Documentation du Code : Optimisation et Simulation de Portefeuilles d'Actifs de Luxe

Auteurs : DOUBABI Mustapha, HADDAD Chirine

1. Introduction

Ce script Python permet d'optimiser un portefeuille d'actifs financiers en utilisant des données historiques sur l'évolution des actions de 10 entreprises du secteur du luxe.

2. Optimisation du Portefeuille

2.1. Calcul de l'Espérance de Rendement et de la Matrice de Covariance

L'espérance de rendement de chaque actif est calculée comme la moyenne des rendements quotidiens basés sur le prix moyen des actions (Moyenne du Maximum et Minimum du jour) :

```
ER = np.zeros((num_assets, 1))
for j in range(num_assets):
    bd = BD[j]
    St = [(bd.iloc[i, 3] + bd.iloc[i, 4]) / 2 for i in range(len(bd) - 1)]
    Rt = [(St[i + 1] / St[i]) - 1 for i in range(len(St) - 1)]
    ER[j] = np.mean(Rt)
```

La matrice de covariance SIGMA est ensuite calculée :

```
SIGMA = np.zeros((num_assets, num_assets))
for i in range(num_assets):
    Rti = np.array([(Sti[k + 1] / Sti[k]) - 1 for k in range(len(Sti) - 1)])
    for j in range(num_assets):
```

```
        Rtj = np.array([(Stj[k + 1] / Stj[k]) - 1 for k in range(len(Stj) - 1)])
        SIGMA[i, j] = np.cov(Rti, Rtj)[0, 1]
```

L'inverse pseudo-généralisée de la matrice SIGMA est ensuite calculée pour permettre l'optimisation du portefeuille.

3. Simulation et Sélection du Portefeuille Optimal

Un grand nombre de portefeuilles aléatoires sont générés afin de trouver celui qui maximise l'espérance de rendement pour un risque donné :

```
NS = 100000
sigmax = 0.8
espmax = -np.inf
xopt = None

for _ in range(NS):
    x = np.random.dirichlet(np.ones(num_assets))
    var = np.dot(x.T, np.dot(SIGMA, x))
    sigma = np.sqrt(var)
    if sigma < sigmax:
        esp = np.dot(x.T, ER)
        if esp > espmax:
            espmax = esp
            xopt = x
```

Ce portefeuille optimal maximise le rendement tout en respectant une contrainte de risque maximal.

4. Stratégie d'Investissement : Avec et Sans Coût de Transaction

Ces simulations permettent d'observer l'impact du coût de transaction sur la performance du portefeuille.

Sans coût de transaction :

```
for j in range(len(BD[0]) - 1):
    quant = [int(C[j] * xopt[i] / ((BD[i].iloc[j, 3] + BD[i].iloc[j, 4]) / 2)) for i in rang
```

```
    Cash.append(C[j] - sum([quant[i] * ((BD[i].iloc[j, 3] + BD[i].iloc[j, 4]) / 2) for
    C.append(float(Cash[j] + sum([quant[i] * ((BD[i].iloc[j + 1, 3] + BD[i].iloc[j +
```

Avec coût de transaction de 5 % :

```
for j in range(len(BD[0]) - 1):
    if j == 0:
        CT = 0
    else:
        CT = sum([(k * abs(quant_F[j - 1][i] - quant_F[j][i]) * ((BD[i].iloc[j + 1, 3] +
    Cash_F.append(C_F[j] - sum([quant_F[j][i] * ((BD[i].iloc[j, 3] + BD[i].iloc[j, 4
    C_F.append(float(Cash_F[j] + sum([quant_F[j][i] * ((BD[i].iloc[j + 1, 3] + BD[i
```

5. Résultats et Visualisation

- **Frontière efficiente des portefeuilles** : une courbe illustrant les portefeuilles optimaux en fonction du risque.
- **Évolution du portefeuille avec et sans coût de transaction.**

6. Outputs