **Vérification de l'installation :** Ouvrez une nouvelle fenêtre de terminal (Command Prompt) et tapez git

### Étape 5: Installation de Git Extension

1. **Téléchargement :** Téléchargez et installez Git Extension à partir du site officiel
   1. Git Extension: [git extension](https://gitextensions.github.io/).
2. **Installation:** Suivez les instructions d'installation spécifiques à votre système d'exploitation.

### Étape 6: Installation de KDIFF3

1. **Téléchargement :** Téléchargez et installez Kdiff3 à partir du site officiel
   1. Kdiff3: [kdiff3](https://sourceforge.net/projects/kdiff3/).
2. **Installation:** Suivez les instructions d'installation spécifiques à votre système d'exploitation.

***Une fois l’installation terminée allez dans Git Extension.***

## Configuration de Git Extension

### Étape 1 : Configuration de l’identité (user et e-mail)

1. Ouvrez Git Extensions.
2. Dans la barre d'outils, cliquez sur "Repository" > "Repository Settings".
3. Sous l'onglet "Settings", vous pouvez configurer votre identité (user et e-mail) dans les champs correspondants.
4. Cliquez sur "OK" pour enregistrer les modifications.

### Étape 2 : Configuration du gestionnaire de Diff/Merge (KDiff3)

Git Extensions utilise généralement un outil externe pour les opérations de différenciation et de fusion. Si vous avez installé un autre outil autre que KDiff3, vous pouvez également le configurer comme suit :

1. Allez dans "Repository" > "Repository Settings".
2. Aller dans "Git" puis "Config".
3. Sous l'onglet "Diff/Merge", configurez votre outil de fusion et de différenciation. Par exemple, pour KDiff3, configurez le chemin vers l'exécutable.
4. Sous l'onglet "Merge Tool": Choisissez "KDiff3" dans la liste déroulante.
5. Sous l’onglet "Path to MergeTool": Spécifiez le chemin vers l'exécutable de KDiff3 avec "C:/Program Files/KDiff3/kdiff3.exe". Vous pouvez le trouver dans le répertoire d'installation de KDiff3.
6. Configurez d'autres paramètres selon vos préférences (optionnel).
7. Cliquez sur "OK" pour enregistrer les modifications.

### Étape 3 : Vérification des configurations

1. Après avoir apporté des modifications, assurez vous de redémarrer Git Extensions pour que les configurations prennent effet.
2. (Optionnel) Vous pouvez également vérifier les configurations en utilisant Git Bash et les commandes Git, par exemple :

git config --global --list

### Étape 4 : Clonage d'un projet avec Git Extension:

1. Ouvrez Git Extensions.
2. Cliquez sur "Clone repository" dans la barre d'outils.
3. Dans l’onglet "Repository to clone" collez le lien Git du projet suivie du mot ".git" comme suit: <https://github.com/mbaAxis/Valo.git> .
4. Dans l’onglet "Destination" choisissez le dossier local où vous souhaitez cloner le projet.
5. Cliquez sur "Clone".

### Étape 5 : Sélectionner une branche sur laquelle travaillée

#### 1. Ouvrir un projet :

* Lancez Git Extensions.
* Cliquez sur "Open" pour ouvrir un projet existant

#### 2. Accéder à la liste des branches :

* Une fois que le projet est ouvert, allez dans l'onglet "Branches" situé dans la barre d'outils.
* Vous devriez voir une liste des branches disponibles dans le dépôt.

#### 3. Sélectionner une branche :

1. Cliquez avec le bouton droit sur la branche que vous souhaitez sélectionner.
2. Dans le menu contextuel, sélectionnez "Switch To" (ou "Checkout" dans certaines versions de Git Extensions) puis “ok”.

#### 4. Confirmer le changement de branche :

1. Vous verrez une boîte de dialogue vous demandant de confirmer le changement de branche.
2. Confirmez votre choix, et Git Extensions basculera sur la branche sélectionnée.

#### 5. Vérifier la branche actuelle :

1. Vous pouvez vérifier la branche actuelle dans la barre d'état en bas de la fenêtre Git Extensions.
2. Il peut également y avoir une indication visuelle de la branche actuelle dans l'onglet "Branches".

**NB** : lien utile pour avec l’utilisation de git extension: [J'apprends Git Extension](https://git-extensions-documentation.readthedocs.io/en/release-4.2/).

***Une fois la branche sélectionnée allez dans Visual studio Code.***

## Installation des plugins (extensions) et configuration de Visual Studio Code

### Étape 1 : Installer quelques extensions

Lancez Visual Studio Code et installer les extensions suivantes:

1. *C#* : pour coder en C#.
2. *Python*: pour coder en python.
3. *GitLens*: permet de voir qui a contribué à chaque ligne de code dans votre projet. ainsi que les informations sur les auteurs peuvent être consultées.
4. *Material Icon*: permet de voir des icônes conçues pour améliorer la lisibilité et l'esthétique des fichiers et dossiers dans l'arborescence du projet.

**Pour cela:**

1. **Ouvrez Visual Studio Code :** Lancez Visual Studio Code sur votre machine.
2. **Extensions :** Cliquez sur l'icône des extensions dans la barre latérale (ou utilisez le raccourci Ctrl+Shift+X).
3. **Recherchez l'extension :** Recherchez par exemple "C#" dans la barre de recherche des extensions.
4. **Installez l'extension :** Cliquez sur l'extension "C#" proposée par Microsoft et cliquez sur "Install".

### Étape 2 : Ouvrir le dossier du projet

1. Cliquez sur l'icône "File" (Fichier) dans la barre d'activités à gauche
2. Cliquez sur le bouton "Open Folder" (Ouvrir un dossier) en haut de la barre latérale.
3. Sélectionnez le dossier que “Valo” (pour ce projet) vous souhaitez ouvrir dans Visual Studio Code et cliquez sur "Sélectionner un dossier" (ou "Select Folder" en anglais).
4. Sélectionner votre notebook python ou fichier python (double clic).
5. S’il n’existe pas le créer avec "File" > "New File".
6. Pour l’exécuter suivre les étapes suivantes.

### Étape 3 : Créer un environnement virtuel python

#### 1. Ouvrir le Terminal dans Visual Studio Code :

* Ouvrez Visual Studio Code.
* Allez dans le menu "View" > "Terminal" ou utilisez le raccourci Ctrl + \ pour ouvrir le terminal intégré.

#### 2. Naviguer vers le répertoire "valo" :

Assurez vous que votre terminal est positionné dans le répertoire de votre projet "valo". Vous pouvez utiliser la commande cd pour changer de répertoire. Par exemple :

cd chemin/vers/valo

#### 3. Créer un environnement virtuel :

Utilisez la commande suivante pour créer un environnement virtuel nommé ".env" (vous pouvez choisir un autre nom si vous le souhaitez) :

python -m venv .env

#### 4. Activer l’environnement virtuel :

.env\Scripts\activate

Vous devriez voir le nom de votre environnement virtuel dans l'invite de commandes, indiquant que l'environnement virtuel est activé.

**NB:** créer un environnement virtuel permet d’éviter le port des dépendances lors des push. toujours se placer dans votre environnement virtuel avant d’exécuter votre code python.

**Attention:** si vous avez une erreur avec un lien d'aide alors:

1. Cliquer sur ce lien et chercher la commande: Set-ExecutionPolicy -ExecutionPolicy RemoteSigned.
2. Ouvrer PowerShell en tant que administrateur.
3. Se placer dans le dossier du projet via la commande: cd "chemin du dossier" (mettre le chemin du dossier comme un string).
4. Saisir la commande suivante: Set-ExecutionPolicy -ExecutionPolicy RemoteSigned.
5. Saisir: O
6. Saisir de nouveau la commande du **“4. Activer l’environnement virtuel”** ci-dessus depuis votre terminal (vous ne devriez plus avoir l’erreur ).

#### 5. Désactiver l’environnement virtuel (optionnel) :

Lorsque vous avez terminé de travailler dans l'environnement virtuel, vous pouvez le désactiver avec la commande :

deactivate

#### 6. Mettre l’environnement virtuel dans le gitignore

Cette étape permet d’éviter que votre environnement soit importer par une personne qui travail sur la même Branche. Ce qui peut générer des erreurs.

1. ouvrir le fichier .gitignore.
2. Mettre l’environnement virtuel. Par exemple si votre environnement virtuel s’appelle “.**env**”, écrire la ligne suivante :

.env/

Ajouter un commentaire pour une meilleur structure.

#### 7. Gérer/créer le fichier requirement.txt

Ceci permet à un autre utilisateur d'éviter de faire les imports que vous avez déjà fait sur la branche en cours. Vous pouvez donc créer le votre qui vous pourriez partager avec l’équipe après votre commit&push.

1. ouvrir le terminal
2. ouvrir le environnement virtuel
3. faire import dans le terminal (par exemple installer matplotlib)

pip install matplotlib

1. Créer votre requirement avec le nom par exemple “**requirement\_username**” . Adapter le **username** au vôtre.

pip freeze > requirement\_username.txt

Si le votre fichier requièrent existe déjà la commande précédente fera la mise à jour de votre fichier.

# Création du fichier .dll

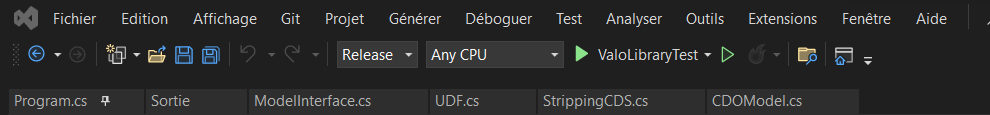
## Execution du code sur visual studio

### Étape 1 : Fonction program.cs

1. Ouvrir Visual studio en tant administrateur.

Après avoir écrit les classes de votre libraire (ex : **Valolibrary**) en langage en C# :

1. Créer un fichier d’exécution (ex : **ValolibraryTest**) dans lequel vous devez écrire une classe program.cs pour tester les fonctions de chaque classe.
2. Définir votre fichier d’exécution **ValolibraryTest** commeprojet de démarrage. Pour cela :
   1. Cliquer sur **Affichage** depuis la barre de menu ensuite sur **Explorateur de solution.** La fenêtre explorateur de solution s’affichera droite gauche de votre écran.
   2. Faites un clic droit sur **ValolibraryTest > Définir en tant que projet de démarrage.** Le bouton Excetution devrait avoir le nom de votre fichier d’exécution, ici sur **ValolibraryTest.**
3. Exécuter votre fichier program.cs
   1. Faites un clic sur le bouton d’execution (il est en forme de triangle en fond vert)



* 1. Rassurer vous qu’il n’y ai pas d’erreur et que les résultats obtenus sont les résultats attendus.

### Étape 2 : Générer la solution

Une fois cela fait :

1. Aller sur **Générer > Générer la solution** dans le menu (en haut sur l’interface visual studio)
2. Si vous faites des modifications par la suite, exécuter d’abord votre code comme à l’étape 1et ensuite aller sur **Générer > Nettoyer la solution** puis **Générer > Régénérer la solution**

**NB**: Si vous avez des erreurs dans lors de l’exécution de votre code, la création de la solution va échouer.

## Configurations

### Étape 1 : sur visual studio

1. Sur l’explorateur de solution, cliquer sur votre librairie **Valolibrary** ensuite aller sur **Projet > Propriétés de Valolibrary.** La fenêtre suivante devrait s’afficher.

Une image contenant capture d’écran, texte, logiciel

Description générée automatiquement

1. Dans le menu **Application** cliquer sur **informations de l’assembly…** .

Une deuxième fenêtre va s’ouvrir. Cocher la case **rendre l’asssemnly visible par la COM.**

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

1. Dans le menu **Build** cocher la case **S’inscrire à COM Interop.**

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Maintenant votre solution sera visible depuis l’interface Excel.

### Étape 2 : sur vba Excel

1. Ouvrir Excel
2. Ensuite ouvrir Visual Basic : via le menu **Développeur > Visual Basic (ou Alt + F11)**
3. Sur le menu **Outils** cliquer sur **Reference** > ensuite sélectionner votre librairie qui se trouve dans le dossier de votre projet. Cliquer sur parcourir et aller dans le chemin où se trouve votre fichier dll et dlt. Sélectionner le fichier **Valolibrary.dlt.**
4. Rassurer vous d’avoir bien sélectionner votre librairie. La fenêtre suivante devra s’afficher :Une image contenant texte, logiciel, Icône d’ordinateur, Page web

   Description générée automatiquement

Maintenant vous pour commencer la partie **intégration** de C# à VBA.

# Intégration de C# à VBA

## Sur Visual studio

Lorsque vous avez écrit les classes de votre librairie :

1. Créer une Class **UDF**
2. Vous devez créer une classe **interface** (ex : **IUDF**) dans laquelle vous aller déclarer toutes les fonctions essentielles que vous voulez rendre visible par la COM sur l’interface Excel.
3. N’oubliez pas les trois lignes de code suivantes :
   1. *[ComVisible(true)]*
   2. *[Guid("839187c8-9765-4e76-a508-61ec3dd1a504")]*
   3. *[InterfaceType(ComInterfaceType.InterfaceIsDual)]*

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Sur l’image ci-dessus la fonction ***GetBSOptionPrice*** a été déclaré avec ces paramètres.

1. Créer ensuite une classe (ex : **UDF**) pour définir ces fonctions (vous pouvez utiliser le principe de l’héritage pour cela). N’oubliez pas les quatre lignes de code suivantes, elles sont importantes pour rendre la Com visible sur Excel.
   1. *[ComVisible(true)]*
   2. *[ProgId(" ValoLibrary.UDF"]*
   3. *[Guid("839187c8-9765-4e76-a508-61ec3dd1a504")]*
   4. *[InterfaceType(ComInterfaceType.InterfaceIsDual)]*

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Sur l’image ci-dessus la fonction ***GetBSOptionPrice*** a définie dans la classe **UDF**. Puisque sa définition existe déjà dans la classe BlackScholes, elle a été appelée de celle-ci en return.

**NB** : les [**Guid (‘…’)]** contiennent des mots clés, ils doivent être différents. Vous pouvez générer des Guid depuis le menu **Recherche** > ensuite écrire **Create Guid.**

## Sur Excel VBA

1. Ouvrir Visual Basic
2. Créer un Module et nommer le (ex : BSFunctions)
3. Déclarer et définir la fonction que vous voulez utiliser sur le classeur excel. Vous pour utiliser le même nom que celui de la classe **UDF** depuis l’IDE Visual Studio.

Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquement

Sur l’image ci-dessous, le module BSFunctions contient la definition de la fonction ***GetBSOptionPrice*** en appelant l’Object crée depuis notre libraire **ValoLibrary.UDF**. Cette fonction est maintenant accessible et utilisable sur le classeur Excel. Voici un exemple :

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Sur l’image ci-dessus la cellule, G7 contient la formule de pricing de prix d’une option utilisant BlackScholes.

Maintenant vous pour commencer la partie **Déploiement** pour le rendre accessible aux utilisateurs.

# Déploiement (in progress)

# Pricing des produits dérivés equity

## Vanilles

### Call/Put

#### Définitions des dérivés

**Call** (option d’achat) : est un produit dérivé qui donne le droit et non de devoir au détendeur d’acheter un actif financier à un prix d’exercice **K** donné et à une maturité fixée à l’avance **T**.

**Put** (option de vente) : est un produit dérivé qui donne le droit et non de devoir au détendeur de vendre un actif financier à un prix d’exercice **K** donné et à une maturité fixée à l’avance **T**.

#### Définitions des Sensibilités

**Le Delta** : mesure la sensibilité du prix de l'option par rapport au prix de l'actif sous-jacent. Un delta de 0,50 signifie que pour chaque augmentation de 1 unité du prix de l'actif sous-jacent, le prix de l'option augmente de 0,50 (pour les options d'achat) ou diminue de 0,50 (pour les options de vente).

**Le Gamma** : mesure la sensibilité du delta de l'option par rapport au prix de l'actif sous-jacent. Il représente la variation du delta par unité de variation du prix de l'actif sous-jacent. Un gamma élevé indique une forte variation du delta en réponse aux fluctuations du prix de l'actif sous-jacent.

**Le Theta** : mesure la sensibilité du prix de l'option au passage du temps. Il représente la variation du prix de l'option par unité de temps qui passe. Un theta négatif indique que la valeur de l'option diminue avec le temps, toutes choses égales par ailleurs.

**Le Vega** : Le vega mesure la sensibilité du prix de l'option par rapport à la volatilité implicite du marché. Il représente la variation du prix de l'option pour une variation d'un point de pourcentage de la volatilité implicite.

Le prix de **Call/Put** est obtenu en utilisant :

1. la formule de BlackScholes avec dividendes utilisant les données de l’utilisateurs (cf. John Hull pour plus de détails sur la formule de BlackScholes).
2. la formule de BlackScholes avec dividendes utilisant les données de marches.
3. La methode de monté Carlo (***cf. John Hull***).

#### Structure du code

La librairie contient des classes suivantes

##### StatisticalFormulas

1. **LinearInterpol** : Cette méthode réalise une interpolation linéaire entre deux points (x0, y0) et (x1, y1) pour trouver la valeur de y correspondante à un certain x donné.
2. **NormalDensity** : Calcule la densité de probabilité d'une variable aléatoire suivant une distribution normale avec une moyenne et un écart type donné.
3. **Cfd** : Cette méthode calcule la fonction de distribution cumulative d'une variable aléatoire suivant une distribution normale. Elle est implémentée à l'aide d'une série de constantes et de calculs mathématiques.
4. **GauCumDisFunction** : Une autre méthode pour calculer la fonction de distribution cumulative d'une variable aléatoire suivant une distribution normale standard. Elle utilise une forme simplifiée de la fonction tangente hyperbolique.
5. **RandomNormal** : Génère un échantillon aléatoire à partir d'une distribution normale avec une moyenne mean et un écart type stddev donnés.
6. **NextGaussian** : Une extension de la classe Random qui génère un échantillon aléatoire suivant une distribution normale standard.

##### GetData

Cette classe contient des méthodes permettant de récupérer des données à partir d'un fichier Excel.

1. La variable **\_filePath** contient le chemin d'accès absolu du fichier Excel contenant les données à charger.
2. **ReleaseMemory** : est utilisée pour libérer la mémoire utilisée par les objets Excel après les avoir utilisés. Cela évite les fuites de mémoire et les problèmes de performances.
3. **Data (string underlying)** : elle prend en paramètre le nom de l'actif sous-jacent et récupère les données correspondantes à partir du fichier Excel.
4. **GetSpot (string underlying)** : elle retourne le prix spot de l'actif sous-jacent spécifié. Actuellement, elle retourne une valeur statique en fonction de l'actif sous-jacent, mais elle pourrait être modifiée pour récupérer le prix spot à partir du fichier Excel.
5. **GetTime(string underlying)** : elle retourne le temps associé à l'actif sous-jacent spécifié. De manière similaire à **GetSpot**, elle retourne actuellement une valeur statique mais pourrait être modifiée pour récupérer le temps à partir du fichier Excel.

##### Calibration

Cette classe contient des méthodes pour effectuer la calibration des données provenant du fichier Excel et pour interpoler les prix en fonction des paramètres tels que le taux sans risque, les dividendes, les maturités et les strikes. Elle contient les fonctions suivantes :

1. **GetRepo(string underlying, double T) :** elle récupère le taux repo pour l'actif sous-jacent spécifié à une maturité donnée T.
2. Elle utilise la méthode **PosMaturitiesToInterpol** pour trouver les positions des maturités à interpoler, puis la méthode **LinearInterpol** de la classe **StatisticFormulas** pour interpoler le taux repo.
3. **GetDividend(string underlying, double T) :** elle récupère le dividende pour l'actif sous-jacent spécifié à une maturité donnée T. Elle utilise la méthode **PosMaturitiesToInterpol** pour trouver les positions des maturités à interpoler, puis la méthode **LinearInterpol** de la classe **StatisticFormulas** pour interpoler le dividende.
4. **PosStrikesToInterpol(string underlying, double k)** : elle trouve les positions des strikes à interpoler pour l'actif sous-jacent spécifié et la valeur du strike k.
5. **PosMaturitiesToInterpol(string underlying, double T)**: elle trouve les positions des maturités à interpoler pour l'actif sous-jacent spécifié et la maturité T.
6. **InterpolatePrice(double K, double T, string underlying) :** elle interpole le prix pour l'actif sous-jacent spécifié en fonction des paramètres K (strike) et T (maturité). Elle utilise les positions des strikes et des maturités à interpoler pour extraire les données nécessaires à partir du fichier Excel, puis utilise la méthode **LinearInterpol** de la classe **StatisticFormulas** pour interpoler le prix.
7. Ces fonctions sont utilisées pour calibrer les données du marché et interpoler les prix en fonction des paramètres spécifiés, facilitant ainsi l'analyse et l'évaluation financière.

##### BlackScholes

Cette classe BlackScholes fournit des méthodes pour calculer diverses valeurs et sensibilités des options en utilisant le modèle de Black-Scholes. Voici une explication des principales méthodes fournies :

1. **BSOptionPrice** : Calcule le prix d'une option selon le modèle de Black-Scholes.
2. Prend en paramètres la quantité d'options, le type d'option (call ou put), la position (long ou short), le prix de l'actif sous-jacent, la volatilité, le taux sans risque, le prix d'exercice, le temps jusqu'à l'expiration et éventuellement le taux de dividende.
3. **DeltaBS, GammaBS, ThetaBS et VegaBS :** calculent respectivement le delta, le gamma, le theta et le vega d'une option selon le modèle de Black-Scholes.
4. **SensiOptionBS** : calcule les sensibilités (delta, gamma, theta, vega) d'une option en une seule fois et les retourne sous forme de tableau.
5. **ImpliedVol** : calcule la volatilité implicite d'une option en utilisant la méthode de Newton-Raphson pour trouver la racine de la fonction de prix de l'option moins le prix observé.
6. **BSOptionPortfolioPrice** : calcule le prix d'un portefeuille d'options en sommant les prix individuels de chaque option dans le portefeuille.

Ces méthodes fournissent un ensemble complet de calculs pour évaluer les options et les portefeuilles d'options en utilisant le modèle de Black-Scholes, ce qui est essentiel pour l'analyse et le trading d'options sur les marchés financiers.

##### BlackScholesMD

Cette classe est une extension de la classe BlackScholes précédemment définie, mais elle est adaptée pour calculer les prix et les sensibilités des options en utilisant des paramètres de modèle dynamiques (utilisant les données de marchés).

1. **OptionPrice** : calcule le prix d'une option en utilisant le modèle de Black-Scholes avec des paramètres dynamiques.
2. **Delta, Gamma, Theta et Vega :** calculent respectivement le delta, le gamma, le theta et le vega d'une option en utilisant des paramètres dynamiques similaires à ceux de la méthode **OptionPrice**.
3. **SensiOption** : calcule les sensibilités (delta, gamma, theta, vega) d'une option en une seule fois et les retourne sous forme de tableau.
4. **OptionPortfolioPrice** : calcule le prix d'un portefeuille d'options en sommant les prix individuels de chaque option dans le portefeuille.

Ces méthodes fournissent une manière flexible et dynamique de calculer les prix et les sensibilités des options en utilisant le modèle de Black-Scholes avec des paramètres qui peuvent être extraits dynamiquement à partir des données disponibles. Cela permet une analyse plus approfondie des options dans divers scénarios de marché.

##### MonteCarlo

Cette classe implémente des méthodes pour estimer les prix des options européennes en utilisant la méthode de Monte Carlo. Elle contient les classes suivantes :

1. **MCEurOptionPrice** : cette méthode est utilisée pour estimer le prix d'une option européenne en utilisant la méthode de Monte Carlo. La méthode effectue un grand nombre de simulations Monte Carlo pour estimer le prix de l'option en utilisant la formule de Black-Scholes pour le calcul du payoff à chaque itération. Elle retourne la moyenne pondérée des payoffs estimés pour obtenir le prix de l'option.
2. **MCEurOptionPortfolioPrice** : elle est utilisée pour estimer le prix d'un portefeuille d'options européennes en utilisant la méthode de Monte Carlo. Elle prend en paramètre un tableau d'objets BSParameters qui contiennent les détails de chaque option dans le portefeuille. La méthode itère sur chaque option dans le portefeuille et utilise la méthode **MCEurOptionPrice** pour estimer le prix de chaque option. Enfin, elle retourne la somme des prix estimés de chaque option dans le portefeuille.

Ces méthodes fournissent une approche numérique pour estimer les prix des options européennes en utilisant des simulations Monte Carlo, ce qui peut être particulièrement utile dans des situations où les formules analytiques ne sont pas facilement applicables ou lorsque des évaluations précises sont nécessaires malgré des conditions de marché complexes.

# Pricing des produits dérivés de crédit

## CDS/CDO

### Définitions

1. **NPV, PV** (Net Present Value, Present Value) : Valeur au jour d'aujourd'hui ou à la date d'évaluation des cash flows futurs.
2. **ZC** (Zero Coupon) : Également appelé facteur d'actualisation (discount factor) . Pour une échéance donnée, il s'agit de la valeur actuelle d'une unité de monnaie à payer à l'échéance.
3. **ZC sans risque** (coupon zéro sans risque) : Valeur d'un ZC si le paiement à l'échéance ne présente pas de risque.
4. **ZC risqué** (Zero Coupon risqué) : Valeur d'un ZC si le paiement à l'échéance présente un risque en cas de défaillance d'une entreprise ou de perte excessive sur un portefeuille de crédit.
5. **Float Leg** : jambe d'un CDS ou d'un CDO qui paie un montant variable en fonction du taux de recouvrement (ou livre une obligation contre paiement de la parité) lors de la survenance d'un événement de crédit.
6. **Fixed fixe** : la jambe d'un CDS ou d'un CDO qui paie une marge fixe jusqu'à ce qu'une défaillance survienne.
7. **CDS/CDO européen** (European Credit Default Swap) : Les dérivés de crédit où le paiement de la jambe flottante est effectué à l'échéance par opposition au paiement immédiat en cas d'événement de Credit.
8. **CDS/CDO américain** (American Credit Default Swap) : Dérivés de crédit pour lesquels le paiement de la jambe flottante est effectué immédiatement en cas d'événement de crédit.

### Structure du code

#### Utilis

1. **SelectionCalculate():** Cette fonction effectue le calcul des cellules sélectionnées dans la feuille de calcul active.
   1. code en C# et VBA
   2. testé pour l’addition de deux cellules
2. **CurrentRegionCalculate()**: la fonction vise à sélectionner la région entière autour de la cellule actuellement sélectionnée, à effectuer le calcul des formules dans cette région, puis à rétablir la sélection d'origine. Cela peut être utile pour forcer le recalcul des valeurs des cellules dans une région spécifique de la feuille de calcul.
   1. code en C# et VBA
   2. testé pour l’addition des cellules

#### UtilitySpline

Cette classe/module contient des fonctions pour effectuer une interpolation spline.  
**SGSpline** est la seule fonction utilisée par d'autres Classes/modules. Elle est appelée par la fonction **GetMarketLossDistribution** dans la classe/module **CDO2model**.  
Comme les méthodes d'interpolation spline sont largement discutées dans la littérature, elles ne sont pas documentées ni commentées ici.

1. **SGSpline(x, n, abscissa\_0, ordinate\_0, Size):** prend des points d'abscisse (x), les points de données initiaux (abscissa\_0, ordinate\_0), et un paramètre de taille (Size). Elle utilise ces informations pour construire une spline cubique en calculant les dérivées secondes (ordinate2) à l'aide de la fonction firstspline. Ensuite, elle utilise la fonction splinter pour évaluer la spline cubique aux points d'abscisse spécifiés dans le tableau x, et elle renvoie les valeurs interpolées dans le tableau Output.
   1. code en C# et VBA
   2. testé avec 5 points
2. **firstspline(a, b, c, d, e, f):** elleimplémente la méthode d'interpolation cubique naturelle. Le but de cette fonction est de calculer les dérivées secondes d'un ensemble de points de données par rapport aux abscisses, créant ainsi une spline interpolatrice cubique continue. Le tableau en **f** résultant contient les dérivées secondes de la spline cubique en chaque point de données. Ces dérivées secondes peuvent être utilisées pour construire la fonction complète d'interpolation cubique.
   1. code en C# et VBA
   2. testé avec 5 points
3. **splinter(a, b, c, d, e, f):** cette fonction effectue une interpolation splinomiale cubique. Ce type d'interpolation est utilisé pour estimer la valeur d'une fonction entre deux/plusieurs points de données adjacents (méthodes par dichotomie). la variable **f** est utilisée pour stocker la valeur évaluée de la spline cubique au point spécifié **e**. Cette valeur est le résultat de l'interpolation cubique réalisée entre les points d'interpolation voisins.
   1. code en C# et VBA
   2. testé avec 5 points

#### UtilitySorting

Cette Classe/module contient la fonction de tri rapide, appelée *QSort*, utilisée par la fonction **GetCorrelInterInnerPortfolio** dans le module **CDO2model**.

Étant donné que les algorithmes de tri dits "rapides" sont largement discutés dans la littérature, ils ne sont ni documentés ni commentés ici.

1. **ChangeDatas(ByRef table, Low, Up, Size):** la fonction permet d'échanger les valeurs des éléments situés à la même position dans différentes colonnes de la table. Après avoir effectué ces échanges, elle renvoie simplement la chaîne de caractères "OK".
   1. code en C# et VBA
   2. testé
2. **DuplicateDatas(DataIn, i, DataOut, n):** ellecopie les données d'une ligne spécifique d'un tableau 2D dans un tableau 1D de sortie.
   1. code en C# et VBA
   2. testé
3. **DataSort(Data, inf, sup, Size):** elleimplémente l'algorithme de tri rapide (*Qsort*) pour trier un tableau 2D en fonction des valeurs de la première colonne. Elle utilise les fonctions DuplicateDatas et ChangeDatas.
   1. code en C# et VBA
   2. testé
4. **TemporarySortData(ByRef Data):** cettefonctioncopie des données de la plage Data dans un tableau bidimensionnel table, puis trier ces données en fonction de la deuxième colonne à l'aide d'une fonction externe appelée DataSort. Le tableau trié est ensuite renvoyé par la fonction.
   1. code en C# et VBA
   2. testé

#### UtilityMatrix

Cette classe/module contient deux fonctions matricielles :

a) Multiplication d'une matrice triangulaire inférieure par un vecteur

b) Décomposition de Cholesky d'une matrice. La décomposition de Cholesky est effectuée sur la matrice de corrélation entre les pertes du portefeuille interne. La multiplication est utilisée pour corréler des variables gaussiennes indépendantes. Les deux fonctions sont utilisées dans la fonction EuropeanCDO2 du modèle CDO2model. Comme la décomposition de Cholesky est largement discutée dans la littérature, elle n'est pas documentée ni commentée ici.

1. **MatrixTimesVector(l, x, NStart, NEnd):** effectue une multiplication entre une matrice triangulaire inférieure l et un vecteur x. Elle renvoie un vecteur résultant.
   1. code en C# et VBA
   2. testé
2. **Choleski(S, NStart, NEnd):** cette fonction utilise l'algorithme de décomposition de Choleski pour transformer la matrice symétrique définie positive S en une matrice triangulaire inférieure L. La matrice L est ensuite renvoyée en tant que résultat de la fonction. La fonction effectue également des vérifications pour s'assurer que la matrice est positive définie et renvoie un message d'erreur le cas échéant.
   1. code en C# et VBA
   2. testé

#### UtilityLittleFunctions

1. **CallSpread(S, k1, k2):** ellecalcule le payoff d'un call spread entre en fonction du prix actuel de l'actif et des prix d'exercice.
   1. code en C# et VBA
   2. non testé
2. **MaxPlus(x):** elleretourne le maximum d'une valeur et de zéro
   1. code en C# et VBA
   2. testé
3. **MinOf(x, y):** elleretourne le minimum de deux valeurs.
   1. code en C# et VBA
   2. testé
4. **MaxOf(x, y):** elleretourne le maximum de deux valeurs.
   1. code en C# et VBA
   2. testé
5. **MaxTab(table, Size):** elle trouve la valeur maximale dans un tableau.
   1. code en C# et VBA
   2. testé

#### UtilityFaure

Ce module contient un générateur quasi-aléatoire de Faure utilisé par la fonction EuropeanCDO2 dans le module CDO2model. Optimisé pour la vitesse.

1. **UniformRandomize(Iter, Size):** cette fonction génère une séquence de nombres pseudo-aléatoires uniformément distribués dans l'intervalle [0, 1] en utilisant une méthode basée sur une table de nombres premiers et la méthode de congruence linéaire. Elle retourne le tableau c3, qui contient la séquence de nombres pseudo-aléatoires générés.
   1. code en C# et VBA
   2. testé

#### UtilityDates

1. **MonthPeriod(period, Optional ReferenceDate):** elle convertit une période spécifiée au format "2m" ou "6M" ou "5y" ou "7Y" en un nombre entier de mois.
   1. code en C# et VBA
   2. testé
2. **ConvertDate(ParamDate, Maturitydate):** elle convertit une date spécifiée au format "3Y" ou "6m" en une date.
   1. code en C# et VBA
   2. testé
3. **DurationYear(EndDate, StartDate):** elle calcule la durée en années entre deux dates.
   1. code en C# et VBA
   2. testé
4. **GetSwapSchedule(ParamDate, Maturity, CpnLastSettle, CpnPeriod, CpnConvention):** ellerenvoie une liste de dates de coupons pour un swap en fonction de la date d'échéance et des conventions de coupons.
   1. code en C# et VBA
   2. testé
5. **SwapSchedule(ParamDate, Maturity, CpnLastSettle, CpnPeriod, CpnConvention):** elle génère un calendrier de dates de coupons pour un swap en fonction des paramètres donnés. Ici le resultats de ne depends pas du paramètre **CpnConvention**.
   1. code en C# et VBA
   2. testé
6. **SwapSchedule2(ParamDate, Maturity, CpnLastSettle, CpnPeriod, CpnConvention)**elle génère un calendrier de dates de coupons pour un swap en fonction des paramètres donnés mais avec les conventions suivantes:
   1. **FirstLong:** Pour la première période, ajuste la date vers la fin du mois si elle ne l'est pas déjà. Les périodes suivantes sont ajoutées normalement.
   2. **LastLong:** Pour la dernière période, ajuste la date vers la fin du mois si elle ne l'est pas déjà. Les périodes précédentes sont ajoutées normalement.
   3. **FirstShort:** Pour la première période, ajuste la date vers le début du mois si elle ne l'est pas déjà. Les périodes suivantes sont ajoutées normalement.
   4. **LastShort:** Pour la dernière période, ajuste la date vers le début du mois si elle ne l'est pas déjà. Les périodes précédentes sont ajoutées normalement.
      1. code en C#
      2. testé (pas validé)

#### UtilityBiNormal

Cette classe/module contient une fonction de distribution normale bivariée utilisée par la fonction **GetCorrelInterInnerPortfolio** dans le module **CDO2model**.

Comme les méthodes de distribution normale sont largement discutées dans la littérature, elles ne sont pas documentées ni commentées ici.

1. **g(x, y, r, b):** renvoie l’exponentielle d’une valeur qui dépend des paramètres.
   1. code en C# et VBA
   2. testé
2. **BivariateNormalDistributiontwo(a, b, rho):** Cette fonction est utilisée par la fonction BivariateNormalDistribution. Elle prend trois paramètres (a, b, et rho) et utilise une combinaison de calculs basés sur des tableaux (tableau1 et tableau2) et la fonction FF pour estimer la distribution normale bivariée.
   1. code en C# et VBA
   2. testé
3. **FF(x, y , at, bt, rho) :** Cette fonction est utilisée pour calculer une partie de la fonction de densité jointe d'une distribution normale bivariée. Elle et est utilisée comme un élément de calcul dans BivariateNormalDistributiontwo.
   1. code en C# et VBA
   2. testé
4. **NormalDensityFunction(x):** Cette fonction calcule la densité de probabilité pour une distribution normale univariée.
5. **MyNormal(x, Mean, Stdev):** Cette fonction calcule également la fonction de distribution cumulative pour une distribution normale univariée. Elle prend trois paramètres (x, Mean, et Stdev) qui représentent la variable aléatoire, la moyenne, et l'écart type respectivement.
   1. code en C# et VBA
   2. testé
6. **NormalCumulativeDistribution(x):** Cette fonction calcule la fonction de distribution cumulative pour une distribution normale univariée. Elle utilise la fonction NormalDensityFunction pour calculer la densité de probabilité normale.
   1. code en C# et VBA
   2. testé
7. **BivariateNormalDistribution(a1, b1, rho):** Cette fonction calcule la fonction de distribution cumulative pour une distribution normale bivariée. Elle prend trois paramètres (a1, b1, et rho) qui représentent les variables aléatoires bivariées et la corrélation entre elles.
   1. code en C# et VBA
   2. testé

#### Stripping IRS

Cette classe/module contient :  
a) une calibration simple de l'extraction des taux zéro-coupon, nommée StripZC, qui retourne les taux sans risque. Elle ne peut prendre que des points à des multiples de 12 mois. Elle n'est pas très adaptée pour les taux à court terme, et l'hypothèse de taux forward constant par morceaux n'est pas la meilleure, mais les dérivés de crédit ne sont pas très sensibles aux taux d'intérêt.  
b) une fonction pour interpoler les taux zéro-coupon à n'importe quelle date, utilisée par de nombreuses autres fonctions dans d'autres classes/modules.

1. **Type IRCurve:** est un type de données personnalisé qui représente une courbe des taux d'intérêt.
   1. code en C# et vba
   2. testé
2. **Type IRCurveList:** est un autre type de données personnalisé qui représente une liste de courbes des taux d'intérêt.
   1. code en C# et vba
   2. testé
3. **InterestRateCurves:** est une variable publique de type IRCurveList. Il destiné à stocker une liste de courbes des taux d'intérêt.
   1. code en C# et vba
   2. testé
4. **GetCurveName(CurveID):** Il s'agit d'une fonction (GetCurveName) qui prend un paramètre CurveID. Elle vérifie si le CurveID fourni se situe dans la plage valide des courbes disponibles dans InterestRateCurves. Si c'est le cas, la fonction renvoie le nom de la courbe avec le CurveID spécifié et définit LastError à False. Si le CurveID n'est pas dans la plage valide, la fonction renvoie la chaîne "Curve ID missing" et ne définit pas LastError à False.
   1. code en C# et vba
   2. testé
5. **GetCurveId :** cette fonction récupère l'ID de la courbe associé à un CurveName donné dans la collection InterestRateCurves. Elle vérifie les entrées valides et gère les erreurs de manière similaire à GetCurveName.
   1. code en C# et vba
   2. testé
6. **vbaStoreZC:** cette fonction stocke les données des obligations zéro-coupon pour une courbe des taux d'intérêt spécifique. Elle vérifie si la courbe existe déjà et en crée une nouvelle au besoin. Elle stocke ensuite divers paramètres et tableaux associés à la courbe, y compris les taux zéro-coupon mensuels (MonthlyZC) calculés à partir des données fournies.
   1. code en C# et vba
   2. testé
7. **vbaComputeMonthlyRiskyZC(CurveName, ParamDate, CDSRollDate):** calcule la courbe des taux zéro-coupon (ZC) mensuelle "mensuel" de la date de roulement d'un contrat Credit Default Swap (CDS).
   1. code en C# et vba
   2. testé
8. **GetFXSpot(CurveName):** récupère le taux de change associé à une courbe donnée (CurveName) dans la collection InterestRateCurves. Gère les erreurs si la courbe n'existe pas.
   1. code en C# et vba
   2. testé
9. **vbaGetRiskFreeZc(ParamDate, Maturitydate, ZC, ZCDate):** interpole un taux zéro-coupon à la date de maturité (Maturitydate) en supposant des taux forward constants. Utilise les tableaux ZC et ZCDate provenant d'une courbe spécifique. Gère les cas où la date de maturité est antérieure à la date paramétrique (ParamDate).
   1. code en C# et vba
   2. testé
10. **GetZC(Maturitydate, CurveName):** récupère le taux zéro-coupon à une date de maturité spécifique (Maturitydate) à partir d'une courbe donnée (CurveName). Utilise la fonction vbaGetRiskFreeZc pour effectuer l'interpolation.
    1. code en C# et vba
    2. testé
11. **vbaGetSwapPVandDerivatives:** Calcule la valeur présente (PV) et ses dérivés en fonction des taux zéro-coupon utilisés entre deux points dans le temps. Implique des calculs liés aux jambes flottante et fixe d'un swap, ainsi qu'à l'interpolation des taux zéro-coupon. Gère les erreurs et renvoie un tableau de résultats.
    1. code en C# et vba
    2. testé
12. **StripZC:** estime les taux zéro-coupon à partir d'une courbe de taux d'intérêt, en prenant en compte les swaps et les ajustements itératifs pour converger vers les taux zéro-coupon appropriés. En resumé cette fonction calibre les coupons zéro risqués pour qu'ils s'adaptent aux courbes de taux d’intérêts dans l'hypothèse d'une intensité par défaut progressive.
    1. code en C# et vba
    2. testé

#### Stripping CDS

Cette classe/module contient des fonctions utilisées pour le stripping des courbes de spread de CDS (Credit Default Swap). Les plus importantes sont :

1. StripDefaultProbability : Cette fonction est appelée depuis Excel pour calibrer un vecteur de probabilité de défaut à partir d'une courbe de spread. Elle est documentée dans le Guide Technique OpenCredit. Cette fonction est essentielle pour convertir les spreads de CDS en probabilités de défaut implicites.
   1. code en C# et vba
   2. testé
2. GetDefaultProbability : Cette fonction est appelée par d'autres classes/modules. Elle interpole une probabilité de défaut pour une date et un émetteur donné. Bien que non documentée, elle est relativement simple et joue un rôle crucial dans le processus d'estimation des probabilités de défaut implicites à partir des données de spread de CDS.
   1. code en C# et vba
   2. testé
3. **StripDefaultProbability** : Cette fonction est utilisée pour calibrer les probabilités de défaut à partir d'une courbe de spread de CDS donnée. Elle calcule de manière itérative les probabilités de défaut à diverses échéances en fonction de la courbe de spread de CDS fournie et d'autres paramètres. Elle gère également les scénarios où la courbe de spread est ajustée (c'est-à-dire modifiée) selon des constantes prédéfinies.
   1. code en C# et vba
   2. testé
4. **StoreDP** : Cette fonction stocke les données de spread de CDS et de probabilité de défaut en mémoire. Elle organise les données pour un accès et une manipulation plus faciles par d'autres fonctions. Elle accepte divers paramètres tels que le nom du CDS, le taux de récupération, la devise, les dates de courbe, la courbe de spread et la probabilité de défaut.
   1. code en C# et vba
   2. testé
5. **GetDefaultProb** : Cette fonction récupère la probabilité de défaut pour un émetteur donné et une date d'échéance donnée. Elle peut également prendre en compte différents scénarios et appliquer un multiplicateur à la probabilité résultante.
   1. code en C# et vba
   2. testé
6. **GetCDSCurveId** : Cette fonction récupère l'identifiant d'une courbe de CDS en fonction de son nom. Elle est utilisée en interne pour identifier les courbes de CDS.
   1. code en C# et vba
   2. testé
7. **GetCDSName** et **GetCDSCurrency** : Ces fonctions récupèrent respectivement le nom et la devise d'une courbe de CDS en fonction de son identifiant.
   1. code en C# et vba
   2. **testé**
8. GetMonthlyDP : Cette fonction récupère les probabilités de défaut mensuelles pour un émetteur donné. Elle prend en compte différents scénarios et renvoie les probabilités de défaut mensuelles pour chaque mois.
   1. code en C# et vba
   2. **testé**
9. GetDefaultProbabilityQuanto : Cette fonction calcule la probabilité de défaut à une échéance donnée pour un CDS (Credit Default Swap) en tenant compte des effets de change entre la devise du CDS et la devise de tarification. Elle peut également prendre en compte les scénarios de décalage de la courbe des spreads de CDS, la volatilité des changes et la corrélation entre les spreads de crédit et les taux de change.
   1. code en C# et vba
   2. **testé**
10. GetRecoveryRate : Cette fonction renvoie le taux de récupération d'un émetteur donné, qui est utilisé dans le calcul de la valeur actualisée nette des paiements de défaut.
    1. code en C# et vba
    2. **testé**
11. GetCDS\_PV et GetCDS\_PV\_3m : Ces fonctions calculent la valeur actuelle nette (VAN) d'un CDS en tenant compte des paiements de coupons et des paiements en cas de défaut. Elles prennent en compte les probabilités de défaut mensuelles, les courbes de taux sans risque et les paramètres spécifiques du CDS.
    1. code en C# et vba
    2. **testé**
12. CDSRefDate : Cette fonction retourne la date de référence pour un CDS à partir de la date actuelle. Elle est utilisée pour déterminer la date de roulage (roll date) du CDS.
    1. code en C# et vba
    2. **testé**

#### CDO Model

Cette classe/ module contient les fonctions suivantes :

1. **Recursion :** est utilisée pour modéliser la distribution des pertes en fonction des probabilités de défaut des émetteurs, en prenant en compte la possibilité d'inclure des grecques (mesures de sensibilité aux changements).
   1. code en C# et vba
   2. testé
2. **GetDefaultDistribution*:*** elle calcule la distribution du nombre de défauts pour les probabilités de défaut initiales ' et les probabilités de défaut choquées, pour une valeur donnée du facteur systémique.. Elle renvoie la matrice de distribution des défauts, qui peut être utilisée pour analyser la probabilité de différents scénarios de défaut en fonction des paramètres spécifiés.
   1. code en C# et vba
   2. testé
   3. GetDefaultDistributionLossUnit calcule la distribution des pertes pour un portefeuille d'émetteurs, en tenant compte des probabilités de défaut, des unités de perte et des coefficients bêta par émetteur. Voici un résumé de ses entrées et sorties.
   4. code en C# et vba
   5. non testé
3. **RecursionLossUnit:** elle calcule la distribution du nombre de défauts en fonction des probabilités de défaut initiales, des probabilités de défaut choquées (si activé), et des unités de perte pour chaque émetteur, pour une valeur donnée du facteur systémique. La distribution est calculée jusqu'à un certain nombre maximal de défauts (défini par l'utilisateur ou calculé en fonction des unités de perte cumulées).
   1. code en C# et vba
   2. testé
4. **EuropeanCDO** : il calcule le prix d'un CDO européen avec une distribution par défaut obtenue en appelant **GetDefaultDistribution**.
   1. code en C# et vba
   2. testé
5. EuropeanCDOLossUnit, calcule la valeur actuelle (VA) et les sensibilités de la VA par rapport aux probabilités de défaut et au bêta pour les protections en actions avec plusieurs seuils de détachement simultanément. Elle est conçue pour analyser les produits dérivés de crédit, en particulier les obligations de dette garantie européennes (CDO).
   1. code en C# et vba
   2. non testé

#### ModelInterface

Cette classe/ module contient les fonctions suivantes :

1. **CDS** : cette fonction calcule la valeur actuelle nette (NPV), la NPV du coupon variable, la NPV du coupon fixe et la valeur en points de base BP risqués d'un CDS américain. Elle appelle une fonction générique **AmericanSwap** qui effectue le calcul de la NPV des CDS américains en tant qu'intégration des CDS européens. Les paramètres d'entrée incluent l'identifiant de l'émetteur, la maturité, le spread, le taux de recouvrement, etc. Les paramètres optionnels sont utilisés pour spécifier des options supplémentaires comme la monnaie de tarification, la corrélation FX, etc.
   1. code en C# et vba
   2. testé
2. **CDO :** cette fonction calcule la valeur actuelle nette (NPV) du coupon variable, la NPV du coupon fixe et la valeur en points de base risqués (BP) d'un CDO américain. Tout comme la fonction CDS, elle appelle la fonction **AmericanSwap** pour effectuer le calcul. Les paramètres d'entrée incluent la maturité, les seuils de détection, la corrélation, le nombre d'émetteurs, etc. Des paramètres optionnels permettent de spécifier des détails supplémentaires tels que le taux de récupération, le hedging CDS, etc.
   1. code en C# et vba
   2. non testé
3. **AmericanSwap :** la fonction est conçue pour évaluer des contrats de swap de crédit (CDS) ou des dérivés de tranche de crédit (CDO).
   1. code en C# et vba
   2. testé
4. **AmountUnit :** cette fonction prend en entrée le nombre de noms (entités) et les montants associés. Elle calcule l'unité de montant en utilisant une méthode de recherche du PGCD (Plus Grand Commun Diviseur) à l'aide de la fonction ProxyPGCD. La précision optionnelle permet de définir une tolérance pour le calcul du PGCD.
   1. code en C# et vba
   2. testé
5. **LossUnit :** cette fonction calcule l'unité de perte en utilisant les montants nominaux et les taux de récupération associés. Elle appelle la fonction AmountUnit pour calculer l'unité de montant. Elle utilise également une méthode de recherche du PGCD pour déterminer l'unité de perte optimale.
   1. code en C# et vba
   2. testé
6. **ProxyPGCD**: Cette fonction calcule le PGCD (Plus Grand Commun Diviseur) entre deux nombres. Elle est utilisée à la fois dans les fonctions AmountUnit et LossUnit pour effectuer les calculs de PGCD.
   1. code en C# et vba
   2. testé
7. **GetLossUnit** : cette fonction est une interface pour obtenir l'unité de perte à partir d'une liste d'émetteurs, de montants nominaux et de taux de récupération. Elle vérifie d'abord les données d'entrée, puis appelle la fonction LossUnit pour effectuer le calcul.
   1. code en C# et vba
   2. testé