Option Trading Algorithm

Un projet en C# qui implémente diverses stratégies de trading d'options, telles que le Straddle, le Bull Spread, le Bear Spread, le Butterfly Spread, le Strangle, le Covered Call, le Protective Put et l'Iron Condor. Le projet comprend également une fonctionnalité pour exporter les données de payoff et les graphiques associés vers un fichier Excel.

Table des matières

Documentation Technique

Introduction

Ce projet vise à fournir une implémentation de diverses stratégies de trading d'options. Il permet aux utilisateurs de :

- Représenter des stratégies d'options complexes.
- Calculer les payoffs et les profits associés à l'ensemble des stratégies demandées. Cela calcule également les payoffs sur un intervalle de prix [MinPrice : Maxprice].
- Exporter les données et les graphiques vers un fichier pour une synthèse des profits et pertes maximales pour chacune des stratégies et feuille de synthèse globale des stratégies.

Sélection des Stratégies

Pour utiliser le programme et sélectionner les stratégies à analyser, modifier le fichier `Program.cs`. Dans ce fichier, vous trouverez une liste de stratégies :

```
List<Strategy> strategies = new List<Strategy>
{
      new Straddle(100.0, DateTime.Now.AddMonths(3), DateTime.Now, market,
pricingMethod: "BlackScholes"),
      new BullSpread(95.0, 105.0, DateTime.Now.AddMonths(3), DateTime.Now, market,
pricingMethod: "Tree"),
      new BearSpread(105.0, 95.0, DateTime.Now.AddMonths(3), DateTime.Now, market,
pricingMethod: "Tree"),
      new ButterflyStrategy(80.0, 100.0, 120.0, DateTime.Now.AddMonths(3),
DateTime.Now, market, pricingMethod: "BlackScholes"),
      new Strangle(90.0, 110.0, DateTime.Now.AddMonths(3), DateTime.Now, market,
pricingMethod: "BlackScholes"),
      new CoveredCall(100.0, DateTime.Now.AddMonths(3), DateTime.Now, market,
pricingMethod: "BlackScholes"),
      new ProtectivePut(100.0, DateTime.Now.AddMonths(3), DateTime.Now, market,
pricingMethod: "BlackScholes"),
      new IronCondor(80.0,90.0,110.0,120.0,DateTime.Now.AddMonths(3),
DateTime.Now, market, pricingMethod: "BlackScholes")
};
```

Par défaut, les stratégies sont définies pour l'instant par des EuropeanOption, ici nous sommes dans `Butterfly.cs` :

```
SetupPositions()
        {
            // Acheter un Call avec le prix d'exercice le plus bas
            EuropeanOption longCallLow = new EuropeanOption(_lowerStrike,
_expirationDate, "Call", _pricingDate);
            double premiumLongCallLow = _priceEngine.PriceOption(longCallLow,
_market, _pricingMethod, _steps);
            AddPosition(new OptionPosition(longCallLow, 1, premiumLongCallLow));

            // Vendre deux options Call avec le prix d'exercice moyen
            EuropeanOption shortCallMiddle = new EuropeanOption(_middleStrike,
_expirationDate, "Call", _pricingDate);
```

```
            double premiumShortCallMiddle =
_priceEngine.PriceOption(shortCallMiddle, _market, _pricingMethod, _steps);
            AddPosition(new OptionPosition(shortCallMiddle, -2,
premiumShortCallMiddle));

            // Acheter un Call avec le prix d'exercice le plus haut
            EuropeanOption longCallHigh = new EuropeanOption(_higherStrike,
_expirationDate, "Call", _pricingDate);
            double premiumLongCallHigh = _priceEngine.PriceOption(longCallHigh,
_market, _pricingMethod, _steps);
            AddPosition(new OptionPosition(longCallHigh, 1, premiumLongCallHigh));
        };
```

On utilise les options Européennes indifféremment des options Américaines car pour l'instant, il n'y a aucune différence entre les deux. En effet, le prix variant si et seulement s'il y a un exercice anticipé, comme nous n'avons pas encore implémenté les dividendes en tant que données, nous n'avons pas encore implémenté le choix entre option américaine et européenne. Cependant, il est important de noter que `TrinomialTree.cs` peut les prendre en compte.

Choix des Paramètres

Lors de l'ajout d'une stratégie, vous pouvez personnaliser les paramètres suivants :

- **Prix d'exercice (Strike Price)** : Le prix auquel l'option peut être exercée.
- **Date d'expiration** : La date à laquelle l'option expire.
- **Date de valorisation** : La date actuelle ou la date à laquelle l'option est évaluée.
- **Méthode de pricing** : La méthode utilisée pour calculer le prix de l'option (par exemple, Black-Scholes, Arbre Trinomial).
- **MinPrice** : Sera le prix Spot minimum qui servira de borne inférieure dans la boucle de comparaison des prix.
- **MaxPrice** : Sera le prix Spot maximum qui servira de borne supérieure dans la boucle de comparaison des prix.
- **Step** : Le pas d'itération sur l'intervalle [MinPrice : Maxprice].

Création des Options

Les options sont modélisées par la classe `EuropeanOption`. Chaque option est caractérisée par :

- **Type d'option** : "Call" ou "Put".
- **Prix d'exercice (StrikePrice)** : Le prix auquel l'option peut être exercée.
- **Date d'expiration (ExpirationDate)** : La date à laquelle l'option expire.

Exemple de création d'un :

```
EuropeanOption callOption = new EuropeanOption
{
    strikePrice: 100.0,
    expirationDate: DateTime.Now.AddMonths(3),
    nature: "Call",
```

```
    pricingDate: DateTime.Now
};
```

Gestion des Positions

Une position représente l'achat ou la vente d'un ou d'un actif sous-jacent. La classe `OptionPosition` est utilisée pour modéliser une position :

- **Option** : L'option associée à la position (ou `null` pour le sous-jacent).
- **Quantité (Quantity)** : Le nombre d'options achetées (positif) ou vendues (négatif).
- **Prime (Premium)** : Le prix payé ou reçu pour la position.

Exemple de création d'une position :

```
OptionPosition position = new OptionPosition
{
    option: callOption,
    quantity: 1,
    premium: premiumValue
};
```

Calcul du Payoff et du Profit

Le payoff d'une position est le gain ou la perte réalisée à l'expiration de l'option en fonction du prix du sous-jacent.

Calcul du Payoff pour un

- **Option Call** : Payoff = max(S - K, 0)
- **Option Put** : Payoff = max(K - S, 0)
- **Position Vendue** : Le payoff est multiplié par -1.

Calcul du Payoff Total d'une Stratégie

Le payoff total est la somme des payoffs de toutes les positions de la stratégie :

```
        public override double CalculatePayoff(double underlyingPrice)
        {
            double totalPayoff = 0.0;
            foreach (var position in Positions)
            {
                double positionPayoff = position.Payoff(underlyingPrice);
                totalPayoff += positionPayoff;
            }
            return totalPayoff;
        }
```

Calcul du Profit Total

Le profit total est calculé en tenant compte du coût initial (la somme des primes payées ou reçues) :

```
double totalProit = totalPayoff - InitialCost();
```

Détail des stratégies pour les mettre en place

Bull Spread

Description : Le Bull Spread est une stratégie haussière utilisant des options Call. Elle permet de limiter les pertes et les gains potentiels en utilisant deux options avec différents prix d'exercice.

- **Composition** :
  - Acheter un Call (`lowerStrikeCall`)
  - Vendre un Call (`higherStrikeCall`), avec `higherStrikeCall > lowerStrikeCall`
- **Payoff Théorique** : Payoff = max(S - lowerStrikeCall, 0) - max(S - higherStrikeCall, 0)
- **Formule du Profit** : Profit = Payoff - (Prime Call Acheté - Prime Call Vendu)

Bear Spread

Description : Le Bear Spread est une stratégie baissière utilisant des options Put. Elle permet de limiter les pertes et les gains potentiels en utilisant deux options avec différents prix d'exercice.

- **Composition** :
  - Acheter un Put (`higherStrikePut`)
  - Vendre un Put (`lowerStrikePut`), avec `higherStrikePut > lowerStrikePut`
- **Payoff Théorique** : Payoff = max(higherStrikePut - S, 0) - max(lowerStrikePut - S, 0)
- **Formule du Profit** : Profit = Payoff - (Prime Put Acheté - Prime Put Vendu)

Butterfly Spread

Description : Le Butterfly Spread est une stratégie de faible volatilité qui utilise trois prix d'exercice différents pour créer un profil de profit limité, combinant des options achetées et vendues.

- **Composition** :
  - Acheter un Call (`lowerStrikeCall`)
  - Vendre deux options Call (`middleStrikeCall`)

    - Acheter un Call (`higherStrikeCall`) avec `higherStrikeCall > middleStrikeCall > lowerStrikeCall`

- **Payoff Théorique** : Payoff = max(S - lowerStrikeCall, 0) - 2 * max(S - middleStrikeCall, 0) + max(S - higherStrikeCall, 0)
- **Formule du Profit** : Profit = Payoff - (Prime Call Acheté Bas + Prime Call Acheté Haut - 2 * Prime Call Vendu)

Calendar Spread

Description : Le Calendar Spread est une stratégie de faible volatilité qui utilise des options avec le même prix d'exercice mais des dates d'expiration différentes.

- **Composition** :
    - Vendre un à court terme
    - Acheter un à long terme avec le même prix d'exercice
- **Payoff Théorique** : Payoff = Valeur de l'option à long terme à l'échéance de la première option - Prime de l'option vendue
- **Formule du Profit** : Payoff - (Prime Option Long Terme - Prime Option Court Terme)

Covered Call

Description : Le Covered Call est une stratégie où l'investisseur détient le sous-jacent et vend un Call sur ce même actif.

- **Composition** :
    - Détenir l'actif sous-jacent
    - Vendre un Call (K)
- **Payoff Théorique** : Payoff = min(S, K)
- **Formule du Profit** : Profit = Payoff + Prime Reçue - Prix d'Achat de l'Actif

---

Protective Put

Description : Le Protective Put est une stratégie de couverture où l'investisseur détient le sous-jacent et achète un Put pour se protéger d'une baisse de prix.

- **Composition** :
    - Détenir l'actif sous-jacent
    - Acheter un Put (K)
- **Payoff Théorique** : Payoff = max(K, S)

- **Formule du Profit** : Profit = Payoff - Prix d'Achat de l'Actif - Prime Pu

---

Iron Condor

L'Iron Condor est une stratégie qui combine un *Bull Put Spread* et un *Bear Call Spread*, en utilisant quatre options avec des prix d'exercice différents.

Notations :

- **K1** : Le prix d'exercice le plus bas, associé à une option Put.
- **K2** : Le deuxième prix d'exercice, également associé à une option Put, avec **K2 > K1**.
- **K3** : Le troisième prix d'exercice, associé à une option Call, avec **K3 > K2**.
- **K4** : Le prix d'exercice le plus élevé, également associé à une option Call, avec **K4 > K3**.

Composition de l'Iron Condor

1. Vendre une option Put avec un prix d'exercice **K2** (Put vendu). (`higherStrikePut`)
2. Acheter une option Put avec un prix d'exercice **K1** (Put acheté). (`lowerStrikePut`), avec `lowerStrikePut < higherStrikePut`
3. Vendre une option Call avec un prix d'exercice **K3** (Call vendu). (`lowerStrikeCall`)
4. Acheter une option Call avec un prix d'exercice **K4** (Call acheté). (`higherStrikeCall`), avec `higherStrikeCall > lowerStrikeCall`

Formule du Payoff

Le payoff de l'Iron Condor peut être exprimé comme suit, en fonction du prix du sous-jacent **S** à l'échéance :

- Si **S <= K1** : Payoff = (K2 - K1)
- Si **K1 < S <= K2** : Payoff = (S - K1)
- Si **K2 < S <= K3** : Payoff = (K2 - K1)
- Si **K3 < S <= K4** : Payoff = (K2 - K1) - (S - K3)
- Si **S > K4** : Payoff = (K2 - K1) - (K4 - K3)

Le payoff de l'Iron Condor est limité, avec une perte maximale égale à la différence entre les strikes de chaque spread, moins les primes nettes reçues. La stratégie permet de profiter d'une faible volatilité en capitalisant sur une fourchette de prix étroite autour de **K2** et **K3**.

Straddle

Description : Le Straddle est une stratégie neutre qui consiste à acheter un Call et un Put avec le même prix d'exercice et la même date d'expiration.

- **Composition** :
    - Acheter un Call strike(K)
    - Acheter un Put strike(K)
- **Payoff Théorique** : Payoff = max(S - K, 0) + max(K - S, 0)

- **Formule du Profit** : Profit = Payoff - (Prime Call + Prime Put)

Strangle

Description : Le Strangle est une stratégie de volatilité qui consiste à acheter un Call et un Put avec des prix d'exercice différents.

- **Composition** :
  - Acheter un Call (`higherStrikeCall`)
  - Acheter un Put (`lowerStrikePut`)
- **Payoff Théorique** : Payoff = max(S - higherStrikeCall, 0) + max(lowerStrikePut - S, 0)
- **Formule du Profit** : Profit = Payoff - (Prime Call + Prime Put)

Étapes d'Installation

1. Cloner le dépôt GitHub :

```
git clone https://github.com/Julien-Zanin/OptionTradingAlgorithm.git
```

2. Naviguer vers le répertoire du projet :

```
cd OptionTradingAlgorithm
```

3. Restaurer les packages NuGet :

```
dotnet restore
```

Structure du Projet

- **OptionTradingAlgorithm** : Projet principal contenant le programme et les implémentations des stratégies.
  - `Program.cs` : Point d'entrée du programme.
  - `Strategies` : Dossier contenant les classes pour chaque stratégie.
    - `Straddle.cs`, `BullSpread.cs`, etc. : Implémentations des stratégies spécifiques.
  - `Modele` : Dossier contenant les modèles financiers.
    - `Market.cs` : Représente les conditions du marché. Cette classe récupère les paramètres liés au marché.
    - `Contract.cs` : Représente les paramètres du contrat.

        - `EuropeanOption.cs` : Modélisation des options européennes.
        - `AmericanOption.cs` : Modélisation des options américaines.
    - `Pricing` : Dossier contenant les méthodes de pricing.
        - `PriceEngine.cs` : Moteur de calcul des prix des options.
        - `BlackScholesPricer.cs` : Implémentation du modèle Black-Scholes.
        - `TrinomialTreePricer.cs` : Implémentation de l'arbre trinomial.
    - `Trading` : Dossier contenant les classes relatives aux positions.
        - `PositionOption.cs` : Représente une position sur un (Achat ou Vente).
        - `TradingStrategy.cs` : Représente les conditions du marché. Cette classe récupère les paramètres liés au marché.

- **OptionTradingAlgorithmTests** : Projet de tests unitaires pour valider le fonctionnement des stratégies.
- **ExcelExport** : Dossier contenant la classe `ExcelExporter.cs` pour l'exportation des données vers Excel.
- **Output** : Dossier généré contenant les fichiers Excel produits par le programme.

Dépendances

- **.NET 6.0** : Framework nécessaire pour exécuter l'application.
- **EPPlus** : Bibliothèque pour manipuler les fichiers Excel.
    - Installée via NuGet :

```
Install-Package EPPlus -Version 5.8.0
```

- **Microsoft.Extensions.Configuration.Json (optionnel)** : Pour gérer les configurations via un fichier `appsettings.json`.
- Bibliothèques de test :
    - `coverlet.collector`, `Microsoft.NET.Test.Sdk`, `xunit`, et `xunit.runner.visualstudio` pour exécuter et analyser les tests.

Considérations Futures

- **Extension des Stratégies** :
    - Ajout de nouvelles stratégies d'options.
    - Implémentation totale des Options Américaines avec l'implémentation des dividendes
- **Interface Utilisateur** :
    - Développement d'une interface graphique pour faciliter l'utilisation du programme.
- **Intégration de Données Réelles** :
    - Connexion à des API financières pour obtenir des données de marché en temps réel (style Yahoo Finance).

- **Intégration de modèles de volatilité** :
    - Implémentation d'une classe volatilité qui permettrait de choisir un modèle de calcul de volatilité tel que le modèle d'Heston mentionné lors de nos échanges.

Difficultées rencontrées

L'implémentation de certaines stratégies telle que l'Iron Condor a pris plus de temps et nous a permis de réaliser certaines erreurs d'implémentations dans le calcul des payoffs. La mise en place ces tests unitaires, bien que laborieux nous a également beaucoup servi à débugger. Le manque de temps nous a fait faillite pour l'intégration de données réelles, qui n'aurait pas rajouté grand chose en lui-même et l'intégration d'une petite interface utilisateur pour guider l'implémentation des données de marché et mise en place des stratégies.