Ex1. Implémantation de la méthode treilli :
Ici, je vais implémenter cette méthode pour calculer l'option de vente. Rappel :

Pour une option Européenne de vente :

$v(k,j)$= p_u * $v(k+1,j+1)$ + p_d * $v(k+1,j)$ *1/R

Et $R=1+r$ où r est le taux sur une période.

V(N,j) = max {(C- u^j * d^(N-j) * S0), 0}

Pour une option American de vente :

V(k,j) = max { (p_u * $v(k+1,j+1)$ + p_d * $v(k+1,j)$ *1/R) , (C- u^j * d^(k-j) * S0) }

Je crée une classe treilli qui contient toutes les valeurs d'inputs : N périod, r taux, Strike K, proba up u, proba down d, Price S0. Ensuite cette classe permet de calculer les deux prix de l'option Européenne et Américaine selon les formulaires vu au cours.

```
class treilliVente():

    #N = 5 r = 0.04 u = 1.0425 d = 0.9592 K = 98 S0 = 100
    def __init__(self, N, r, u, d, K, S0):
        self.N = N
        self.r = r
        self.u = u
        self.d = d
        self.K = K
        self.S0 = S0
        self.R = 0
        self.p_u = 0
        self.p_d = 0
        self.probability()

    def probability(self):
        self.R=1+(self.r/52)
        self.p_u = (self.R-self.d)/(self.u-self.d)
        self.p_d = (self.u-self.R)/(self.u-self.d)
```

Ici, j'ai testé avec les données vu aux cours : N = 5, r = 0.04, u = 1.0425, d = 0.9592, K = 98, S0 = 100

```
def calcul_price_option(self):
    S = numpy.zeros([self.N+1,self.N+1])
    S[0,0] = self.S0
    for i in range(self.N+1):
        for j in range(i+1):
            S[i,j] = self.S0 * pow(self.u,j) * pow(self.d,(i-j))
    return S

def EuroVente(self):
    S = self.calcul_price_option()
    #R, p_u, p_d = probability(T,N,d,u,r)
    V = numpy.zeros([self.N+1,self.N+1])
    print(S[self.N, 3])
```

```python
    for j in range(self.N+1):
        V[self.N,j] = max(self.K-S[self.N, j], 0)

    for i in range(self.N-1,-1,-1):
        for j in range(i+1):
            V[i,j] = (self.p_u*V[i+1,j+1] + self.p_d*V[i+1,j])/self.R
    return V



def AmericanVente(self):
    S = self.calcul_price_option()
    V = numpy.zeros([self.N+1,self.N+1])

    for j in range(self.N+1):
        V[self.N,j] = max(self.K-S[self.N, j], 0)

    for i in range(self.N-1,-1,-1):
        for j in range(i+1):
            V[i,j] = max((self.p_u*V[i+1,j+1] + self.p_d*V[i+1,j])/self.R,
max(self.K-S[i,j],0))
    return V
```

Arbre de calcul et le prix de l'option Eu :

```
i= 4  V[ 4 , 0 ] =  13.348105822167028  V[ 4 , 1 ] =  5.99666421977598  V[ 4 , 2 ] =  0.0  V[ 4 , 3 ] =  0.0  V[ 4 , 4 ] =  0.0
i= 3  V[ 3 , 0 ] =  9.672073059409046  V[ 3 , 1 ] =  3.0018374813985216  V[ 3 , 2 ] =  0.0  V[ 3 , 3 ] =  0.0
i= 2  V[ 2 , 0 ] =  6.338547049575091  V[ 2 , 1 ] =  1.5026734755319764  V[ 2 , 2 ] =  0.0
i= 1  V[ 1 , 0 ] =  3.922282075540712  V[ 1 , 1 ] =  0.7522151309188664
i= 0  V[ 0 , 0 ] =  2.3385234949285376
Le prix de l'option Europeenne est:  2.3385234949285376
```

Arbre de calcul et le prix de l'option Ame :

```
i= 4  V[ 4 , 0 ] =  13.348105822167028  V[ 4 , 1 ] =  5.99666421977598  V[ 4 , 2 ] =  0.0  V[ 4 , 3 ] =  0.0  V[ 4 , 4 ] =  0.0
i= 3  V[ 3 , 0 ] =  9.747399731199977  V[ 3 , 1 ] =  3.0018374813985216  V[ 3 , 2 ] =  0.0  V[ 3 , 3 ] =  0.0
i= 2  V[ 2 , 0 ] =  6.37625441794588  V[ 2 , 1 ] =  1.5026734755319764  V[ 2 , 2 ] =  0.0
i= 1  V[ 1 , 0 ] =  3.9411578017031093  V[ 1 , 1 ] =  0.7522151309188664
i= 0  V[ 0 , 0 ] =  2.347972391882954
Le prix de l'option American est:  2.347972391882954
```

Tester avec les données plus grandes :

N = 300, r = 0.04, u = 1.0425, d = 0.9592, K = 98,   S0 = 100

```
Le prix de l'option Europeenne est:  15.368832269557247
Le prix de l'option American est:  18.309718847798987
```

N = 400, r = 0.04, u = 1.0366, d = 0.9646, K = 98,   S0 = 100

```
Le prix de l'option Europeenne est:  12.527886147209792
Le prix de l'option American est:  16.351984387576064
```

On voit que pour les même de données, l'option de Américaine peut donner le meilleur prix.

Ex2 :

Ex2:

Fonction de perte : $\gamma(x,y) = (b-y)^T x$

* Noté γ : gamma

1, Le problème $\min_{x \in X} CVaR(x)$ est approximé par

$$\min_{x \in X, \gamma \in R} \gamma + \frac{1}{(1-\alpha)S} \sum_{s=1}^{S} (\gamma(x, y_s) - \gamma)^+$$

Soit z_s une variable qui représente $\gamma(x, y_s) - \gamma)^+$. On a $z_s \geq 0$. Donc le problème est équivalent au problème d'optimisation :

Titre 1

$$(*) \begin{cases} \min \ \gamma + \frac{1}{(1-\alpha)S} \sum_{s=1}^{S} z_s \rightarrow (1) \\ z_s \geq 0 \text{ for } s = \{1, \dots, S\} \\ z_s \geq (b-y_s)^T . x - \gamma \\ \sum_{i=1}^{n} \mu_i x_i \geq R \\ \sum x_i = 1 \quad (i = \{1, \dots n\} \\ 0 \leq x_i \leq 1 \end{cases}$$

2, Pour résoudre le problème minimisation du modèle (*), en python nous pouvons utiliser scipy.optimize.linprog.

Cette fonction peut résoudre le problème de minimisation d'une fonction linéaire objective soumise à des contraintes linéaires d'égalité et d'inégalité

La programmation linéaire ~~sea~~ résout les problèmes de la forme suivante :

$$(**) \begin{cases} \min c^T x \ : \\ A_{ub} . x \leq b_{ub} \\ A_{eq} . x = b_{eq} \\ l \leq x \leq u \end{cases}$$

Dont : c : 1-Darray : les coefficients de la fonction objective linéaire à minimiser (lié à la fonction (1))

A-ub : 2-D array : La matrice de contraintes d'inégalité . Chaque ligne de A-ub spécifie les coefficients d'une contrainte d'inégalité linéaire sur x

~~B-ub~~

b-ub : 1-D array : Le vecteur de contrainte d'inégalité . chaque élément représente une limite supérieure sur la valeur correspondante de A-ub . x

A-eq : 2-D array . La matrice de contraintes d'égalité - Chaque ligne de A-eq spécifie les coefficients d'une contraintes d'égalité linéaire sur x

b-eq : le vecteur de contrainte d'égalité relié à A-eq . x = b-eq

On doit transformer le modèle (*) vers (**) pour pouvoir utiliser la fonction linprog en python .

Pour ~~calculer~~ les prix de l'option , ~~ope~~ on peut utiliser la fonction calcul-option-price ( ) vu dans la question 1 , qui est calculé par la formule : $s[i,j] = S_0 * u^j * d^{i-j}$

J'ai testé avec 4 options, calculé leurs prix en utilisant le principe de la méthode treilli.

```
Résultat pour S = 4 #scenario
assets.append(asset(N= S, r= 0.04, u=1.0425, d=0.9592, S0=100))
assets.append(asset(N=S, r=0.04, u=1.0325, d=0.9685, S0=100))
assets.append(asset(N=S, r=0.04, u=1.0389, d=0.9625, S0=100))
assets.append(asset(N=S, r=0.04, u=1.0752, d=0.93, S0=100))

uj = [0.11, 0.12, 0.15, 0.14]
alpha = 0.95
R = 0.14
```

```
result :
calcul price en periode de l'option 1 par example :
[[100.           0.           0.           0.           0.        ]
 [ 95.92       104.25         0.           0.           0.        ]
 [ 92.006464    99.9966     108.680625     0.           0.        ]
 [ 88.25260027  95.91673872 104.2464555  113.29955156   0.        ]
 [ 84.65189418  92.00333578  99.99320012 108.67692986 118.1147825 ]]
calcul price en periode de l'option 2 par example :
[[100.           0.           0.           0.           0.        ]
 [ 96.85       103.25         0.           0.           0.        ]
 [ 93.799225    99.997625   106.605625     0.           0.        ]
 [ 90.84454941  96.84769981 103.24754781 110.07030781   0.        ]
 [ 87.98294611  93.79699727  99.99525006 106.60309312 113.64759282]]
VaR:  1.0810539608335135e-13
CVaR:  5.138422035761411e-13
Portfolio : [0.24149092 0.22496668 0.23441595 0.29912645]
```