Partie 1 : Pricing d'une option en variations discrètes

1. Définition de la méthode treillis :

Le model, treillis est un modèle binomial appelé CRR qui fournit une méthode numérique pour le calcul du prix de l'option en temps discret. Le modèle est un modèle discret pour la dynamique du sous-jacent.

2. Objectif

Notre objectif consiste à implémenter la méthode treillis sur python. Cette méthode permet de calculer les prix des options CALL et PUT européennes et américaines selon le prix de l'actif à t=0, la période (en semaines), le taux annuel convertis, le strike K, et les taux d'augmentation où diminution d'un actif (Up et Down).

Ensuite tester la méthode sur différentes instances en faisant varier les paramètres n (le nombre de périodes), r (le taux), D et U. On essayera de considérer aussi des instances assez larges

3. Réalisations

J'ai réalisé une méthode en python permettant de calculer les prix des options. J'ai réalisé ensuite une interface graphique qui adapte ma méthode python, et qui a pour but de faciliter la saisie et l'affichage du prix de l'option. Cette interface affiche également l'arborescence des variations des prix pour toutes les périodes t.

Ensuite, j'ai étudié le comportement des option européenne et américaine en variant les données. J'ai modélisé ces variations en graphes pour faciliter la compréhension du comportement des options.

J'ai utilisé les bibliothèques Numpy, math Plot pour l'affichage des graphes et App Jar pour l'interface graphique.

4. L'implémentation de la méthode treillis en python :

J'ai implémenté une méthode en python qui prend en paramètres le prix de l'actif S, le Strike de l'option K, le taux convertis en taux quotidien, les variations d'augmentation et de diminutions (U et D), le nombre de périodes n, le type de l'option Call ou PUT, et l'origine de l'option européenne ou américaine.

Notre méthode nous retourne le prix de l'option à t=0 accompagné d'une matrice où chaque colonne désigne une période et chaque ligne affiche la variation du prix à l'instant t.

```
def treillis(actifPrice, strike, taux, up, down, periode, optionType,
optionOrigine):
    # init the matrix
    global columnValue
    m = periode * 2 + 1
    n = periode
    matrix = np.zeros((m, n + 1))
```

```
    # init the call put situation
    clefCallPut = -1
    if optionType == "PUT":
        clefCallPut = 1

    # calculate probabilities pU and pD
    ratio = 1 + taux
    pU = (ratio - down) / (up - down)

    tauxRatio = 1 / ratio

    # calculate the first elements
    j = m - 1
    i = 0
    while j >= 0:
        actifAtT = ((up ** i) * (down ** (n - i)) * actifPrice)
        value = strike - actifAtT
        valueN = max(value * clefCallPut, 0)
        matrix[j][periode] = valueN
        j = j - 2
        i = i + 1
    # calculate the rest of the elements with reccurence
    n = n - 1
    i = 1
    while n >= 0:
        j = (periode * 2) - i
        h = n + 1
        k = 0
        while j >= 0:
            columnValue = (1 / 1 + taux) * ((1 - pU) * matrix[j + 1][n + 1]
+ (pU) * matrix[j - 1][n + 1])
            if optionOrigine == "USA":
                if optionType == "PUT":
                    amiricaineValue = strike - ((up ** (j+1)) * (down **
(periode - (j+1))) * actifPrice)
                    columnValue = max(columnValue, amiricaineValue)

            matrix[j][n] = columnValue
            j = j - 2
            k = k + 1
            if k == h:
                break

        n = n - 1
        i = i + 1

    return (matrix,columnValue)
```

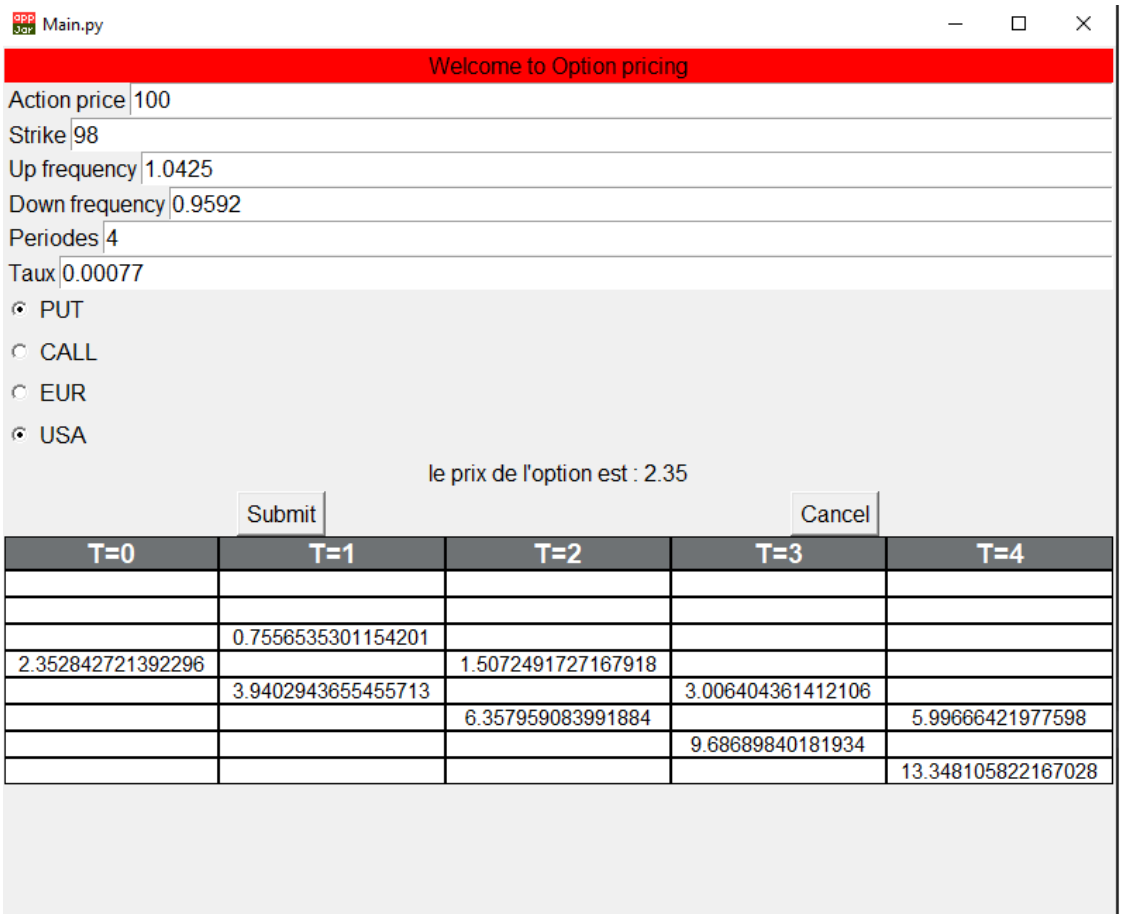
Figure 1: Le code source de la méthode de pricing

Cette capture d'écran affiche le code source de la méthode. La méthode commence d'abord par calculer les probabilités de hausse et de baisse Pu et Pd, pour ensuite calculer les éléments à la date de maturité de l'option N, et finalement calculer le reste des éléments par les formules de récurrence.

Pour mieux afficher les résultats du pricing des options, j'ai implémenté une interface graphique en utilisant le package Appjar de python. Ce package facile d'utilisation permet d'implémenter

rapidement une interface graphique qui contient des champs de texte facilitant la saisie des paramètres de calcul.

Cette interface nous affiche un tableau contenant les variations des prix de l'option pour chaque période.



T=0	T=1	T=2	T=3	T=4
	0.7556535301154201			
2.352842721392296		1.5072491727167918		
	3.9402943655455713		3.006404361412106	
		6.357959083991884		5.99666421977598
			9.68689840181934	
				13.348105822167028

Figure 2: l'interface graphique pour la méthode de pricing

5. Test de la méthode sur différentes instances :

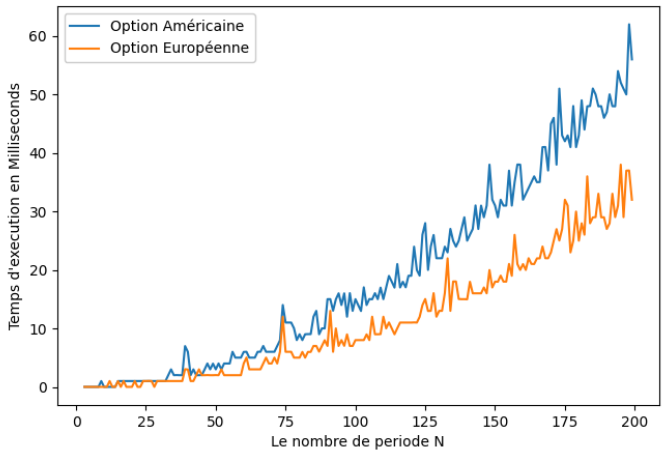
Exemple du cours :

Pour l'implémentation des graphes suivants j'ai repris l'exemple du cours, où les paramètres de la méthode sont les suivant :

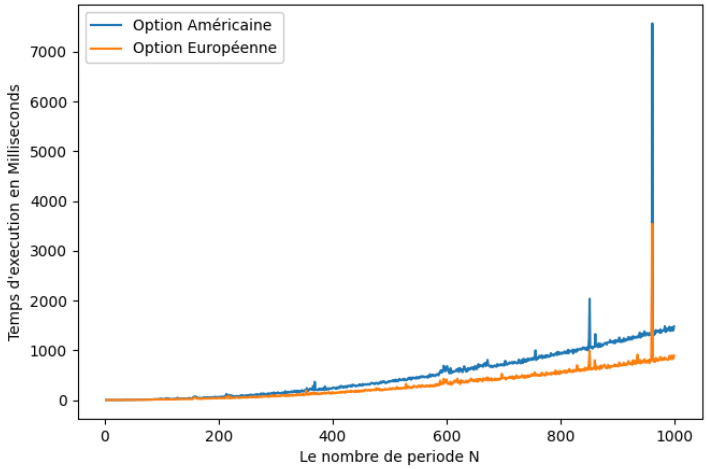
- Prix de l'actif = 100
- Strike k = 90

Les autres paramètres sont variés selon les graphes, et sont affichés à gauche de chaque graphe.

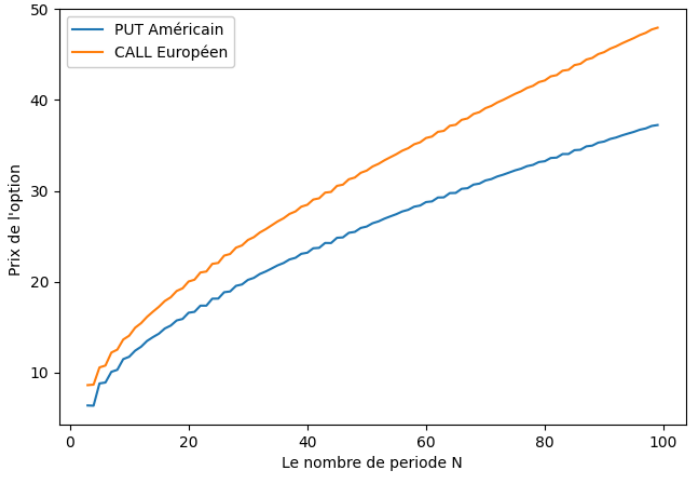
N = 0 -> 200, U=1.1, D=0.9, r=4%



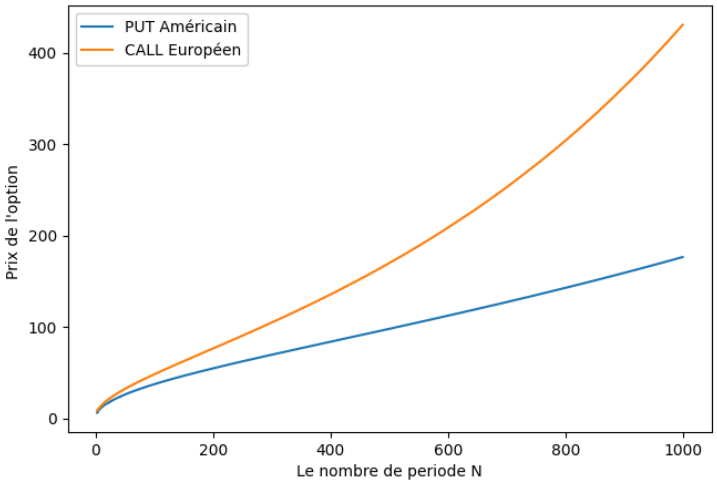
N = 0 -> 1000, U=1.1, D=0.9, r=4%



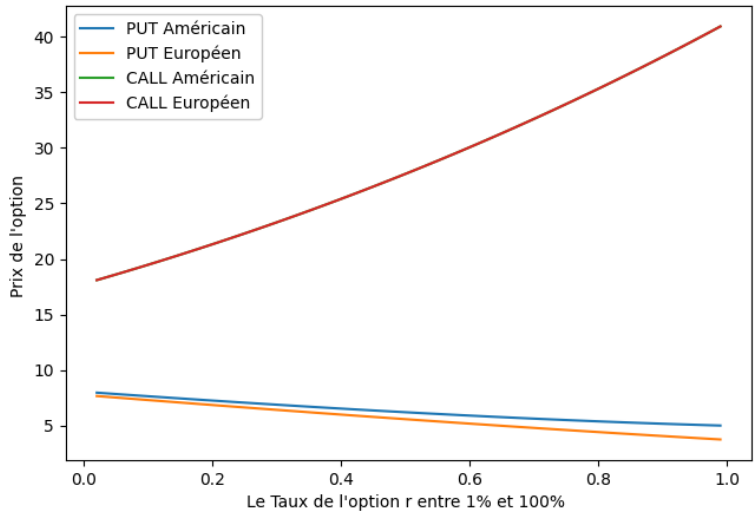
N= 0 -> 100, U=1.1, D=0.9, r=4%



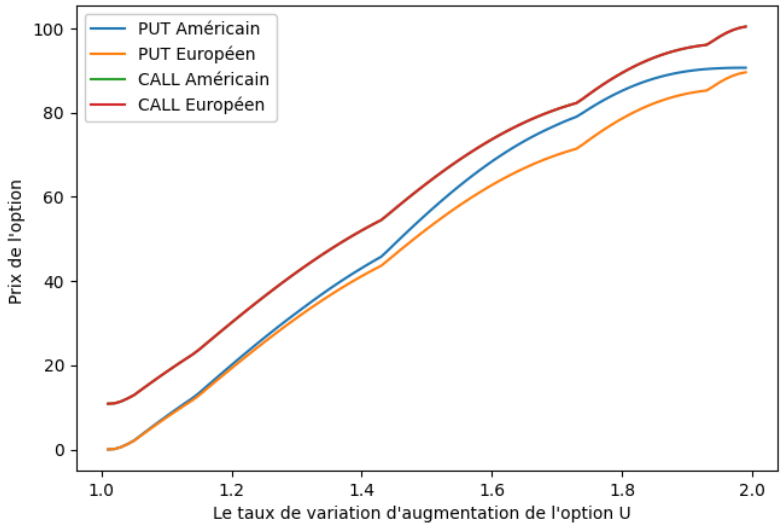
N= 0 -> 1000, U=1.1, D=0.9, r=4%

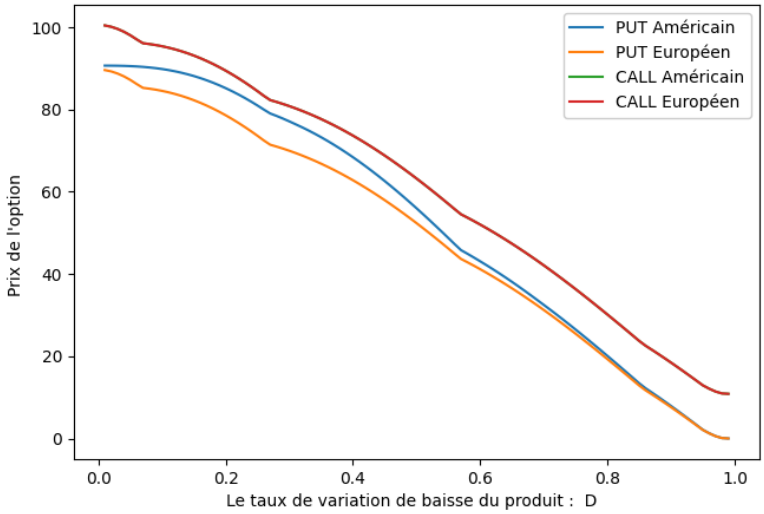


r= 0.01% -> 0.99%, N= 1000, U=1.1, D=0.9



N = 1000, r = 4%, U= 1.0 -> 1.99, D= 0.0-> 1.0 (pour les 2 graphes ci-dessous)





Partie 2 : Optimiser le risque d'un portefeuille

Optimiser le risque d'un portefeuille repose sur la résolution des modèles de minimisation de risque, notamment sur les modèles de minimisation de la CVaR. Pour la résolution, j'ai utilisé l'outil Cplex de IBM que j'ai trouvé disponible sur internet pour les étudiants avec un nombre de contraintes illimitées.

Pour cette 2e partie, j'ai construit un portefeuille avec des actifs réels qui sont : S&P qui est un indice boursier basé sur plusieurs entreprises, Gov Bond qui sont des emprunts d'état, et Small cap qui est une société dans le domaine de la bourse.

Je vais commencer par construire un portefeuille Markowitz. Ce portefeuille doit me garantir en rendement R. J'ai comme données la matrice de covariance et les espérances de rendements des 3 actifs. Ce portefeuille va m'aider à bien comprendre les liens et les différences entre un portefeuille avec une variance minimale et un portefeuille avec un risque minimal.

Je vais ensuite, trouver un portefeuille avec un rendement égale à celui de Markowitz et calculer le VaR et la CVaR de ce portefeuille en utilisant les méthodes du cours que j'ai programmé en python. Ensuite, on va passer au modèle de minimisation de la CVaR en générant des simulations sur les prix des actifs. Puis à la comparaison des 2 portefeuilles avant et après la minimisation.

Enfin, on passera au modèle de maximisation du rendement du portefeuille en respectant un certain seuil de risque.

Avant d'entamer la résolution des modèles, je vais expliquer mes méthodes python que j'ai implémenté pour calculer la VaR et la CVaR d'un portefeuille sans minimisation, simuler les scénarios des actifs, et les différents tests effectués avec ses simulations.

1. Construire un portefeuille (Model Markowitz) :

Pour construire le portefeuille j'ai utilisé le modèle Markowitz du cours, qui consiste à minimiser la variance pour un rendement maximal donné. On n'est pas obligé de résoudre le model Markowitz pour l'utiliser dans la minimisation de la CVaR, il suffit juste de trouver un portefeuille qui satisfait les conditions du portefeuille Markowitz ($\Sigma x_i = 1$, $\Sigma \mu_i x_i \geq R$).

Pour une meilleure observation, j'ai décidé de passer par le model Markowitz.

Le modèle est le suivant :

$$\text{Min} \sum_{i,j} \rho_{ij}\sigma_i\sigma_j x_i x_j$$
$$\sum_i x_i = 1$$

$$\sum_i \mu_i x_i \geq R$$
$$x_i \geq 0 \quad \forall\, i = 1, \dots, n.$$

La première étape était d'implémenter ce modèle dans l'outil d'optimisation Cplex.

Pour réussir cette implémentation, je me suis exercé sur plusieurs exemples dans la documentation. Ces exercices m'ont permis de générer mon premier model.

```
/* DATA VARIABLES ___________*/
int Actifs=...;
range numberActifs = 1..Actifs;
float R= ...;
float Cov[numberActifs][numberActifs]=...;
float Rendements[numberActifs]=...;
int wealth = ...;


/* DECISION VARIABLE ___________*/
dvar float+ x[numberActifs];
dexpr float objective = sum(i ,j in  numberActifs)Cov[i][j]*x[i]*x[j];


/* OBJECTIF FUNCTION ___________*/
minimize objective;


/* Constraintes ________*/
subject to {
  sum(i in numberActifs)x[i]== wealth;

 sum(i in numberActifs) (x[i]*Rendements[i]) >= R ;

}

float TotalReturn = sum(i in numberActifs) Rendements[i]*x[i];
float TotalVariance = sum(i,j in numberActifs) Cov[i][j]*x[i]*x[j];

execute DISPLAY {
  writeln("Total Expected Return: ", TotalReturn);
  writeln("Total Variance       : ", TotalVariance);
}
```

Figure 3 : Modèle Markowitz pour la minimisant la variance

1.1 Les actifs :

La 2e étape consiste à trouver les actifs, la matrice de covariance, et les espérances de rendement. Pour ensuite les utiliser comme données.

On dispose des 3 actifs S&P, Gov bonds, et Small cap. Notre objectif est d'atteindre 1.1% de gains avec un minimum de variance entre les actifs. L'espérance de rendement de chaque actif et la matrice de covariance sont affichées dans les tableaux ci-dessous.

Actif	Espérance de gains
S&P	0.0101110
Gov bonds	0.0043532
Small cap	0.0137058

Figure 4: Espérances de gains pour les 3 actifs

	S&P	Gov bonds	Small cap
S&P	0.00324625	0.00022983	0.00420395
Gov bonds	0.00022983	0.00049937	0.00019247
Small cap	0.00420395	0.00019247	0.00764097

Figure 5: Matrice de covariance des 3 actifs

1.3 Solution du model Markowitz :

Cplex nous a généré la solution ci-dessous.

S&P	Gov bonds	Small cap
0.452008747332515	0.115573	0.432414

Figure 6 : Solution générée par Cplex pour un rendement de 1.1%

La solution du modèle Markowitz consiste à allouer 45.2 % pour S&P, 11.5% pour Gouv Bond et 43.2% pour l'actif Small cap. Cette allocation d'actifs nous garantit un rendement minimal de 1.1% et une variance totale de 0.004.

1.2 Trouver un portefeuille Markowitz avec un rendement égale à 1.1% :

Le portefeuille ci-dessous satisfait les conditions Markowitz ($\Sigma x_i = 1$, $\Sigma \mu_i x_i \geq R$).

S&P	Gov bonds	Small cap
0.75	0	0.25

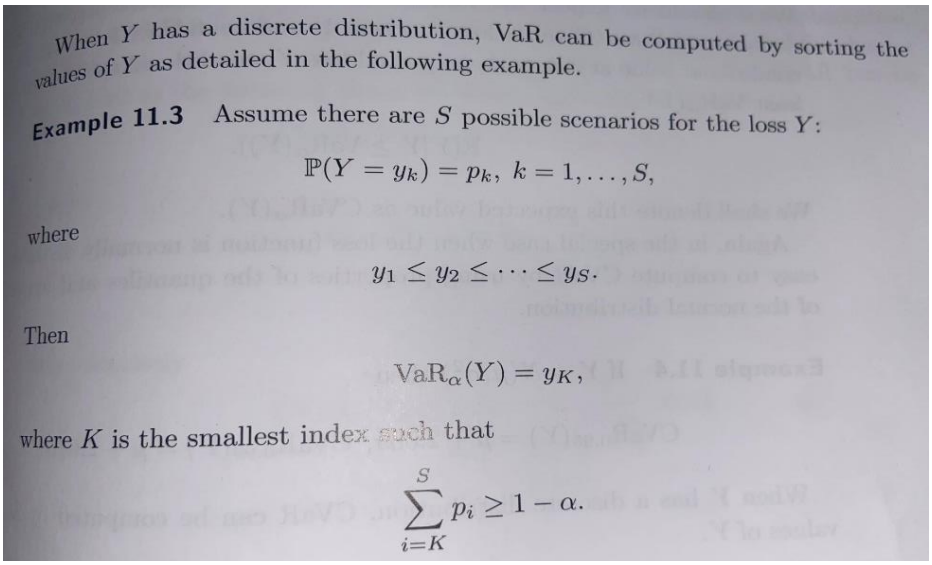
Rendement R = 1.1%.

Notre objectif est de calculer la CVaR et la VaR de ce portefeuille. Ensuite, trouver un autre portefeuille avec une VaR et une CVaR minimal.

2. Calcule de la CVAR et la VAR pour ce portefeuille :

Pour implémenter la CVAR et la VaR, j'ai repris les mêmes méthodes de calcul du cours. Je me suis permis de chercher des méthodes et des exemples dans d'autres livres, Je cites (Optimization methodes in finance , Seconde édition)

2.1 Méthode de calcul de la VaR



When Y has a discrete distribution, VaR can be computed by sorting the values of Y as detailed in the following example.

Example 11.3 Assume there are S possible scenarios for the loss Y:

$$\mathbb{P}(Y = y_k) = p_k,\ k = 1, \ldots, S,$$

where

$$y_1 \leq y_2 \leq \cdots \leq y_S.$$

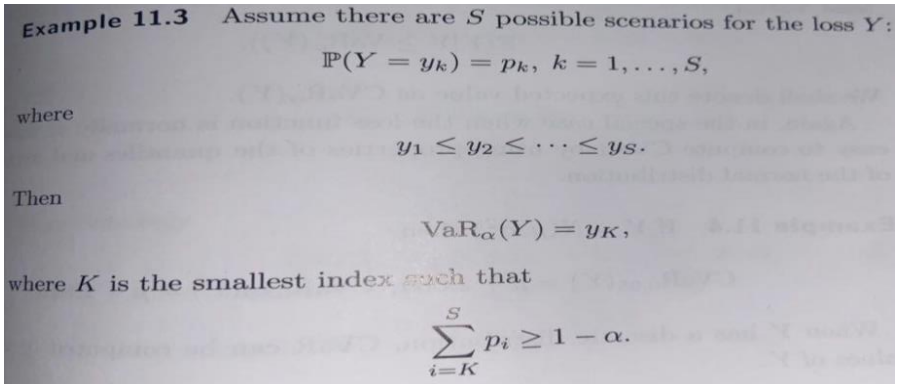
Then

$$\mathrm{VaR}_\alpha(Y) = y_K,$$

where K is the smallest index such that

$$\sum_{i=K}^{S} p_i \geq 1 - \alpha.$$

2.2 Méthode de calcul de la CVaR



Example 11.3 Assume there are S possible scenarios for the loss Y:

$$\mathbb{P}(Y = y_k) = p_k,\ k = 1, \ldots, S,$$

where

$$y_1 \leq y_2 \leq \cdots \leq y_S.$$

Then

$$\mathrm{VaR}_\alpha(Y) = y_K,$$

where K is the smallest index such that

$$\sum_{i=K}^{S} p_i \geq 1 - \alpha.$$

Y, ici désigne la valeur de la fonction de risque Y= f(x,y)= $(b-y)^T x$, où b désigne les prix futurs de l'actif i, y désigne l'ensemble des scénario possibles et x l'allocation de l'actif dans le portefeuille de Markowitz.

J'ai réussi à implémenter une méthode python qui utilise les simulations des actifs dans le futur et pas les données historiques des actifs pour calculer la VaR et la CVaR. En utilisant la méthode de simulation Monte-Carlo et en générant suffisamment de scénarios. J'ai calculé grâce à cette méthode les valeurs de la VaR et la CVaR de mon portefeuille.

```
def calculateurCVarAndVar(benchmark,simulationNumber,portfeuille,α):
    simulationMatrix =
GenerateSimulationMatrix(len(portfeuille),simulationNumber)
    Scenario_ResultAndProbability = []
    P = 1 / simulationNumber
    for i in range(simulationNumber - 1):
        Y = 0
        for j in range(len(portfeuille)):
            Y = Y + ((benchmark[j] - simulationMatrix[i][j]) *
portfeuille[j])

        Scenario_ResultAndProbability.append([Y, P])
    SortedMatrixVarValue = np.array(Scenario_ResultAndProbability)
```

```
    SortedMatrixVarValue.sort(axis=0)
    k = int(α /(P))
    Var =  SortedMatrixVarValue[k][0]
    somme = ((α /(P))-k)*(1/simulationNumber)
    for i in range(k,simulationNumber):
        somme = somme+SortedMatrixVarValue[k][0]*SortedMatrixVarValue[k][1]
    CVAR= somme/(1-α)
    return Var,CVAR
```

Les valeurs de la VaR et la CVaR de mon portefeuille avec 1000 simulations sont affichés ci-dessous :

S&P	Gov bonds	Small cap
0.75	0	0.25

Rendement R = 1.1%.

	α = 90%	α = 95%	α = 99%
VAR	0.35	0.36	0.41
CVAR	0.37	0.39	0.45

3. Simulations :

Plusieurs méthodes de simulations sont proposées. On peut citer la méthode Monte-Carlo ou la méthode Box Muller. Selon le contexte les méthodes de simulation varient. Ces méthodes avancées sont proposées lorsque nos scénarios ne sont pas équiprobables.

J'ai implémenté deux méthodes de simulation en python. Comme les scénarios sont équiprobable j'ai supposé une variation maximale pour chaque actif. Un actif ne peut pas dépasser les 100% de sa valeur actuelle. On peut notamment augmenter cette valeur si on veut, ou bien on peut fixer une valeur maximale pour tous les actifs. C'est d'onc l'utilisation de la loi uniforme continue entre (0, N). Certes, ceci n'est pas suffisant pour tous les portefeuilles. Mais, cette loi est suffisante pour le nôtre où les actifs ont des prix proches relativement.

Après mes recherches, j'ai découvert que les vecteurs (Ys) des scénarios ne contient pas forcements des probabilités entre [0,1], mais des valeurs probables des actifs donc les scénarios. Cette erreur m'a bloqué pendant un certain temps.

Par exemple pour un portefeuille composé de 3 actifs, où les prix des actifs sont [100,115,120]

Y(1) = [90,120,130], est le 1er scénario possible pour ce portefeuille.

Y(2) = [50,160,80], est le 2e scénario possible pour ce portefeuille… .

Dans le sujet du projet on a bien précisé que le vecteur (Ys) est un vecteur de probabilités. Dans le cours et dans la vraie vie, généralement, le vecteur (Ys) contient des simulations donc des variations ou des scénarios des prix futurs de l'actif. Mais, contenir des probabilités n'est pas faux et ceci déponds de notre fonction de risque. Donc comme demandée, mon vecteur (Ys) des scénarios contient des probabilités. On peut facilement changer ceci en augmentant le paramètre (0,1) à (0, N) dans les paramètres de la loi uniforme continue.

Dans le cours on suppose que tous les scénarios ont la même probabilité 1/S ou S est le nombre de simulations. Dans la vraie vie les simulations n'ont pas les mêmes probabilités et donc la façon de le générer est différente.

```
def GenNSim(n):
    p = []
    sommeAll=0
    sommeLast=0
    for i in range(n):
        value = np.random.uniform(0, 1, 1)[0]
        sommeAll = sommeAll + value
        p.append(value)
    for i in range(n-1):
        p[i]=p[i]/sommeAll
        p[i]=round(p[i],3)
        sommeLast=sommeLast+p[i]
    p[n-1]=round((1-sommeLast),3)

    return p

def GenerateSimulationMatrix(ArraySize,SimulationNumber):
    l = []
    f = open("Similations.txt", "w+")
    for i in range(1, SimulationNumber):
        k = GenNSim(ArraySize)
        l.append(k)
    f.write(str(l))
    f.write(";")
    f.close()
    return l
```

Figure 7 : Méthode de calcul de la CVaR et la VaR

4. Model minimisant la CVAR :

Pour minimiser la CVAR j'ai repris le modèle vu en cours.

$$1)\begin{cases} min\ \gamma + \frac{1}{(1-\alpha)S} \sum_{s=1}^{S} z_s \\ z_s \geq 0, s = 1, \dots, S \\ z_s \geq f(x, y_s) - \gamma,\ s = 1, \dots, S \\ x \in X. \end{cases}$$

Ce model nous retourne le portefeuille Markowitz X.

En optimisant ce problème, la valeur de γ trouvée représente la VaR. La valeur de la CVaR est la valeur finale de la fonction Objectif.

4.1 Implémentation du model en CPLEX :

Cette partie était relativement difficile. Je suis passée par une phase de documentation et de tests pour finalement avoir le model correct de la minimisation de la CVAR.

Le modèle est affiché ci-dessous.

```
float Rendements[numberActifs]=...;
float a = ...;
int s = ...;
range Simulations = 1..s;
float yProbabilites [Simulations][numberActifs]=...;
float bValues [numberActifs]=...;
/* DECISION VARIABLE ___________*/

dvar float+ x[1..Actifs];
dvar float k;
dvar float ZFunction[Simulations];
dexpr float Objective = k + (1/((1-a)*s)) * sum(s in Simulations) ZFunction[s];

/* OBJECTIF FUNCTION ___________*/
minimize  Objective;

/* CVAR MODEL CONSTRAINTES ___________*/
subject to {
  forall(s in Simulations){
    ZFunction[s]>=(sum(i in numberActifs) x[i]*(bValues[i]-yProbabilites[s][i]);
    ZFunction[s]>=0;
  }
  forall(i in numberActifs){
    x[i] >= 0;
    }
  sum(i in numberActifs)x[i]==1;
  sum(i in numberActifs) (x[i]*Rendements[i]) >= R;

}
float TotalReturn = sum(i in numberActifs) Rendements[i]*x[i];
float Varx = k;
float Cvar = Objective;
```

Figure 8: Modèle Cplex pour l'optimisation de la CVAR

4.1 Les données :

Les données sont :

- Le rendement R.
- Le pourcentage du risque α.
- Les valeurs futures du portefeuille.
- Les scénarios issus des simulations.
- Prix espérés des actifs du portefeuille.

S&P	Gov bonds	Small cap
0.2	0.6	0.3

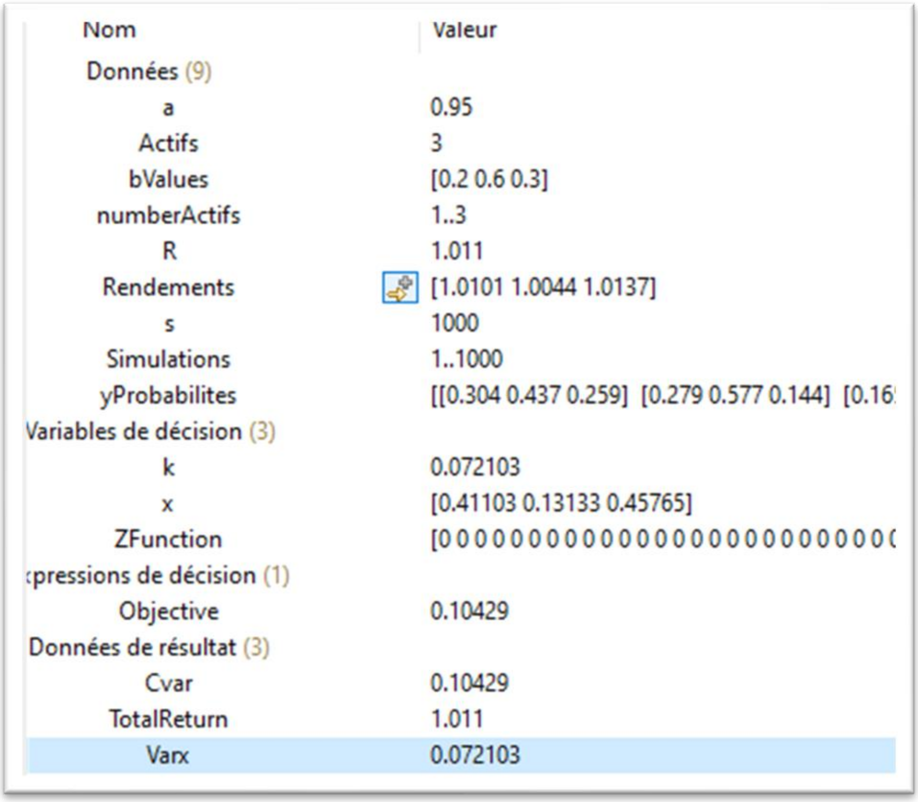
Figure 9: prix espérés des actifs dans le futur (b)

On prend α entre : 0.9 et 0.99.

On espère un rendement de 1 ,1%.

4.2 Les résultats :

Les résultats de la minimisation sont ci-dessous.



Nom	Valeur
Données (9)	
a	0.95
Actifs	3
bValues	[0.2 0.6 0.3]
numberActifs	1..3
R	1.011
Rendements	[1.0101 1.0044 1.0137]
s	1000
Simulations	1..1000
yProbabilites	[[0.304 0.437 0.259] [0.279 0.577 0.144] [0.16
Variables de décision (3)	
k	0.072103
x	[0.41103 0.13133 0.45765]
ZFunction	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
xpressions de décision (1)	
Objective	0.10429
Données de résultat (3)	
Cvar	0.10429
TotalReturn	1.011
Varx	0.072103

Figure 10: résultat de minimisation de la CVaR.

Pour α = 95% le portefeuille Markowitz trouvé est le suivant :

S&P	Gov bonds	Small cap
0.411	0.131	0.457

Figure 11: portefeuille Markowitz avec une CVaR minimal.

Avec un rendement total de : 1.1%.

Avec les même simulations précédentes (1000 simulations), les résultats sont comme suit :

	α = 90%	α = 95%	α = 99%
VAR	0.03	0.7	0.11
CVAR	0.07	0.10	0.14

Figure 12: résultats de la minimisation de la CVaR

4.3 Remarques :

On remarque une baisse dans les valeurs de la CVAR et la VAR par rapport aux valeurs du l'ancien modèle. Donc, le model a bien optimisé la CVaR et la VaR.

Pour s'assurer j'ai utilisé les mêmes données dans ma méthode python pour le calcul de la CVAR et la VAR et les données sont exactement les mêmes.

La précision de ces valeurs dépondra de la fonction de risque donc de multiples paramètres comme b, la simulation des probabilités.

En augmentant la valeur de α la valeur du risque donc, la CVaR augmente.

Le portefeuille généré est proche de notre portefeuille Markowiz initial. Ceci ne résume rien, mais comme le portefeuille est réel et les covariances sont réel une variance minimale entre les portefeuilles, implique dans ce cas moins de risque. Mais, ceci n'est pas valable pour tous les cas.

4.4 Variation des CVAR et VAR

Pour observer le comportement du model, J'ai minimisé mon portefeuille en variant les paramètres suivants :

- Le pourcentage α
- Le nombre de simulations

En tournant plusieurs fois le model de minimisation avec ces différentes valeurs j'ai eu les résultats suivants :

Avec α = 90% :

α	Nombre de simulations	S&P	Gov Bond	Small cap	Var	CVaR
90%	1000	0.47941	0.10504	0.41555	0.035	0.078
90%	5000	0.44761	0.11727	0.43513	0.029	0.066
90%	10000	0.4607	0.11223	0.42707	0.030	0.066
90%	20000	0.46115	0.11206	0.42679	0.03	0.06

Avec α = 95% :

α	Nombre de simulations	S&P	Gov Bond	Small cap	Var	CVaR
95%	1000	0.41103	0.13133	0.45765	0.072	0.10
95%	5000	0.43034	0.1239	0.44576	0.057	0.086
95%	10000	0.41164	0.13109	0.45727	0.059	0.087
95%	20000	0.45338	0.12866	0.45338	0.060	0.089

Avec α = 96% :

α	Nombre de simulations	S&P	Gov Bond	Small cap	Var	CVaR
96%	1000	0.38605	0.14093	0.47303	0.084	0.11
96%	5000	0.42775	0.1249	0.44735	0.065	0.093
96%	10000	0.39679	0.1368	0.46641	0.067	0.093
96%	20000	0.40519	0.13357	0.46124	0.068	0.095

Avec α = 97% :

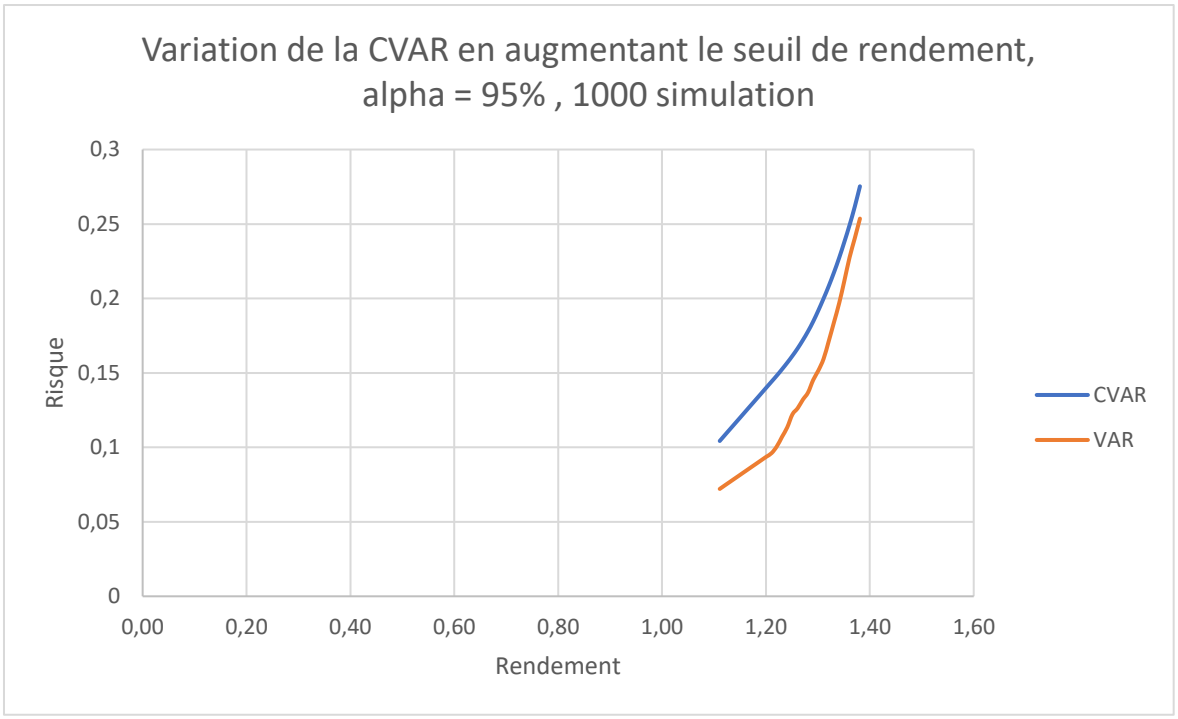
α	Nombre de simulations	S&P	Gov Bond	Small cap	Var	CVaR
97%	1000	0.37221	0.14625	0.48154	0.093	0.11
97%	5000	0.40792	0.13252	0.45956	0.076	0.10
97%	10000	0.37684	0.14447	0.4787	0.076	0.10
97%	20000	0.38074	0.14297	0.47629	0.078	0.10

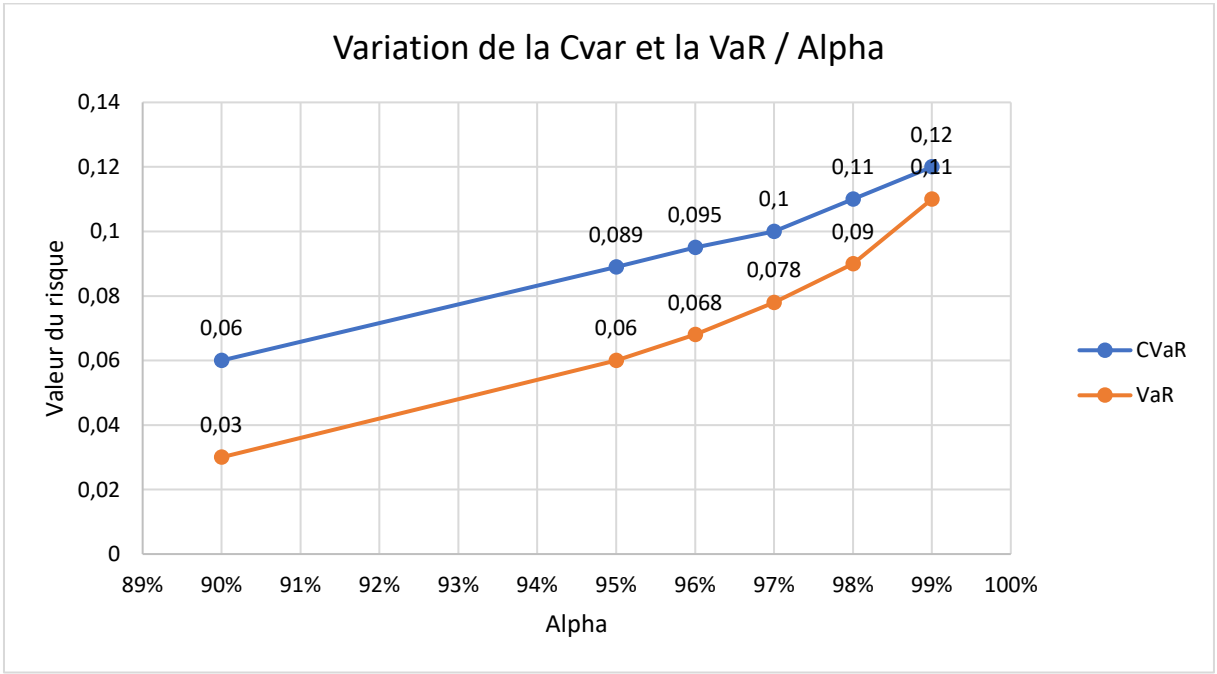
Avec α = 98% :

α	Nombre de simulations	S&P	Gov Bond	Small cap	Var	CVaR
98%	1000	0.32331	0.16504	0.51165	0.10	0.12
98%	5000	0.37353	0.14574	0.48073	0.087	0.10
98%	10000	0.33715	0.15972	0.50313	0.09	0.11
98%	20000	0.33709	0.15975	0.50317	0.09	0.11

Avec α = 99% :

α	Nombre de simulations	S&P	Gov Bond	Small cap	Var	CVaR
99%	1000	0.29757	0.17494	0.5275	0.11	0.14
99%	5000	0.32283	0.16523	0.51195	0.10	0.12
99%	10000	0.28727	0.17889	0.53383	0.10	0.12
99%	20000	0.29013	0.1778	0.53208	0.11	0.12





4.5. Conclusion :

Suivant ces résultats j'ai pu arriver aux conclusions suivantes :

- Notre model prouve que la VAR est toujours inferieure a la CVAR.
- En augmentant le nombre de simulation, on augmente nos chances d'être plus précis dans le calcul de la CVAR et la VAR.
- Si on veut avoir plus de rendement on est exposé à un risque de perte élevé.

5. Model maximisant le rendement en respectant un seuil de risque maximal :

Dans cette partie, on va implémenter un modèle qui considère une borne sur le risque tout en minimisant l'espérance de gains. Le model est le suivant :

$$\begin{cases} \max \sum_{i} \mu_i x_i \\ \gamma + \frac{1}{(1-\alpha_j)S} \sum_{s=1}^{S} z_s \leq U_{\alpha_j}, j = 1, \dots, J \\ z_s \geq 0, s = 1, \dots, S \\ z_s \geq f(x, y_s) - \gamma, \quad s = 1, \dots, S \\ x \in X \end{cases}$$

En optimisant ce problème, la valeur de γ trouvée représente la Var.

5.1. L'implémentation du model en Cplex :

L'implémentation du model en CPLEX est affichée dans la figure suivante :

```
float alphaValues [1..alphaNumber] = ...;
int s = ...;
range Simulations = 1..s;
float yProbabilites [Simulations][numberActifs]=...;
float bValues [numberActifs]=...;

/* DECISION VARIABLE ___________*/

dvar float+ x[1..Actifs];
dvar float k;
dvar float ZFunction[Simulations];
dexpr float objective = sum(i  in  numberActifs)Rendements[i]*x[i];


/* OBJECTIF FUNCTION ___________*/
maximize objective;


/* Constraintes ________*/
subject to {
  forall(j in 1..alphaNumber){
     k + (1/((1-alphaValues[j])*s)) * sum(s in Simulations) ZFunction[s] <= cVarSeuil[j];

  }

    forall(s in Simulations){
    ZFunction[s]>=(sum(i in numberActifs) x[i]*(bValues[i]-yProbabilites[s][i]))-k;
    ZFunction[s]>=0;
  }
       sum(i in numberActifs)x[i]== 1;
}

float TotalReturn = sum(i in numberActifs) Rendements[i]*x[i];
```

5.2. Les données :

Dans cette partie nous allons varier α entre 95% et 99% et pour chaque α on va indiquer le seuil de la CVaR à ne pas dépasser.

On suppose donc les données suivantes :

α	95%	96%	97%	98%	99%
CVaR	0.09	0.1	0.11	0.12	0.14

Prix espérés des actifs du portefeuille :

S&P	Gov bonds	Small cap
0.2	0.6	0.3

- Les simulation (mis dans le fichier des simulations)
- La fonction de risque Y= $f(x,y) = (b\text{-}y)^T x$, où b désigne les prix futurs de l'actif i, y désigne l'ensemble des scénario possibles et x l'allocation de l'actif dans le portefeuille de Markowitz.

5.3 Les résultats :

En appliquant ce model aux données mentionnées dans la partie (5.2), on arrive aux résultats suivants :

Les allocations du portefeuille sont :

S&P	Gov bonds	Small cap
0.37307	0.17212	0.4548

Value at risk (VaR)	0.077
Rendement total :	1.010

5.4 Varier le seuil de la CVAR :

On va donc varier les valeurs de la CVAR en mettant des valeurs supérieures aux valeurs mises dans les données et des valeurs inférieurs à celles mises dans les données et observer.

Variation avec des CVaR Supérieures :

α	95%	96%	97%	98%	99%
CVaR	0.12	0.13	0.14	0.15	0.16

On a comme résultats :

Les allocations du portefeuille sont :

S&P	Gov bonds	Small cap
0.38978	0.093782	0.51644

Value at risk (VaR)	0.11
Rendement total :	1.011

5.4.1 Variation avec des CVaR inférieurs :

α	95%	96%	97%	98%	99%
CVaR	0.06	0.07	0.08	0.09	0.1

Les allocations du portefeuille sont :

S&P	Gov bonds	Small cap
0.3549	0.26517	0.37994

Value at risk (VaR)	0.05
Rendement total :	1.00

5.4.2 Variation avec des CVaR assez supérieurs :

α	95%	96%	97%	98%	99%

CVaR	1	1.1	1.2	1.3	1.4

Les allocations du portefeuille sont :

S&P	Gov bonds	Small cap
0	0	1

Value at risk (VaR)	0.22
Rendement total :	1.0137058

Bilan

Annexes

- https://www.wikipedia.org/
- https://livre.fnac.com/mp6402460/Optimization-Methods-in-Finance-Mathematics-Finance-and-Risk
- https://fr.wikipedia.org/wiki/Mod%C3%A8le_binomial

- https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Monte-Carlo
- https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Box-Muller
- Optimization of conditional value-at-risk, Implemented in Portfolio Safeguard by AORDA.com
- http://appjar.info/
- https://github.com/jiangyinchen/Cplex-for-Markowitz-portfolio-optimization-/blob/master/OptimizationUsingCplex.ipynb
- https://www.ibm.com/support/knowledgecenter/SSSA5P_20.1.0/COS_KC_home.html