Optimisation en finance

Optimisation de la CVaR et construction d'un index fund

OUAZZANI CHAHDI Nizar

Table des matières

1 Introduction

L'objectif de la première partie de ce travail et d'utiliser des modèles d'optimisation mathématiques programmées sur le langage **Python** pour déterminer la répartition des parts investis dans un nombre d'action afin de construire un portefeuille optimal. Il sera question d'utiliser la *Conditional Value at Risk* (CVaR) comme mesure de risque. Dans un premier temps, j'utiliserai le modèle d'optimisation de la CVaR afin de minimiser le risque lors de la constitution d'un portefeuille d'investissement. Puis dans un deuxième temps, j'utiliserai la CVaR comme mesure de risque pour optimiser le choix du portefeuille selon le modèle de Markowitz.

Dans la deuxième partie, il sera question de construire un *index fund* à partir d'un ensemble d'actions composant le marché. Dans un premier temps, j'utiliserai le modèle simplifié de la détermination des composantes de l'index. Puis dans un deuxième temps, j'utiliserai un modèle qui relaxe les contraintes selon la démarche de la relaxation Lagrangienne permettant d'avoir un temps de calcul moindre.

2 Démarche de réalisation du projet

D'abord, il m'a fallu comprendre le problème d'optimisation à résoudre (paramètres d'entrées, résultats attendus, interprétations). Pour ce faire, je me suis basé en grande partie sur les ressources bibliographiques [1] et [3] pour la première partie du projet (optimisation de la CVaR) puis des [2], [4] et [5] pour la deuxième partie (construction de l'index fund).

Puis, je suis passé à la lecture des modes d'emploi des librairies d'optimisation utilisées disponibles dans le langage **Python** (que j'ai choisi d'utiliser pour implémenter les solveurs étant donné qu'il n'y avait pas de contrainte sur le choix du langage dans l'énoncé du projet) afin de pouvoir implémenter de manière exacte les modèles mathématiques.

Plus particulièrement la manière d'implémenter les contraintes du problème et de faire la distinction entre problème résolu dans l'espace des réels (le cas du modèle de Markowitz) et problème résolu dans l'espace des entiers (le cas du modèle de construction de l'index fund) puisque dans ce cas les solutions sont entières et donc nécessiterait l'utilisation d'un programme en nombre entier.

Cette différence est importance pour le choix des solveurs à utiliser puisque la résolution de programme en nombres entiers n'est disponible que dans un nombre limité de solveurs.

Afin de résoudre les différents problèmes d'optimisation du projet j'ai utilisé plusieurs modules d'optimisation disponibles sur Python :

- Le module optimize de la librarie **scipy** de *Python* (librairie de calcul scientifique utilisant des implémentations de structures de données C et C++ pour l'optimisation des temps et vitesses de calcul) pour la résolution de programme linéaire en nombre réels.
- La librairie **cvxpy** de Python pour résoudre des programmes en nombres entiers (se basant également sur une implémentation en C++ - Note : avant d'utiliser cvxpy, il faut avoir préalablement installer Visual Studio C++).
- Une fois les problèmes modélisés, j'ai utilisé d'autres librairies standards tel que **numpy** (pour la manipulation des vecteurs et matrices pour la définition des contraintes et fonctions objectives par exemple) **matplotlib** pour tracer des graphes et **pandas** pour la lecture et la manipulation de fichier csv (en l'occurrence pour la lecture de données historiques d'actions réelles).

3 Formulation du problème d'optimisation de la VaR et CVaR

3.1 Présentation et intuition autour de la CVaR et de la VaR

La Value at Risk est la proportion maximale du portefeuille qui peut être perdue sur une période de temps avec un niveau spécifique de confiance (en général en prend comme niveau 95%).
La Conditional Value at Risk permet de sa permet de calculer une moyenne de perte sur une période de temps avec un niveau spécifique de confiance.
Exemple :

- Si la VaR(95%)=3% alors, on aura un chance de 5% (au pire 5% des rendements), il y ait une perte de 3% ou plus sur une journée.
- Si CVaR(95%)=4,5% alors au pire 5% des rendements, il y aura une perte moyenne de 4,5%.

On remarque sur cet exemple que la CVaR mesure de manière plus précise les risques pris parce qu'elle donne une moyenne de perte alors que la VaR donne une borne inférieure de perte potentielle sans tenir compte des valeurs extrêmes (y compris les risques extrêmes du marché de type crises).

Afin d'illustrer ces deux concepts, j'ai calculé à partir de données générées automatiquement de rendements/retours (depuis une loi normale) la VaR et la CVaR à 95%. On remarque que la VaR est de -2,53%, donc avec une chance de 5%, on peut perdre au minimum -2,53 point sur notre investissement alors que la CVaR nous indique qu'au même niveau de confiance, on peut perdre en moyenne un rendement de -2,40 point.

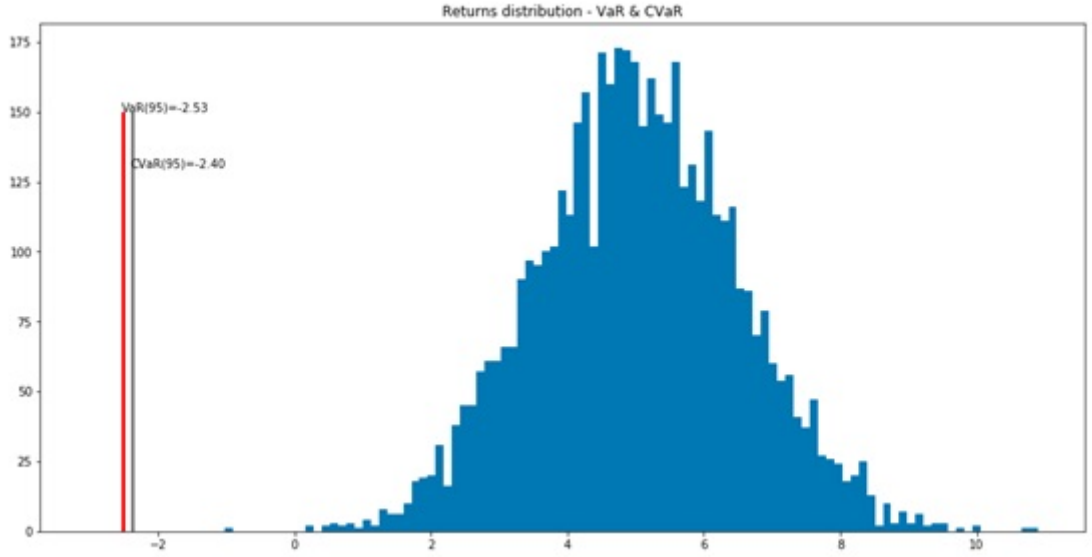

Figure 1 – Intuition derrière la CVaR et la VaR.

3.2 Implémentation du programme d'optimisation de la CVaR et de la VaR

La formulation matricielle du problème de minimisation de la CVaR ainsi que l'ensemble des paramètres d'entrées est la suivante :

- n est le nombre d'actions.
- S est le nombre de scénarios du vecteur des probabilités Y.
- β niveau de confiance (1-α).
- VaR Value at Risk.
- R_{min} rendement minimal attendu.
- R matrice des rendements des n actifs.
- $\hat{R}$ vecteur des rendements moyens (vecteur de n elements)
- $\hat{R}_i$ rendement moyen de l'$action_i$
- $x_j \in$ **[0,1]** proportion investie dans l'action$_i$

$$\min \quad VaR + \frac{1}{(1-\beta) * S}((\hat{R}^T - Y) * x, VaR)^+ \tag{1}$$

$$\text{sous contraintes} \quad \sum_{j=1}^{n} x_j = 1 \tag{2}$$

$$R_{min} \leqslant \sum_{j=1}^{n} \hat{R}_j * x_j \tag{3}$$

Cette formulation définie des paramètres initiaux (n, J, beta, Rmin, une matrice des rendements) et les variables du systèmes (x_i le poids dans l'$action_i$ et la Value at Risk).

Après avoir défini les paramètres initiaux (n=4, J=1000, beta=0,05, Rmin=2%), ci-dessous le problème d'optimisation implémenté, en utilisant la démarche de la librairie **scipy.optimize**.

Selon le paradigme de **scipy.optimize**, il faut définir 4 paramètres : une fonction définissant la fonction objective du programme à optimiser, une fonction définissant les contraintes d'égalités et d'inégalités et un vecteur définissant les intervalles d'appartenance des variables du programmes.

Après avoir défini les fonctions ci-dessus, on peut démarrer le processus d'optimisation en utilisant la fonction **minimize** (il s'agit d'un problème de minimisation).

En compilant le programme d'optimisation sur des rendements générés aléatoirement de 4 actifs suivants des lois normales de paramètres (**espérance= [0.05,0.1,0.02,0.16]** et **variance=[0.07,0.13,0.05,0.15]**) et un niveau de confiance de 5%, je retrouve le résultat suivant :

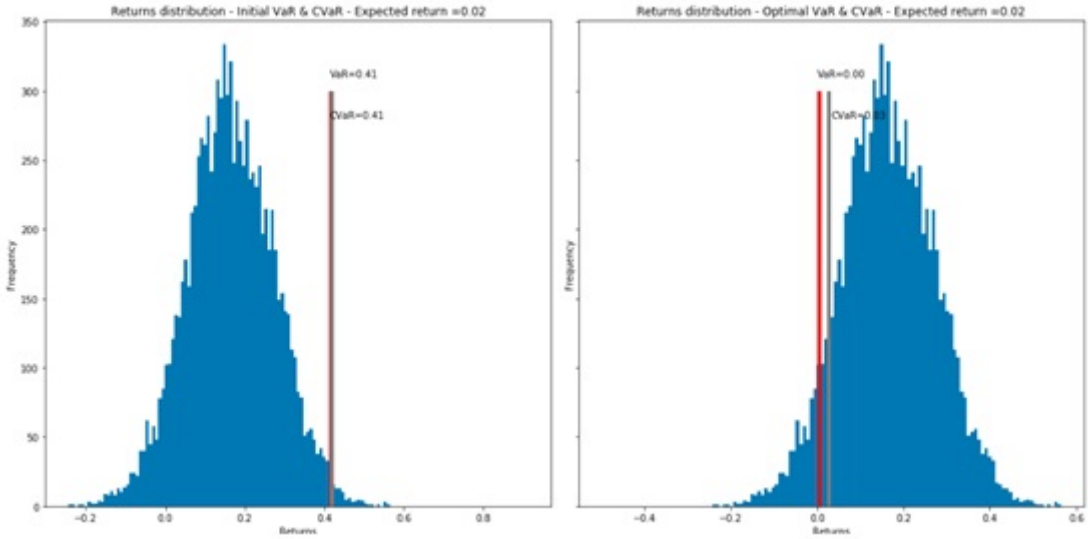

Figure 2 – Avant après l'optimisation de la CVaR et la VaR.

Nous remarquons que dans l'état initial la VaR et la CVaR du portefeuille (avec la composition des x_i initiale) étaient d'approximativement 41% alors qu'après la minimisation la VaR a passé à 0,00% et 0,03% respectivement.
Avec la nouvelle répartition de x_i, le programme permet de diminuer considérablement les pertes mises en jeu.

3.3 Evolution de quelques paramètres lors de la phase d'optimisation

Dans l'exemple donné ci-dessus, la phase d'optimisation a passé par 140 itérations.

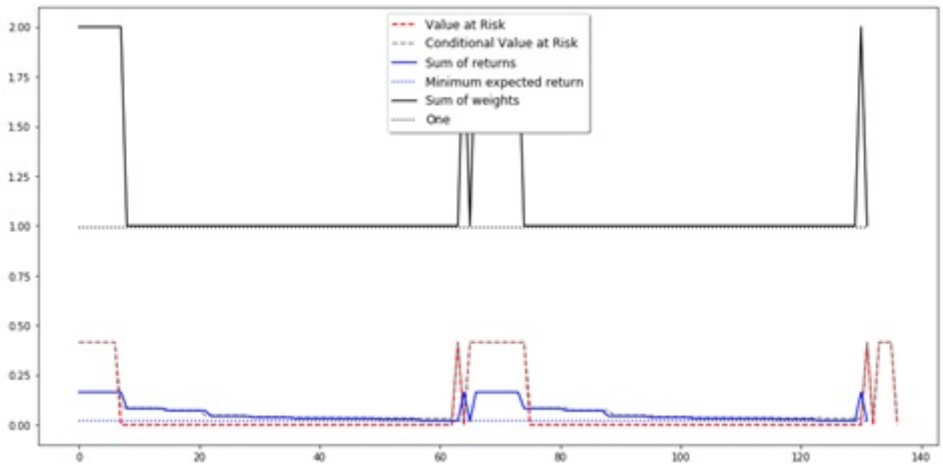

Figure 3 – Evolution de la CVaR, la VaR, la contrainte sur la somme des rendements et la somme des poids lors de la phase d'optimisation.

On remarque qu'au bout d'une vingtaine d'ittération l'optimisation converge, puis au bout de 80 itérations la contrainte sur la somme des poids est violée ce qui fait que le processus d'optimisation reprend une deuxième fois et se stabilise en atteignant 140 itérations.

3.4 Frontière de décision

Dans cette partie, j'ai généré la composition de 10000 portefeuilles consitués de 4 actifs chacun aléatoirement à partir de lois normales, puis fait varier le rendement minimum attendu et tracé l'ensemble des 10000 portefeuilles et les portefeuilles optimaux (la composition des poids optimale) pour chaque rendement attendu en indiquant la CVaR correspondante.

La frontière optimale est composée par l'ensemble des portefeuilles ayant un rendement maximal pour un niveau de risque (CVaR) donné. On retrouve bien le résultat exposé en cours.

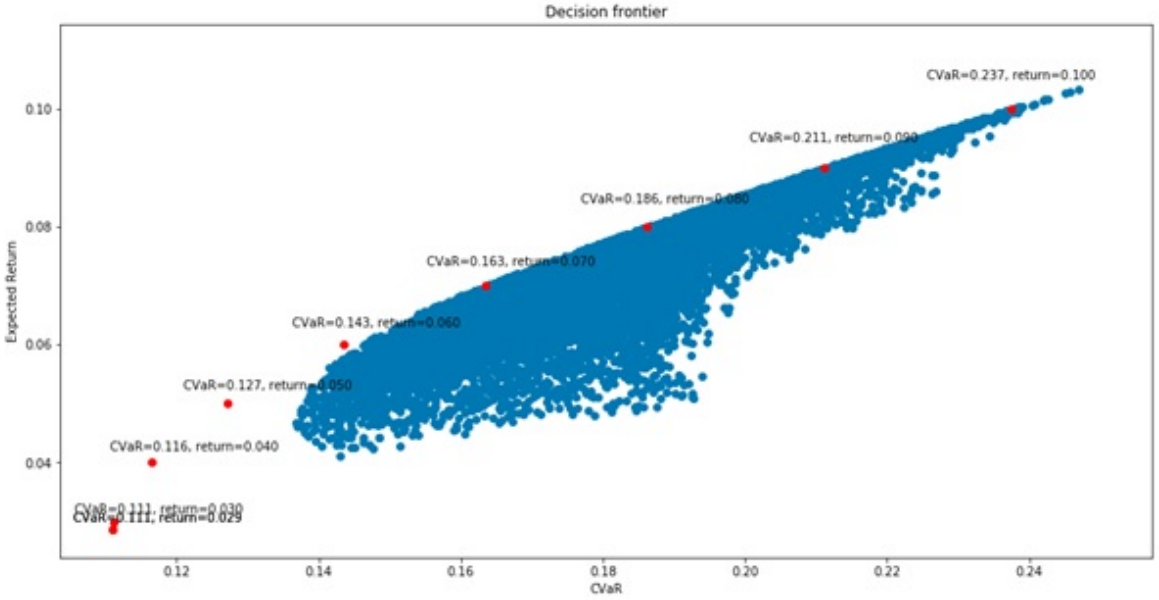

FIGURE 4 – Frontière de décision.

3.5 Effet de la variation des paramètres

3.5.1 Effet de la variation de β sur la CVaR et la VaR.

Dans ce paragraphe, on étudie l'effet de la variation de β sur la CVaR et la VaR optimales.

On remarque qu'avec 4 actions et un $R_{min} = 0.01$, la VaR reste quasi constante en évoluant le β alors que la CVaR croît au fur et à mesure de l'augmentation du β. Ceci reflète d'une part que la CVaR est une mesure de risque plus précise que la VaR et d'autre part que plus on augmente β plus la valeur de la CVaR augmente ce qui est en accord avec l'intuition formulée dans la première partie du rapport.

On remarque par ailleurs que plus on augmente β plus la CVaR s'éloigne de la VaR, ce qui est tout à fait normal car pour β=**0** la CVaR tend vers la VaR.

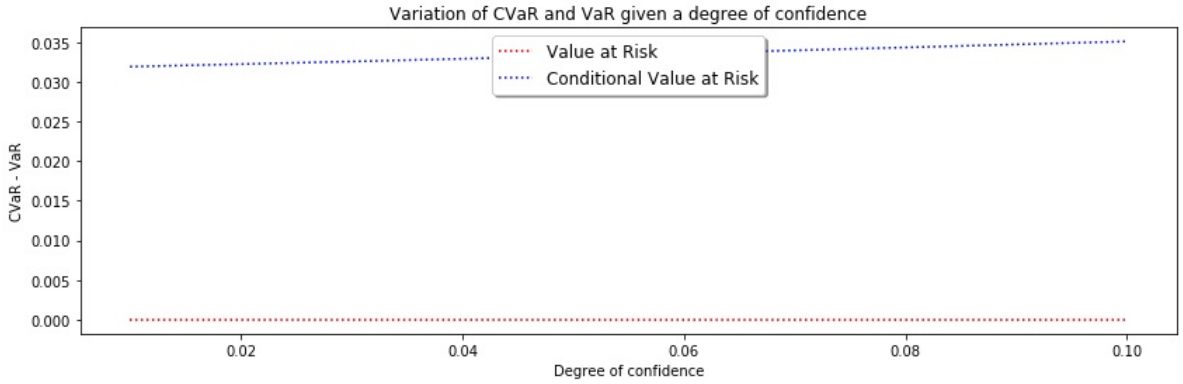

FIGURE 5 – Effet de la variation de β.

3.5.2 Effet de la variation de R_{min} sur la CVaR et la VaR.

Dans ce paragraphe, on étudie l'effet de la variation de R_{min} sur la CVaR et la VaR optimales.

On remarque que la CVaR augmente de manière constante en augmentant le R_{min}, ceci est tout à fait normal puisque plus R_{min} le rendement minimum exigé est grand plus le niveau de risque à prendre (en l'occurence la CVaR) est grand.

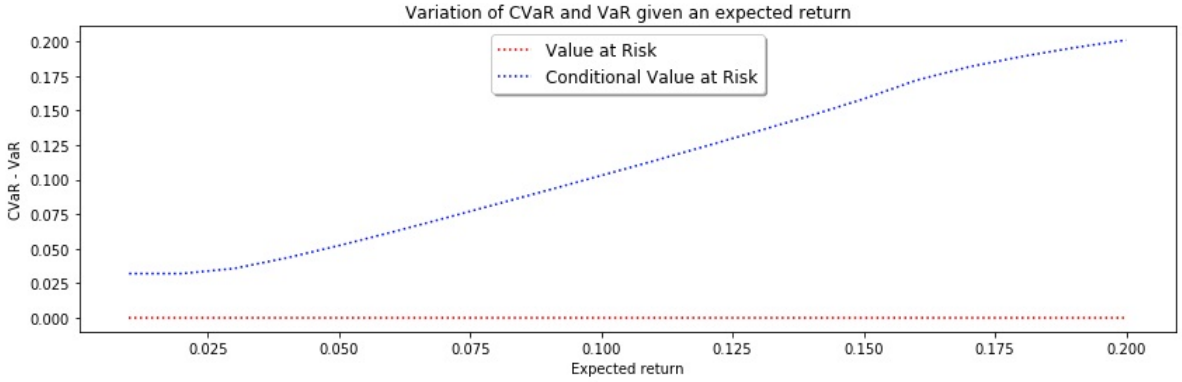

FIGURE 6 – Effet de la variation de R_{min}.

3.5.3 Effet de la variation de n sur le temps d'exécution

Le constat à faire dans ce paragraphe est simple et est prévisible : Le fait d'augmenter le nombre d'actions dans la composition du portefeuille à optimiser augmentera de manière naturelle le temps d'optimisation du programme linéaire puisqu'il y aura plus de variables et de contraintes à traiter.

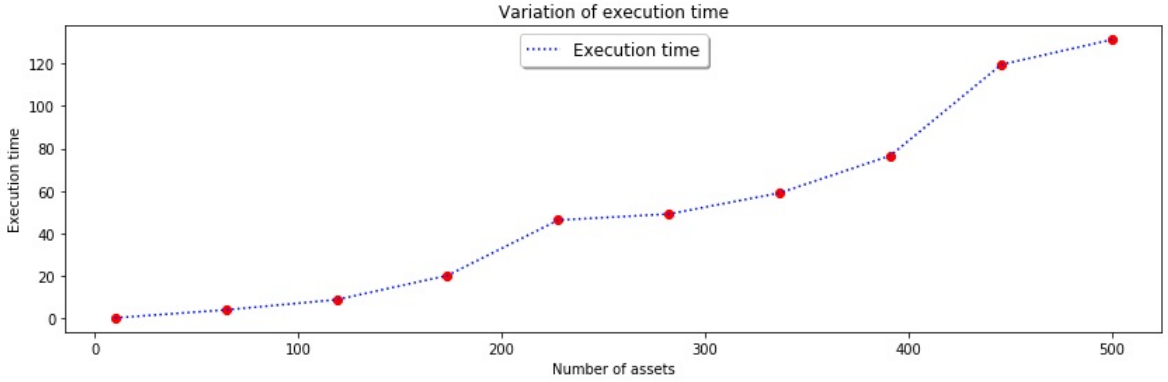

FIGURE 7 – Effet de la variation du nombre d'actifs sur le temps d'exécution.

4 Formulation du modèle de Markowitz en utilisant la CVaR

L'objectif de cette deuxième partie est d'utiliser le modèle de Markovitz maximisant le rendement avec un niveau de risque (mesuré par la CVaR) fixé à l'avance.

4.1 Implémentation du programme d'optimisation de portefeuille selon le modèle de Markowitz

La formulation matricielle du problème est la suivante :

$$\max \quad \sum_{j=1}^{n} \hat{R}_j * x_j \tag{4}$$

$$\text{Sous contraintes} \quad \sum_{j=1}^{n} x_j = 1 \tag{5}$$

$$CVaR \leqslant CVaR_{max} \tag{6}$$

Dans ce modèle l'objectif est de maximiser le rendement. Pour pouvoir utiliser la fonction minimiser de **scipy.optimize.minimize** j'ajoute un moins devant la fonction objective dans mon implémentation (minimiser la fonction objective revient à maximiser l'opposé de la fonction objective).

4.2 Effet de la variation des paramètres

4.2.1 Effet de la variation de $CVaR_{max}$ le rendement du portefeuille.

En prenant un nombre d'actifs de 50, j'ai fait varier la $CVaR_{max}$ entre 0.01 et 0.02.

Le graphique suivant montre l'évolution du rendement optimal et la VaR pour chaque valeur de $CVaR_{max}$

Le premier constat à faire est que le rendement augmente en augmentant la $CVaR_{max}$, ce qui est prévisible étant donné que l'augmentation de la $CVaR_{max}$ est équivalente à l'augmentation du niveau du risque prix, ce à quoi on s'attendrait à être rémunéré plus.

Le deuxième constat est que la valeur de la VaR augmente en augmentant la $CVaR_{max}$, ce qui est également logique puisque les deux métriques mesurent le risque. Par conséquent, on s'attend à ce qu'elles varient dans le même sens.

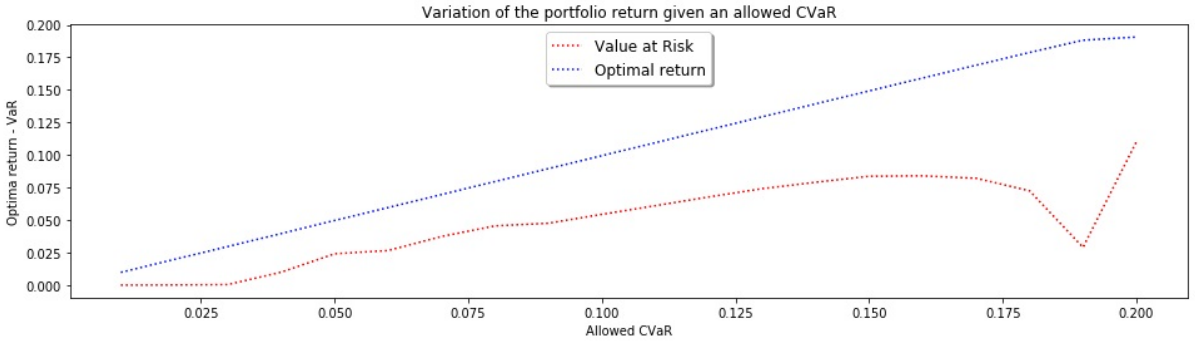

FIGURE 8 – Effet de la variation de la $CVaR_{max}$ sur **R** et **VaR**.

4.2.2 Effet de la variation de n le temps d'exécution.

On retrouve dans le graphe suivant le constat que l'on a fait dans la partie précédente : l'augmentation du nombre d'actifs dans le portefeuille fait augmente le temps d'exécution de l'optimisation du programme linéaire.

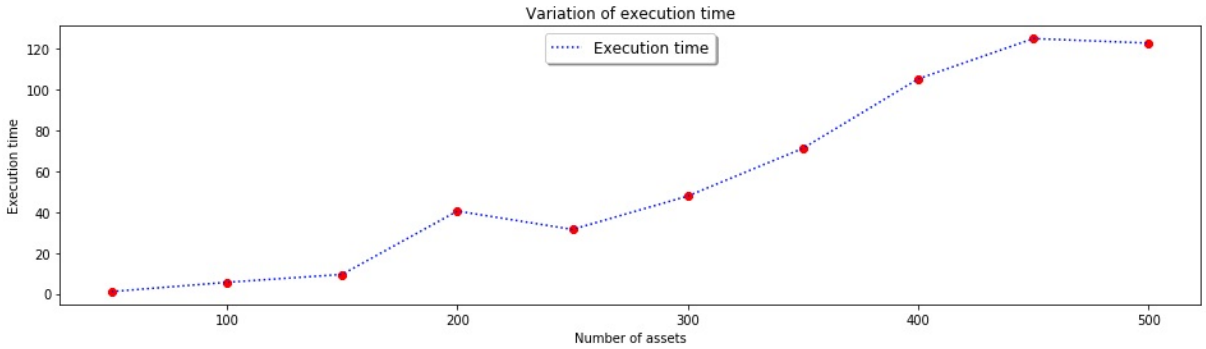

FIGURE 9 – Effet de la variation du nombre d'actifs sur le temps d'exécution.

5 Formulation du problème de la construction d'un index fund

Un index fund est une sélection d'un petit groupe de l'ensemble de l'univers des actions permettant de suivre le mouvement global du marché.

Le modèle présenté ci-dessous cherche à classifier les actions dans des groupes d'actions ayant un comportement similaire. Dans chaque groupe, le modèle choisi un représentant des membres du groupes.

L'entrée du modèle consiste en une matrice de $(\alpha_{ij})_{0<i,j<n+1}$ indiquant la similarité entre paire d'action. Dans notre modèle, j'utilise le coefficient de corrélation entre deux vecteurs d'historique de prix de deux actions comme mesure de similarité.

5.1 1ère formulation du problème

La formulation du problème est la suivante :

— n le nombre d'actions

— α_{ij} indique la similarité entre chacune des paires (i, j) d'actions du marché.

— y_j 0,1, 1 si l'action j est selectionnée comme representative, 0 sinon. $j = \{1, 2, .., n\}$

— x_{ij} 0,1, 1 si l'action i est dans le cluster représenté par l'action j, 0 sinon. $i = \{1, 2, .., n\}$

— q est le nombre de clusters. Le problème requière que chaque action soit assignée à un et un seul cluster.

$$\max \quad \sum_{(i,j)\ [1,n]} \alpha_{ij} x_{ij} \tag{7}$$

$$\text{Sous les contraintes} \quad \sum_{j=1}^{n} y_j = q \tag{8}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1, 2, .., n\} \tag{9}$$

$$x_{ij} \leqslant y_j \qquad i \in \{1, 2, .., n\} \qquad j \in \{1, 2, .., n\} \tag{10}$$

5.1.1 Utilisation du modèle avec n=5

L'objectif de cette partie est de donner un exemple simplifié où j'utilise le modèle afin de pouvoir comparer les résultats donnés par l'optimisation et ce à quoi on s'attendrait normalement. Dans cette optique, j'ai utilisé les données historiques de 5 entreprises françaises cotées en bourse ('VINCI','BOUYG','SANOFI','SG','BNP').

La matrice de corrélation entre les 5 actifs est données ci-dessous :

Computing the correlation matrix

```
In [10]: correlation_matrix=np.corrcoef([vinci,bouyg,sanofi,sg,bnp])
         print('      [vinci    -    bouyg    -    sanofi    -    sg    -    bnp    ]')
         correlation_matrix

               [vinci    -    bouyg    -    sanofi    -    sg    -    bnp    ]
Out[10]: array([[1.        , 0.69465691, 0.56456922, 0.42855814, 0.63032581],
                [0.69465691, 1.        , 0.17303234, 0.76440501, 0.76653873],
                [0.56456922, 0.17303234, 1.        , 0.2374724 , 0.42817254],
                [0.42855814, 0.76440501, 0.2374724 , 1.        , 0.92116859],
                [0.63032581, 0.76653873, 0.42817254, 0.92116859, 1.        ]])
```

FIGURE 10 – Matrice de corrélation - Capture de Python.

On peut voir à partir de la matrice de corrélation que :

— l'actif BNP représente bien 'VINCI' (0.76),'BOUYG' (0.76),'SG' (0,92).

— l'actif SG représente moins bien les autres actifs que 'BNP'.

— l'actif BOUYG représente mieux que BNP l'actif 'VINCI'.

— l'actif VINCI représente mieux l'actif 'SANOFI'.

En choisissant un **q=2**, on retrouve le résultat suivant :

```
Clusters representatives :
[0, 0, 1, 0, 1]
Allocation matrix :
[[0, 0, 0, 0, 1], [0, 0, 0, 0, 1], [0, 0, 1, 0, 0], [0, 0, 0, 0, 1], [0, 0, 0, 0, 1]]
Allocation :
VINCI  is represented by  BNP
BOUYG  is represented by  BNP
SANOFI  is represented by  SANOFI
SG  is represented by  BNP
BNP  is represented by  BNP
```

Conclusion

The index fund is composed of two stocks : BNP & SANOFI.

FIGURE 11 – Résultat de l'optimisation - Capture de Python.

On retrouve bien l'intuition que l'on a énoncé en début de cette partie.

5.1.2 Effet de la variation de n sur le temps d'exécution

Dans cette section on étudie l'effet de l'augmentation du nombre d'actifs et des clusters dans l'index fund sur le temps d'excécution.

On remarque que de manière générale, la courbe du temps d'exécution a une tendance croissante. Plus particulièrement, en augmentant q, la courbe acquiert un comportement asymptotique d'une exponentielle.

Cela est tout afin normal, car d'une part, en augmentant le nombre d'actifs, on augmente considérablement le nombre de contraintes sur x_{ij}, et d'autre part, l'augmentation de q fait que les contraintes sur y_j soient multipliées.

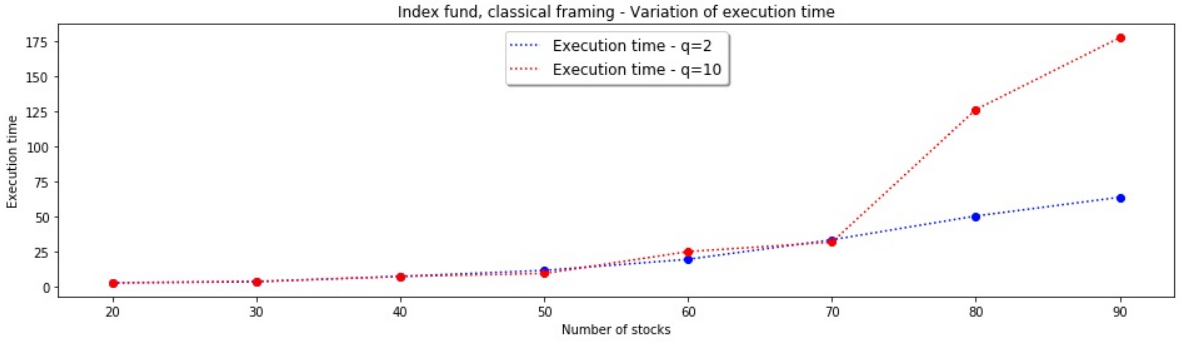

FIGURE 12 – Effet de la variation de **n** avec **q=2** et **q=10**.

5.2 2ème formulation du problème

Dans cette deuxième partie, l'objectif est d'utilisation la relaxation lagrangienne pour alléger les contraintes imposées sur les x_{ij} dans la formulation précédente. On introduit donc un vecteur de multiplicateurs de Lagrange **U**.
La formulation du problème initial devient :

- n le nombre d'actions
- α_{ij} indique la similarité entre chacune des paires (i, j) d'actions du marché.
- y_j 0,1, 1 si l'action j est selectionnée comme representative, 0 sinon. $j = \{1, 2, .., n\}$
- q est le nombre de clusters. Le problème requière que chaque action soit assignée à un et un seul cluster.

$$L(U) = \max_{y_1, y_2 .. y_n} \sum_{i=1}^{n} C_i y_i + \sum_{i=1}^{n} u_i \tag{11}$$

$$\text{Où} \quad C_i = \sum_{j=1}^{n} (\alpha_{ij} - u_j)^+ \tag{12}$$

$$\text{Sous les contraintes} \quad \sum_{j=1}^{n} y_j = q \tag{13}$$

$$y_j \in \{0, 1\}, j = 1, 2, .., n \tag{14}$$

$$U = [u_1, u_2, .., u_n] \tag{15}$$

Afin de pouvoir trouver la solution optimale (cf. [5]), il faut résoudre le problème suivant :

$$\min_{U} L(U) \tag{16}$$

Etant donné que la fonction : $\mathbf{f :}\ u \mapsto L(u)$ est convexe et non différentiable dans les points où $\alpha_{ij} = u_j$ (cf. [5]), il n'est pas possible d'utiliser l'algorithme de déscente de gradient qui nécessite que la fonction sur laquelle l'algorithme est appliqué soit au moins une fonction de C^1.Dans ce cas, la méthode de subgradient (cf. [2]) peut être utilisée.

Dans mon programme, je n'ai pas réussi à implémenter la méthode de subgradient. Je me suis donc contenté de varier aléatoirement les composantes du vecteur **U**, puis de prendre la résultante $L(U)_{min}$. Cette méthode n'assure pas d'avoir un minimum global dans l'espace des **U**. Cependant, en générant un assez grand nombre d'instances de **U**, il y a une grande probabilité de tomber sur le minimum global.
Une fois le minimum trouvé, les q représentatifs des clusters seront les q éléments ayant les q plus grands C_i.

5.2.1 Utilisation du modèle avec n=5

L'objectif de ce paragraphe est de comparer les résultats du modèle dans sa deuxième formulation avec les résultats que l'on a eu lors de l'expérimentation utilisant la première formulation.

Le vecteur des C_i est donné dans ce cas par :

```
In [80]: C[optimal_utility_fct.index(min(optimal_utility_fct))]

Out[80]: [0, 0, 0, 0.060127513170310 84, 0.1389589228877005]
```

Figure 13 – Résultat de l'optimisation - Capture de Python.

Cela revient à dire que les actifs représentant le marché sont 'BNP' et 'SG' (ceux ayant les plus grands C_i non nuls).

BNP représente le cluster 'VINCI','BOUYG','SANOFI' et SG représente le cluster 'SG'.

Ce résultat est différent de ce que l'on a eu lors du premier essai, mais reste toutefois acceptable dans le sens où cette affectation reflèterait bien le mouvement du marché au vu de la matrice de corrélation.

6 Code source du projet

Vous pouvez retrouver l'ensemble du code source développé dans le cadre de ce projet dans mon répertoire Github dans les répertoires suivants :

— **Project 5 - Portfolio Optimization using VaR CVaR** pour la première et la deuxième partie du projet.

— **Project 6 - Construction of an index fund** pour la troisième partie du projet.

7 Conclusion

Ce projet a été pour moi l'occasion de comprendre en profondeur le sens derrière les différentes thématiques traitées en cours d'Optimisation en Finance. Il m'a permis, en particulier, de comprendre l'utilité de l'utilisation de différentes mesures de risques afin d'avoir une vision plus vaste sur la problématique traitée, et d'être conforter aux difficultés rencontrées lorsque l'on passe d'un modèle mathématique à son implémentation sur machine en vue de sa résolution.

J'ai pu également développer de nouvelles compétence en modélisation de programmes linéaires sur le langage **Python**.

Références

[1] Dr. Ted Ralphs, Financial, Optimization ISE 347/447, lecture 24
`https://coral.ie.lehigh.edu/ ted/files/ie447/lectures/Lecture24.pdf`

[2] Dr. Ted Ralphs, Financial, Optimization ISE 347/447, lecture 15
`https://coral.ie.lehigh.edu/ ted/files/ie447/lectures/Lecture15.pdf`

[3] Simone Forghieri : Portfolio Optimization using CVaR,
`https://tesi.luiss.it/12528/1/forghieri-simone-tesi-2014.pdf`

[4] Optimization Methods in Finance,
`http://cs.brown.edu/courses/cs1951g/slides/04-IndexFunds.pdf`

[5] fabio.furini@dauphine.eu : integer programming models,
`https://www.lamsade.dauphine.fr/ furini/lib/exe/fetch.php?media=wiki:mip2.pdf`