TUTORIEL SUR LE SOLVEUR LINPROG DE LE BIBLIOTHEQUE SICPY DE PYTHON

PROGRAMMATION LINEAIRE

Nombreux sont les problèmes susceptibles d'être formulés en tant que maximisation ou minimisation d'un objectif en fonction de ressources limitées et de contraintes mutuellement rivales. Si l'on arrive à exprimer l'objectif sous la forme d'une fonction linéaire de certaines variables et que l'on peut spécifier les contraintes concernant les ressources sous la forme d'égalités ou d'inégalités linéaire sur ces variables, alors on a un problème de *programmation linéaire*.

PROGRAMMATION LINEAIRE

Un problème de programmation linéaire est donc un problème ayant :

- **X:** n variables ou paramétres inconnues
- **F**: une fonction lineaire sur les n variables à maximiser ou minimiser
- **Ce**: p contraintes d’égalité linéaire sur les n variables
- **Ci**: q contraintes d’inegalité linéaire sur les n variables
- **E**: ensemble convexe des valeurs admissibles

L’objectif est donc de trouver un sous ensemble de variables de E qui maximise ou minimise F tout en respectant les contraintes Ce et Ci, cela est possible en utilisant un solveur . Dans suite nous allons montrer comment proceder en utilisant le solveur **linprog**

LE SOLVEUR LINPROG

La bibliotheque *scipy* de python posséde la function *linprog* qui permet de resoudre un probleme de progrmmation lineaire, Pour utiliser linprog 02 considérations doivent etre prise avant de commencer à ecrire toute ligne de code à savoir:

- Le problème doit etre formuler en problème de minimisation
- Les contraintes d’inégalités doivent etre toutes etre des contraintes d’inferiorités ‘’<=“.

LE SOLVEUR LINPROG

Linprog est une function de la bibliothéque scipy utilisée avec un parametre obligatoire d'autres optionels à savoir :

- **C**: un vecteur qui contient les coeffients de la function objective à minimiser (c'est le seul paramètre obligatoire) .
- **A_ub**: Tableau 2-D qui, multiplié par la matrice x, donne les valeurs des contraintes d'inégalité de la borne supérieure en x.
- **b_ub**: Tableau 1-D de valeurs représentant la limite supérieure de chaque contrainte d'inégalité (ligne) dans A_ub.
- **A_eq**: Tableau 2-D qui, multiplié par la matrice x, donne les valeurs des contraintes d'égalité en x .

LE SOLVEUR LINPROG
(les paramètres)

- **b_eq:** Tableau 1-D de valeurs représentant la valeur de chaque contrainte d'égalité (ligne) dans A_eq .
- **bound :** (min, max) paires pour chaque élément de x, définissant les limites de ce paramètre Par défaut, les limites sont (0, null) (non négatives) . Si une séquence contenant un seul tuple est fournie, alors min et max seront appliqués à toutes les variables du problème.
- **Method:** Type de solveur. Pour le moment, seul 'simplex' est pris en charge
- **Options:** Un dictionnaire d'options de solveur.

  maxiter: reel Nombre maximum d'itérations à effectuer.

  disp: booléen à définir sur True pour imprimer les messages de convergence.

LE SOLVEUR LINPROG (les paramètres)

- **Callback:** Si une fonction de rappel est fournie, elle sera appelée à chaque itération de l'algorithme simplex. Le rappel doit avoir la signature (xk, ** kwargs) où *xk* est le vecteur de solution actuel et *kwargs* est un dictionnaire contenant les éléments suivants:

  "tableau": le tableau actuel de l'algorithme Simplex

  "nit": l'itération actuelle.

  "pivot": Le pivot (ligne, colonne) utilisé pour la prochaine itération.

  «phase»: si l'algorithme est en phase 1 ou en phase 2.

  "base": les indices des colonnes des variables de base.

LE SOLVEUR LINPROG

La function linprog retourne un objet possédant les propriétés suivante:

- **x**: Le vecteur variable indépendant qui optimise le problème de programmation linéaire
- **slack**: variable contenant des variables d'écart, chaque variable d'ecart correspond a une constraint d'inegalité
- **success**: un boléen qui vaut True si l'algorithme a trouvé la solution optimal avec succés
- **nit:** un entier representant le nombre d'iteration
- **status**: un entier representant l'etat de sortie de l'algorithme
- **message**: chaîne décrivant l'état de sortie de l'optimisation.

EXEMPLE DE RESOLUTION D'UN PROBLEME DE MINIMISATION

Considérons le probleme de minimisation suivant :

Min z = 10x1 + 15x2 + 25x3

S,C

1x1 + 1x2 + 1x3 >= 1000

1x1 – 2x2 + 0x3 >= 0

0x1 + 0x2 + 1x3 >= 340

X1, x2, x3 >= 0

Les pages suivantes présentent le code python permettant de resoudre ce probléme grace a la fonction *linprog* de la bibliotheque *scipy* .

EXEMPLE DE RESOLUTION D'UN PROBLEME DE MINIMISATION

```python
# Importer les bibliothéques requises
import numpy as np
from scipy.optimize import linprog

# Creation de la matrice de contraintes d'inégalité
# Note: Les contraintes d'inégalité sont sous la forme <=
A = np.array([[-1, -1, -1], [-1, 2, 0], [0, 0, -1]])


b = np.array([-1000, 0, -340])


# Creation du vecteur contenant les coefficients de la fonction objective
# Note: Lorsqu'il sagira d'un probleme de maximisation, il vas falloir changer le signe des coefficients
c = np.array([10, 15, 25])


# Resolution du probleme de programmation lineaire
res = linprog(c, A_ub=A, b_ub=b)


# On affiche le resultat
print('valeur optimal:', round(res.fun, ndigits=2),
      '\nx valeur:', res.x,
      '\nNombre d\'iteration:', res.nit,
      '\nStatus:', res.message)
```

EXEMPLE DE RESOLUTION D'UN PROBLEME DE MINIMISATION

Enregistrer le fichier sous le nom *"linprog,py" et exécuter*:

```
C:\Users\lenovo\Desktop>python linprog.py
valeur optimale: 15100.0
x valeur: [6.59999996e+02 1.00009440e-07 3.40000000e+02]
Nombre d'iterations: 7
Status: Optimization terminated successfully.

C:\Users\lenovo\Desktop>
```

Vous devriez avoir le resultat ci dessus .

EXEMPLE DE RESOLUTION D'UN PROBLEME DE MAXIMISATION

Étant donné que la fonction linprog de la bibliothèque SciPy de Python est programmée pour résoudre les problèmes de minimisation, il est nécessaire d'effectuer une transformation vers la fonction objectif d'origine. Tout problème de minimisation peut être transformé en problème de maximisation en multipliant les coefficients de la fonction objectif par -1 (c'est-à-dire en changeant leurs signes).

EXEMPLE DE RESOLUTION D'UN PROBLEME DE MAXIMISATION

Considérons le probleme de maximisation suivant :

$$Max\ z = 5x1 + 7x2$$

$$S,C$$

$$1x1 + 0x2 <= 16$$

$$2x1 - 3x2 <= 19$$

$$1x1 + 1x2 <= 8$$

$$X1, x2 >= 0$$

Les pages suivantes presentent le code python permettant de resoudre ce probléme grace a la function *linprog* de la bibliotheque *scipy* .

EXEMPLE DE RESOLUTION D'UN PROBLEME DE MAXIMISATION

```python
# Importer les bibliothéques requises
import numpy as np
from scipy.optimize import linprog

# Creation de la matrice de contraintes d'inégalité
# Note: Les contraintes d'inégalité sont sous la forme <=
A = np.array([[1, 0], [2, 3], [1, 1]])


b = np.array([16, 19, 8])


# Creation du vecteur contenant les coefficients de la fonction objective
# Note: Lorsqu'il sagira d'un probleme de maximisation, il vas falloir changer le signe des coefficients
c = np.array([-5, -7])


# Resolution du probleme de programmation lineaire
res = linprog(c, A_ub=A, b_ub=b)


# On affiche le resultat
print('valeur optimale:', round(res.fun*-1, ndigits=2),
      '\nx valeur:', res.x,
      '\nNombre d\'iterations:', res.nit,
      '\nStatus:', res.message)
```

EXEMPLE DE RESOLUTION D'UN PROBLEME DE MAXIMISATION

Enregistrer le fichier sous le nom *"linprog,py" et exécuter*:

```
C:\Users\lenovo\Desktop>python linprog.py
valeur optimale: 46.0
x valeur: [5. 3.]
Nombre d'iterations: 5
Status: Optimization terminated successfully.

C:\Users\lenovo\Desktop>
```

Vous devriez avoir le resultat ci dessus .

MERCI…

TUTORIEL SUR LESOLVEUR LINPROG