Optimisation d'une Stratégie d'Arrêt Optimal pour un Payoff Asiatique

Mohamed CHAIB
Samuel JEAN-JACQUES

October 31, 2024

Introduction

Dans ce projet, nous nous intéressons à l'application des techniques d'arrêt optimal dans un cadre financier. Plus précisément, nous étudierons une stratégie d'optimisation du temps d'arrêt pour un actif dont les prix journaliers suivent un mouvement brownien géométrique sous la mesure de probabilité neutre au risque :

$$S_{n+1} = S_n \exp\left(-\frac{1}{2}\sigma^2 \Delta t + \sigma\sqrt{\Delta t}\,\epsilon_{n+1}\right), \tag{1}$$

où S_0 est le prix initial, σ représente la volatilité, et ϵ_{n+1} sont des variables aléatoires gaussiennes standard indépendantes.

On définit également la moyenne des prix jusqu'au jour n comme suit :

$$A_n = \frac{S_0 + S_1 + \cdots + S_n}{n+1}. \tag{2}$$

L'objectif est de maximiser l'espérance du ratio entre cette moyenne et le prix courant en appliquant une stratégie d'arrêt optimal. Pour cela, nous définissons un temps d'arrêt optimal $\tau_a = \min\{\min\{n \in \{0,\ldots,N\} \mid A_n \geq aS_n\}, N\}$ afin de maximiser $\mathbb{E}\left[\frac{A_{\tau_a}}{S_{\tau_a}}\right]$. On comparera les valeurs simulées avec les résultats théoriques et expérimentera différentes stratégies pour affiner cette optimisation.

1 Implémentation du Simulateur

Pour implémenter notre simulateur et simuler les dynamiques de prix, on va utiliser le langage Python, idéal pour sa facilité d'utilisation et les bibliothèques qu'il propose afin que l'on puisse implémenter des matrices, des graphiques et des calculs d'optimisation comme nous allons faire, pour cela on commence par importer les librairies nécessaires au fonctionnement de notre code :

```
import numpy as np
import optuna
```

```python
import matplotlib.pyplot as plt
```

Listing 1: Importation des packages

1.1 Création de la classe Simulator

Dans cette section, nous développons une classe `Simulator` en Python, qui permet de simuler la dynamique des prix d'une action basée sur le mouvement brownien géométrique. La classe comprend des méthodes pour simuler les trajectoires de prix, calculer la moyenne courante, et générer des visualisations.

1.1.1 Initialisation de la classe Simulator

Nous allons initialiser notre Simulateur en lui introduisant les variables qui interviennent dans la simulation des prix journaliers de l'actif sous-jacent :

- S_0 : Prix initial de l'actif sous-jacent
- σ : Volatilité
- Δ_t : Intervalle de temps (pas journalier)
- N : Nombre d'étapes de temps (de jours de trading)

```python
class Simulator:
    def __init__(self, S0, sigma, delta_t, N):
        self.S0 = S0
        self.sigma = sigma
        self.delta_t = delta_t
        self.N = N
```

Listing 2: Initialisation des variables utilisées dans la classe Simulator

1.1.2 Simulation des trajectoires de prix

Pour simuler les trajectoires de prix, nous générons d'abord des variables $(\epsilon_{i,j})_{\substack{1\leq i\leq M\\1\leq j\leq N}}$ telles que :

$$\forall i \in \{1,\dots,M\}, \forall j \in \{1,\dots,N\}, \quad \epsilon_{i,j} \sim \mathcal{N}(0,1)$$

On génère alors une matrice "incréments" de taille $M \times N$, constituée de variables aléatoires gaussiennes centrées réduites.

Pour chaque trajectoire i, on initialise le prix à sa valeur initiale S_0 et on simule les variations de prix de manière itérative. La dynamique des prix suit un mouvement brownien géométrique, de sorte que pour chaque jour j, le prix $S_{i,j}$ est mis à jour selon la formule :

$$S_{i,j+1} = S_{i,j} \exp\left(-\frac{1}{2}\sigma^2\Delta t + \sigma\sqrt{\Delta t}\,\epsilon_{i,j}\right)$$

où σ représente la volatilité, et Δt est le pas de temps (par exemple, $\frac{1}{252}$ pour une simulation journalière). Cette formule assure une prise en compte du drift (tendance) corrigé de la volatilité et de l'effet aléatoire des variations des prix.

Ensuite, la moyenne cumulative $A_{i,j}$ pour chaque trajectoire i jusqu'au jour j est calculée comme suit :

$$A_{i,j} = \frac{1}{j+1} \sum_{k=0}^{j} S_{i,k}$$

Ainsi, la moyenne cumulative A est obtenue de manière vectorisée pour éviter les boucles et optimiser la simulation de nombreuses trajectoires, fournissant une estimation de la moyenne des prix à chaque étape pour chacune des M trajectoires.

```
    def simulate_paths(self, M):
        """Simule M trajectoires des prix S et calcule la moyenne cumulative A"""
        # Matrice de bruit gaussien pour M trajectoires sur N jours
        increments = np.random.normal(0, 1, (M, self.N))

        # Initialisation des trajectoires de S et assignation du prix initial
        S = np.zeros((M, self.N + 1))
        S[:, 0] = self.S0

        # Calcul vectorise de chaque trajectoire S
        S[:, 1:] = S[:, [0]] * np.exp(
            np.cumsum((-0.5 * self.sigma**2 * self.delta_t) + self.sigma * np.sqrt(
    self.delta_t) * increments, axis=1)
        )

        # Calcul vectorise de la moyenne cumulative A
        A = np.cumsum(S, axis=1) / np.arange(1, self.N + 2)

        return S, A
```

Listing 3: Simulation de la trajectoire de prix pour M simulations

1.1.3 Affichage des trajectoires simulées

La fonction `plot_trajectories` permet de tracer plusieurs trajectoires simulées du prix de l'actif S et des moyennes cumulatives associées A, sans recourir à des boucles explicites pour l'affichage. On génère tout d'abord M trajectoires de prix et leurs moyennes cumulatives sur N jours avec la fonction `simulate_paths`. Ensuite, un vecteur temporel est créé pour représenter les jours de la période, et les trajectoires de S et A sont tracées en une seule opération avec un style pointillé pour A.

```
    def plot_trajectories(self, M=5):
        """Trace quelques trajectoires de l'actif S et des moyennes A sans boucle.
    """
        # Simule M trajectoires
        S, A = self.simulate_paths(M)
        time = np.arange(self.N + 1)

        plt.figure(figsize=(12, 6))

        # Tracer toutes les trajectoires S et A en une seule operation
        plt.plot(time, S.T, label=[f'Trajectoire S {i+1}' for i in range(M)], alpha
    =0.7)
        plt.plot(time, A.T, linestyle='--', label=[f'Moyenne A {i+1}' for i in range
    (M)], alpha=0.7)
```

```
12
13          plt.title("Trajectoires simulees de l'actif et des moyennes associ es")
14          plt.xlabel("Temps (jours)")
15          plt.ylabel("Prix de l'actif / Moyenne cumulative")
16          plt.legend(loc="upper left", bbox_to_anchor=(1, 1))  # Legende deportee pour
        plus de lisibilite
17          plt.show()
```

Listing 4: Affichage des Trajectoires Simulées de l'Actif et des Moyennes Cumulatives

1.2 Utilisation des méthodes de Monte Carlo pour le calcul du prix du payoff $\frac{A_N}{S_N}$

Les méthodes de Monte Carlo sont une classe d'algorithmes qui utilisent des simulations aléatoires pour estimer des valeurs numériques. En finance, elles sont couramment utilisées pour estimer le prix d'instruments financiers lorsque les formules analytiques sont complexes ou inexistantes. En générant de nombreuses trajectoires possibles d'un actif sous-jacent selon un modèle probabiliste, on peut estimer la valeur d'un payoff complexe en calculant la moyenne des valeurs simulées.

L'objectif est d'estimer l'espérance $E\left(\frac{A_N}{S_N}\right)$, où

$$A_N = \frac{1}{N+1}\sum_{n=0}^{N} S_n$$

représente la moyenne cumulative des prix simulés jusqu'à un instant final N, et S_N le prix final de l'actif. Pour cela, le code utilise les méthodes de Monte Carlo en générant M trajectoires indépendantes du processus stochastique S, modélisé par un mouvement brownien géométrique, selon la formule :

$$S_{n+1} = S_n \exp\left(\left(-\frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}\,\epsilon_{n+1}\right)$$

où σ est la volatilité, Δt est l'incrément de temps, et ϵ_{n+1} est une variable aléatoire normale standard. Pour chaque trajectoire i, on calcule le ratio $\frac{A_N^{(i)}}{S_N^{(i)}}$. La moyenne de ces M ratios fournit une estimation de l'espérance recherchée, donnée par :

$$\frac{1}{M}\sum_{i=1}^{M}\frac{A_N^{(i)}}{S_N^{(i)}}$$

Cette méthode est cohérente car cette moyenne empirique converge vers la véritable espérance $E\left(\frac{A_N}{S_N}\right)$ lorsque $M \to \infty$, soit :

$$E\left(\frac{A_N}{S_N}\right) = \lim_{M\to\infty}\frac{1}{M}\sum_{i=1}^{M}\frac{A_N^{(i)}}{S_N^{(i)}}$$

grâce à la loi des grands nombres. De plus, la précision de cette estimation s'améliore à mesure que M augmente, avec une réduction de l'erreur d'estimation en $\frac{1}{\sqrt{M}}$, ce qui signifie que l'erreur

quadratique moyenne (RMSE) est inversement proportionnelle à la racine carrée du nombre de simulations. Cette convergence garantit qu'en augmentant M, les résultats obtenus par simulation Monte Carlo deviennent comparables aux résultats théoriques. Ainsi, l'approche Monte Carlo fournit une estimation fiable de l'espérance recherchée en utilisant la convergence de la moyenne empirique sur un grand nombre de simulations.

Cela donne le code Python suivant :

```
    def calculate_payoff(self, M):
        # Generer M trajectoires simultanement pour S et A, de taille (M, N+1)
        S, A = self.simulate_paths(M)

        # Selection des valeurs finales des trajectoires (A_N et S_N)
        A_N = A[:, -1]
        S_N = S[:, -1]

        # Calcul du ratio A_N / S_N pour chaque trajectoire, puis de la moyenne
        payoff = np.mean(A_N / S_N)
        return payoff
```

Listing 5: Calcul du Payoff Espéré avec la Méthode de Monte Carlo

1.3 Résultats de la simulation de S et A sur un horizon temporel donné

1.3.1 Calcul de la valeur théorique

Recherche de la valeur théorique du prix du payoff $\frac{A_n}{S_n}$:

$$A_n = \frac{S_0 + S_1 + \cdots + S_n}{n+1}$$

Soit $A = \frac{1}{2}\sigma^2\Delta t$ et $B = \sigma\sqrt{\Delta t}$.

$$S_1 = S_0 \exp(A + B\xi_1)$$

$$S_2 = S_1 \exp(A + B\xi_2) = S_0 \exp(A + B\xi_1)\exp(A + B\xi_2) = S_0 \exp(2A + B(\xi_1 + \xi_2))$$

On remarque que :

$$S_n = S_0 \exp\left(nA + B\sum_{i=1}^{n} \xi_i\right)$$

et que pour $j \in \{1, \ldots, n\}$, on a :

$$S_j = S_0 \exp\left(jA + B\sum_{i=1}^{j} \xi_i\right)$$

donc

$$A_n = \frac{S_0 + \sum_{j=1}^{n} S_j}{n+1}$$

On a :

$$\mathbb{E}\left[\frac{A_n}{S_n}\right] = \frac{1}{n+1}\left[\mathbb{E}\left(\frac{S_0 + \sum_{j=1}^n S_j}{S_n}\right)\right] = \frac{1}{n+1}\left[\mathbb{E}\left(\frac{S_0}{S_n}\right) + \sum_{j=1}^n \mathbb{E}\left(\frac{S_j}{S_n}\right)\right]$$

Calcul de $\mathbb{E}\left[\frac{S_0}{S_n}\right]$:

$$\mathbb{E}\left[\frac{S_0}{S_n}\right] = \mathbb{E}\left[\frac{\cancel{S_0}}{\cancel{S_0}\exp\left(nA + B\sum_{i=1}^n \xi_i\right)}\right] = \mathbb{E}\left[\exp\left(-nA - B\sum_{i=1}^n \xi_i\right)\right] = \exp\left(-nA\right)\mathbb{E}\left[\exp(\sum_{i=1}^n (-B\xi_i)\right]$$

avec $\xi_i \sim \mathcal{N}(0,(0,2)^2)$, donc $\xi_i = 0,2X_i$ avec $X_i \sim \mathcal{N}(0,1)$, et $(\xi_i)_{i\geq 0}$ i.i.d, ce qui implique $(X_i)_{i\geq 0}$ i.i.d. et on obtient :

$$\mathbb{E}\left[\frac{S_0}{S_n}\right] = \exp(-nA)\,\mathbb{E}\left[\exp\left(\sum_{i=1}^n -0,2BX_i\right)\right] = \exp(-nA)\prod_{i=1}^n \underbrace{\mathbb{E}\left[\exp(-0,2BX_i)\right]}_{=\exp(\frac{0.2^2B^2}{2})} \quad \text{car } (X_i) \text{ iid}$$

Finalement,

$$\begin{aligned}
\mathbb{E}\left[\frac{S_j}{S_n}\right] &= \mathbb{E}\left[\frac{S_0\exp\left(jA + B\sum_{i=1}^j \xi_i\right)}{S_0\exp\left(nA + B\sum_{i=1}^n \xi_i\right)}\right] \\
&= \mathbb{E}\left[\exp\left(jA + B\sum_{i=1}^j \xi_i - nA - B\sum_{i=1}^n \xi_i\right)\right] \\
&= \mathbb{E}\left[\exp\left(-A(n-j) - B\sum_{i=j+1}^n \xi_i\right)\right] \\
&= \exp(-A(n-j))\mathbb{E}\left[\exp\left(-B\sum_{i=j+1}^n \xi_i\right)\right] \\
&= \exp(-A(n-j))\prod_{i=j+1}^n \mathbb{E}\left[\exp(-B\xi_i)\right] \quad \text{car } \xi_i \text{ sont iid} \\
&= \exp(-A(n-j))\left(\exp\left(\frac{0.2^2B^2}{2}\right)\right)^{n-j}
\end{aligned}$$

On en déduit :

$$\mathbb{E}\left[\frac{A_n}{S_n}\right] = \frac{1}{n+1}\left(\exp(-nA)\left(\exp\left(\frac{0.04B^2}{2}\right)\right)^n + \sum_{j=1}^n \exp\left(-A(n-j)\right)\exp\left(\frac{0.04B^2}{2}\right)^{n-j}\right)$$

Ainsi :

$$\mathbb{E}\left[\frac{A_n}{S_n}\right] = \frac{1}{n+1}\left[\exp\left(-nA + n \cdot 0.02B^2\right) + \sum_{j=1}^{n} \exp\left(-A(n-j) + (n-j) \cdot 0.02B^2\right)\right]$$

■

On remplace avec

$$n = 22,$$
$$A = -0.5 \times 0.2^2 \times \frac{1}{252} = -\frac{0.02}{252},$$
$$B = 0.2 \times \sqrt{\frac{1}{252}}$$

On implémente ce raisonnement sur Python pour calculer la valeur théorique :

```python
import numpy as np

# Parametres donnes
S0 = 10           # Prix initial
sigma = 0.2       # Volatilite
delta_t = 1 / 252  # Increment de temps (jours)
n = 22            # Nombre de jours

# Calcul de A et B en fonction de sigma et delta_t
A = -0.5 * sigma**2 * delta_t
B = sigma * np.sqrt(delta_t)

# Calcul de l'esperance de E[An / Sn] avec la formule extraite
def esperance_An_Sn(n, A, B):
    # Premiere partie de la formule
    first_term = np.exp(-n * A) * np.exp(0.02 * B**2 / 2)**n

    # Deuxieme partie : somme des termes pour j = 1 a n
    sum_terms = sum(np.exp(-A * (n - j)) * np.exp((n - j) * 0.02 * B**2) for j in
    range(1, n + 1))

    # Calcul de l'esperance E[An / Sn]
    esperance = (1 / (n + 1)) * (first_term + sum_terms)

    return esperance

# Calcul de l'esperance
esperance_An_Sn_value = esperance_An_Sn(n, A, B)

# Affichage du resultat
print(f"L'esperance de A_n / S_n pour n={n} est : {esperance_An_Sn_value:.4f}")
```

Listing 6: Calcul de la valeur théorique de $\mathrm{E}\left(\frac{A_n}{S_n}\right)$

Ce qui nous permet à terme de trouver pour n=22 :

- $\mathbb{E}\left(\frac{A_n}{S_n}\right) = 1.0009$

1.3.2 Comparaison des valeurs simulées avec la valeur théorique

On exécute les codes affichés ci-dessus avec les paramètres suivants :

- $S_0 = 10$,
- $\sigma = 0.2$,
- $\Delta_t = \frac{1}{252}$,
- $N = 22$

Pour faire tourner notre algorithme, on va simuler 1000 trajectoires (M=1000) pour que les prix des payoffs simulés soient comparables aux résultats théoriques on va tracer 5 trajectoires pour visualiser le comportement du mouvement brownien géométrique avec nos paramètres

Le code Python de notre simulation donne :

```python
# Parametres de simulation globaux
S0 = 10        # Prix initial
sigma = 0.2    # Volatilite
delta_t = 1 / 252  # Increment de temps
N = 22         # Nombre de jours

simulator = Simulator(S0, sigma, delta_t, N)

payoff_estime = simulator.calculate_payoff(M=1000)
print(f"Prix estime du payoff E(A_N / S_N) : {payoff_estime}")

# Trace des trajectoires simulees
simulator.plot_trajectories(M=5)
```

Listing 7: Simulation Monte Carlo pour l'Estimation du Payoff Espéré et Visualisation des Trajectoires

Une exécution du code nous donne :

- **Payoff attendu** : 1.0010543461105241
- **Temps d'exécution** : 1.26 secondes

On peut aussi déterminer un intervalle pour le prix du payoff espéré avec Monte Carlo pour évaluer la variabilité et la précision de l'estimation, afin valider la cohérence du modèle en comparant les résultats simulés aux attentes théoriques ou aux données empiriques, garantissant ainsi une meilleure fiabilité de l'estimation du payoff.

Construisons un intervalles de payoffs obtenus en faisant 100000 simulations Monte Carlo :

```python
payoff_estimations = [simulator.calculate_payoff(M=1000) for _ in range(100000)]

# Calcul de l'intervalle de resultats (minimum et maximum)
min_payoff = np.min(payoff_estimations)
max_payoff = np.max(payoff_estimations)
mean_payoff = np.mean(payoff_estimations)

min_payoff, max_payoff, mean_payoff
```

Listing 8: Estimation de l'Intervalle du Payoff Espéré avec 100000 Simulations Monte Carlo

Les résultats d'une exécution sont (pour 10^6 simulations) :

- **Intervalle des résultats** : [0.9971396802419431, 1.0063894919185559]
- **moyenne** = 1.0017575486047168
- **Temps d'exécution** : 126.78 secondes

On voit que avec un nombre très grand de simulations, le payoff espéré se rapproche fortement de la valeur théorique déterminée précédemment, ce qui est en accord avec l'utilisation des méthodes de Monte Carlo.

Affichage des courbes de payoff :

Voici un exemple d'une représentation de 5 graphiques obtenus avec en lignes continues les prix et en lignes pointillées les moyennes :

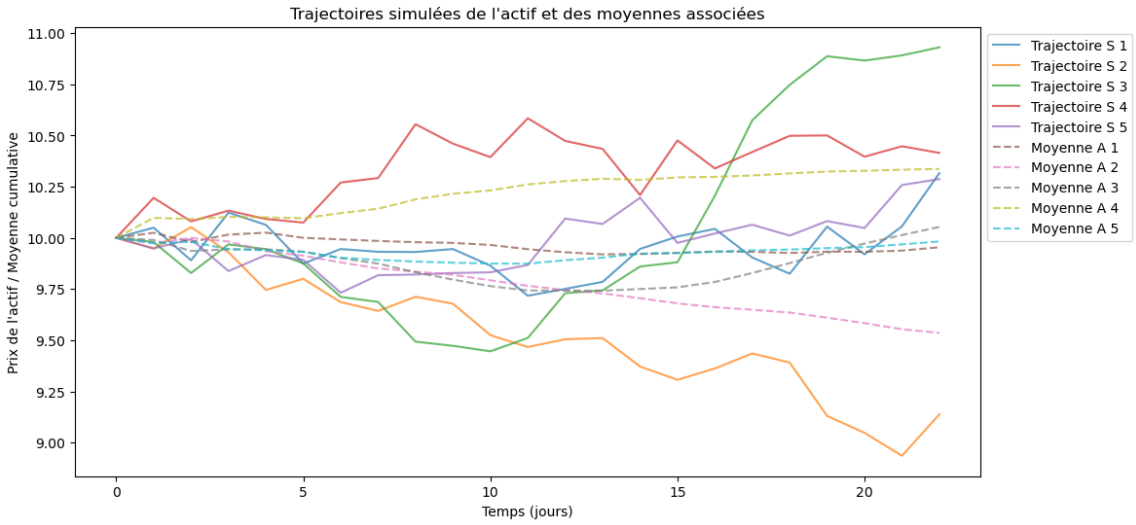

Figure 1: Graphique des Trajectoires Simulées de l'Actif et des Moyennes Cumulatives

2 Stratégie d'arrêt optimal

2.1 Définition du problème

Nous souhaitons maximiser l'espérance

$$\mathbb{E}\left[\frac{A_\tau}{S_\tau}\right]$$

où τ est un temps d'arrêt optimal appartenant à l'ensemble $\{0, 1, \ldots, N\}$. Les notations utilisées sont :

- A_τ : la moyenne des valeurs de l'actif jusqu'au temps τ,

- S_τ : la valeur de l'actif au temps τ,

On veut alors définir un moment optimal pour "arrêter" et réaliser le payoff dans le but de maximiser l'espérance $E\left[\frac{A_\tau}{S_\tau}\right]$. On détermine alors le moment optimal pour tirer le meilleur avantage du rapport entre la moyenne des prix et le prix actuel de l'actif. Ce choix stratégique réduit l'impact de fluctuations extrêmes du prix à court terme et stabilise le payoff de l'option en s'appuyant sur une tendance moyenne. En identifiant un temps d'arrêt optimal τ, on peut maximiser la valeur espérée de l'option, sécuriser des gains plus stables, et minimiser les risques associés à la volatilité du marché. Cette approche peut nous aider à améliorer le rendement de notre position tout en optimisant la gestion de risque dans un contexte de marché potentiellement incertain.

Pour optimiser ce payoff espéré, on va employer deux stratégies :

- La 1ère étant une stratégie de seuil consistant à optimiser $\mathbb{E}\left[\frac{A_{\tau_a}}{S_{\tau_a}}\right]$ avec pour $a \geq 1$, un temps optimal τ_a définit par

$$\tau_a = \min\{\min\{n \in \{0, \ldots, N\} \mid A_n \geq aS_n\}, N\}$$

- La 2ème visant à déterminer un temps d'arrêt optimal afin d'approcher la maximisation de $\mathbb{E}\left[\frac{A_\tau}{S_\tau}\right]$ utilisant l'enveloppe de Snell

2.2 Maximisation de $\mathbb{E}\left[\frac{A_{\tau_a}}{S_{\tau_a}}\right]$

2.2.1 Initialisation de la Classe "Strategy"

On conçoit la classe `Strategy` pour implémenter et tester une stratégie de seuil optimale sur des trajectoires de prix simulées. Elle utilise la classe `Simulator` pour déterminer le temps d'arrêt optimal τ_a, défini comme le premier instant où la moyenne cumulative A_n dépasse un multiple a du prix actuel S_n.

L'implémentation nous donne :

```
class Strategy:
    def __init__(self, simulator):
        self.simulator = simulator
```

Listing 9: Initialisation de la classe 'Strategy' pour maximiser le payoff espéré par l'utilisation de la classe 'Simulator'

2.2.2 Estimation du payoff obtenu avec la stratégie de seuil

Le code suivant a pour objectif d'estimer le payoff d'une stratégie de seuil en utilisant des simulations de Monte Carlo (cf 1.2). Dans cette stratégie, on s'arrête dès que la moyenne cumulative des prix, A, dépasse un certain multiple du prix actuel S (défini par le seuil a).
Tout comme dans la partie 1.2, on commence par générer M trajectoires pour les prix S et les moyennes A, puis calcule le payoff final comme le ratio $\frac{A_N}{S_N}$ en fin de trajectoire.
Ensuite, on identifie le temps d'arrêt optimal pour chaque trajectoire, soit le premier instant où $A \geq a \cdot S$, et ajuste le temps d'arrêt à la fin de la trajectoire si ce seuil n'est jamais atteint.

Les valeurs de A et S sont récupérées aux temps d'arrêt pour calculer le ratio moyen $\frac{A_{\tau_a}}{S_{\tau_a}}$ pour toutes les simulations. Enfin, on renvoie l'espérance du payoff obtenu avec la stratégie de seuil, ainsi que le payoff final moyen pour comparer la performance de la stratégie.

Cela nous donne le code suivant :

```
        def monte_carlo_payoff(self, a, M=1000):
            """Strategie de seuil : arret lorsque A >= a * S."""
            # Genere les trajectoires de S et A pour M simulations
            S, A = self.simulator.simulate_paths(M)

            # Calcul du payoff final AN / SN
            payoff_final = A[:, -1] / S[:, -1]

            # Critere de temps d'arret : A >= a * S
            stopping_criteria = (A >= a * S)
            stopping_times = np.argmax(stopping_criteria, axis=1)

            # V rifie les arrets valides et d finit les arrets finaux si non
    rencontres
            valid_stops = stopping_criteria[np.arange(M), stopping_times]
            stopping_times[~valid_stops] = self.simulator.N

            # Recupere A et S aux temps d'arret
            A_tau = A[np.arange(M), stopping_times]
            S_tau = S[np.arange(M), stopping_times]

            # Calcule le ratio moyen et retourne l'esperance et le payoff final
            return np.mean(A_tau / S_tau), np.mean(payoff_final)
```

Listing 10: Calcul du payoff d'une stratégie de seuil par simulation de Monte Carlo

On évoquera ensuite la recherche du τ_a optimal

2.2.3 Optimisation du paramètre τ_a

Dans notre cadre, on utilise la bibliothèque Optuna pour optimiser le paramètre de seuil a en maximisant l'espérance du payoff attendu de la stratégie d'arrêt. Pour cela, on définit la fonction objectif $f(a) = \mathbb{E}\left[\frac{A_{\tau_a}}{S_{\tau_a}}\right]$ et cherchons à la maximiser. Dans le code, cette maximisation est transformée en minimisant son opposé $-f(a) = -\mathbb{E}\left[\frac{A_{\tau_a}}{S_{\tau_a}}\right]$. À chaque essai de valeur pour a, Optuna évalue $f(a)$ par une approximation Monte Carlo :

$$f(a) \approx \frac{1}{M} \sum_{i=1}^{M} \frac{A_{\tau_a^{(i)}}}{S_{\tau_a^{(i)}}},$$

où M est le nombre de simulations et $\tau_a^{(i)}$ représente le temps d'arrêt pour la i-ième simulation.

Optuna utilise ensuite un modèle probabiliste (basé notamment sur une estimation par noyau) pour estimer les valeurs potentielles de $f(a)$ et applique un critère d'acquisition, défini par

$$\text{Critère}(a) = \mathbb{E}[\max(f(a^*) - f(a), 0)],$$

où $f(a^*)$ est le meilleur payoff observé jusqu'à présent. Ce critère d'acquisition permet de choisir des valeurs de a qui maximisent l'amélioration attendue, en exploitant les résultats observés.

À chaque itération, Optuna affine son modèle de $f(a)$ pour converger vers l'optimum global ou un bon optimum local pour $f(a)$, permettant ainsi de maximiser efficacement $\mathbb{E}\left[\frac{A_{\tau_a}}{S_{\tau_a}}\right]$ sans évaluer toutes les valeurs de a possibles dans l'intervalle.

Le code utilisé pour adopter cette méthode est le suivant (ce code n'est pas dans la classe `Strategy` car il utilise la fonction `monte_carlo_payoff` de la classe `Strategy`):

```
# Fonctions d'optimisation pour optuna

def objective_threshold(trial):
    """Objectif pour optimiser la strategie de seuil avec le parametre a."""
    a = trial.suggest_uniform('a', 1, 3)
    strategy = Strategy(Simulator(S0, sigma, delta_t, N))
    payoff, _ = strategy.monte_carlo_payoff(a)
    return -payoff  # Maximisation en minimisant l'oppose

# Execution de l'optimisation avec optuna pour la strategie de seuil
study_threshold = optuna.create_study(direction="minimize")
study_threshold.optimize(objective_threshold, n_trials=100)
```

Listing 11: Optimisation du paramètre de seuil pour maximiser le payoff avec Optuna

On peut désormais exécuter ces codes et évaluer nos résultats.

2.2.4 Résultats

On évalue à présent la performance de cette stratégie sur des simulations. En optimisant a, nous cherchons à maximiser l'espérance du payoff lorsque le temps d'arrêt est défini par la condition $A_n \geq a \cdot S_n$. Ensuite, nous appliquons ce paramètre optimal dans une nouvelle simulation pour estimer le payoff moyen obtenu avec la stratégie de seuil, ainsi que le payoff final $\frac{A_N}{S_N}$ au dernier instant. On pourra alors évaluer l'efficacité de la stratégie de seuil optimisée.

```
# Resultats de la strategie de seuil
best_a = study_threshold.best_params['a']
simulator = Simulator(S0, sigma, delta_t, N)
strategy = Strategy(simulator)
payoff_threshold, payoff_final = strategy.monte_carlo_payoff(best_a, M=1000)
print(f"Meilleur parametre a pour la strategie de seuil : {best_a}")
print(f"Esperance maximale estimee avec seuil : {payoff_threshold:.4f}")
print(f"Prix estime du payoff A_N / S_N : {payoff_final:.4f}")
```

Listing 12: Évaluation des résultats de la stratégie de seuil avec le paramètre optimal

Le résultat sur notre simulation nous donne les résultats suivants :

- Meilleur paramètre a pour la stratégie de seuil 1.0177167008835728
- Espérance maximale estimée avec seuil : 1.0106
- Prix estimé du payoff $\frac{A_N}{S_N}$: 1.0030

Cette stratégie est alors gagnante car elle maximise le payoff attendu en arrêtant dès que $A_n \geq 1.0177 \cdot S_n$. Elle offre un rendement espéré de 1.0106 contre 1.0011 sans stratégie d'arrêt, permettant d'augmenter notre rendement dès que les conditions sont favorables, sans attendre la fin de la période.
Cette approche optimise le rendement total sur le long terme en capturant des opportunités au moment idéal.

Cette stratégie est donc efficace en pratique, mais est-elle la meilleure ?
On va adopter une autre stratégie et la comparer avec la stratégie de seuil.

2.3 Stratégie d'arrêt optimal par utilisation de l'enveloppe de Snell

2.3.1 Enveloppe de Snell

Pour cette stratégie, on utilise **l'enveloppe de Snell**, une suite de variables aléatoires qui donne la valeur optimale du problème à chaque étape n, en tenant compte de toutes les décisions futures possibles.

L'enveloppe de Snell V_n est définie par la relation de récurrence suivante :

$$V_n = \max\left(\frac{A_n}{S_n}, \mathbb{E}[V_{n+1}|\mathcal{F}_n]\right) \tag{3}$$

où :

- V_n représente la valeur optimale de continuation si on ne s'arrête pas au temps n
- $\mathcal{F}_n$ est la filtration (ensemble d'informations disponibles) au temps n.

2.3.2 Application

On modélise tout d'abord le Payoff Instantané qui est défini pour chaque instant t par :

$$Z_t = \frac{A_t}{S_t}$$

où :

- A_t représente la moyenne cumulative jusqu'à l'instant t.
- S_t est le prix de l'actif à t.

Z_t est calculé pour toutes les trajectoires et tous les instants de la simulation, créant une matrice Z de dimension $M \times N$ (avec M le nombre de simulations et N le nombre d'instants).

On construit ensuite l'Enveloppe de Snell via l'Algorithme de Bellman pour obtenir la meilleure espérance de payoff atteignable si l'on décide d'arrêter ou de continuer à chaque instant t. On note cette enveloppe **snell_values** et elle est calculée par backward recursion, en partant de la dernière période jusqu'à la première, c'est le principe même de l'algorithme de Bellman.

Récurrence

Pour chaque instant $t = N, N-1, N-2, \ldots, 0$, on applique la récurrence de Bellman pour mettre à jour l'enveloppe de Snell :

$$\text{snell\_values}[:, t] = \max(Z[:, t], \mathbb{E}[\text{snell\_values}[:, t+1 \mid \mathcal{F}_t])$$

Cela se traduit dans le code par :

$$\text{snell\_values}[:, t] = \max(Z[:, t], \text{snell\_values}[:, t+1])$$

où l'espérance conditionnelle $\mathbb{E}[\text{snell\_values}[:, t+1] \mid \mathcal{F}_t]$ est approximée par la valeur future snell_values[: , $t+1$] de l'enveloppe.
Cela signifie que, pour chaque instant t, la stratégie d'arrêt optimale est soit de :

- S'arrêter avec un payoff Z_t

ou

- Continuer avec l'espérance de payoff futur donné par $\text{snell\_values}_{t+1}$.

Ainsi, snell_values_t représente la valeur optimale de payoff atteignable si on commence à décider de s'arrêter ou de continuer à partir de l'instant t.

Une fois l'enveloppe de Snell calculée, le temps d'arrêt optimal pour chaque simulation est le premier instant t où le payoff instantané Z_t est supérieur ou égal à l'enveloppe de Snell :

$$\tau = \min\{t \in [0, N] \mid Z_t \geq \text{snell\_values}_t\}$$

Cela garantit que l'on s'arrête dès que le payoff instantané est au moins égal à la meilleure espérance de payoff possible si l'on continue.

Enfin, on calcule l'espérance du payoff obtenu au temps d'arrêt optimal τ pour toutes les simulations, donnée par :

$$\frac{1}{M} \sum_{i=1}^{M} \frac{A_{\tau_i}}{S_{\tau_i}}$$

où τ_i est le temps d'arrêt optimal pour la i-ième simulation. Le code retourne cette moyenne comme l'espérance du payoff pour la stratégie d'arrêt optimale (tout comme dans la section 1.2, en utilisant l'enveloppe de Snell pour maximiser le rendement attendu.

2.3.3 Code

On incorpore le code qui applique le raisonnement effectué dans la section 2.3.2 dans la classe `Strategy`:

```
    def snell_envelope_strategy(self, M=1000):
        """Strat gie basee sur l'enveloppe de Snell pour un arret optimal."""
        # Genere les trajectoires de S et A pour M simulations
        S, A = self.simulator.simulate_paths(M)

        # Calcul du payoff Z = A / S a chaque temps
        Z = A / S     # Matrice contenant les payoffs instantanee pour chaque jour

        # Calcul de l'enveloppe de Snell en recurrence inversee
        snell_values = np.zeros_like(Z) # snell_values est une matrice avec les
    memes dimensions que la matrice Z
        snell_values[:, -1] = Z[:, -1]

        for t in range(self.simulator.N - 1, -1, -1): # On commence a N-1, on
    termine    0
            snell_values[:, t] = np.maximum(Z[:, t], snell_values[:, t + 1])
        # La matrice snell_values est remplit au fur et a mesure c'est l'enveloppe
    de Snell
        # snell_values est la matrice de l'enveloppe de Snell, qui contient la
    valeur optimale (le payoff maximal possible)
        # que  l on  peut esperer pour chaque jour et chaque simulation.

        # Temps d'arret optimal base sur l'enveloppe de Snell
        stopping_times = np.argmax(Z >= snell_values, axis=1)
        # np.argmax renvoie l'indice du 1er    lment    True

        # Recupere A et S aux temps d'arret pour chaque simulation
        A_tau = A[np.arange(M), stopping_times]
        S_tau = S[np.arange(M), stopping_times]

        return np.mean(A_tau / S_tau)
```

Listing 13: Implémentation de la stratégie d'arrêt optimale basée sur l'enveloppe de Snell

2.3.4 Résultats

Toujours avec les paramètres données dans l'énoncé, on fait tourner les codes suivants pour afficher nos résultats :

```
# Resultats de la strategie avec l'enveloppe de Snell
simulator = Simulator(S0, sigma, delta_t, N)
strategy = Strategy(simulator)

payoff_s = strategy.snell_envelope_strategy(M=1000)
print(f"Esperance maximale estimee avec l'enveloppe de Snell : {payoff_s:.4f}")
```

Listing 14: Évaluation de l'espérance de payoff avec la stratégie d'arrêt optimale basée sur l'enveloppe de Snell

Le code exécuté nous donne les résultats suivants :

- Espérance maximale estimée avec l'enveloppe de Snell : 1.0308

L'espérance maximale estimée avec l'enveloppe de Snell, égale à 1.0308, représente le rendement espéré optimal en appliquant une stratégie d'arrêt qui maximise le payoff à chaque instant, selon la meilleure estimation possible fournie par l'enveloppe de Snell. Comparée aux autres stratégies, cette approche montre un avantage significatif, car elle intègre de manière dynamique l'évaluation des gains futurs à chaque instant de la simulation.

Par exemple, dans une stratégie de seuil où le meilleur payoff était de 1.0106 avec un seuil fixe, on n'optimise pas les décisions d'arrêt en fonction des valeurs futures de chaque simulation, ce qui limite son efficacité. De même, un payoff final sans stratégie d'arrêt aboutissait à 1.0011, ce qui montre que l'absence de prise de décision anticipée minimise le rendement attendu.

Ainsi, l'enveloppe de Snell offre une stratégie plus sophistiquée, en tenant compte de la valeur future attendue à chaque instant pour maximiser le rendement, et démontre un rendement supérieur par rapport aux méthodes plus simples

3 Conclusion

Dans un contexte financier, cette stratégie est avantageuse car elle permet de réaliser des profits en capturant les meilleurs moments pour sortir d'une position, en adaptant chaque décision à la dynamique future des prix, ce qui augmente la rentabilité par rapport aux stratégies statiques.