
TP3: Implémentation de l'algorithme du Simplexe en Python

1 Introduction

L'objectif de ce TP est de programmer une version simplifiée de l'algorithme du simplexe primal en Python. Dans un premier temps, nous allons introduire les notions de base de la programmation en Python, en mettant l'accent sur le package NumPy. Ensuite, nous concevrons des fonctions qui seront utilisées pour développer l'algorithme du simplexe. Enfin, l'algorithme développé sera comparé, sur un petit exemple, avec le simplexe natif implémenté dans python.

2 Rappels sur NumPy

NumPy (Numerical Python) est une bibliothèque fondamentale pour le calcul scientifique en Python. Elle fournit notamment :

- Un objet `ndarray`, tableau multidimensionnel performant et flexible.
- Des fonctions mathématiques pour opérer sur ces tableaux.
- Des outils pour l'algèbre linéaire, la transformation de Fourier, et les nombres aléatoires.

2.1 Création de tableaux

- `np.array([1, 2, 3])` : crée un tableau 1D.
- `np.zeros((m, n))` : crée un tableau de zéros de taille $m \times n$.
- `np.ones((m, n))` : crée un tableau de uns de taille $m \times n$.
- `np.arange(start, stop, step)` : crée un tableau d'entiers de `start` à `stop` avec un pas de `step`.
- `np.linspace(start, stop, num)` : crée un tableau de `num` nombres également espacés entre `start` et `stop`.

2.2 Opérations sur les tableaux

- Indexation : `A[i, j]` accède à l'élément à la ligne `i` et la colonne `j` du tableau `A`.
- Slicing : `A[:, j]` accède à toutes les lignes de la colonne `j` de `A`.
- Reshape : `A.reshape((p, q))` modifie la forme du tableau `A` en un tableau de taille $p \times q$.
- Concaténation : `np.append(A, B, axis=0)` concatène les tableaux `A` et `B` verticalement si `axis=0`, horizontalement si `axis=1`.

2.3 Fonctions mathématiques

- `np.amin(A)` : renvoie la valeur minimale du tableau `A`.
- `np.amax(A)` : renvoie la valeur maximale du tableau `A`.
- `np.argmin(A)` : renvoie l'indice de la valeur minimale de `A`.
- `np.argmax(A)` : renvoie l'indice de la valeur maximale de `A`.
- `np.sum(A)` : calcule la somme de tous les éléments de `A`.
- `np.sqrt(A)` : applique la racine carrée à chaque élément de `A`.

2.4 Génération de nombres aléatoires

- `np.random.rand(m, n)` : génère un tableau $m \times n$ de nombres aléatoires uniformément distribués entre 0 et 1.
- `np.random.randn(m, n)` : génère un tableau $m \times n$ de nombres aléatoires suivant une distribution normale standard.

2.5 Visualisation avec Matplotlib

Matplotlib est une bibliothèque de traçage pour Python. Elle est souvent utilisée en conjonction avec NumPy pour visualiser des données.

- `plt.plot(x, y)` : trace la courbe définie par les points (x, y) .
- `plt.scatter(x, y)` : crée un nuage de points défini par (x, y) .
- `plt.hist(x)` : trace l'histogramme des données x .
- `plt.show()` : affiche la figure courante.

3 Exercices Pratiques avec NumPy

Objectif : Saisir et expliquer le résultat des commandes suivantes.

3.1 Importation des packages

```
1 import numpy as np
2 from scipy.optimize import linprog
3 import math
4 import matplotlib.pyplot as plt
```

Explication : Ces commandes importent les bibliothèques nécessaires :

- `numpy` pour le calcul numérique.
- `scipy.optimize.linprog` pour résoudre des problèmes de programmation linéaire.
- `math` pour les fonctions mathématiques standards.
- `matplotlib.pyplot` pour la visualisation.

3.2 Manipulation de tableaux

```
1 A = np.array([[1,2],[3,4]])
```

Explication : Crée un tableau A de dimension 2×2 .

```
1 A[0,0]
```

Résultat : Accède à l'élément en première ligne, première colonne de A (valeur 1).

```
1 A[:,1]
```

Résultat : Sélectionne toutes les lignes de la deuxième colonne de A (vecteur colonne $[2, 4]$).

```
1 A[0,:]
```

Résultat : Sélectionne la première ligne de A (vecteur ligne $[1, 2]$).

```
1 n1, nc = A.shape
2 print(n1, nc)
```

Résultat : Stocke le nombre de lignes et de colonnes de A dans `n1` et `nc`, puis les affiche (2 2).

```
1 c = np.array([4, -2, -2])
2 print(type(c))
3 print(c.ndim)
```

Explication : `c` est un tableau NumPy de dimension 2 (tableau 1×3). `type(c)` renvoie `<class 'numpy.ndarray'>`. `c.ndim` affiche le nombre de dimensions (2).

```
1 m = np.amin(c)
2 print(m)
```

Résultat : Affiche la valeur minimale de `c` (-2).

```
1 print(np.argmin(c))
```

Résultat : Affiche l'indice de la valeur minimale de `c` (1, puisque -2 apparaît à l'indice 1).

```
1 n = np.amax(c)
2 print(np.argmax(c))
```

Résultat : Affiche l'indice de la valeur maximale de `c` (0, puisque 4 est à l'indice 0).

3.3 Reshape et concaténation

```
1 a = np.array([[1.2, 2.5], [3.2, 1.8], [1.1, 4.3]])
2 v = np.arange(0, 10).reshape(2, 5)
3 print(a.reshape(3, 2))
```

Explication : `a` est un tableau 3×2 . `v` est un tableau 2×5 avec les nombres de 0 à 9. La fonction `reshape` ne modifie pas le tableau `a` ici car il est déjà de taille 3×2 .

```
1 print(np.zeros(shape=(2,4)))
```

Résultat : Affiche un tableau de zéros de taille 2×4 .

```
1 b = np.array([[4.1, 2.6]])
2 c = np.append(a, b, axis=0)
3 print(c)
```

Explication : Ajoute la ligne `b` à la fin du tableau `a` verticalement (`axis=0`). Le nouveau tableau `c` est de taille 4×2 .

```
1 d = np.array([[7.8], [6.1], [5.4]])
2 print(np.append(a, d, axis=1))
```

Explication : Ajoute la colonne `d` au tableau `a` horizontalement (`axis=1`). Le résultat est un tableau de taille 3×3 .

3.4 Insertion et suppression d'éléments

```
1 print(np.insert(a, 1, b, axis=0))
```

Explication : Insère la ligne `b` à la position 1 dans `a` (après la première ligne). Le tableau résultant est de taille 4×2 .

```
1 print(np.delete(a, 1, axis=0))
```

Explication : Supprime la ligne à l'indice 1 de `a` (la deuxième ligne). Le tableau résultant est de taille 2×2 .

3.5 Slicing avancé

```
1 v = np.array([[1.2, 2.5], [3.2, 1.8], [1.1, 4.3]])
2 print(v[0:2, :])
```

Explication : Sélectionne les lignes d'indice 0 et 1 de `v`.

```
1 print(v[-1, :])
```

Explication : Sélectionne la dernière ligne de `v`.

```
1 print(v[-2:, :])
```

Explication : Sélectionne les deux dernières lignes de `v`.

```
1 print(np.max(v, axis=0))
```

Explication : Affiche le maximum de chaque colonne de `v`.

3.6 Boucles et fonctions

```
1 s = 0
2 for i in range(0, v.shape[0]):
3     for j in range(0, v.shape[1]):
4         print(v[i, j])
5         s = s + v[i, j]
6 print("Somme = ", s)
```

Explication : Parcourt chaque élément du tableau `v`, l'affiche, et calcule la somme totale des éléments.

```
1 def racine(x):
2     return math.sqrt(x)
3 print(racine(4))
```

Explication : Définit une fonction `racine` qui calcule la racine carrée d'un nombre. Affiche la racine de 4 (2.0).

```

1 def ajouter_et_afficher():
2     l = list()
3     l.append(2)
4     print(l)
5
6 # Test de la fonction
7 ajouter_et_afficher()

1 def division(a, b):
2     try:
3         resultat = a / b
4     except ZeroDivisionError:
5         print("Erreur : division par z ro.")
6         return None
7     except TypeError:
8         print("Erreur : les arguments doivent tre des nombres.")
9         return None
10    else:
11        return resultat

1 def fibonacci_memo(n):
2     if n in memo:
3         return memo[n]
4     if n <= 0:
5         print("Erreur : n doit tre un entier positif.")
6         return None
7     elif n == 1 or n == 2:
8         memo[n] = 1
9     else:
10        memo[n] = fibonacci_memo(n-1) + fibonacci_memo(n-2)
11    return memo[n]
12
13 # Tests
14 print(fibonacci_memo(10)) # Affiche 55

```

Explication : Crée une liste vide l, y ajoute l'élément 2, puis l'affiche.

3.7 Visualisation de données

```

1 x = np.linspace(0, 10, 30)
2 y = np.sin(x)
3 plt.scatter(x, y, marker='o')
4 plt.show()

```

Explication : Génère 30 points entre 0 et 10, calcule le sinus de ces points, puis crée un nuage de points.

```

1 rng = np.random.RandomState(0)
2 x = rng.randn(100)
3 y = rng.randn(100)
4 colors = rng.rand(100)
5 sizes = 1000 * rng.rand(100)
6 plt.scatter(x, y, c=colors, s=sizes, alpha=0.3, cmap='viridis')
7 plt.colorbar()
8 plt.show()

```

Explication : Génère 100 points aléatoires pour x et y, des couleurs et des tailles aléatoires, puis crée un nuage de points coloré avec une échelle de couleurs.

4 Implémentation de l'Algorithme du Simplexe

4.1 Présentation de l'algorithme

L'algorithme du simplexe est une méthode itérative pour résoudre les problèmes de programmation linéaire. Il consiste à se déplacer de sommet en sommet sur le polyèdre des contraintes jusqu'à atteindre la solution optimale.

Le tableau initial du simplexe pour un problème de minimisation est donné par :

$\alpha_{1,1} \cdots \alpha_{1,n+m}$	\bar{b}_1
$\vdots \cdots \vdots$	\vdots
$\alpha_{m,1} \cdots \alpha_{m,n+m}$	\bar{b}_m
$\bar{c}_1 \cdots \bar{c}_{n+m}$	z

4.2 Règles de manipulation des tableaux

1. **Initialisation** : $\bar{b}_i = b_i$, $\bar{c}_j = c_j$, $\alpha_{ij} = a_{ij}$, $z = 0$.
2. **Choix de la colonne pivot** (variable à entrer en base) :
 - (a) Si $\bar{c}_j \geq 0$, $\forall j \in \{1, \dots, n+m\}$ alors **STOP** : la solution optimale est trouvée.
 - (b) Sinon, choisir une colonne s telle que $\bar{c}_s < 0$ (utiliser la règle de Bland pour éviter les cycles).
3. **Choix de la ligne pivot** (variable à sortir de la base) :
 - (a) Si $\alpha_{is} \leq 0$, $\forall i \in \{1, \dots, m\}$ alors **STOP** : la fonction objectif n'est pas bornée inférieurement.
 - (b) Sinon, choisir une ligne r telle que :

$$\frac{\bar{b}_r}{\alpha_{rs}} = \min \left\{ \frac{\bar{b}_i}{\alpha_{is}} : \alpha_{is} > 0 \right\}$$

4. **Mise à jour du tableau** :
 - (a) Diviser la ligne pivot par le pivot α_{rs} .
 - (b) Mettre à zéro les autres éléments de la colonne pivot en utilisant des opérations sur les lignes.
 - (c) Mettre à jour les autres éléments du tableau :

$$\alpha'_{ij} = \alpha_{ij} - \frac{\alpha_{is}\alpha_{rj}}{\alpha_{rs}}, \quad \bar{b}'_i = \bar{b}_i - \frac{\alpha_{is}\bar{b}_r}{\alpha_{rs}}, \quad \bar{c}'_j = \bar{c}_j - \frac{\bar{c}_s\alpha_{rj}}{\alpha_{rs}}, \quad z' = z - \frac{\bar{c}_s\bar{b}_r}{\alpha_{rs}}$$

4.3 Travail à réaliser

1. **Écrire une fonction Python** `generate_tabinitial(A, b, c)` qui prend en arguments la matrice A , les vecteurs b et c , et qui construit et retourne le tableau initial du simplexe.
2. **Écrire une fonction Python** `positive(v)` qui prend en paramètre un vecteur v et qui retourne `True` si toutes les composantes du vecteur sont positives, sinon retourne `False`, ainsi que la valeur minimale et l'indice du minimum.
3. **Écrire une fonction Python** `rapportmin(b, a)` qui prend en arguments deux vecteurs colonnes b et a , et qui retourne l'indice du rapport minimum positif des composantes de b par celles de a .
4. **Écrire une fonction Python** `pivotgauss(T, r, s)` qui prend en argument une matrice tableau T , le numéro de la ligne pivot r et le numéro de la colonne pivot s , et qui effectue le pivot de Gauss sur T .
5. **Implémenter l'algorithme du simplexe** en utilisant les fonctions définies précédemment. Votre programme doit être capable de résoudre un problème de programmation linéaire en minimisation sous forme standard.
6. **Comparer les résultats obtenus** avec ceux de la fonction `linprog` de la bibliothèque `scipy.optimize`.

4.4 Conseils pour l'implémentation

- Veillez à bien gérer les indices des variables de décision et des variables d'écart.
- Utilisez des structures de données appropriées pour stocker la base courante et suivre l'évolution des variables.
- Pensez à inclure des conditions d'arrêt pour éviter les boucles infinies en cas de non convergence.
- Testez votre programme sur des exemples simples.

4.5 Exemple d'utilisation

Considérons le problème suivant :

$$\begin{aligned} \min \quad & z = -3x_1 - 2x_2 \\ \text{sous contraintes} \quad & x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Transformez ce problème en forme standard, puis utilisez votre implémentation de l'algorithme du simplexe pour le résoudre.

4.6 Comparaison avec linprog

Utilisez la fonction `linprog` de `scipy.optimize` pour résoudre le même problème et comparez les résultats obtenus (valeur optimale, valeurs des variables de décision, etc.).

```
1 res=linprog(c, A_ub=A, b_ub=b)
2 print(res)
3 print('Optimal value:', res.fun, '\nX:', res.x)
```