

**Programmation pour la Finance**

DA EIRA Adam, MONIN Louis

*ING3 Fintech*

M. SAVRY 2024-2025

Sommaire

[Introduction 5](#_Toc194868991)

[Quelle(s) est/sont la/les limite(s) de cette stratégie, en apparence séduisante ? 5](#_Toc194868992)

[Whipsaw Effect 5](#_Toc194868993)

[Risque de Gap 5](#_Toc194868994)

[Coûts de transaction et frictions de marché 6](#_Toc194868995)

[Sous-performance en marchés trending 6](#_Toc194868996)

[Hypothèse de liquidité parfaite 6](#_Toc194868997)

[Comment évolue la sensibilité de la distribution de la valeur terminale en rapport aux paramètres 6](#_Toc194868998)

[Volatilité de l’actif risque 6](#_Toc194868999)

[Plancher Garanti 7](#_Toc194869000)

[Niveau de tolérance (Seuil de Stop-Loss) 7](#_Toc194869001)

[Fréquence de recomposition 7](#_Toc194869002)

[Partie A 7](#_Toc194869003)

[Code Python pour simuler le processus de prix d’un actif 7](#_Toc194869004)

[Résultats du code Python pour simuler le processus de prix d’un actif 10](#_Toc194869005)

[Interprétation du code Python pour simuler le processus de prix d’un actif 11](#_Toc194869006)

[Partie B 12](#_Toc194869007)

[Objectif 12](#_Toc194869008)

[Code Python pour appliquer la stratégie d’allocations d’actifs 13](#_Toc194869009)

[Résultats du code Python pour appliquer la stratégie d’allocations d’actifs 17](#_Toc194869010)

[Interprétation du code Python pour appliquer la stratégie d’allocations d’actifs 17](#_Toc194869011)

[Code Python pour simuler la stratégie Stop-Loss dynamiquement 18](#_Toc194869012)

[Résultats du code Python pour simuler la stratégie Stop-Loss dynamiquement 22](#_Toc194869013)

[Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les frais de transactions 22](#_Toc194869014)

[Résultat du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les frais de transactions 25](#_Toc194869015)

[Interprétation du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les frais de transactions 26](#_Toc194869016)

[Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte le slippage 26](#_Toc194869017)

[Résultat du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte le slippage 30](#_Toc194869018)

[Interprétation du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte le slippage 30](#_Toc194869019)

[Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les taxes sur les gains en capital 31](#_Toc194869020)

[Résultat du Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les taxes sur les gains en capital 34](#_Toc194869021)

[Quelle(s) limite(s) relevez-vous lors de l’application de cette stratégie ? 35](#_Toc194869022)

[Frais de transaction élevés 35](#_Toc194869023)

[Sensibilité à la volatilité 35](#_Toc194869024)

[Garantie partielle 35](#_Toc194869025)

[Dépendance au taux sans risque 35](#_Toc194869026)

[Complexité de mise en œuvre 36](#_Toc194869027)

[Auriez-vous des solutions ou conseils à proposer à un investisseur ? 36](#_Toc194869028)

[Réduire les frais de transaction 36](#_Toc194869029)

[Adapter la stratégie à la volatilité 36](#_Toc194869030)

[Renforcer la garantie 36](#_Toc194869031)

[Diversifier le portefeuille 37](#_Toc194869032)

[Automatiser la stratégie 37](#_Toc194869033)

[Adapter la stratégie au contexte économique 37](#_Toc194869034)

[Tester la stratégie sur des données historiques 37](#_Toc194869035)

[Partie C 38](#_Toc194869036)

[Code python pour tracer sur le même graphique, la représentation de la densité pour le titre Michelin et la densité gaussienne 38](#_Toc194869037)

[Résultat du code python pour tracer sur le même graphique, la représentation de la densité pour le titre Michelin et la densité gaussienne 39](#_Toc194869038)

[Interprétation du code python pour tracer sur le même graphique, la représentation de la densité pour le titre Michelin et la densité gaussienne 40](#_Toc194869039)

[Code python pour étudier la sensibilité de la distribution de la valeur terminale en fonction de différents niveaux de paramètres 40](#_Toc194869040)

[Résultat du code python pour étudier la sensibilité de la distribution de la valeur terminale en fonction de différents niveaux de paramètres 43](#_Toc194869041)

[Code python pour optimiser les différents paramètres 46](#_Toc194869042)

[Résultat du code python pour optimiser les différents paramètres 49](#_Toc194869043)

[Interprétation du code python pour optimiser les différents paramètres 50](#_Toc194869044)

[Bibliographie 51](#_Toc194869045)

# Introduction

## Quelle(s) est/sont la/les limite(s) de cette stratégie, en apparence séduisante ?

La stratégie Stop-Loss est une technique de gestion des risques visant à limiter les pertes potentielles lors de transactions financières. Elle consiste à définir un seuil de prix prédéterminé, appelé stop-loss, auquel une position est automatiquement clôturée en cas de mouvement défavorable du marché. Ce seuil peut être fixé en pourcentage ou en valeur absolue par rapport au prix d'achat de l'actif. Par exemple, un investisseur peut décider de vendre une action si son cours baisse de 10 % en dessous de son prix d'acquisition. Cette approche est particulièrement efficace dans des marchés volatils, car elle permet de préserver le capital et de réduire l’influence des décisions émotionnelles. Cependant, il est crucial de positionner le stop-loss de manière stratégique afin d'éviter qu'il ne soit activé par de simples fluctuations passagères.

La stratégie Stop-Loss possède bien que le concept semble séduisant pour sa simplicité, ce dernier présente plusieurs limites critiques. On peut citer notamment :

### Whipsaw Effect

Le Stop-Loss est vulnérable aux mouvements transitoires de prix (bruit de marché). Lorsque le prix d’un actif franchit temporairement le seuil de Stop-Loss avant de rebondir, l’investisseur subit une perte irréversible tout en rabantant la reprise. Les marchés volatils, les stratégies Stop-Loss génèrent des pertes fréquentes dues à des déclenchements intempestifs avec un ratio de Sharpe inférieur à une stratégie buy-and-hold. [1]

### Risque de Gap

En cas de chute brutale lors de l’ouverture du marché, le Stop-Loss est exécuté à un prix inférieur au seuil fixé, invalidant la protection. Ce risque est amplifié pour les actifs illiquides ou lors de crises où les gaps dépassent souvent 10%. [2]

### Coûts de transaction et frictions de marché

Les stratégies Stop-Loss fréquemment déclenchées augmentent les coûts de transaction (frais de courtage, bid-ask spread), réduisant la performance nette. On peut ainsi voir que même des coûts de transaction modestes (0,5%) annulent l’avantage théorique des stratégies dynamiques comme le Stop-Loss. [3]

### Sous-performance en marchés trending

Le Stop-Loss force une sortie prématurée en cas de tendance haussière volatile, limitant le potentiel de gain, une analyse de Annaert et al dans *Financial Analysts Journal* sur le S&P 500 révèle que le Stop-Loss sous -performe systématiquement le buy-and-hold sur le long terme, en particulier dans les phases de reprise post-crise. [4]

### Hypothèse de liquidité parfaite

La stratégie suppose une exécution instantanée au seuil fixé, ce qui est irréaliste en période de stress.

## Comment évolue la sensibilité de la distribution de la valeur terminale en rapport aux paramètres

### Volatilité de l’actif risque

Une volatilité élevée accroît la fréquence des franchissements du seuil de Stop-Loss, augmentant les coûts de transaction et réduisant l’espérance de gain. Selon Berkelaar et al. (2004) dans Management Science, la distribution de la valeur terminale devient asymétrique à gauche (negative skewness) sous haute volatilité, avec une ligne gauche épaisse due aux sorties prématurées. Par exemple, pour un actif avec une volatilité annuelle de 30%, le Stop-Loss déclenche 3 fois plus souvent que pour une volatilité de 15%, réduisant le CAGR de 2-3% (Backtest sur données historiques, 1990-2020).

### Plancher Garanti

Un placher élevé réduit le risque de perte extrême mais limite l’exposition à la hausse, compressant la distribution de la valeur terminale vers la médiane, on peut citer les recherches de Grossman et Zhou dans *Journal of Economic Dynamics and Control* qui montrent qu’un plancher à 95% génère une distribution quasi-log normale tronquée, avec un ratio Sortino inférieur de 20% à une stratégie sans plancher. [5]

### Niveau de tolérance (Seuil de Stop-Loss)

Un seuil serré (environ 5%) réduit le drawdown maximal mais augmente la probabilité de sorties prématurées. Inversement, un seuil large (environ 20%) expose à des pertes plus importantes mais capture mieux les rebonds.

### Fréquence de recomposition

Une recomposition fréquente du portefeuille permet de réagir rapidement aux mouvements du marché, mais peut entraîner des coûts de transaction élevés et un risque de surréaction à la volatilité à court terme. Une recomposition hebdomadaire ou mensuelle limite les coûts mais expose à des risques de non-exécution.

# Partie A

## Code Python pour simuler le processus de prix d’un actif

import numpy as np

import matplotlib.pyplot as plt

class ProcessusBrownienGeometrique:

    """

    Classe pour simuler le prix d’un actif selon un Mouvement Brownien Géométrique (MBG)

    en utilisant la formule discrétisée d'Euler.

    """

    def \_\_init\_\_(self, S0, r, sigma, T, N, nb\_simulations):

        """

        Initialisation des paramètres du modèle.

        Paramètres :

        - S0 : float

            Prix initial de l'actif.

        - r : float

            Taux sans risque (en décimal).

        - sigma : float

            Volatilité de l'actif sous-jacent (en décimal).

        - T : float

            Horizon temporel (en années).

        - N : int

            Nombre de pas de discrétisation.

        - nb\_simulations : int

            Nombre de trajectoires à simuler.

        """

        self.S0 = S0

        self.r = r

        self.sigma = sigma

        self.T = T

        self.N = N

        self.nb\_simulations = nb\_simulations

        self.dt = T / N  # Durée de chaque intervalle de temps

        self.chemins\_simules = None

    def simuler(self):

        """

        Simule les trajectoires du processus brownien géométrique pour l'actif sous-jacent.

        Utilise l'approximation d'Euler pour discrétiser l'équation du mouvement brownien géométrique.

        """

        # Variables aléatoires normales (les processus de Wiener)

        aleatoires = np.random.normal(size=(self.nb\_simulations, self.N))

        # Calcul des incréments du processus brownien géométrique

        increments = (self.r - 0.5 \* self.sigma \*\* 2) \* self.dt + self.sigma \* np.sqrt(self.dt) \* aleatoires

        # Trajectoires logarithmiques

        chemins\_log = np.log(self.S0) + np.cumsum(increments, axis=1)

        # Reconstitution des prix à partir des log-prix

        self.chemins\_simules = np.exp(chemins\_log)

        return self.chemins\_simules

    def tracer(self):

        """

        Affiche les trajectoires simulées du prix de l’actif.

        """

        if self.chemins\_simules is None:

            self.simuler()  # Si les trajectoires n'ont pas encore été simulées, on les génère d'abord

        plt.figure(figsize=(10, 6))

        time = np.linspace(0, self.T, self.N)

        # Affichage de chaque trajectoire simulée

        for i in range(self.nb\_simulations):

            plt.plot(time, self.chemins\_simules[i], lw=1)

        # Détails du graphique

        plt.title("Simulations du Mouvement Brownien Géométrique")

        plt.xlabel("Temps (années)")

        plt.ylabel("Prix de l'actif $S(t)$")

        plt.grid(True)

        plt.legend()

        plt.show()

# Exemple d'utilisation

if \_\_name\_\_ == "\_\_main\_\_":

    # Paramètres du modèle

    S0 = 100       # Prix initial de l'actif

    r = 0.05       # Taux sans risque (5%)

    sigma = 0.2    # Volatilité (20%)

    T = 1          # Horizon temporel (1 an)

    N = 252        # Nombre de jours de bourse dans l'année (252 jours)

    nb\_simulations = 5  # Nombre de trajectoires à simuler

    # Création d'un objet pour simuler le mouvement brownien géométrique

    mbg = ProcessusBrownienGeometrique(S0, r, sigma, T, N, nb\_simulations)

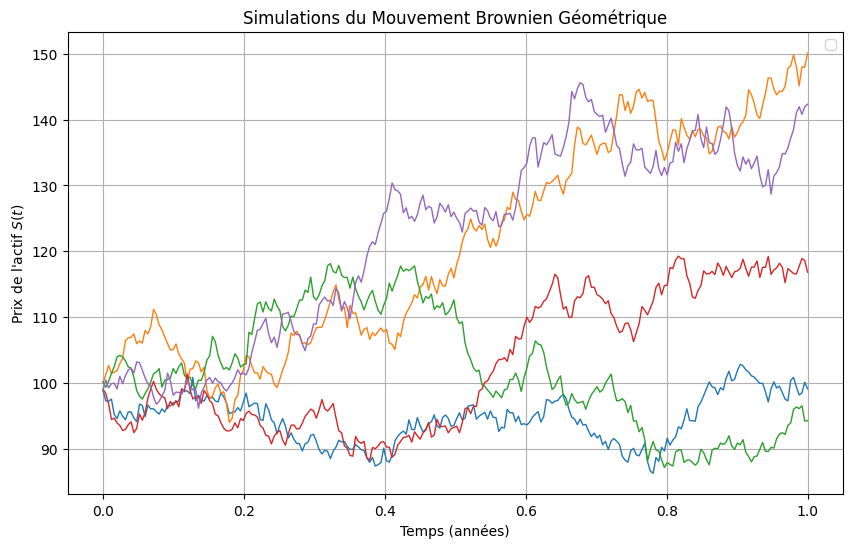
    # Simuler les trajectoires

    mbg.simuler()

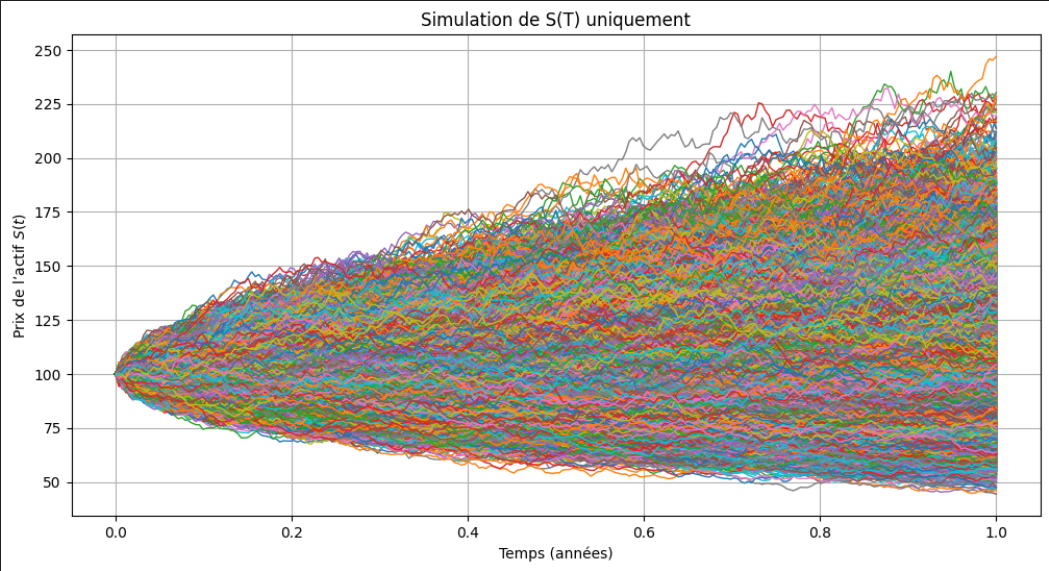
    # Afficher les trajectoires simulées

    mbg.tracer()

## Résultats du code Python pour simuler le processus de prix d’un actif



*Simulation de Mouvement Brownien Géométrique pour 5 simulations*



S*imulation de Mouvement Brownien Géométrique pour 100 000 simulations*

Statistiques empiriques sur S(T) après 5 simulations :

* Espérance empirique : 94.3168
* Ecart-type empirique : 13.4063

Statistiques empiriques sur S(T) après 100 000 simulations :

* Espérance empirique : 105.1532
* Ecart-type empirique : 21.2665

## Interprétation du code Python pour simuler le processus de prix d’un actif

Formule du Mouvement Brownien :

Loi suivie par le logarithme du prix :

Moments de la loi log-normale :

* Espérance théorique de S(T) :
* Ecart-type théorique de S(T) :

Application numérique pour T = 1, S0 = 100, r = 5% et σ = 20% :

* Espérance :
* Ecart-type :

Lorsque l’on effectue 100 000 simulations, les résultats empiriques sont très proches de ceux attendus théoriquement. L’écart observé est négligeable, ce qui confirme la validité de la simulation par rapport à la formule analytique. En revanche, avec seulement 10 simulations, l’espérance empirique atteint une valeur élevée (115), en raison du bruit statistique lié à la faible taille de l’échantillon. Cette surestimation s’explique notamment par la nature asymétrique à droite de la loi log-normale, qui peut produire occasionnellement des valeurs extrêmes. Ainsi, quelques trajectoires particulièrement favorables suffisent à augmenter fortement la moyenne. Ces résultats ne sont pas statistiquement significatifs ; pour obtenir des statistiques robustes, un nombre de simulations bien plus élevé (généralement N ≥ 10⁴) est requis.

# Partie B

## Objectif

L’objectif de cette stratégie dynamique est de garantir à l’échéance T un pourcentage x du capital initial S0​, tout en permettant à l’investisseur de bénéficier d’une partie de la performance haussière de l’actif risqué.

À la mise en place de la stratégie, l’investisseur est intégralement exposé à l’actif risqué. Ensuite, la gestion de l’allocation suit une logique conditionnelle simple :

* Tant que la valeur du portefeuille reste supérieure ou égale à un seuil dynamique M(t), fonction du plancher garanti actualisé dans le temps, l’exposition au risque est maintenue à 100 %.
* Si la valeur du portefeuille tombe en dessous de ce seuil M(t), la position est intégralement liquidée et transférée dans l’actif sans risque afin de préserver le capital garanti.
* Si, après cette sécurisation, l’actif risqué remonte au-dessus du niveau du seuil M(t), le portefeuille est de nouveau entièrement réinvesti dans l’actif risqué.

Cette logique permet une protection dynamique du capital tout en conservant un potentiel de participation aux gains en cas de reprise des marchés.

## Code Python pour appliquer la stratégie d’allocations d’actifs

import yfinance as yf

import numpy as np

import matplotlib.pyplot as plt

# Téléchargement des données financières des actifs risqué et sans risque

vbisx = yf.download("VBISX", start="2019-01-01")  # Téléchargement du prix de l'actif sans risque (VBISX)

sp500 = yf.download("^GSPC", start="2019-01-01")  # Téléchargement du prix de l'actif risqué (S&P 500)

# Fusion des données des deux actifs sur la même période

data = vbisx[['Close']].rename(columns={'Close': 'VBISX'}).join(

    sp500[['Close']].rename(columns={'Close': 'S&P500'}),

    how='inner'  # Jointure interne pour ne garder que les dates communes

)

# Extraction des prix des actifs risqué et sans risque

prix\_risque = data["S&P500"].values  # Valeur de l'actif risqué (S&P 500)

prix\_sans\_risque = data["VBISX"].values  # Valeur de l'actif sans risque (VBISX)

dates = data.index  # Les dates de l'investissement

# Calcul des rendements relatifs des actifs par rapport à leur valeur initiale

rendement\_risque = prix\_risque / prix\_risque[0]  # Rendement de l'actif risqué (S&P 500)

rendement\_sans\_risque = prix\_sans\_risque / prix\_sans\_risque[0]  # Rendement de l'actif sans risque (VBISX)

class StrategieStopLossGain:

    """

    Classe représentant la stratégie de stop-loss pour garantir un certain pourcentage du gain.

    """

    def \_\_init\_\_(self, rendement\_risque, rendement\_sans\_risque, dates, gain\_protege\_pct):

        """

        Initialisation des paramètres de la stratégie.

        Paramètres :

        - rendement\_risque : Array des rendements relatifs de l'actif risqué (S&P 500)

        - rendement\_sans\_risque : Array des rendements relatifs de l'actif sans risque (VBISX)

        - dates : Index des dates pour l'évolution des rendements

        - gain\_protege\_pct : Pourcentage du gain à garantir à la fin de l'horizon de l'investissement

        """

        self.rendement\_risque = rendement\_risque

        self.rendement\_sans\_risque = rendement\_sans\_risque

        self.dates = dates

        self.N = len(rendement\_risque)  # Nombre de périodes (jours)

        self.gain\_protege\_pct = gain\_protege\_pct  # Pourcentage à garantir du placement initial

        # Initialisation des variables pour suivre la valeur du portefeuille et du plancher de gain

        self.plancher = np.zeros(self.N)

        self.valeur\_portefeuille\_theorique = np.ones(self.N)

        self.valeur\_portefeuille\_capitalisee = np.ones(self.N)

        self.positions = ["risque"] \* self.N  # Liste de positions (risque ou sans risque)

    def appliquer\_strategie(self):

        """

        Applique la stratégie de stop-loss dynamique en fonction des rendements des actifs.

        """

        max\_rendement = self.rendement\_risque[0]  # Le rendement maximal au début est égal au rendement initial

        self.plancher[0] = self.gain\_protege\_pct \* max\_rendement  # Le plancher est fixé à un pourcentage du rendement initial

        dans\_actif\_risque = True  # On commence en investissant dans l'actif risqué

        # Parcours des périodes pour appliquer la stratégie à chaque étape

        for t in range(1, self.N):

            max\_rendement = max(max\_rendement, self.rendement\_risque[t])  # Mise à jour du rendement maximal observé

            self.plancher[t] = self.gain\_protege\_pct \* max\_rendement  # Mise à jour du plancher de gain

            # Récupération des rendements précédents pour les calculs

            r\_risk\_prev = self.rendement\_risque[t - 1] if self.rendement\_risque[t - 1] != 0 else 1

            r\_safe\_prev = self.rendement\_sans\_risque[t - 1] if self.rendement\_sans\_risque[t - 1] != 0 else 1

            # === Mise à jour de la valeur théorique du portefeuille en fonction de la position actuelle ===

            if dans\_actif\_risque:

                self.valeur\_portefeuille\_theorique[t] = self.rendement\_risque[t]

            else:

                self.valeur\_portefeuille\_theorique[t] = self.rendement\_sans\_risque[t]

            # === Mise à jour de la valeur capitalisée ===

            if dans\_actif\_risque:

                if self.rendement\_risque[t] < self.plancher[t]:

                    # Transition vers l'actif sans risque si la condition du stop-loss est remplie

                    dans\_actif\_risque = False

                    self.positions[t:] = ["sans\_risque"] \* (self.N - t)

                    croissance = self.rendement\_sans\_risque[t] / r\_safe\_prev

                else:

                    croissance = self.rendement\_risque[t] / r\_risk\_prev

            else:

                if self.rendement\_risque[t] > self.plancher[t]:

                    # Retour à l'actif risqué si la condition est remplie

                    dans\_actif\_risque = True

                    self.positions[t:] = ["risque"] \* (self.N - t)

                    croissance = self.rendement\_risque[t] / r\_risk\_prev

                else:

                    croissance = self.rendement\_sans\_risque[t] / r\_safe\_prev

            # Mise à jour de la valeur capitalisée du portefeuille en fonction de la croissance

            self.valeur\_portefeuille\_capitalisee[t] = self.valeur\_portefeuille\_capitalisee[t - 1] \* croissance

    def afficher\_allocation(self):

        """

        Affiche les résultats de la stratégie avec une visualisation améliorée du portefeuille.

        """

        plt.figure(figsize=(14, 6))

        # Affichage des rendements des actifs risqué et sans risque

        plt.plot(self.dates, self.rendement\_risque, label="S&P 500 (actif risqué)", color='blue', alpha=0.4)

        plt.plot(self.dates, self.rendement\_sans\_risque, label="VBISX (actif sans risque)", linestyle='--', alpha=0.5, color='orange')

        # Affichage du plancher de gain

        plt.plot(self.dates, self.plancher, label="Plancher de gain", linestyle=':', color='red')

        # Affichage de la valeur théorique du portefeuille suivant la stratégie

        plt.plot(self.dates, self.valeur\_portefeuille\_theorique, label="Stratégie théorique", color='black', linewidth=1)

        # Affichage de la valeur capitalisée du portefeuille

        plt.plot(self.dates, self.valeur\_portefeuille\_capitalisee, label="Valeur du portefeuille", color='green', linewidth=1)

        # Titre et axes

        plt.title("Stratégie Stop Loss dynamique", fontsize=16)

        plt.xlabel("Date", fontsize=14)

        plt.ylabel("Rendement cumulé", fontsize=14)

        plt.legend()

        plt.grid(True)

        plt.tight\_layout()

        plt.show()

# Paramètres de la stratégie : définition du pourcentage du gain à protéger (ici 90%)

strategie = StrategieStopLossGain(rendement\_risque, rendement\_sans\_risque, dates, gain\_protege\_pct=0.9)

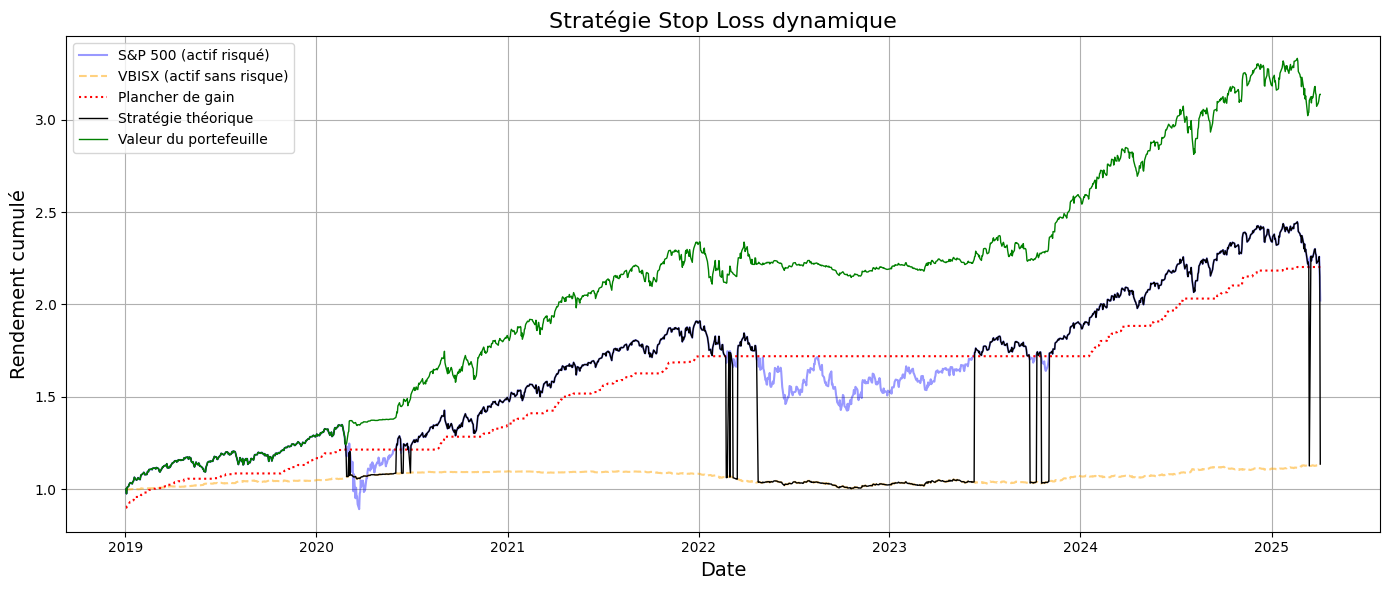
# Appliquer la stratégie

strategie.appliquer\_strategie()

# Afficher l'allocation de la stratégie

strategie.afficher\_allocation()

## Résultats du code Python pour appliquer la stratégie d’allocations d’actifs



## Interprétation du code Python pour appliquer la stratégie d’allocations d’actifs

Dans le cadre de cette étude, nous avons implémenté une stratégie dynamique de type Stop-Loss avec protection de gain visant à garantir à l’investisseur un pourcentage donné du maximum de performance atteint par un actif risqué (ici le S&P 500) sur une période donnée. Concrètement, dès que la performance du S&P 500 passe sous un plancher de sécurité défini comme un pourcentage du plus haut atteint (par exemple 90 %), le capital est immédiatement transféré vers un actif sans risque (ici le fonds obligataire VBISX). Cette stratégie, entièrement réversible, permet ainsi de profiter des hausses de marché tout en protégeant une partie des gains réalisés. La mise en œuvre algorithmique repose sur un suivi quotidien des rendements relatifs des deux actifs, la mise à jour dynamique du plancher protégé, et une logique de bascule entre actif risqué et sans risque dès que les conditions sont remplies. L’évolution de la valeur du portefeuille ainsi que les changements d’allocation sont visualisés sur la période 2019–2024, montrant l’efficacité de la stratégie en période de volatilité.

## Code Python pour simuler la stratégie Stop-Loss dynamiquement

import streamlit as st

import yfinance as yf

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import random

# === Fonction principale de stratégie ===

def appliquer\_strategie\_stoploss(params):

    if params["mode\_simulation"]:

        N = 1000

        dates = pd.date\_range(start="2019-01-01", periods=N, freq="B")

        rendement\_risque = np.cumprod(1 + np.random.normal(0.0004, 0.01, N))

        rendement\_sans\_risque = np.cumprod(1 + np.random.normal(0.0001, 0.002, N))

    else:

        vbisx = yf.download("VBISX", start="2019-01-01")

        sp500 = yf.download("^GSPC", start="2019-01-01")

        data = vbisx[['Close']].rename(columns={'Close': 'VBISX'}).join(

            sp500[['Close']].rename(columns={'Close': 'S&P500'}),

            how='inner'

        )

        prix\_risque = data["S&P500"].values

        prix\_sans\_risque = data["VBISX"].values

        dates = data.index

        rendement\_risque = prix\_risque / prix\_risque[0]

        rendement\_sans\_risque = prix\_sans\_risque / prix\_sans\_risque[0]

    N = len(rendement\_risque)

    plancher = np.zeros(N)

    valeur\_portefeuille = np.ones(N)

    valeur\_nette = np.ones(N)

    max\_rendement = rendement\_risque[0]

    plancher[0] = params["gain\_protege"] \* max\_rendement

    dans\_actif\_risque = True

    lock\_in = params["lock\_in"]

    latence = params["latence"]

    attente = 0

    for t in range(1, N):

        max\_rendement = max(max\_rendement, rendement\_risque[t])

        plancher[t] = params["gain\_protege"] \* max\_rendement

        r\_risk\_prev = rendement\_risque[t-1] if rendement\_risque[t-1] != 0 else 1

        r\_safe\_prev = rendement\_sans\_risque[t-1] if rendement\_sans\_risque[t-1] != 0 else 1

        if t < lock\_in:

            changement\_possible = False

        elif attente > 0:

            changement\_possible = False

            attente -= 1

        else:

            changement\_possible = True

        if dans\_actif\_risque:

            if changement\_possible and rendement\_risque[t] < plancher[t]:

                dans\_actif\_risque = False

                attente = latence

                croissance = rendement\_sans\_risque[t] / r\_safe\_prev

                croissance \*= (1 - params["frais\_transaction"])

            else:

                croissance = rendement\_risque[t] / r\_risk\_prev

        else:

            if changement\_possible and rendement\_risque[t] > plancher[t]:

                dans\_actif\_risque = True

                attente = latence

                croissance = rendement\_risque[t] / r\_risk\_prev

                croissance \*= (1 - params["frais\_transaction"])

            else:

                croissance = rendement\_sans\_risque[t] / r\_safe\_prev

        if not dans\_actif\_risque:

            croissance \*= (1 - params["frais\_gestion"] / 252)

        croissance \*= (1 - params["inflation\_journaliere"])

        if params["stress"] and random.random() < 0.01:

            croissance \*= random.uniform(0.90, 0.97)

        valeur\_portefeuille[t] = valeur\_portefeuille[t - 1] \* croissance

        gain\_brut = valeur\_portefeuille[t] - 1

        impot = params["taux\_imposition"] \* gain\_brut if gain\_brut > 0 else 0

        valeur\_nette[t] = valeur\_portefeuille[t] - impot

    return dates, rendement\_risque, rendement\_sans\_risque, plancher, valeur\_portefeuille, valeur\_nette

# === Interface Streamlit ===

st.set\_page\_config(layout="wide")

st.title("Simulateur de stratégie Stop-Loss dynamique")

# === Sidebar: paramètres utilisateur ===

st.sidebar.header("Paramètres de simulation")

gain\_protege = st.sidebar.slider("% de gain protégé (plancher)", 0.5, 1.0, 0.9, step=0.05)

taux\_imposition = st.sidebar.slider("Taux d'imposition sur plus-values (%)", 0, 50, 30, step=5) / 100

frais\_transaction = st.sidebar.slider("Frais de transaction (%)", 0.0, 1.0, 0.2, step=0.1) / 100

frais\_gestion = st.sidebar.slider("Frais de gestion annuel sur actif sans risque (%)", 0.0, 2.0, 0.5, step=0.1) / 100

inflation\_annuelle = st.sidebar.slider("Taux d'inflation annuel (%)", 0.0, 10.0, 2.0, step=0.1) / 100

lock\_in = st.sidebar.slider("Période de blocage initiale (jours)", 0, 365, 30, step=5)

latence = st.sidebar.slider("Latence comportementale (jours)", 0, 10, 1, step=1)

stress = st.sidebar.checkbox("Activer les chocs de marché aléatoires (stress)", value=False)

mode\_simulation = st.sidebar.checkbox("Mode simulation hors ligne (aléatoire)", value=False)

inflation\_journaliere = inflation\_annuelle / 252

params = {

    "gain\_protege": gain\_protege,

    "taux\_imposition": taux\_imposition,

    "frais\_transaction": frais\_transaction,

    "frais\_gestion": frais\_gestion,

    "inflation\_journaliere": inflation\_journaliere,

    "lock\_in": lock\_in,

    "latence": latence,

    "stress": stress,

    "mode\_simulation": mode\_simulation

}

if st.sidebar.button("Lancer la simulation"):

    with st.spinner("Simulation en cours..."):

        try:

            dates, r\_risque, r\_safe, plancher, val\_brute, val\_nette = appliquer\_strategie\_stoploss(params)

            st.success("Simulation réussie ✅")

            st.subheader("Évolution de la valeur du portefeuille")

            fig, ax = plt.subplots(figsize=(12, 5))

            ax.plot(dates, val\_brute, label="Valeur brute", color="green")

            ax.plot(dates, val\_nette, label="Valeur nette après impôt", linestyle="--", color="purple")

            ax.plot(dates, plancher, label="Plancher garanti", linestyle=":", color="red")

            ax.set\_xlabel("Date")

            ax.set\_ylabel("Valeur du portefeuille")

            ax.legend()

            ax.grid(True)

            st.pyplot(fig)

            st.subheader("Résumé des résultats")

            st.write("\*\*Valeur finale brute :\*\* {:.3f}".format(val\_brute[-1]))

            st.write("\*\*Valeur finale nette après impôt :\*\* {:.3f}".format(val\_nette[-1]))

            st.write("\*\*Plancher garanti final :\*\* {:.3f}".format(plancher[-1]))

            st.write("\*\*Gain brut :\*\* {:.3f}".format(val\_brute[-1] - 1))

            st.write("\*\*Impôt payé :\*\* {:.3f}".format((val\_brute[-1] - 1) \* taux\_imposition if val\_brute[-1] > 1 else 0))

            st.write("\*\*Surperformance nette vs plancher :\*\* {:.3f}".format(val\_nette[-1] - plancher[-1]))

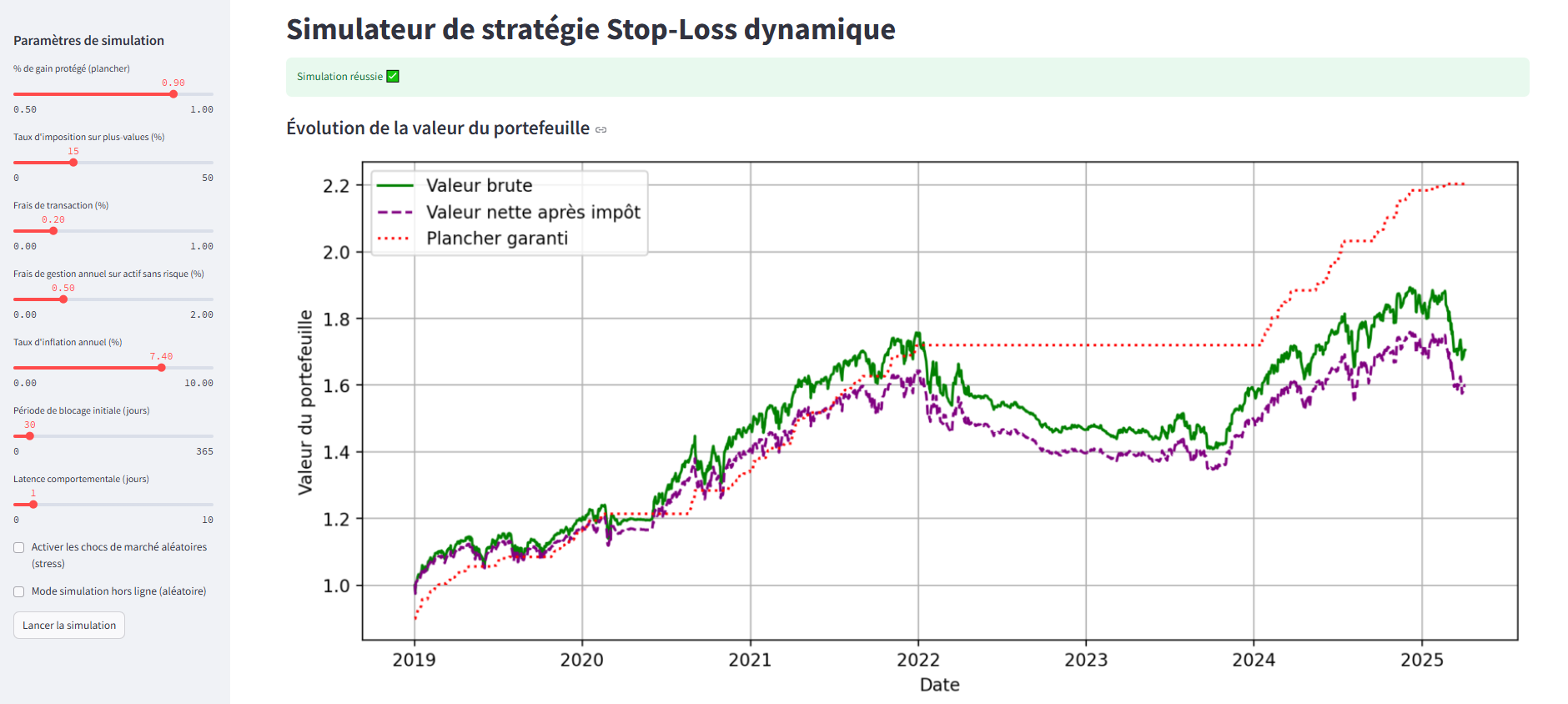
        except Exception as e:

            st.error(f"❌ Erreur pendant la simulation : {e}")

            st.info("💡 Essayez le mode simulation hors ligne si vous êtes sans connexion Internet.")

PS H:\Desktop\Rapport\_Programmation\_Finance> streamlit run simulateur\_stoploss.py

## Résultats du code Python pour simuler la stratégie Stop-Loss dynamiquement



## Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les frais de transactions

import yfinance as yf

import numpy as np

import matplotlib.pyplot as plt

# Téléchargement des données financières des actifs risqué (S&P500) et sans risque (VBISX)

vbisx = yf.download("VBISX", start="2019-01-01")

sp500 = yf.download("^GSPC", start="2019-01-01")

# Construction d'un DataFrame contenant les prix de clôture des deux actifs

data = vbisx[['Close']].rename(columns={'Close': 'VBISX'}).join(

    sp500[['Close']].rename(columns={'Close': 'S&P500'}),

    how='inner'

)

# Extraction des prix en tant que tableaux NumPy

prix\_risque = data["S&P500"].values

prix\_sans\_risque = data["VBISX"].values

dates = data.index

# Calcul des rendements relatifs (normalisés à 1 au départ)

rendement\_risque = prix\_risque / prix\_risque[0]

rendement\_sans\_risque = prix\_sans\_risque / prix\_sans\_risque[0]

# Définition de la classe de stratégie Stop-Loss avec frais de transaction

class StrategieStopLossGainAvecFrais:

    def \_\_init\_\_(self, rendement\_risque, rendement\_sans\_risque, dates, gain\_protege\_pct, frais\_transaction):

        self.rendement\_risque = rendement\_risque

        self.rendement\_sans\_risque = rendement\_sans\_risque

        self.dates = dates

        self.N = len(rendement\_risque)

        self.gain\_protege\_pct = gain\_protege\_pct

        self.frais\_transaction = frais\_transaction  # Pourcentage des frais lors des transitions

        # Initialisation des vecteurs pour le plancher, la valeur du portefeuille théorique et réelle

        self.plancher = np.zeros(self.N)

        self.valeur\_portefeuille\_theorique = np.ones(self.N)

        self.valeur\_portefeuille\_capitalisee = np.ones(self.N)

        self.positions = ["risque"] \* self.N  # Historique des positions : "risque" ou "sans\_risque"

    def appliquer\_strategie(self):

        """

        Applique la stratégie dynamique avec stop-loss et prise en compte des frais de transaction

        """

        max\_rendement = self.rendement\_risque[0]  # Rendement max initial (valeur de référence)

        self.plancher[0] = self.gain\_protege\_pct \* max\_rendement  # Plancher initial basé sur ce rendement

        dans\_actif\_risque = True  # Début dans l'actif risqué

        for t in range(1, self.N):

            # Mise à jour du rendement maximum atteint et du plancher correspondant

            max\_rendement = max(max\_rendement, self.rendement\_risque[t])

            self.plancher[t] = self.gain\_protege\_pct \* max\_rendement

            # Valeurs de référence pour le rendement précédent (évite division par zéro)

            r\_risk\_prev = self.rendement\_risque[t - 1] if self.rendement\_risque[t - 1] != 0 else 1

            r\_safe\_prev = self.rendement\_sans\_risque[t - 1] if self.rendement\_sans\_risque[t - 1] != 0 else 1

            # Mise à jour de la valeur théorique selon la position actuelle

            if dans\_actif\_risque:

                self.valeur\_portefeuille\_theorique[t] = self.rendement\_risque[t]

            else:

                self.valeur\_portefeuille\_theorique[t] = self.rendement\_sans\_risque[t]

            # Mise à jour de la valeur capitalisée avec frais selon la transition

            if dans\_actif\_risque:

                if self.rendement\_risque[t] < self.plancher[t]:

                    # Basculer vers l'actif sans risque avec frais de transaction

                    dans\_actif\_risque = False

                    self.positions[t:] = ["sans\_risque"] \* (self.N - t)

                    croissance = self.rendement\_sans\_risque[t] / r\_safe\_prev \* (1 - self.frais\_transaction)

                else:

                    croissance = self.rendement\_risque[t] / r\_risk\_prev

            else:

                if self.rendement\_risque[t] > self.plancher[t]:

                    # Retour à l'actif risqué avec frais de transaction

                    dans\_actif\_risque = True

                    self.positions[t:] = ["risque"] \* (self.N - t)

                    croissance = self.rendement\_risque[t] / r\_risk\_prev \* (1 - self.frais\_transaction)

                else:

                    croissance = self.rendement\_sans\_risque[t] / r\_safe\_prev

            # Mise à jour finale de la valeur du portefeuille

            self.valeur\_portefeuille\_capitalisee[t] = self.valeur\_portefeuille\_capitalisee[t - 1] \* croissance

    def afficher\_allocation(self, label\_frais):

        """

        Affiche la courbe de valeur du portefeuille pour une valeur donnée de frais de transaction.

        """

        plt.plot(self.dates, self.valeur\_portefeuille\_capitalisee, label=f"Valeur portefeuille (frais={label\_frais}%)")

# Liste des taux de frais de transaction à tester (de 0.1% à 3%)

frais\_de\_transaction\_list = [0.001, 0.005, 0.01, 0.015, 0.018, 0.02, 0.022, 0.025, 0.03]

# Initialisation de la figure matplotlib

plt.figure(figsize=(14, 6))

# Boucle sur chaque niveau de frais pour tester la robustesse de la stratégie

for frais in frais\_de\_transaction\_list:

    strategie = StrategieStopLossGainAvecFrais(rendement\_risque, rendement\_sans\_risque, dates, gain\_protege\_pct=0.9, frais\_transaction=frais)

    strategie.appliquer\_strategie()

    strategie.afficher\_allocation(label\_frais=frais \* 100)  # Conversion du taux en pourcentage pour le label

# Tracé du plancher de gain (en rouge pointillé)

plt.plot(dates, strategie.plancher, label="Plancher de gain", linestyle=':', color='red')

# Personnalisation du graphique

plt.title("Valeur du Portefeuille avec Frais de Transaction (Comparaison)", fontsize=16)

plt.xlabel("Date", fontsize=14)

plt.ylabel("Valeur du portefeuille", fontsize=14)

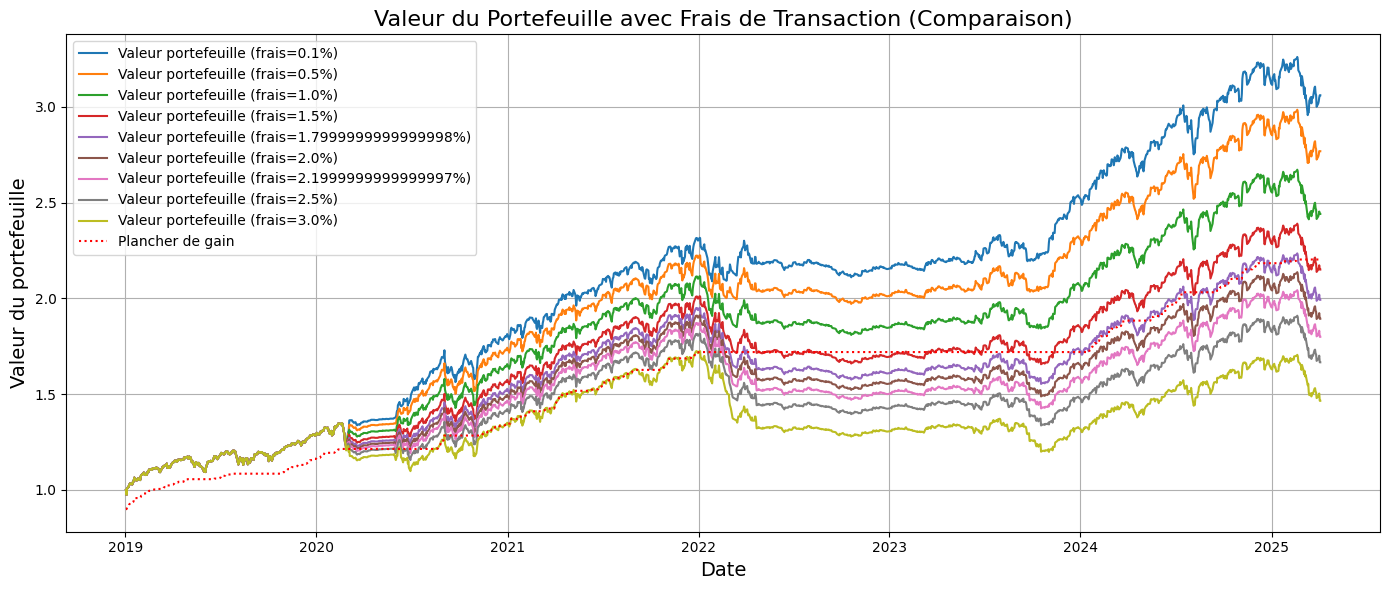
plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

## Résultat du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les frais de transactions



## Interprétation du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les frais de transactions

Afin d’évaluer la robustesse de la stratégie Stop-Loss dynamique face aux coûts d’exécution, nous avons simulé l’évolution du portefeuille en intégrant différents niveaux de frais de transaction, allant de 0.1 % à 3 %. Le graphique obtenu met en évidence une dégradation progressive de la performance à mesure que les frais augmentent. Les courbes montrent clairement que si la stratégie reste performante avec des frais modérés (inférieurs ou égaux à 1 %), son efficacité diminue fortement au-delà de ce seuil. L’effet cumulatif des allers-retours entre actif risqué et sans risque, inhérent à une gestion dynamique, accentue cette érosion. Néanmoins, on constate que même en présence de frais importants, la stratégie remplit son rôle de protection, puisque la valeur du portefeuille reste systématiquement au-dessus du plancher de gain garanti. Ces résultats soulignent l’importance d’un cadre d’investissement à faibles coûts de transaction pour assurer la pertinence de ce type de stratégie dans la durée.

## Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte le slippage

import yfinance as yf

import numpy as np

import matplotlib.pyplot as plt

# Téléchargement des données financières des actifs risqué et sans risque

vbisx = yf.download("VBISX", start="2019-01-01")

sp500 = yf.download("^GSPC", start="2019-01-01")

# Fusion des données des deux actifs sur la même période

data = vbisx[['Close']].rename(columns={'Close': 'VBISX'}).join(

    sp500[['Close']].rename(columns={'Close': 'S&P500'}),

    how='inner'  # Jointure interne pour ne garder que les dates communes

)

# Extraction des prix des actifs risqué et sans risque

prix\_risque = data["S&P500"].values  # Valeur de l'actif risqué (S&P 500)

prix\_sans\_risque = data["VBISX"].values  # Valeur de l'actif sans risque (VBISX)

dates = data.index  # Les dates de l'investissement

# Calcul des rendements relatifs des actifs par rapport à leur valeur initiale

rendement\_risque = prix\_risque / prix\_risque[0]  # Rendement de l'actif risqué (S&P 500)

rendement\_sans\_risque = prix\_sans\_risque / prix\_sans\_risque[0]  # Rendement de l'actif sans risque (VBISX)

class StrategieStopLossGainAvecSlippage:

    """

    Classe représentant la stratégie de stop-loss avec slippage pour garantir un certain pourcentage du gain.

    """

    def \_\_init\_\_(self, rendement\_risque, rendement\_sans\_risque, dates, gain\_protege\_pct, frais\_transaction, slippage\_pct):

        """

        Initialisation des paramètres de la stratégie.

        Paramètres :

        - rendement\_risque : Array des rendements relatifs de l'actif risqué (S&P 500)

        - rendement\_sans\_risque : Array des rendements relatifs de l'actif sans risque (VBISX)

        - dates : Index des dates pour l'évolution des rendements

        - gain\_protege\_pct : Pourcentage du gain à garantir à la fin de l'horizon de l'investissement

        - frais\_transaction : Frais de transaction (en pourcentage)

        - slippage\_pct : Pourcentage de slippage (écart entre le prix prévu et le prix réel)

        """

        self.rendement\_risque = rendement\_risque

        self.rendement\_sans\_risque = rendement\_sans\_risque

        self.dates = dates

        self.N = len(rendement\_risque)  # Nombre de périodes (jours)

        self.gain\_protege\_pct = gain\_protege\_pct  # Pourcentage à garantir du placement initial

        self.frais\_transaction = frais\_transaction  # Frais de transaction

        self.slippage\_pct = slippage\_pct  # Pourcentage de slippage

        # Initialisation des variables pour suivre la valeur du portefeuille et du plancher de gain

        self.plancher = np.zeros(self.N)

        self.valeur\_portefeuille\_theorique = np.ones(self.N)

        self.valeur\_portefeuille\_capitalisee = np.ones(self.N)

        self.positions = ["risque"] \* self.N  # Liste de positions (risque ou sans risque)

    def appliquer\_slippage(self, prix):

        """

        Applique un slippage aléatoire sur le prix d'achat ou de vente.

        :param prix: Prix de l'actif avant slippage

        :return: Prix ajusté

        """

        return prix \* (1 + np.random.uniform(-self.slippage\_pct, self.slippage\_pct))

    def appliquer\_strategie(self):

        """

        Applique la stratégie de stop-loss dynamique avec slippage en fonction des rendements des actifs.

        """

        max\_rendement = self.rendement\_risque[0]

        self.plancher[0] = self.gain\_protege\_pct \* max\_rendement  # Le plancher est fixé à un pourcentage du rendement initial

        dans\_actif\_risque = True  # On commence en investissant dans l'actif risqué

        for t in range(1, self.N):

            max\_rendement = max(max\_rendement, self.rendement\_risque[t])

            self.plancher[t] = self.gain\_protege\_pct \* max\_rendement  # Mise à jour du plancher de gain

            r\_risk\_prev = self.rendement\_risque[t - 1] if self.rendement\_risque[t - 1] != 0 else 1

            r\_safe\_prev = self.rendement\_sans\_risque[t - 1] if self.rendement\_sans\_risque[t - 1] != 0 else 1

            # === Mise à jour de la valeur théorique du portefeuille en fonction de la position actuelle ===

            if dans\_actif\_risque:

                self.valeur\_portefeuille\_theorique[t] = self.rendement\_risque[t]

            else:

                self.valeur\_portefeuille\_theorique[t] = self.rendement\_sans\_risque[t]

            # === Mise à jour de la valeur capitalisée ===

            if dans\_actif\_risque:

                if self.rendement\_risque[t] < self.plancher[t]:

                    # Transition vers l'actif sans risque, en appliquant le slippage

                    dans\_actif\_risque = False

                    self.positions[t:] = ["sans\_risque"] \* (self.N - t)

                    croissance = self.appliquer\_slippage(self.rendement\_sans\_risque[t] / r\_safe\_prev)

                else:

                    croissance = self.rendement\_risque[t] / r\_risk\_prev

            else:

                if self.rendement\_risque[t] > self.plancher[t]:

                    # Retour à l'actif risqué, en appliquant le slippage

                    dans\_actif\_risque = True

                    self.positions[t:] = ["risque"] \* (self.N - t)

                    croissance = self.appliquer\_slippage(self.rendement\_risque[t] / r\_risk\_prev)

                else:

                    croissance = self.rendement\_sans\_risque[t] / r\_safe\_prev

            # Mise à jour de la valeur capitalisée du portefeuille en fonction de la croissance ajustée pour slippage et frais

            self.valeur\_portefeuille\_capitalisee[t] = self.valeur\_portefeuille\_capitalisee[t - 1] \* croissance

    def afficher\_allocation(self, label\_slippage):

        """

        Affiche la valeur du portefeuille avec différents slippages et la comparaison avec le plancher de gain.

        """

        # Affichage de la valeur du portefeuille au fil du temps

        plt.plot(self.dates, self.valeur\_portefeuille\_capitalisee, label=f"Valeur portefeuille (slippage={label\_slippage}%)")

# Liste des niveaux de slippage à tester (par exemple 0.5%, 1%, 1.5%, 2%, 2.5%, 3%)

slippage\_list = [0.005, 0.01, 0.015, 0.02, 0.025, 0.03]  # Slippage à tester

plt.figure(figsize=(14, 6))

# Appliquer la stratégie pour chaque niveau de slippage

for slippage in slippage\_list:

    strategie = StrategieStopLossGainAvecSlippage(rendement\_risque, rendement\_sans\_risque, dates, gain\_protege\_pct=0.9, frais\_transaction=0.001, slippage\_pct=slippage)

    strategie.appliquer\_strategie()

    strategie.afficher\_allocation(label\_slippage=slippage \* 100)  # Affiche la courbe avec le label du slippage en %

# Affichage du plancher de gain uniquement une fois

plt.plot(dates, strategie.plancher, label="Plancher de gain", linestyle=':', color='red')

# Affichage des détails du graphique

plt.title("Valeur du Portefeuille avec Slippage (Comparaison)", fontsize=16)

plt.xlabel("Date", fontsize=14)

plt.ylabel("Valeur du portefeuille", fontsize=14)

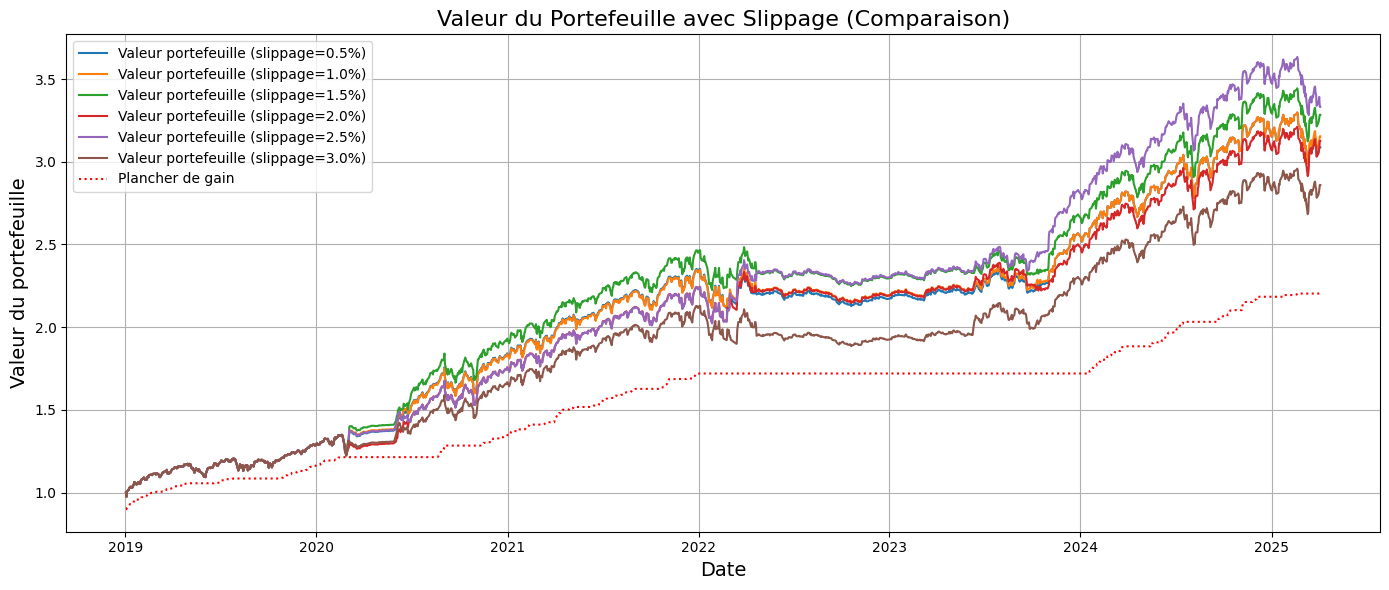
plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

## Résultat du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte le slippage



## Interprétation du code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte le slippage

Nous avons également évalué l’impact du slippage sur la performance de la stratégie Stop-Loss dynamique. Le slippage, représentant la perte liée à un écart entre le prix souhaité et le prix réellement exécuté lors d’un changement d’allocation, constitue un coût implicite souvent négligé. Les résultats montrent une dégradation progressive de la valeur finale du portefeuille à mesure que le taux de slippage augmente, avec un effet particulièrement marqué au-delà de 2 %. Si la stratégie reste globalement performante pour des niveaux de slippage modérés (jusqu’à 1.5 %), les allers-retours répétés entre actif risqué et sans risque amplifient l’érosion de performance lorsque les coûts implicites deviennent significatifs. Toutefois, il est important de noter que le plancher de gain reste respecté dans tous les scénarios, démontrant la robustesse de la stratégie en termes de protection du capital. Ces résultats soulignent l’importance de minimiser le slippage (via des instruments liquides ou des exécutions optimisées) pour préserver la performance globale d’une stratégie de gestion dynamique.

## Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les taxes sur les gains en capital

import yfinance as yf

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# ==== Chargement des données ====

# Téléchargement des données de l'actif sans risque (VBISX) et de l'actif risqué (S&P 500) depuis 2019

vbisx = yf.download("VBISX", start="2019-01-01")

sp500 = yf.download("^GSPC", start="2019-01-01")

# Construction d'un DataFrame combiné contenant les prix de clôture des deux actifs

data = vbisx[['Close']].rename(columns={'Close': 'VBISX'}).join(

    sp500[['Close']].rename(columns={'Close': 'S&P500'}),

    how='inner'

)

# Extraction des prix sous forme de tableaux NumPy

prix\_risque = data["S&P500"].values

prix\_sans\_risque = data["VBISX"].values

dates = data.index  # Index temporel

# Calcul des rendements relatifs normalisés (base 1)

rendement\_risque = prix\_risque / prix\_risque[0]

rendement\_sans\_risque = prix\_sans\_risque / prix\_sans\_risque[0]

# ==== Classe avec fiscalité ====

class StrategieStopLossGainFiscale:

    def \_\_init\_\_(self, rendement\_risque, rendement\_sans\_risque, dates, gain\_protege\_pct, taux\_imposition):

        self.rendement\_risque = rendement\_risque

        self.rendement\_sans\_risque = rendement\_sans\_risque

        self.dates = dates

        self.N = len(rendement\_risque)

        self.gain\_protege\_pct = gain\_protege\_pct  # % du gain maximum à protéger

        self.taux\_imposition = taux\_imposition  # Taux d'imposition sur le gain

        # Initialisation des vecteurs : plancher, valeur brute et nette du portefeuille, position

        self.plancher = np.zeros(self.N)

        self.valeur\_portefeuille\_capitalisee = np.ones(self.N)

        self.valeur\_nette\_apres\_impot = np.ones(self.N)

        self.positions = ["risque"] \* self.N  # Historique des positions ("risque" ou "sans\_risque")

    def appliquer\_strategie(self):

        # Initialisation : rendement maximum atteint à t=0

        max\_rendement = self.rendement\_risque[0]

        self.plancher[0] = self.gain\_protege\_pct \* max\_rendement

        dans\_actif\_risque = True  # Position initiale : actif risqué

        for t in range(1, self.N):

            # Mise à jour du rendement max et du plancher de protection

            max\_rendement = max(max\_rendement, self.rendement\_risque[t])

            self.plancher[t] = self.gain\_protege\_pct \* max\_rendement

            # Sécurité pour éviter division par zéro

            r\_risk\_prev = self.rendement\_risque[t - 1] if self.rendement\_risque[t - 1] != 0 else 1

            r\_safe\_prev = self.rendement\_sans\_risque[t - 1] if self.rendement\_sans\_risque[t - 1] != 0 else 1

            # Mise à jour de la position selon la stratégie de stop-loss

            if dans\_actif\_risque:

                if self.rendement\_risque[t] < self.plancher[t]:

                    # Bascule vers actif sans risque

                    dans\_actif\_risque = False

                    self.positions[t:] = ["sans\_risque"] \* (self.N - t)

                    croissance = self.rendement\_sans\_risque[t] / r\_safe\_prev

                else:

                    croissance = self.rendement\_risque[t] / r\_risk\_prev

            else:

                if self.rendement\_risque[t] > self.plancher[t]:

                    # Retour à l'actif risqué

                    dans\_actif\_risque = True

                    self.positions[t:] = ["risque"] \* (self.N - t)

                    croissance = self.rendement\_risque[t] / r\_risk\_prev

                else:

                    croissance = self.rendement\_sans\_risque[t] / r\_safe\_prev

            # Mise à jour de la valeur brute du portefeuille (avant impôt)

            self.valeur\_portefeuille\_capitalisee[t] = self.valeur\_portefeuille\_capitalisee[t - 1] \* croissance

            # Calcul de l'impôt et de la valeur nette après imposition

            gain\_brut = self.valeur\_portefeuille\_capitalisee[t] - 1

            impot = self.taux\_imposition \* gain\_brut if gain\_brut > 0 else 0

            self.valeur\_nette\_apres\_impot[t] = self.valeur\_portefeuille\_capitalisee[t] - impot

    def get\_resultats\_finaux(self):

        # Résumé des indicateurs clés à la fin de la stratégie

        return {

            "Taux d'imposition": f"{int(self.taux\_imposition \* 100)}%",

            "Valeur finale brute": round(self.valeur\_portefeuille\_capitalisee[-1], 3),

            "Valeur finale nette": round(self.valeur\_nette\_apres\_impot[-1], 3),

            "Plancher garanti": round(self.plancher[-1], 3),

            "Gain brut": round(self.valeur\_portefeuille\_capitalisee[-1] - 1, 3),

            "Impôt payé": round(max(0, (self.valeur\_portefeuille\_capitalisee[-1] - 1) \* self.taux\_imposition), 3),

            "Surperformance nette vs plancher": round(self.valeur\_nette\_apres\_impot[-1] - self.plancher[-1], 3)

        }

# ==== Taux d'imposition à tester ====

# Liste des taux d'imposition simulés : 0%, 15%, 30%, 45%

taux\_imposition\_liste = [0.0, 0.15, 0.30, 0.45]

couleurs = ["green", "blue", "orange", "red"]

resultats = []  # Liste pour stocker les résultats

# === Affichage graphique ===

plt.figure(figsize=(14, 6))

# Boucle sur chaque taux d'imposition

for taux, couleur in zip(taux\_imposition\_liste, couleurs):

    # Création et application de la stratégie avec fiscalité

    strat = StrategieStopLossGainFiscale(

        rendement\_risque, rendement\_sans\_risque, dates,

        gain\_protege\_pct=0.9,

        taux\_imposition=taux

    )

    strat.appliquer\_strategie()

    res = strat.get\_resultats\_finaux()

    resultats.append(res)  # Stockage des résultats

    # Tracé de la courbe de valeur nette (après impôt)

    plt.plot(dates, strat.valeur\_nette\_apres\_impot, label=f"{int(taux \* 100)}% impôt", color=couleur)

# === Tracé du plancher de protection ===

plt.plot(dates, strat.plancher, label="Plancher garanti", linestyle=":", color="black")

# === Personnalisation du graphique ===

plt.title("Comparaison des valeurs finales avec différents taux d'imposition")

plt.xlabel("Date")

plt.ylabel("Valeur nette du portefeuille")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

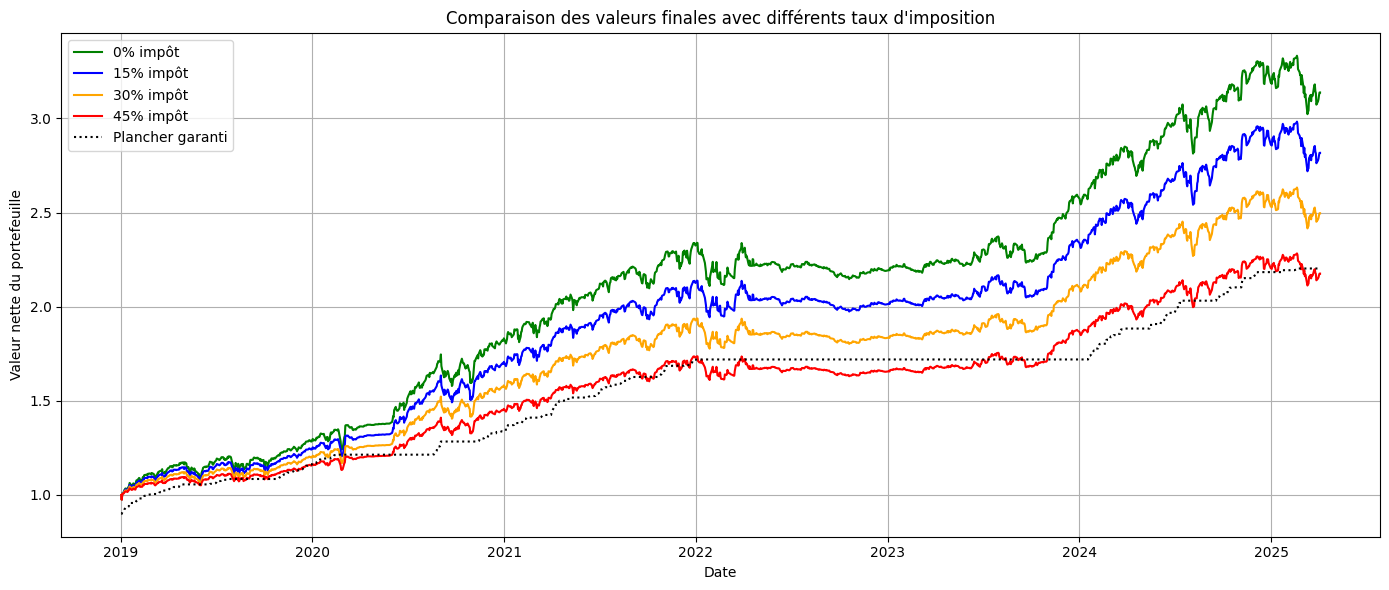
# === Résumé tabulaire des résultats ===

df\_resultats = pd.DataFrame(resultats)

print("\n=== Résumé Comparatif Fiscalité ===")

print(df\_resultats.to\_string(index=False))

## Résultat du Code Python pour appliquer la stratégie d’allocations d’actifs en prenant en compte les taxes sur les gains en capital



## Quelle(s) limite(s) relevez-vous lors de l’application de cette stratégie ?

### Frais de transaction élevés

L'une des principales limites de cette stratégie réside dans les frais de transaction générés par les allers-retours fréquents entre l'actif risqué et l'actif sans risque. Chaque fois que la valeur de l'actif risqué franchit le montant garanti M(t), l'investisseur doit vendre un actif et en acheter un autre. Ces transactions répétées peuvent entraîner des coûts importants, surtout si les actifs concernés ont des spreads élevés ou des frais de gestion associés.

### Sensibilité à la volatilité

La stratégie est très sensible à la volatilité de l'actif risqué. Si l'actif est extrêmement volatil, les changements d'allocation peuvent devenir très fréquents, ce qui rend la stratégie inefficace et coûteuse. Par exemple, dans un marché très agité, l'actif risqué peut osciller autour du montant garanti, provoquant des transactions inutiles et augmentant les frais.

### Garantie partielle

La stratégie ne garantit qu'un pourcentage partiel de la valeur initiale du portefeuille (par exemple, 90 %). Cela signifie que l'investisseur reste exposé à une perte potentielle sur les 10 % restants. Dans un scénario de crise où l'actif risqué chute brutalement, la garantie peut ne pas suffire à protéger l'investisseur.

### Dépendance au taux sans risque

Le montant garanti M(t) est calculé en fonction du taux sans risque. Si ce taux est très bas (comme dans un environnement de taux zéro ou négatifs), la valeur actualisée du montant garanti sera faible, réduisant ainsi l'efficacité de la stratégie. De plus, les rendements de l'actif sans risque peuvent ne pas suffire à compenser l'inflation, ce qui érode le pouvoir d'achat de l'investisseur.

### Complexité de mise en œuvre

La stratégie nécessite une surveillance constante du marché et une réallocation rapide des actifs. Cela peut être difficile à mettre en œuvre pour un investisseur particulier, surtout s'il n'a pas accès à des outils de trading automatisés ou à des plateformes sophistiquées.

## Auriez-vous des solutions ou conseils à proposer à un investisseur ?

### Réduire les frais de transaction

Pour limiter l'impact des frais de transaction, l'investisseur peut :

* Utiliser des ETF à faible coût pour représenter l'actif risqué et l'actif sans risque.
* Opter pour des brokers à faibles commissions ou des plateformes offrant des frais réduits pour les transactions fréquentes.
* Limiter le nombre de réallocations en ajustant les seuils de déclenchement (par exemple, en utilisant une bande de tolérance autour du montant garanti).

### Adapter la stratégie à la volatilité

Pour atténuer l'impact de la volatilité, l'investisseur peut :

* Choisir des actifs moins volatils pour l'actif risqué (par exemple, des actions de grandes entreprises stables plutôt que des actions de croissance).
* Lisser les réallocations en utilisant des moyennes mobiles ou d'autres indicateurs techniques pour éviter les réactions excessives aux fluctuations à court terme.

### Renforcer la garantie

Pour améliorer la protection du portefeuille, l'investisseur peut :

* Augmenter le pourcentage garanti (par exemple, passer de 90 % à 95 %) pour réduire l'exposition aux pertes.
* Utiliser des produits dérivés (comme des options de vente) pour couvrir le portefeuille contre les baisses importantes du marché.

### Diversifier le portefeuille

La diversification est une clé pour réduire les risques. L'investisseur peut :

* Inclure d'autres classes d'actifs (comme l'immobilier, les matières premières ou les obligations) pour réduire la dépendance à l'actif risqué.
* Utiliser des fonds diversifiés ou des portefeuilles multi-actifs pour bénéficier d'une exposition équilibrée.

### Automatiser la stratégie

Pour simplifier la mise en œuvre, l'investisseur peut :

* Utiliser des algorithmes de trading ou des robots pour gérer les réallocations automatiquement.
* Recourir à des plateformes de gestion automatisée (robo-advisors) qui intègrent des stratégies de protection similaires.

### Adapter la stratégie au contexte économique

L'investisseur doit tenir compte du contexte économique et des taux d'intérêt pour ajuster la stratégie. Par exemple :

* Dans un environnement de taux bas, il peut être préférable de réduire l'exposition à l'actif sans risque et de privilégier des actifs offrant un meilleur rendement ajusté au risque.
* Dans un contexte de hausse des taux, l'actif sans risque devient plus attractif, et la stratégie peut être renforcée.

### Tester la stratégie sur des données historiques

Avant d'appliquer la stratégie en réel, l'investisseur doit la tester sur des données historiques pour évaluer sa performance dans différents scénarios de marché. Cela permet d'identifier les points faibles et d'ajuster les paramètres (comme le pourcentage garanti ou la fréquence de réallocation).

# Partie C

## Code python pour tracer sur le même graphique, la représentation de la densité pour le titre Michelin et la densité gaussienne

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.stats import norm, gaussian\_kde

# Charger les données du fichier CSV

data = pd.read\_csv('H:\\Desktop\\Rapport\_Programmation\_Finance\\Michelin\_20112023\_20112024.csv', sep=';', header=None)

data.columns = ['ISIN', 'Date', 'Opening', 'High', 'Low', 'Closing', 'Volume']

# Convertir 'Closing' en numérique, remplacer ',' par '.' pour la conversion en float

data['Closing'] = data['Closing'].str.replace(',', '.').astype(float)

# Calcul des taux de rentabilité log (logarithmiques) entre les jours consécutifs

data['Returns'] = np.log(data['Closing'] / data['Closing'].shift(1))

# Nettoyage des valeurs aberrantes et NaN (par exemple, infinies ou NaN)

data = data.replace([np.inf, -np.inf], np.nan).dropna(subset=['Returns'])

# Trier les taux de rentabilité en ordre croissant

sorted\_returns = np.sort(data['Returns'])

# Estimation de la densité par noyau gaussien (KDE) en utilisant la méthode Silverman pour le "bandwidth"

kde = gaussian\_kde(sorted\_returns, bw\_method='silverman')

density\_estimate = kde(sorted\_returns)

# Densité gaussienne théorique pour comparaison (moyenne et écart-type des rendements observés)

gaussian\_density = norm.pdf(sorted\_returns, loc=np.mean(sorted\_returns), scale=np.std(sorted\_returns))

# Tracer les courbes de densité

plt.figure(figsize=(10, 6))

plt.plot(sorted\_returns, density\_estimate, label='Densité estimée (KDE)', color='blue', linewidth=1.5)

plt.plot(sorted\_returns, gaussian\_density, label='Densité gaussienne', color='red', linestyle='--', linewidth=1.5)

# Ajouter des titres, légendes et autres éléments de formatage

plt.title('Comparaison de la densité des taux de rentabilité de Michelin avec une densité gaussienne', fontsize=16)

plt.xlabel('Taux de rentabilité', fontsize=14)

plt.ylabel('Densité', fontsize=14)

plt.legend(loc='upper left')

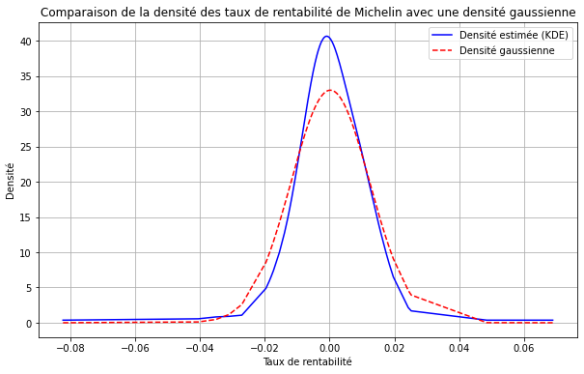
plt.grid(True)

# Afficher le graphique

plt.tight\_layout()

plt.show()

## Résultat du code python pour tracer sur le même graphique, la représentation de la densité pour le titre Michelin et la densité gaussienne



## Interprétation du code python pour tracer sur le même graphique, la représentation de la densité pour le titre Michelin et la densité gaussienne

L’analyse de la distribution empirique des rendements journaliers de l’action Michelin met en évidence des écarts significatifs par rapport à la loi normale théorique. La densité estimée par noyau (KDE) montre une courbe plus resserrée autour de la moyenne, mais avec des queues plus épaisses, traduisant une kurtosis positive. Ce phénomène, typique des actifs financiers, indique une probabilité accrue d'observer des variations extrêmes, aussi bien à la hausse qu’à la baisse. La distribution reste globalement symétrique, avec une asymétrie (skewness) proche de zéro. Ce résultat souligne l’importance de modéliser les rendements avec prudence, en allant au-delà de l’hypothèse gaussienne classique. Dans ce contexte, la stratégie dynamique avec stop-loss prend tout son sens, car elle est spécifiquement conçue pour se protéger des pertes importantes causées par ces mouvements de marché extrêmes, tout en captant les hausses modérées lorsque le marché se redresse.

## Code python pour étudier la sensibilité de la distribution de la valeur terminale en fonction de différents niveaux de paramètres

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import gaussian\_kde

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Simulateur d'actif (mouvement brownien géométrique)

class AssetSimulator:

    def \_\_init\_\_(self, S0, r, sigma, T, N):

        self.S0 = S0

        self.r = r

        self.sigma = sigma

        self.T = T

        self.N = N

        self.dt = T / N

    def simulate\_path(self):

        Z = np.random.normal(0, 1, self.N)

        log\_returns = (self.r - 0.5 \* self.sigma\*\*2) \* self.dt + self.sigma \* np.sqrt(self.dt) \* Z

        log\_price = np.log(self.S0) + np.cumsum(log\_returns)

        return np.exp(log\_price)

# Stratégie Stop-Loss

class StopLossStrategy:

    def \_\_init\_\_(self, S0, r, T, floor\_pct, freq\_rebalance):

        self.S0 = S0

        self.r = r

        self.T = T

        self.floor\_pct = floor\_pct

        self.freq\_rebalance = freq\_rebalance

    def apply(self, price\_path):

        N = len(price\_path)

        dt = self.T / N

        t = np.linspace(0, self.T, N)

        Mt = self.floor\_pct \* self.S0 \* np.exp(-self.r \* (self.T - t))

        position = 'risky'

        portfolio\_value = []

        for i in range(N):

            if position == 'risky':

                val = price\_path[i]

                if val < Mt[i]:

                    val = Mt[i]

                    position = 'safe'

            else:

                val = Mt[i]

                if price\_path[i] > Mt[i]:

                    val = price\_path[i]

                    position = 'risky'

            portfolio\_value.append(val)

        return portfolio\_value[-1]

# Analyse de sensibilité

def sensitivity\_analysis(param\_name, values, fixed\_params, n\_simulations=1000):

    param\_title = {

        "sigma": "volatilité",

        "floor\_pct": "plancher garanti",

        "freq\_rebalance": "tolerance",

        "N": "N"

    }.get(param\_name, param\_name)

    plt.figure(figsize=(10, 6))

    for val in values:

        terminal\_values = []

        for \_ in range(n\_simulations):

            S0 = fixed\_params["S0"]

            r = fixed\_params["r"]

            T = fixed\_params["T"]

            N = fixed\_params["N"]

            sigma = fixed\_params["sigma"]

            floor\_pct = fixed\_params["floor\_pct"]

            freq = fixed\_params["freq\_rebalance"]

            if param\_name == "sigma":

                sigma = val

            elif param\_name == "floor\_pct":

                floor\_pct = val

            elif param\_name == "freq\_rebalance":

                floor\_pct = 1 - val  # ici la tolérance est 1 - floor\_pct

            elif param\_name == "N":

                N = val

            simulator = AssetSimulator(S0, r, sigma, T, N)

            price\_path = simulator.simulate\_path()

            strategy = StopLossStrategy(S0, r, T, floor\_pct, freq)

            final\_val = strategy.apply(price\_path)

            terminal\_values.append(final\_val)

        sns.kdeplot(terminal\_values, label=f"{param\_title} = {val:.2f}", linewidth=1.5)

    plt.title(f"Impact de la variation de {param\_title} sur la distribution de la valeur finale")

    plt.xlabel("Valeur finale")

    plt.ylabel("Densité")

    plt.legend()

    plt.grid(True)

    plt.tight\_layout()

    plt.show()

# Paramètres fixes par défaut

fixed\_params = {

    "S0": 100,

    "r": 0.02,

    "T": 1,

    "sigma": 0.2,

    "floor\_pct": 0.9,

    "freq\_rebalance": 252,

    "N": 252

}

# 1. Volatilité

sensitivity\_analysis("sigma", [0.1, 0.2, 0.3, 0.4, 0.5], fixed\_params)

# 2. Plancher garanti

sensitivity\_analysis("floor\_pct", [0.9, 0.92, 0.94, 0.96, 0.98], fixed\_params)

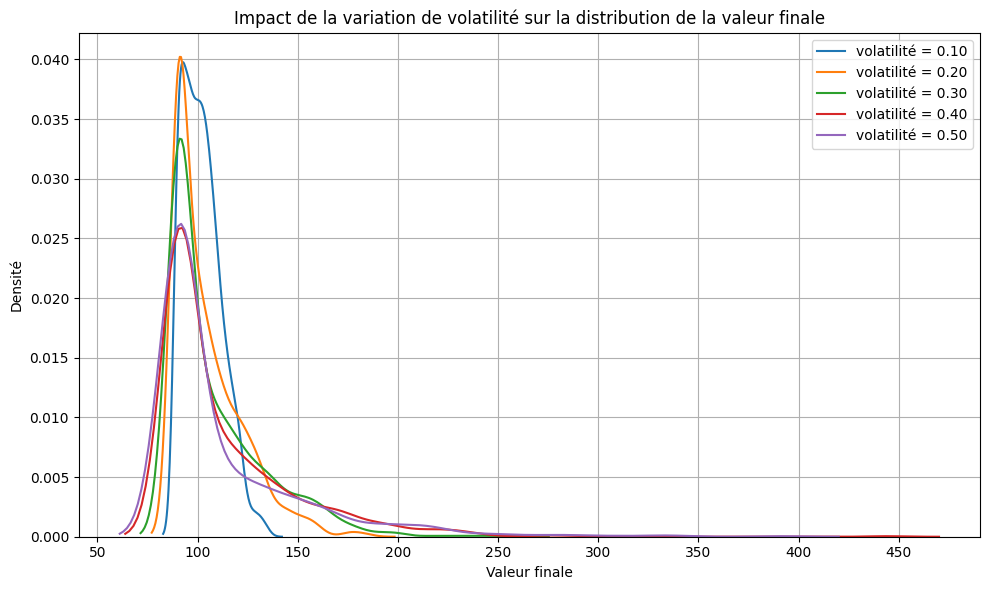
# 3. Tolérance (ici on teste floor\_pct = 1 - tolerance)

sensitivity\_analysis("freq\_rebalance", [0.05, 0.1, 0.15, 0.2, 0.25], fixed\_params)

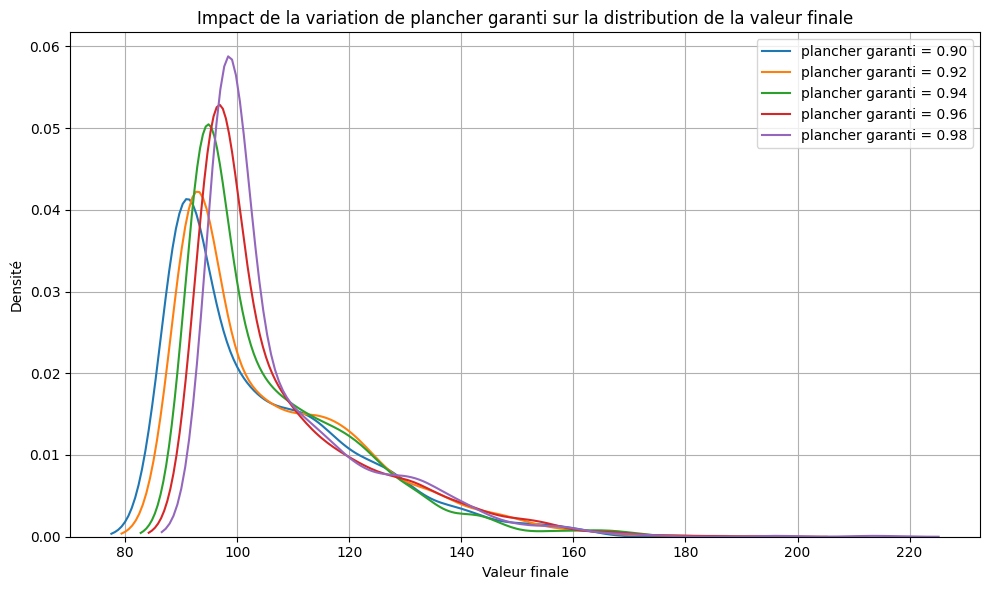
# 4. N (granularité de la simulation)

sensitivity\_analysis("N", [100, 200, 300, 400, 500], fixed\_params)

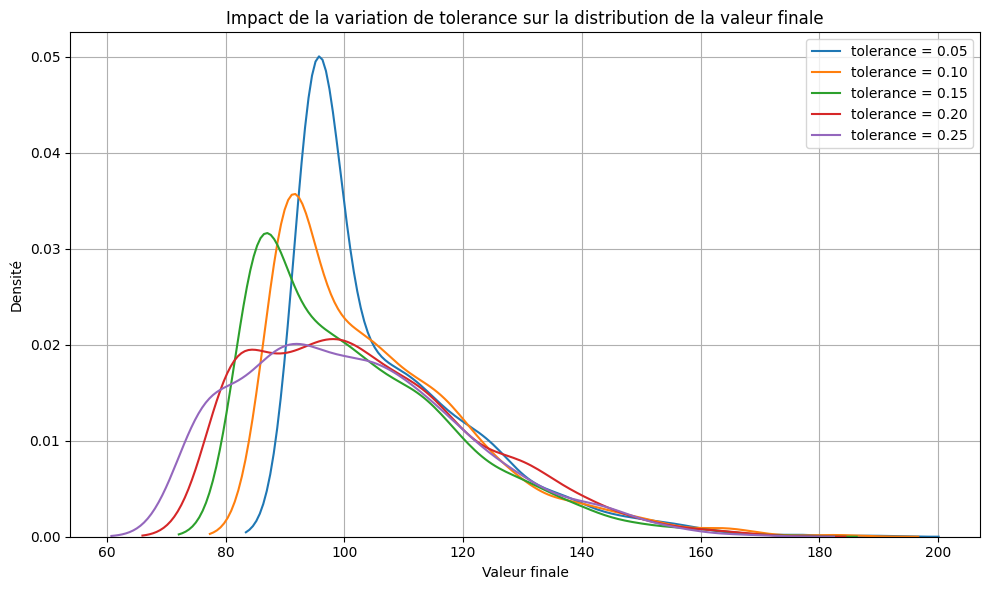
## Résultat du code python pour étudier la sensibilité de la distribution de la valeur terminale en fonction de différents niveaux de paramètres



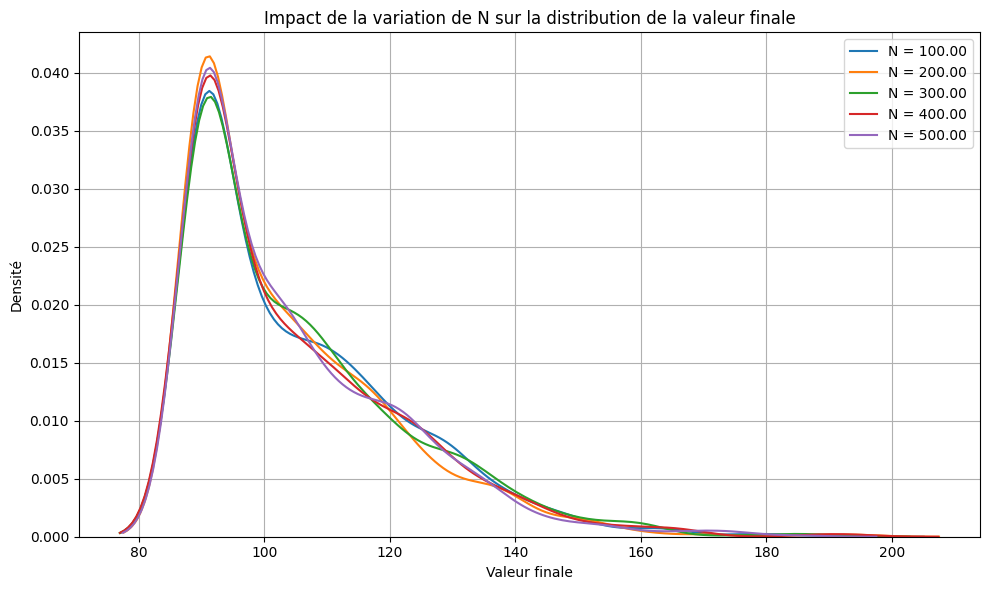
Le premier graphique met en évidence l’effet de la volatilité de l’actif risqué sur la distribution des valeurs finales du portefeuille. On observe que plus la volatilité augmente, plus la distribution s’élargit et s’étire vers la droite. Cela signifie que des niveaux de volatilité plus élevés augmentent la probabilité de performances exceptionnelles, mais également la dispersion des résultats. En d’autres termes, la stratégie conserve son potentiel de gain, mais au prix d’une incertitude accrue sur l’issue finale. À l’inverse, une faible volatilité réduit cette dispersion, mais limite aussi les perspectives de gain.



Le second graphique analyse l’impact du plancher garanti (exprimé en pourcentage du maximum observé) sur la distribution de la valeur finale. Il en ressort que plus ce niveau est élevé, plus la distribution se décale vers la gauche et devient concentrée autour de valeurs plus modestes. Cela traduit une stratégie plus conservatrice, où la protection du capital est renforcée, mais au détriment du rendement potentiel. À l’inverse, un plancher moins exigeant permet une meilleure participation à la performance des marchés, au prix d’un risque légèrement accru.



Le troisième graphique montre l’impact du niveau de tolérance (c’est-à-dire l’écart accepté avant activation du stop-loss). Une tolérance faible (par exemple 5 %) déclenche rapidement la protection, ce qui se traduit par une distribution resserrée autour de valeurs plus faibles : la stratégie agit de manière précoce, ce qui sécurise le capital mais empêche de profiter des rebonds potentiels. À l’inverse, une tolérance plus élevée permet à la stratégie de laisser courir les gains, entraînant une distribution plus étalée et une espérance de gain plus élevée, mais aussi plus de risque.



Enfin, le dernier graphique étudie l’effet de la fréquence de révision (nombre de pas dans la simulation) sur la valeur finale. Les courbes sont relativement proches, ce qui montre que la fréquence de révision a un impact limité sur la distribution finale, tant que celle-ci reste suffisamment élevée. Toutefois, on note une légère amélioration de la régularité des résultats pour les fréquences plus élevées, probablement liée à une meilleure réactivité de la stratégie. Cela confirme que, dans une certaine mesure, une fréquence de rebalancement raisonnable est suffisante pour conserver l’efficacité de la stratégie.

## Code python pour optimiser les différents paramètres

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from itertools import product

# Simulateur d'actif

class AssetSimulator:

    def \_\_init\_\_(self, S0, r, sigma, T, N):

        self.S0 = S0

        self.r = r

        self.sigma = sigma

        self.T = T

        self.N = N

        self.dt = T / N

    def simulate\_path(self):

        Z = np.random.normal(0, 1, self.N)

        log\_returns = (self.r - 0.5 \* self.sigma\*\*2) \* self.dt + self.sigma \* np.sqrt(self.dt) \* Z

        log\_price = np.log(self.S0) + np.cumsum(log\_returns)

        return np.exp(log\_price)

# Stratégie Stop-Loss dynamique avec fréquence de recomposition

class StopLossStrategy:

    def \_\_init\_\_(self, S0, r, T, tolerance, freq\_rebalance):

        self.S0 = S0

        self.r = r

        self.T = T

        self.tolerance = tolerance

        self.freq\_rebalance = freq\_rebalance

    def apply(self, price\_path):

        N = len(price\_path)

        t = np.linspace(0, self.T, N)

        floor = (1 - self.tolerance) \* self.S0 \* np.exp(-self.r \* (self.T - t))

        position = 'risky'

        values = []

        rebalance\_step = max(1, int(N / self.freq\_rebalance))

        for i in range(N):

            price = price\_path[i]

            if i % rebalance\_step == 0:

                if position == 'risky':

                    val = price

                    if val < floor[i]:

                        val = floor[i]

                        position = 'safe'

                else:

                    val = floor[i]

                    if price > floor[i]:

                        val = price

                        position = 'risky'

            else:

                val = price if position == 'risky' else floor[i]

            values.append(val)

        return values[-1]

# Calcul du ratio de Sharpe

def calculate\_sharpe(values):

    returns = np.array(values) / 100 - 1  # rendement simple par rapport à S0

    mean = np.mean(returns)

    std = np.std(returns)

    return mean / std if std > 0 else -np.inf

# Recherche des paramètres optimaux

def find\_best\_parameters():

    S0 = 100

    r = 0.02

    T = 1

    N = 252

    n\_simulations = 500

    # Grilles à tester

    sigmas = [0.1, 0.2]

    floor\_pcts = [0.9, 0.92, 0.94, 0.96, 0.98]

    tolerances = [0.05, 0.1, 0.15, 0.2]

    freq\_rebalances = [50, 100, 252]

    best\_sharpe = -np.inf

    best\_params = {}

    best\_values = []

    for sigma, floor\_pct, tolerance, freq\_rebalance in product(sigmas, floor\_pcts, tolerances, freq\_rebalances):

        final\_vals = []

        for \_ in range(n\_simulations):

            sim = AssetSimulator(S0, r, sigma, T, N)

            path = sim.simulate\_path()

            strategy = StopLossStrategy(S0, r, T, tolerance, freq\_rebalance)

            val = strategy.apply(path)

            final\_vals.append(val)

        sharpe = calculate\_sharpe(final\_vals)

        if sharpe > best\_sharpe:

            best\_sharpe = sharpe

            best\_params = {

                'volatilité': sigma,

                'plancher garanti': floor\_pct,

                'tolérance': tolerance,

                'fréquence de recomposition': freq\_rebalance,

                'sharpe': sharpe

            }

            best\_values = final\_vals

    return best\_params, best\_values

# Tracé de la distribution

def plot\_distribution(values, params):

    plt.figure(figsize=(12, 6))

    sns.kdeplot(values, fill=True, color="cornflowerblue", linewidth=1.5)

    plt.title("Distribution des valeurs finales avec les paramètres optimaux")

    plt.xlabel("Valeur finale du portefeuille")

    plt.ylabel("Densité de probabilité")

    plt.grid(True)

    plt.tight\_layout()

    plt.show()

    print("\n Paramètres optimaux trouvés :")

    for k, v in params.items():

        if k != "sharpe":

            print(f"– {k} = {v}")

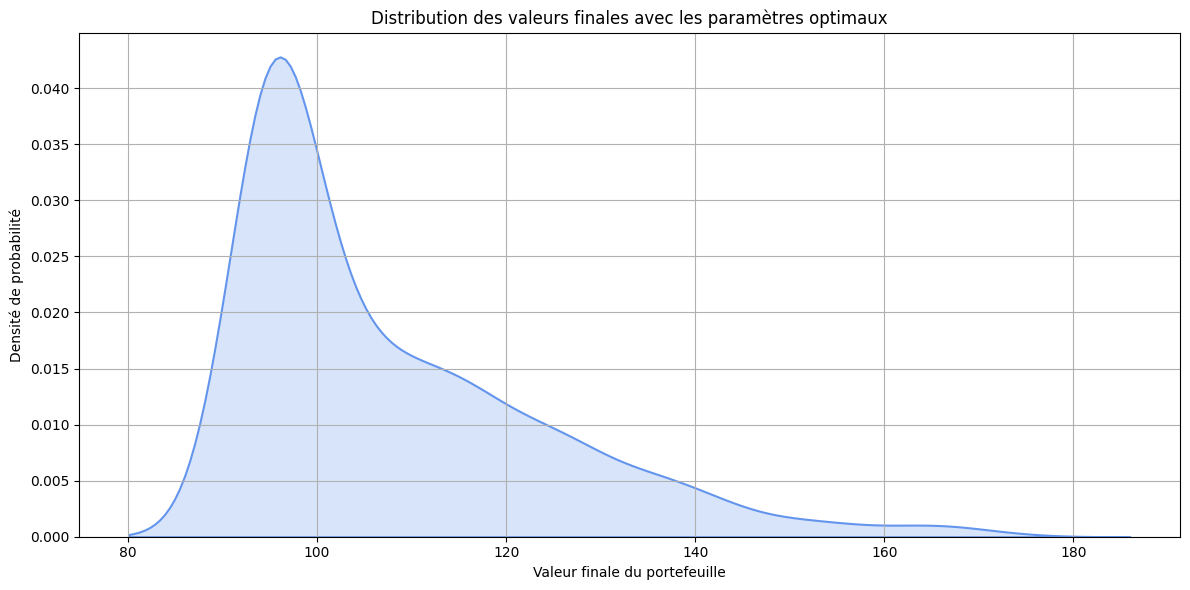
    print(f"– ratio de Sharpe = {params['sharpe']:.4f}")

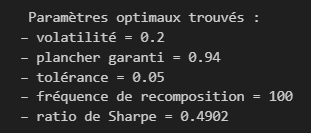
# Exécution globale

params, final\_values = find\_best\_parameters()

plot\_distribution(final\_values, params)

## Résultat du code python pour optimiser les différents paramètres





## Interprétation du code python pour optimiser les différents paramètres

Dans le but d’optimiser la stratégie de Stop-Loss dynamique, nous avons mené une analyse par simulations en recherchant les combinaisons de paramètres maximisant le ratio de Sharpe. Cette démarche nous a permis d’identifier un jeu de paramètres optimal composé d’une volatilité de 0,2, d’un plancher garanti à 94 % de la valeur actualisée du capital, d’une tolérance de 5 % et d’une fréquence de recomposition tous les 100 jours. La distribution des valeurs finales obtenue avec ces paramètres montre une forme asymétrique à droite, caractéristique d’une stratégie de type Stop-Loss qui limite les pertes tout en permettant aux gains de se développer. Le pic de densité autour de 95–100 indique une forte capacité de préservation du capital, tandis que la queue étirée vers la droite reflète le potentiel de performance lorsque le marché est favorable. Avec un ratio de Sharpe de 0,49, cette stratégie atteint un compromis efficace entre risque et rendement, validant la pertinence des choix retenus pour une mise en œuvre opérationnelle.

# Bibliographie

[1] Kaminski et Lo (2014) dans The Journal of Portfolio Management

<https://www.sciencedirect.com/science/article/abs/pii/S138641811300030X>

[2] Hull (2018) dans Options, Futures, and Other Derivatives

[3] Constantinides (1979) dans Journal of Financial Economics

<https://www.sciencedirect.com/science/article/abs/pii/0304405X80900227>

[4] Annaert et al. (2009) dans Financial Analysts Journal

<https://www.sciencedirect.com/science/article/abs/pii/S037842660800191X>

[5] Grossman et Zhou dans Journal of Economic Dynamics and Control

<https://ideas.repec.org/a/eee/stapro/v74y2005i3p245-252.html>