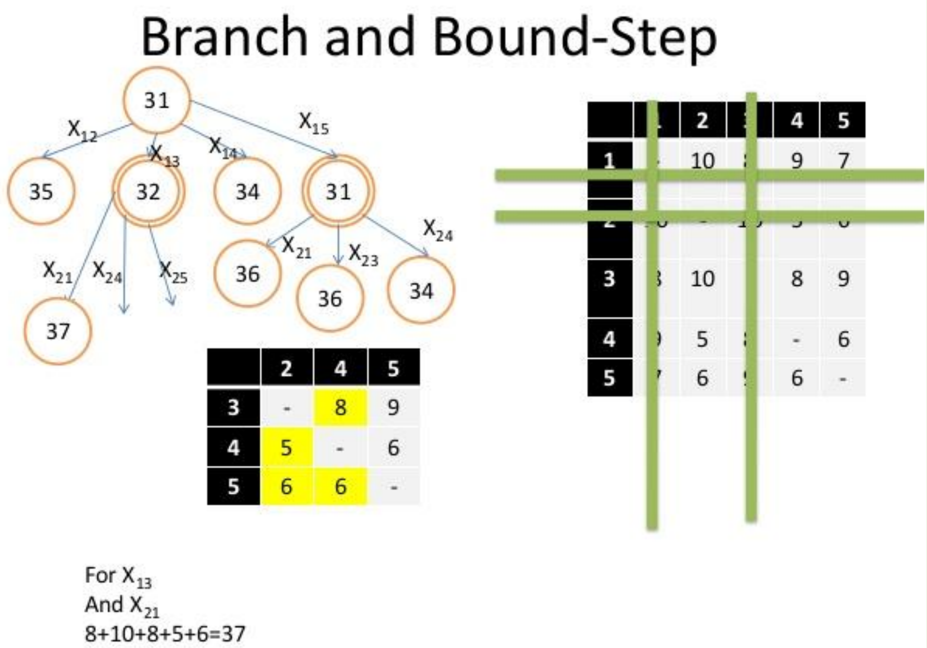
MASTER DS

Projet du module Recherche Opérationnelle

PARTIE 2 (programmation linéaire entière B&B)



BASTA Mohammed

Cadi Ayyad - FSSM

2017 - 2018

INTRODUCTION

Cette partie concerne l'implémentation de l'algorithme de Branch and Bound (B&B) sous Matlab.

J'ai essayer de traiter un problème simple déjà résolu.

La méthode de Branch and Bound est une des importantes méthodes dans la programmation linéaire, elle permet de résoudre des problèmes complexe dans la vie réelle comme le problème de sac-à-dos : « bientôt les vacances … Comment faire pour remplir mon sac le mieux possible ? »

D'abord on va voir la notion de BRANCH AND BROUND ensuite les outils que j'ai utilisé et enfin la démonstration sous MATLAB.

I. PROGRAMME LINEAIRE :

Soit le problème sous forme canonique suivant [PL_BNB 2017]:

$$
(PL) \left\{
\begin{array}{l}
\text{Min } Z = x1 - 2\,x2 \\
\text{sc :} \\
\quad -4\,x1 + 6\,x2 \leq 9 \\
\quad x1 + x2 \leq 4 \\
\quad x1\,,\, x2 \geq 0 \\
\quad x1\,,\, x2 \text{ entiers}
\end{array}
\right.
$$

On cherche à résoudre le PL ci-dessus avec la méthode de B&B pour avoir des valeurs optimales et entières.

II. OUTILS :

Définitions :

BRANCH_AND_BOUND PROCÉDÉ DE RÉSOLUTION DE PROBLÈMES DE PROGRAMMATION D'ENTIERS PUR [BB_def] :

- ❖ Les méthodes de branch-and-bound sont des méthodes basées sur une énumération "intelligente" des solutions admissibles d’un problème d’optimisation combinatoire.
- ❖ Idée : prouver l’optimalité d’une solution en partitionnant l’espace des solutions.
- ❖ "Diviser pour régner"
- ❖ Application à la programmation linéaire en nombres entiers : utilise toute la puissance de la programmation linéaire pour déterminer de bonnes bornes.
- ❖ On appelle relaxation linéaire d’un programme linéaire en nombres entiers le programme linéaire obtenu en supprimant les contraintes d’intégralité sur les variables.

Méthode [BB_methode]**:**

a. Si la solution de LP n'est pas entière, soit xi une variable prenant une valeur fractionnaire x ∗ i dans la solution optimale de LP.

b. Le problème peut être divisé en deux sous-problèmes en imposant

   xi ≤ bx∗i ou xi ≥ bx*i + 1

   où bx∗i est le plus grand entier inférieur à x∗i .

c. La solution optimale de P est la meilleure des solutions optimales des deux problèmes P1 et P2.

$$(P1)\begin{cases} \max c\,T\,x \\ \text{s.c.}\quad Ax \le b \\ \qquad xi \le bx * i\ c \\ \qquad x \ge 0, \text{ entier.} \end{cases}$$

$$(P2)\begin{cases} \max c\,T\,x \\ \text{s.c.}\quad Ax \le b \\ \qquad xi \ge bx * i\ c + 1 \\ \qquad x \ge 0, \text{ entier.} \end{cases}$$

Fonctions d'aide :

- Assert() : gestion des exceptions
- linprog() : résolution des problèmes linéaire
- numel() : nombre d'éléments dans une matrice
- isnan() : return oui si la variable est numérique
- zeros() : créer un tableau avec des zéros
- plot() : pour présenter les données sur un graphe
- num2str() : convertir numérique en caractère

III. Implémentation :

Entrée :

[1 -2] [-4 6 ;1 1] [9 4]

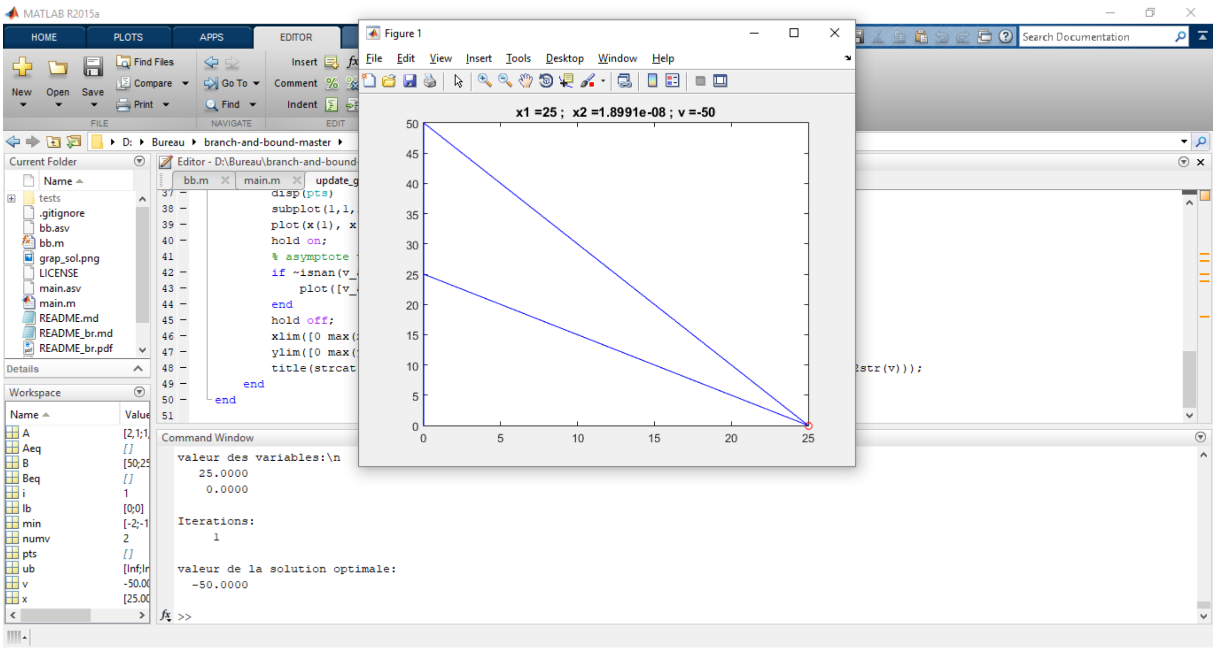
Sortie :

X1* = 25

X2* = 0

Z* = -50

Capture d'écran :



Résultat :

<u>Input :</u> soit copier la ligne indiquée avant dans la partie 'entrée' soit saisir chaque élément dans sa place :

```
>> main
entrez les coeficients de la fonction a minimiser dans le vecteur:
[-2; -1]
entrez les elements de la matrice A:
[2 1; 1 1]
entrez les elements du vecteur B:
[50; 25]
```

Output : sous forme d'une liste des valeurs des variables dans chaque itération :

```
Optimization terminated.
    0.0000    2.5000
    0.0000    9.0000
   25.0000    0.0000
   25.0000    0.0000
    1.5000    2.5000
    1.5000    2.5000
    2.1000    1.9000
    0.9141    1.9571
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
    2.1025    0.7951
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
    3.7500    2.2500
   25.0000    0.0000
   25.0000    0.0000
   25.0000    0.0000
```

Et puis les valeurs optimales :

```
valeur des variables:\n
   25.0000
    0.0000

Iterations:
     1

valeur de la solution optimale:
  -50.0000
```

Remarque : ci-joint les fonctions utilisées dans mon implémentation.

IV. Référence :

[BB_def] :
http://student.ulb.ac.be/~bfortz/bb.pdf

[BB_methode] :
http://ocw.nctu.edu.tw/upload/classbfs1211091041160581.pdf

documentation :
https://www.mathworks.com/help/matlab/
http://student.ulb.ac.be/~bfortz/bb.pdf
https://www.mathworks.com/help/matlab/ref/plot.html