Master 1 Data-Sciences

OPTIMISATION LINÉAIRE

PROJET

Mathis CORDIER

Mai 2019

Table des matières

Exercice 1: Commun à tous les projets

On considère un problème de minimisation sous forme standard :

$$\begin{cases} \min\ f(x) = \sum_{j=1}^{n} c_j x_j = {}^t\boldsymbol{c}\boldsymbol{x} \\ \sum_{j=1}^{n} a_{ij} x_j \leq b_i,\ i = 1, \cdots, r \Leftrightarrow A\boldsymbol{x} \leq \boldsymbol{b} \\ x_j \geq 0, j = 1, \cdots, n \Leftrightarrow \boldsymbol{x} \geq 0. \end{cases}$$

où $\boldsymbol{x} = {}^t(x_1, \cdots, x_n) \in \mathbb{R}^n$, $\boldsymbol{c} = {}^t(c_1, \cdots, c_n) \in \mathbb{R}^n$, $\boldsymbol{b} = {}^t(b_1, \cdots, b_r) \in \mathbb{R}^r$, $A \in \mathcal{M}_{r,n}(\mathbb{R})$.
Ecrire une fonction Python `FourierMotzkin(A,b,c)` qui retourne la liste $(\min f, x_1, \cdots, x_n)$ par méthode de Fourier-Motzkin.

Exercice 2: Problème de mélange

Une raffinerie produit de l'essence e_1 (SP98) et e_2 (SP95) à partir de 3 composants c_1 (brut1), c_2 (brut2) et c_3 (brut 3). Les limitations sont de :

	Production par jour
c_1	30000 barils/jour
c_2	20000 barils/jour
c_3	10000 barils/jour

Certaines proportions doivent être respectées dans la production de SP95 et SP98 :

	Part de c_1	Part de c_2	Part de c_3
SP95	au plus 50%	au moins 10%	
SP98	au plus 30%	au moins 40%	au moins 50%

Les prix d'achat par baril sont de :

	Prix d'achat par baril
c_1	36 euros
c_2	48 euros
c_3	60 euros

Les prix de vente aux distributeurs par baril sont de :

	Prix de vente par baril
SP95	54 euros
SP98	66 euros

La demande maximum des stations de distribution est de :

	Demande maximum
SP95	15000 barils
SP98	35000 barils

1. Quelle est la structure de production journalière qui maximise la marge d'exploitation ?
2. On souhaite maintenir l'activité simultanément sur SP98 et SP95. Pour cela, on veut maximiser la marge la plus faible sur les deux produits. Calculer cette marge maximale.

Remarque : pour les calculs numériques, vous pouvez-devez utiliser la fonction `optimize.linprog` du module `scipy`.

EXERCICE 1

Nous souhaitons créer une fonction python réalisant la méthode de Fourier-Motzkin pour minimiser une fonction affine $f(x) = c^T x$ sur un polyèdre défini par $Ax \leqslant b$ et $x \geqslant 0$.

Les paramètres à saisir à l'appel de la fonction seront les suivants :

- $A \in \mathcal{M}_{p,q}(\mathbb{R})$
- $b \in \mathbb{R}^p$, $c \in \mathbb{R}^q$
- $x \in \mathbb{R}^q$

Ces éléments seront saisis comme des arrays de la libraire numpy de python.

```
import numpy as np
```

Les fonctions FM numérotées sont faite de façon à se succéder.

I) Une première fonction $FM1$ a pour but de joindre le vecteur colonne $-b$ à droite la matrice A :

$$FM1 : \mathcal{M}_{p,q}(\mathbb{R}) \times \mathbb{R}^p \longrightarrow \mathcal{M}_{p,q+1}(\mathbb{R})$$
$$(A, b) \longmapsto \left[\begin{array}{c|c} A & -b \end{array}\right]$$

```
def FM1(A,b):  # A,b de type np.array
    B = np.copy(A)
    c = -np.copy(b)
    c = c.reshape(len(c),1)
    B = np.concatenate((B,c),axis=1)
    return B
```

II) $FM2$ a ensuite pour but de ramener les coefficients de la première colonne à 0 s'ils sont nuls, -1 s'ils sont strictement négatifs et 1 s'ils sont strictement positifs :

```
def FM2(B):  # B de type np.array
    C = np.copy(B)
    (p,q) = np.shape(C)
    for i in range(p):
        a = abs(C[i,0])
        if a>0:
            C[i,:] = C[i,:]/a
    return C
```

III) $FM3$ a pour but d'éliminer la variable x_1 dans l'algorithme de Fourier-Motzkin. Pour cela, elle traite différemment les lignes dont la valeur sur la première colonne est 0, 1 ou -1. En effet, un pivot égal à -1 signifie que la ligne impose une majoration sur x_1 et inversement une minoration quand le pivot est égal à 1. De plus, lorsque le pivot vaut 0, la variable x_1 n'est pas concernée par la contrainte, alors nous conservons cette ligne dans la matrice finale. Une fois tout cela identifié, nous obtenons une matrice pour les pivots égaux à 1 et une autre pour les pivots égaux à -1.

Nous voulons alors réunir ces deux matrices et cela correspond à l'étape de l'algorithme :

$$\max(h_1(x), \dots, h_{n_1}(x)) \leqslant x_1 \leqslant \min(H_1(x), \dots, H_{n_2}(x))$$

Cela revient à minorer un à un les h_i par tous les H_i. On obtient alors la matrice finale en ajoutant les lignes dont le pivot était 0 (dont on tronque le 0). Nous avons alors éliminer x_1 et la matrice a donc perdu une dimension.

L'élimination de x_1 entraîne alors la perte d'une colonne, nous avons donc :

$$FM3 : \mathcal{M}_{p,q}(\mathbb{R}) \longrightarrow \mathcal{M}_{p,q-1}(\mathbb{R})$$
$$A \longmapsto B$$

Où en entrée :

$$A = \left[\begin{array}{c|c} -1 & \\ \vdots & B_1 \\ -1 & \\ \hline 1 & \\ \vdots & B_2 \\ 1 & \\ \hline 0 & \\ \vdots & B_3 \\ 0 & \end{array}\right]$$

Avec $B_1 \in \mathcal{M}_{p-1,k_1}(\mathbb{R})$, $B_2 \in \mathcal{M}_{p-1,k_2}(\mathbb{R})$ et $B_3 \in \mathcal{M}_{p-1,k_3}(\mathbb{R})$

Nous noterons $B_i^{(j)}$ la j-ème ligne de la matrice B_i pour la suite.

La fonction renvoie :

$$B = \left[\begin{array}{c} B_1^{(1)} - B_2^{(1)} \\ \vdots \\ B_1^{(1)} - B_2^{(k_2)} \\ \hline \vdots \\ \hline B_1^{(k_1)} - B_2^{(1)} \\ \vdots \\ B_1^{(k_1)} - B_2^{(k_2)} \\ \hline B_3 \end{array}\right]$$

```
def FM3(C):  # C a le type np.array
    (p,q) = np.shape(C)
    G,D,E = np.array([np.ones(q)]),np.array([np.ones(q)]),np.array([np.ones(q)])
    for i in range(p):
        x = np.copy(C[i])
        if C[i][0] < .0:
            G = np.concatenate((G,[x]),axis=0)
        elif C[i][0] > .0:
            x=-x
            D = np.concatenate((D,[x]),axis=0)
        else:
            E = np.concatenate((E,[x]),axis=0)

    G = np.delete(G,0,0) # On elimine la premiere ligne.
    D = np.delete(D,0,0)
    E = np.delete(E,0,0)

    G = np.delete(G,0,1) # On elimine la premiere colonne
    D = np.delete(D,0,1)
    E = np.delete(E,0,1)

    (pg,qg) = np.shape(G)
    (pd,qd) = np.shape(D)
    if pd == 0:  # si pd=0, D est vide
        return E
    elif pg == 0:  # si pg=0, G est vide
        return E
    else:
        for k in range(pg):
            for l in range(pd):
                x = G[k,:]-D[l,:]
                E = np.concatenate((E,[x]),axis=0)

    return E
```

IV) $FM4$ répète l'élimination des x_i jusqu'à obtenir l'intervalle dans lequel se situe le dernier x_i soit $x_{q+1} = f(x)$.

Nous conservons les matrices résultants des éliminations successives des x_i de manière a pouvoir remonter simplement ensuite pour trouver le vecteur x dont nous obtiendrons le minimum.

En effet, $FM4$ suffit pour obtenir le minimum recherché mais il ne donne pas le vecteur pour lequel ce minimum est atteint.

```
# On calcule ici l'intervalle dans lequel se situe le dernier élément de x
def FM4(A,b):
    E = FM1(A,b)  # B : Concatenation de A et -b
    E = FM2(E)    # C : Première colonne composée de 0,1 et -1
    (p,q) = np.shape(E)

    Lx = []

    for k in range(q-2):
        E = FM3(E)
        E = FM2(E)
        Lx.append(E)

    (p,q) = np.shape(E)
    mini = -np.inf
    maxi = np.inf
    for k in range(p):
        if E[k,0]<0:
            mini = max(mini, E[k,1])
        elif E[k,0]>0:
            maxi = min(maxi, -E[k,1])

    if mini>maxi : raise Exception("Polyèdre vide")

    return mini,Lx
```

V) La fonction de remontée du système a pour but de récupérer les valeurs des x_i pour lequel le minimum est atteint. Pour cela, nous utilisons les matrices d'élimination successives. Grâce à l'obtention des bornes de x_{q+1}, nous obtenons des majorations et des minorations sur x_q.

De manière générale, le minimum étant atteint simplement sur un sommet du polyèdre, nous obtiendrons une valeur unique pour x_q car les bornes inférieure et supérieure obtenues seront égales. Cependant, dans des cas particuliers, le minimum peut être atteint sur une arête, nous choisissons alors de prendre le sommet pour lequel le x_i concerné est minimal.

Nous répétons ensuite cette opération jusqu'à obtenir tous les éléments de x.

```
# On remonte ici le système en utilisant les matrices E consécutives de FM4
def remontee(L,c):
    mini,vect = L
    X = []
    X0 = mini
    vect.reverse()
    Z = np.array([[1.,0],[0,X0],[0,1]])
    for mat in vect[1:]:
        tmp = np.dot(mat,Z)
        bmax = np.inf
        bmin = -np.inf
        for ligne in tmp:
            if ligne[0]==-1:
                bmin = max(bmin,ligne[1])
            elif ligne[0]==1:
                bmax = min(bmax,-ligne[1])
        X0 = bmin
        Z = np.vstack((Z[0],np.array([0,X0]),Z[1:]))
        X.append(X0)
    X.reverse()
    X = np.hstack(( (mini - np.dot(c[1:],X)) / c[0], X))
    return X
```

VI) Pour finir, la fonction FM est la fonction réalisant l'algorithme de Fourier-Motzkin. Cette fonction rajoute la valeur de $f(x) = c^T x$ comme dernier paramètre du vecteur x, c'est celui qui sera minimisé.

```
def FM(A,b,c):
    # On rajoute un paramètre caractérisant la fonction <c,x> dans A
    # On minimise alors le dernier paramètre
    # Il correspond à la valeur de la fonction
    d = np.concatenate((np.zeros(len(A)),[1,-1]))
    A = np.vstack((A,-c,c))
    A = np.vstack((A.T,d)).T
    b = np.concatenate((b,[0,0]))
    res = FM4(A,b)
    mini_vect = remontee(res,c)
    return (round(res[0],3) , mini_vect)
```

EXERCICE 2

1. Maximisation de la marge journalière

Dans un premier temps nous allons expliciter les notations que l'on utilisera par la suite. Nous noterons A la matrice des contraintes sur le vecteur x. Le nombre de lignes de A est le nombre de contraintes qu'elle implique et son nombre de colonnes est la taille du vecteur x. Nous regarderons un système tel que $Ax \leqslant b$ où b est un vecteur de la taille du nombre de lignes de A.

Nous avons ici 5 variables à prendre en compte : le nombre de barils de SP98, de SP95, de Brut C1, de Brut C2 et de Brut C3. Les produits SP98 et SP95 sont entièrement produits à partir des produits C1, C2 et C3.

Nous avons toutes les contraintes de production suivantes :

- La production journalière est de :
  - C1 : 30000 barils
  - C2 : 20000 barils
  - C3 : 10000 barils
- Le demande journalière maximale est de :
  - SP98 : 35000 barils
  - SP95 : 15000 barils
- Le prix d'achat des barils est de :
  - C1 : 36 €
  - C2 : 48 €
  - C3 : 60 €
- Le prix de vente des barils est de :
  - SP98 : 66 €
  - SP95 : 54 €

- La part de brut dans le SP98 est de :
  - C1 : Au plus 30
  - C2 : Au moins 40
  - C3 : Au moins 50
- La part de brut dans le SP95 est de :
  - C1 : Au plus 50
  - C2 : Au moins 10
  - C3 : 0

Par la suite, nous aurons besoin de décomposer les variables des bruts C1 et C2 chacune en deux pour pouvoir s'adapter aux contraintes ci-dessus. Nous aurons alors pour chacun des deux bruts une variable correspondant au nombre de barils de ce brut servant à la production du SP98 et une autre correspondant au nombre de barils de ce brut servant à la production du SP95.
C3 lui n'est pas concerné car il fait uniquement parti de la composition du SP98.

Nous avons donc 7 variables que nous représenterons de la manière suivante :

$$x = \begin{bmatrix} e_{98} \\ e_{95} \\ c_1^{(98)} \\ c_1^{(95)} \\ c_2^{(98)} \\ c_2^{(95)} \\ c_3 \end{bmatrix} = \begin{bmatrix} \text{Nombre de barils de SP98 produits} \\ \text{Nombre de barils de SP95 produits} \\ \text{Nombre de barils de C1 consommés pour du SP98} \\ \text{Nombre de barils de C1 consommés pour du SP95} \\ \text{Nombre de barils de C2 consommés pour du SP98} \\ \text{Nombre de barils de C2 consommés pour du SP95} \\ \text{Nombre de barils de C3 consommés} \end{bmatrix}$$

Nous noterons alors $c_1 = c_1^{(98)} + c_1^{(95)}$ et $c_2 = c_2^{(98)} + c_2^{(95)}$

Nous souhaitons maximiser la marge d'exploitation journalière de l'entreprise. Cette marge est simplement calculée de la manière suivante :

$$M = [\text{ Total des ventes }] - [\text{ Total des achats }]$$

La fonction qui correspond à cette marge maximale est alors :

$$M(x) = 66e_{98} + 54e_{95} - 36c_1 - 48c_2 - 60c_3$$

Cela revient en fait à minimiser la fonction suivante :

$$\begin{aligned} f(x) &= -66e_{98} - 54e_{95} + 36c_1 + 48c_2 + 60c_3 \\ &= -66e_{98} - 54e_{95} + 36[c_1^{(98)} + c_1^{(95)}] + 48[c_2^{(98)} + c_2^{(95)}] + 60c_3 \\ &= -66e_{98} - 54e_{95} + 36c_1^{(98)} + 36c_1^{(95)} + 48c_2^{(98)} + 48c_2^{(95)} + 60c_3 \\ &= c^T x \end{aligned}$$

Où $c = \begin{bmatrix} -66 \\ -54 \\ 36 \\ 36 \\ 48 \\ 48 \\ 60 \end{bmatrix}$ ← *Prix de vente du SP98*
← *Prix de vente du SP95*
Prix d'achat du C1
P rix d'achat du C2
← *Prix d'achat du C3*

Nous voulons maintenant résumer les conditions de production sous la forme d'un système composé des 7 variables explicitées précédemment. On mettra par la suite ce système sous forme $Ax \leqslant b$.
Ce système formera alors un polyèdre de $\mathbb{R}^7$ sur lequel nous chercherons à minimiser la droite définie par $f(x)$.

Nous partons du principe que les variables sont toutes positives et nous ne le préciserons pas dans le système. En effet, nous le spécifierons dans python lors de l'appel à la fonction "linprog" de scipy.optimize.

Nous avons alors le système de contraintes linéaires suivant :

$$
\begin{cases}
e_{98} \leqslant 35000 \\
e_{95} \leqslant 15000 \\
c_1^{(98)} + c_1^{(95)} \leqslant 30000 \\
c_2^{(98)} + c_2^{(95)} \leqslant 20000 \\
c_3 \leqslant 10000 \\
e_1 = c_1^{(98)} + c_2^{(98)} + c_3 \\
e_2 = c_1^{(95)} + c_2^{(95)} \\
0.3e_{98} \geqslant c_1^{(98)} \\
0.4e_{98} \leqslant c_2^{(98)} \\
0.5e_{98} \leqslant c_3 \\
0.5e_{95} \geqslant c_1^{(95)} \\
0.1e_{95} \leqslant c_2^{(95)}
\end{cases}
\iff
\begin{cases}
e_{98} \leqslant 35000 \\
e_{95} \leqslant 15000 \\
c_1^{(98)} + c_1^{(95)} \leqslant 30000 \\
c_2^{(98)} + c_2^{(95)} \leqslant 20000 \\
c_3 \leqslant 10000 \\
e_1 - c_1^{(98)} - c_2^{(98)} - c_3 \leqslant 0 \\
-e_1 + c_1^{(98)} + c_2^{(98)} + c_3 \leqslant 0 \\
e_2 - c_1^{(95)} - c_2^{(95)} \leqslant 0 \\
-e_2 + c_1^{(95)} + c_2^{(95)} \leqslant 0 \\
-0.3e_{98} + c_1^{(98)} \leqslant 0 \\
0.4e_{98} - c_2^{(98)} \leqslant 0 \\
0.5e_{98} - c_3 \leqslant 0 \\
-0.5e_{95} + c_1^{(95)} \leqslant 0 \\
0.1e_{95} - c_2^{(95)} \leqslant 0
\end{cases}
$$

On obtient alors de manière explicite :

$$
A = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & -1 & 0 & -1 & 0 & -1 \\
-1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & -1 & 0 & -1 & 0 \\
0 & -1 & 0 & 1 & 0 & 1 & 0 \\
-0.3 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & -0.5 & 0 & 1 & 0 & 0 & 0 \\
0.4 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0.1 & 0 & 0 & 0 & -1 & 0 \\
0.5 & 1 & 0 & 0 & 0 & 0 & -1
\end{bmatrix}
$$

Contraintes de demande maximale par jour

Contraintes de production maximale par jour

Contraintes de fabrication de e_{98} sur les bruts

Contraintes de fabrication de e_{95} sur les bruts

Contraintes des taux de c_1, c_2, c_3 dans e_{98}, e_{95}

$$b = \begin{bmatrix} 35000 \\ 15000 \\ 30000 \\ 20000 \\ 10000 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \leftarrow \textit{ Demande maximale en SP98} \\ \leftarrow \textit{ Demande maximale en SP95} \\ \leftarrow \textit{ Production maximale de C1} \\ \leftarrow \textit{ Production maximale de C2} \\ \leftarrow \textit{ Production maximale de C3} \end{matrix} \quad \text{et} \quad c = \begin{bmatrix} -66 \\ -54 \\ 36 \\ 36 \\ 48 \\ 48 \\ 60 \end{bmatrix}$$

Nous avons $A \in \mathcal{M}_{14\times 7}(\mathbb{R})$, $b \in \mathbb{R}^{14}$, $c \in \mathbb{R}^7$ et $x \in \mathbb{R}^7$. Les opérations $Ax \leqslant b$ et $c^T x$ sont alors bien définies. Nous pouvons aussi dire qu'il y a 14 contraintes linéaires sur x.

Maintenant que nous avons explicité la matrice A et les vecteurs x, b et c, nous pouvons demander à la fonction "linprog" de scipy.optimize de résoudre notre problème de minimisation. Nous indiquons dans le paramètre "bounds" que nous souhaitons que toutes les valeurs du vecteur x soient positives.
(Voir saisie en annexe)

```
res = linprog ( c, A_ub=A, b_ub=b, bounds = (0, None) )
```

Nous obtenons alors :

```
res.fun = - 444 000.0
```

```
res.x = [20000., 15000., 2000., 7500., 8000., 7500., 10000.]
```

Alors $\min_x f(x) = -444000$ et il est atteint pour $x_{max} = \begin{bmatrix} 20000 \\ 15000 \\ 2000 \\ 7500 \\ 8000 \\ 7500 \\ 10000 \end{bmatrix} \begin{matrix} \leftarrow e_{98} \\ \leftarrow e_{95} \\ \leftarrow c_1^{(98)} \\ \leftarrow c_1^{(95)} \\ \leftarrow c_2^{(98)} \\ \leftarrow c_2^{(95)} \\ \leftarrow c_3 \end{matrix}$

Alors **la marge journalière maximale est de** **444 000 €** sous les contraintes de production suivantes :

- **Approvisionnement :**
  - C1 : 9500 barils
    - 2000 pour la production de SP98
    - 7500 pour la production de SP95
  - C2 : 15500 barils
    - 8000 pour la production de SP98
    - 7500 pour la production de SP95
  - C3 : 10000 barils
- **Production :**
  - SP98 : 20000 barils
    - 2000 de C1
    - 8000 de C2
    - 10000 de C3
  - SP95 : 15000 barils
    - 7500 de C1
    - 7500 de C2

2. Maximisation de la plus petite marge journalière sur les deux essences

Dans cette partie nous voulons maximiser la plus petite des marges faite sur le SP98 et le SP95. Nous notons m_{98} la marge faite sur le SP98 et m_{95} la marge faite sur le SP95. Nous cherchons alors a maximiser m où $m = \min\{m_{98}, m_{95}\}$.
Cela revient à maximiser m avec :

$$\begin{cases} m & \leqslant & m_{98} \\ m & \leqslant & m_{95} \end{cases}$$

De plus, nous pouvons exprimer m_{98} et m_{95} en fonction des variables de la question précédente : e_{98}, e_{95}, $c_1^{(98)}$, $c_1^{(95)}$, $c_2^{(98)}$, $c_2^{(95)}$ et c_3 :

$$\begin{cases} m_{98} & = & 66e_{98} - 36c_1^{(98)} - 48c_2^{(98)} - 60c_3 \\ m_{95} & = & 54e_{95} - 36c_1^{(95)} - 48c_2^{(95)} \end{cases}$$

Nous avons donc :

$$\begin{cases} m & \leqslant & 66e_{98} - 36c_1^{(98)} - 48c_2^{(98)} - 60c_3 \\ m & \leqslant & 54e_{95} - 36c_1^{(95)} - 48c_2^{(95)} \end{cases}$$

C'est sous ces contraintes que nous chercherons à maximiser m.

Nous allons maintenant modifier la matrice A pour y ajouter les nouvelles contraintes. Nous remarquons aussi que nous ne maximisons plus la même fonction et que les dimensions changent également car nous rajoutons un paramètre. En effet :

Nous voulons maximiser $M'(x') = c'^T x'$ où $\quad c' = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}$ et $x' = \begin{bmatrix} \boxed{x} \\ m \end{bmatrix} = \begin{bmatrix} e_{98} \\ e_{95} \\ c_1^{(98)} \\ c_1^{(95)} \\ c_2^{(98)} \\ c_2^{(95)} \\ c_3 \\ m \end{bmatrix}$

Comme dans le premier exercice, nous voulons maximiser m, ce qui revient à minimiser $-m$, c'est pour cela que nous mettons -1 comme dernier élément de c'.
Nous avons bien $g(x') = -M'(x') = -c'^T x' = -m$

$$A' = \left(\begin{array}{ccccccc|c} & & & A & & & & 0 \\ \hline -66 & 0 & 36 & 0 & 48 & 0 & 60 & 1 \\ 0 & -54 & 0 & 36 & 0 & 48 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 1 & 0 & 0 \\ -0.3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0.4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0.5 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ \hline -66 & 0 & 36 & 0 & 48 & 0 & 60 & 1 \\ 0 & -54 & 0 & 36 & 0 & 48 & 0 & 1 \end{array}\right)$$

De même pour b', $b' = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 35000 \\ 15000 \\ 30000 \\ 20000 \\ 10000 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}$ *9+2 = 11 fois*

Nous pouvons alors appliquer la fonction ”linprog” sur A', b' et c'.

Nous avons $A' \in \mathcal{M}_{16\times 8}(\mathbb{R})$, $b' \in \mathbb{R}^{16}$, $c' \in \mathbb{R}^8$ et $x' \in \mathbb{R}^8$. Les opérations $A'x' \leqslant b'$ et $c'^T x'$ sont alors bien définies. Nous pouvons aussi dire qu'il y a 16 contraintes linéaires sur x'.

Maintenant que nous avons explicité la matrice A' et les vecteurs x', b' et c', nous pouvons demander à la fonction "linprog" de scipy.optimize de résoudre notre problème de minimisation. Nous indiquons dans le paramètre "bounds" que nous souhaitons que toutes les valeurs du vecteur x' soient positives.
(Voir saisie en annexe)

```
res = linprog ( c', A_ub=A', b_ub=b', bounds = (0, None) )
```

Nous obtenons alors :

```
res.fun = - 180 000.0
res.x = [ 13636.36363636, 15000., 1363.63636364, 7500., 5454.54545455, 7500., 6818.18181818, 180000. ]
```

Alors $\min_{x'} f(x') = -180000$ et il est atteint pour $x'_{max} = \begin{bmatrix} 13636.36 \\ 15000 \\ 1363.64 \\ 7500 \\ 5454.55 \\ 7500 \\ 6818.18 \\ 180000 \end{bmatrix} \begin{matrix} \leftarrow e_{98} \\ \leftarrow e_{95} \\ \leftarrow c_1^{(98)} \\ \leftarrow c_1^{(95)} \\ \leftarrow c_2^{(98)} \\ \leftarrow c_2^{(95)} \\ \leftarrow c_3 \\ \leftarrow m \end{matrix}$

On arrive donc à une marge journalière de 180 000 € en voulant maximiser la plus petite des deux marges faite sur le SP98 et SP95.
Nous avons alors $m_{98} = m_{95} = 180000$ €.
Or $m = m_{98} + m_{95} = 2 \times 180000 = 360000$ €.
Alors **la marge totale journalière sera de** **360 000 €**.

Nous remarquons que le mode de production de l'essence SP95 n'a pas changé, les changements se sont faits au niveau de la production de SP98. En effet, par ce mode de calcul nous arrivons à une baisse de la production de l'essence SP98 et donc également une baisse de la marge d'exploitation et de consommation des bruts.
Cette maximisation de la plus petite des deux marges est donc atteinte sous les contraintes de production suivantes :

- **Approvisionnement :**
  - C1 : 8863.64 barils
    - 1363.64 pour la production de SP98
    - 7500 pour la production de SP95
  - C2 : 12954.55 barils
    - 5454.55 pour la production de SP98
    - 7500 pour la production de SP95
  - C3 : 6818.18 barils
- **Production :**
  - SP98 : 13636.36 barils
    - 1363.64 de C1
    - 5454.54 de C2
    - 6818.18 de C3
  - SP95 : 15000 barils
    - 7500 de C1
    - 7500 de C2

ANNEXE

EXERCICE 1 : Code brut

```
import numpy as np

def FM1(A,b):  # A,b de type np.array
    B = np.copy(A)
    c = -np.copy(b)
    c = c.reshape(len(c),1)
    B = np.concatenate((B,c),axis=1)
    return B

def FM2(B):  # B de type np.array
    C = np.copy(B)
    (p,q) = np.shape(C)
    for i in range(p):
        a = abs(C[i,0])
        if a>0:
            C[i,:] = C[i,:]/a
    return C

def FM3(C):  # C a le type np.array
    (p,q) = np.shape(C)
    G,D,E = np.array([np.ones(q)]),np.array([np.ones(q)]),np.array([np.ones(q)])
    for i in range(p):
        x = np.copy(C[i])
        if C[i][0] < .0:
            G = np.concatenate((G,[x]),axis=0)
        elif C[i][0] > .0:
            x=-x
            D = np.concatenate((D,[x]),axis=0)
        else:
            E = np.concatenate((E,[x]),axis=0)
    G = np.delete(G,0,0) # On elimine la premiere ligne.
    D = np.delete(D,0,0)
    E = np.delete(E,0,0)
    G = np.delete(G,0,1) # On elimine la premiere colonne
    D = np.delete(D,0,1)
    E = np.delete(E,0,1)
    (pg,qg) = np.shape(G)
    (pd,qd) = np.shape(D)
    if pd == 0:  # si pd=0, D est vide
        return E
    elif pg == 0:  # si pg=0, G est vide
        return E
    else:
        for k in range(pg):
            for l in range(pd):
                x = G[k,:]-D[l,:]
                E = np.concatenate((E,[x]),axis=0)
    return E
# On calcule ici l'intervalle dans lequel se situe le dernier élément de x
def FM4(A,b):
    E = FM1(A,b)  # B : Concatenation de A et -b
    E = FM2(E)    # C : Première colonne composée de 0,1 et -1
    (p,q) = np.shape(E)
    Lx = []
    for k in range(q-2):
        E = FM3(E)
        E = FM2(E)
        Lx.append(E)
    (p,q) = np.shape(E)
    mini = -np.inf
    maxi = np.inf
    for k in range(p):
        if E[k,0]<0:
            mini = max(mini, E[k,1])
        elif E[k,0]>0:
            maxi = min(maxi, -E[k,1])
    if mini>maxi : raise Exception("Polyèdre vide")
    return mini,Lx
```

```
# On remonte ici le système en utilisant les matrices E consécutives de FM4
def remontee(L,c):
    mini,vect = L
    X = []
    X0 = mini
    vect.reverse()
    Z = np.array([[1.,0],[0,X0],[0,1]])
    for mat in vect[1:]:
        tmp = np.dot(mat,Z)
        bmax = np.inf
        bmin = -np.inf
        for ligne in tmp:
            if ligne[0]==-1:
                bmin = max(bmin,ligne[1])
            elif ligne[0]==1:
                bmax = min(bmax,-ligne[1])
        X0 = bmin
        Z = np.vstack((Z[0],np.array([0,X0]),Z[1:]))
        X.append(X0)
    X.reverse()
    X = np.hstack(( (mini - np.dot(c[1:],X)) / c[0], X))
    return X

def FM(A,b,c):
    # On rajoute un paramètre caractérisant la fonction <c,x> dans A
    # On minimise alors le dernier paramètre
    # Il correspond à la valeur de la fonction
    d = np.concatenate((np.zeros(len(A)),[1,-1]))
    A = np.vstack((A,-c,c))
    A = np.vstack((A.T,d)).T
    b = np.concatenate((b,[0,0]))
    res = FM4(A,b)
    mini_vect = remontee(res,c)
    return (round(res[0],3) , mini_vect)
```

EXERCICE 1 : Tests

```
#-----------------------------------------------------------
A=np.array([[3.,2.,1.],
            [-3.,-2.,-1.],
            [2.,3.,4.],
            [-2.,-3.,-4.],
            [3.,2.,5.],
            [-3.,-2.,-5.],
            [-1.,0.,0.],
            [0.,-1.,0.],
            [0.,0.,-1.]])
b=np.array([15.,-6.,12.,-5.,7.,-4.,0.,0.,0.])
c=np.array([1.,1.,1.])
# ON VEUT X=[1.6,0.6,0], min=2.2
#-----------------------------------------------------------
A=np.array([[-1.,-1,-1],
            [-3,1,1],
            [1,-3,1],
            [2,1,-3],
            [0,1,1]])
b=np.array([1.,1,1,1,4])
c=np.array([1.,0,-1])
# ON VEUT X=[1,1,3], min=-2
#-----------------------------------------------------------
```

```
from scipy.optimize import linprog
```

EXERCICE 2 : Q1

```
A = np.array([[1.,0,0,0,0,0,0],
              [0,1,0,0,0,0,0],
              [0,0,1,1,0,0,0],
              [0,0,0,0,1,1,0],
              [0,0,0,0,0,0,1],
              [1,0,-1,0,-1,0,-1],
              [-1,0,1,0,1,0,1],
              [0,1,0,-1,0,-1,0],
              [0,-1,0,1,0,1,0],
              [-0.3,0,1,0,0,0,0],
              [0,-0.5,0,1,0,0,0],
              [0.4,0,0,0,-1,0,0],
              [0,0.1,0,0,0,-1,0],
              [0.5,0,0,0,0,0,-1]]))
b = np.array([35000.,15000,30000,20000,10000,0,0,0,0,0,0,0,0])
c = np.array([-66,-54,36,36,48,48,60])
```

EXERCICE 2 : Q2

```
A = np.array([[1.,0,0,0,0,0,0,0],
              [0,1,0,0,0,0,0,0],
              [0,0,1,1,0,0,0,0],
              [0,0,0,0,1,1,0,0],
              [0,0,0,0,0,0,1,0],
              [1,0,-1,0,-1,0,-1,0],
              [-1,0,1,0,1,0,1,0],
              [0,1,0,-1,0,-1,0,0],
              [0,-1,0,1,0,1,0,0],
              [-0.3,0,1,0,0,0,0,0],
              [0,-0.5,0,1,0,0,0,0],
              [0.4,0,0,0,-1,0,0,0],
              [0,0.1,0,0,0,-1,0,0],
              [0.5,0,0,0,0,0,-1,0],
              [-66,0,36,0,48,0,60,1],
              [0,-54,0,36,0,48,0,1]])
b = np.array([35000.,15000,30000,20000,10000,0,0,0,0,0,0,0,0,0,0])
c = np.array([0.,0,0,0,0,0,0,-1])
```