Feuille de TP n°2
Résolution graphique en dimension 2

Objectif : Programmer la méthode de la résolution graphique pour un problème d'optimisation linéaire en dimension 2.

1 La bibliothèque `matplotlib`

Dans cette séance, on va utiliser la bibliothèque `matplotlib` qui contient des fonctions permettant l'affichage graphique sous Python.

1.1 Charger la bibliothèque `matplotlib`

Pour utiliser la bibliothèque `matplotlib`, il faut ajouter en début de script la ligne suivante :

```
import matplotlib.pyplot as plt
```

1.2 Tracé de point, droite et région du plan

Voici quelques fonctions de la bibliothèque `matplotlib` que nous utiliserons dans ce TP. On commence par les fonctions `plt.figure()` et `plt.show()` qui permettent respectivement d'ouvrir et d'afficher une figure. Dans ce TP, on cherchera principalement à représenter graphiquemment trois types d'objet, à savoir les points, les droites et les régions.

Pour tracer un point, on va utiliser la fonction `plt.plot()` :

```
plt.figure()

plt.plot(1,0,'or') # rond rouge
plt.plot(2,1,'ob') # rond bleu

plt.show()
```

Pour tracer une droite, on va utiliser la fonction `plt.contour()` qui permet de tracer les lignes de niveaux d'une fonction donnée. On peut par exemple écrire

```
plt.figure()

abscisses = np.linspace(-1,10,100)
ordonnees = np.linspace(-1,10,100)

def droite(x,y) :
    return 2*x+2*y-2

```

```
z = droite(abscisses,ordonnees)

plt.contour(x,y,z,[0],colors='k')

plt.show()
```

pour tracer la droite $\{(x,y) \in \mathbb{R}^2 \mid 2\,x + 2\,y = 2\}$ sur le carré $[-1\,;10] \times [-1\,;10]$, cette droite pouvant être interprétée comme la ligne de niveau 0 de la fonction

$$f : (x,y) \mapsto 2\,x + 2\,y - 2$$

Notons que cette même droite peut aussi être vue comme la ligne de niveau 2 de la fonction

$$g : (x,y) \mapsto 2\,x + y$$

auquel cas le code suivant

```
plt.figure()

abscisses = np.linspace(-1,10,100)
ordonnees = np.linspace(-1,10,100)

def droite2(x,y) :
    return 2*x+2*y

z = droite2(abscisses,ordonnees)

plt.contour(x,y,z,[2],colors='k')

plt.show()
```

produira le même résultat. Le nombre `100` correspond à l'échantillonnage choisi pour x et y. Plus ce chiffre est grand, meilleure est la résolution, mais plus long est le calcul. L'option `colors='k'` permet de spécifier la couleur de la droite tracée, ici en noir.

Exercice 1 – Lignes de niveaux d'une forme linéaire On considère la forme linéaire suivante :

$$f : (x_1, x_2) \mapsto 2\,x_1 + x_2$$

(a) Tracer les lignes de niveaux 0, -1 et 1 de cette fonction, en rouge (`'r'`), en vert (`'g'`) et en bleu (`'b'`) respectivement.

(b) Tracer la ligne de niveau de f qui passe par le point $(2,1)$.

Pour représenter une région du plan, comme un polyèdre par exemple, on va commencer par générer un maillage du plan, c'est-à-dire générer un ensemble de points régulièrement répartis sur une partie du plan. On utilise pour cela la fonction `np.meshigrid()` de la bibliothèque `numpy`. Par exemple, pour générer l'ensemble des points (x_i, y_i) avec $x_i \in \{0, 0.5, \cdots, 9.5\}$ et $y_i \in \{1, 2, \cdots, 5\}$, on pourra écrire

```
abscisses = np.linspace(0,10,20)
ordonnees = np.linspace(1,5,5)

X,Y = np.meshgrid(abscisses,ordonnees)
```

Si l'on souhaite par exemple représenter le demi-plan

$$2\,x + y \leq 0$$

il suffit de tester pour chacun de ces points si cette inégalité est vérifiée, puis d'attribuer 1 au point si c'est le cas, et 0 sinon, c'est-à-dire de considérer la fonction

$$t(x,y) = \begin{cases} 1 & \text{si } 2\,x + y \leq 0 \\ 0 & \text{sinon} \end{cases}$$

Enfin, on utilise la fonction `plt.pcolor` qui attribue une couleur à chaque valeur différente de $t(x_i, y_i)$ au point (x_i, y_i), de sorte que les points (x_i, y_i) appartenant au demi-plan considéré apparaîtront en une couleur donnée sur la figure et les autres en une autre couleur. On peut tester le code suivant :

```
def t(x,y)
    if 2*x+y>0:
        return 0
    return 1

abscisses = np.linspace(-1,10,100) # on se place sur le carré
     [-1,10]*[-1,10]
ordonnees = np.linspace(-1,10,100)
X,Y = np.meshgrid(abscisses,ordonnees)
n,p = X.shape # dimension de X

# on définit Z[k,j] = t(X[k,j],Y[k,j])
Z = np.zeros((n,p)) # on initialise les valeurs de Z à 0
for k in range(n):
    for j in range(p):
        if t(X[k,j],Y[k,j]) == 1:
            Z[k,j] = 1

# affichage
plt.figure()
plt.pcolor(X,Y,Z)
plt.show()
```

Exercice 2 – Polyèdre On considère le polyèdre suivant :

$$\mathcal{C} = \left\{ (x_1, x_2) \in \mathbb{R}^2 \;\middle|\; \begin{array}{rcl} x_1 + 2\,x_2 & \leq & 8 \\ x_1 + x_2 & \leq & 5 \\ 9\,x_1 + 4\,x_2 & \leq & 36 \\ x_1 \;,\; x_2 & \geq & 0 \end{array} \right\}$$

(a) Définir en python la fonction

$$t(x,y) = \begin{cases} 1 & \text{si } (x,y) \in \mathcal{C} \\ 0 & \text{sinon} \end{cases}$$

(b) Afficher le polyèdre $\mathcal{C}$ en modifiant le code donné en exemple plus haut.

2 Sommets d'un polyèdre

Dans cette section, on va s'intéresser aux sommets d'un polyèdre donné. On rappelle que ces sommets sont des intersections entre deux droites portant les arêtes du polyèdre.

Exercice 3 – Fonction mystère Que fait la fonction suivante ?

```
def mystere(n,p)
    for k in range(n):
        for j in range(k+1,p):
            print(''('',k,'','',j,'')'')
```

Exercice 4 – Intersection de droites Considérons les deux droites

$$\mathcal{D}_1 : x_1 + 2\,x_2 = 8 \qquad \text{et} \qquad \mathcal{D}_2 : x_1 + x_2 = 5$$

(a) Définir en Python les matrice et vecteur

$$D = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \qquad \text{et} \qquad d = \begin{pmatrix} 8 \\ 5 \end{pmatrix}$$

(b) Calculer avec Python le rang de la matrice D. En déduire que D est inversible.

(c) Utiliser la fonction `np.linalg.solve()` pour calculer les coordonnées de l'intersection entre les droites $\mathcal{D}_1$ et $\mathcal{D}_2$.

(d) Écrire une fonction `intersection` qui prend pour argument une matrice D de taille 2×2, un vecteur $d \in \mathbb{R}^2$, qui teste si D est inversible et qui résout le système linéaire $D\,X = d$ si D est inversible. Voici le prototype d'une telle fonction :

```
def intersection(D,d)
    ... series d'instructions ...
    ...
    ...
    return ...
```

Exercice 5 – Pré-détermination des sommets d'un polyèdre On revient au polyèdre défini dans l'**exercice 2**, qui est l'intersection de 5 demi-plans, lesquels sont associés à une droite-frontière.

(a) Définir en Python les matrice et vecteur

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 9 & 4 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \text{et} \qquad b = \begin{pmatrix} 8 \\ 5 \\ 36 \\ 0 \\ 0 \end{pmatrix}$$

(b) En s'inspirant de la fonction `mystere` de l'**exercice 3**, extraire en Python toutes les sous-matrices carrées de A.

(c) Modifier le code de la question **(b)** pour calculer les intersections entre paires de droites-frontières du polyèdre $\mathcal{C}$, à l'aide de la fonction `intersection` de l'**exercice 4**.

(d) Tracer le polyèdre $\mathcal{C}$ et les 5 droites-frontières.

(e) À l'aide de la fonction `t` introduite dans l'**exercice 2**, tracer en bleu les points obtenus dans la question **(c)** qui sont des sommets du polyèdre $\mathcal{C}$ et en rouge les autres.

3 Résolution graphique

On peut à présent implémenter la résolution graphique d'un problème d'optimisation linéaire de dimension 2 en Python. On rappelle les différentes étapes à suivre :

1. représentation graphique du polyèdre des contraintes ;
2. tracé des lignes de niveau (par exemple 0, 1, -1) de la fonction objectif ;
3. détermination de tous les sommets ;
4. tracé des lignes de niveau passant par les sommets.

Exercice 6 – Résolution graphique On considère le problème d'optimisation linéaire suivant :

$$\begin{array}{lrcl} \text{Maximiser} & f(x_1,x_2) = 2\,x_1 + \;\; x_2 & & \\ \text{sous les contraintes} & x_1 + 2\,x_2 & \leq & 8 \\ & x_1 + \;\; x_2 & \leq & 5 \\ & 9\,x_1 + 4\,x_2 & \leq & 36 \\ & x_1 \;\; , \;\; x_2 & \geq & 0 \end{array}$$

En utilisant les codes introduits plus haut, résoudre graphiquement ce problème.