Implémentation Python de l'algorithme de simplexe en utilisant le critère de Bland

21 avril 2021

[1]:
```
from numpy import *
from fractions import Fraction

class Simplex:
    #Constructeur de la classe Simplex
    def __init__(self, obj):
        self.obj = [1] + obj #Attribut pour la fonction objectif
        self.rows = [] #attribut pour contenir les lignes du tableau du simplex
        self.b = [] #Attribut pour le vecteur colonne b
        self.nb_variables = len(obj) #Attribut pour compter le nombre de variable␣
 ↪hors base initialement
        self.nb_constraints = 0#Attribut pour compter le nombre de variable de␣
 ↪base initialement
        self.accept_fraction = False#Attibut permettant de dire si oui ou non␣
 ↪des fractions seront utilisées
        self.minmax="MAX"

    def add_constraint(self, expression, value):
        self.rows.append([0] + expression)
        self.b.append(value)#On ajoute une valeur au vecteur colonne b
        self.nb_constraints += 1# On incremente à chaque nouvelle contrainte␣
 ↪cette variable
        #On remet ensuite à jour l'entête du tableau du simplexe
        self.header_tableau = ["J"] + ["x"+str(i+1) for i in range(self.
 ↪nb_variables)] \
                                        + ["x"+str(len(range(self.
 ↪nb_variables))+i+1)\
                                          for i in range(self.nb_constraints)] \
                                        + ["b"]

        self.basic_variables = ["x"+str(len(range(self.nb_variables))+i+1) for i␣
 ↪in range(self.nb_constraints)]

    #Méthode pour choisir la colonne du pivot
    def _pivot_column(self):
```

```
        #Critère de Bland(Utilisation du plus petit indice de variable
                             #parmi les variables potentielles pour la variable␣
↪entrante)
        low = 0
        idx = 0
        #Pour un problème de maximisation, on prend la colonne avec le plus␣
↪grand coefficient de coût positif
        if self.minmax=="MAX":
            for i in range(1, len(self.obj)-1):
                if self.obj[i] < low:
                    low = self.obj[i]
                    idx = i
                    break
            if idx == 0: return -1
            return idx
        #Pour un problème de minimisation,, on prend la colonne avec le plus␣
↪petit coefficient de coût négatif
        else:
            for i in range(1, len(self.obj)-1):
                if self.obj[i] > low:
                    low = self.obj[i]
                    idx = i
                    break
            if idx == 0: return -1
            return idx
    #Méthode pour choisir la ligne du pivot
    def _pivot_row(self, col):
        rhs = [self.rows[i][-1] for i in range(len(self.rows))]  #On prend le b
        lhs = [self.rows[i][col] for i in range(len(self.rows))] #On prend les␣
↪elements de la colonne du pivot
        ratio = []
        for i in range(len(rhs)):
            if lhs[i] == 0:
                ratio.append(99999999 * abs(max(rhs)))#denominateur=0, on ajoute␣
↪un grand nombre
                continue
            ratio.append(rhs[i]/lhs[i]) #Ajout du ratio

        res= argmin(self.basic_variables)  if len(set(ratio)) == 1 else␣
↪argmin(ratio)

        return res  #Critère de Bland(Utilisation du plus petit indice de␣
↪variable
                             #parmi les variables de base potentielles)

    #Méthode pour afficher le tableau du simplexe
```

```
    def display(self):
        #Si on souhaite avoir un affichage avec les fractions
        if self.accept_fraction:

            simplexe_table = '{:<8}'.format("J") \
                   + "".join(['{:<8}'.format("x"+str(i+1)) for i in range(self.
↪nb_variables)])   \
                   + "".join(['{:<8}'.format("x"+str(len(range(self.
↪nb_variables))+i+1)) for i in range(self.nb_constraints)]) \
                   + '{}'.format("b")


            for i, row in enumerate(self.rows):
                simplexe_table += "\n"
                simplexe_table += '{:<8}'.format(self.basic_variables[i]) \
                       + "".join(["{:<8}".format(str(Fraction(item).
↪limit_denominator(3))) for item in row[1:]])
            simplexe_table += "\n"
            simplexe_table += '{:<8}'.format("Z") \
                    + "".join(["{:<8}".format(str(Fraction(-item).
↪limit_denominator(3))) for item in self.obj[1:]])

            print(simplexe_table)
        #Si on préfère un affichage sans les fractions
        else:
            # L'affichage sera fait avec 2 chiffres après la virgule
            simplexe_table = '{:<''}'.format("J") \
                   + "".join(['{:<8}'.format("x"+str(i+1)) for i in range(self.
↪nb_variables)])   \
                   + "".join(['{:<8}'.format("x"+str(len(range(self.
↪nb_variables))+i+1)) for i in range(self.nb_constraints)]) \
                   + '{:<8}'.format("b")


            for i, row in enumerate(self.rows):
                simplexe_table += "\n"
                simplexe_table += '{:<8}'.format(self.basic_variables[i]) \
                       + "".join(["{:>8.2f}".format(item) for item in row[1:]])
            simplexe_table += "\n"
            simplexe_table += '{:<8}'.format("Z") + "".join(["{:>8.2f}".
↪format(-item) for item in self.obj[1:]])

            print(simplexe_table)

    #Méthode du pivotage de GAUSS
    def _pivot(self, row, col):
```

```
        pivot = self.rows[row][col]
        self.rows[row] /= pivot #On divise la ligne du pivot par le pivot
        for r in range(len(self.rows)):
            if r == row: continue #On ignore lq ligne du pivot(déjà traité)
            self.rows[r] = self.rows[r] - self.rows[r][col]*self.rows[row] #On pivote chaque ligne
        self.obj = self.obj - self.obj[col]*self.rows[row]

    def _check(self):
        if self.minmax=="MAX":
            if min(self.obj[1:-1]) >= 0: return 1 #Il s'agit de la condition d'arrêt de l'algorithme du simplexe
            #Tant qu'on aura un coefficient de coût positif, on va boucler dans l'algorithme du simplexe
            return 0
        elif self.minmax=="MIN":
            if max(self.obj[1:-1]) <= 0: return 1 #Il s'agit de la condition d'arrêt de l'algorithme du simplexe
            #Tant qu'on aura un coefficient de coût négatif, on va boucler dans l'algorithme du simplexe
            return 0
        else:
            raise ValueError("Erreur, ce n'est pas un programme linéaire")

    def solve(self):
        for i in range(len(self.rows)):
            self.obj += [0]
            ident = [0 for r in range(len(self.rows))]
            ident[i] = 1#Matrice identité pour les variable de base
            self.rows[i] += ident + [self.b[i]]#On rajout les lignes du b
            self.rows[i] = array(self.rows[i], dtype=float)#Conversion de type en array
        self.obj = array(self.obj + [0], dtype=float)#Rajout d'un 0 en fin de la fonction objectif qui sera au
                                                    #niveau de b

        # Résolution
        print('---------------------------------------------------------')
        self.display()
        while not self._check():
            c = self._pivot_column()#On calcule la colonne du pivot
            r = self._pivot_row(c)#On calcule la ligne du pivot en se basant sur la colonne
            self._pivot(r,c)#On fait l'opération de pivotage par la méthode de GAUSS
            #On déduit la ligne et la colonne du pivot
```

```python
            print('Colonne du pivot: %s\nLigne du pivot: %s'%(c,r+1))
            #On déduit la variable entrante et la variable sortante
            print('Variable entrante : {}'.format(self.header_tableau[c]))
            print('Variable sortante : {}'.format(self.basic_variables[r]))
            print('----------------------------------------------------------')
            # Mise à jour de la base
            for index, item in enumerate(self.basic_variables):
                if self.basic_variables[index] == self.basic_variables[r]:
                    self.basic_variables[index] = self.header_tableau[c]

            self.display()#Affichage du tableau du simplexe

if __name__ == '__main__':

    #1er exemple

    """
    2x1 + x2 + x3 <= 4
    x1 + 2x2 + x3 <= 8
    x3          <= 5
    min z = -2x1 - 3x2 - x3
    x1,x2,x3 >= 0
    """

    print('-----------------------1er exemple-----------------------')
    print("                      MINIMISATION                       ")
    t = Simplex([2,3,1])#On met la fonction objectif(coefficients multipliés par␣
 ↪-1)
    t.minmax="MIN"#Il s'agit d'un problème de minimisation
    t.add_constraint([2, 1, 1], 4)
    t.add_constraint([1, 2, 1], 8)
    t.add_constraint([0, 0, 1], 5)
    t.accept_fraction = True
    t.solve()
    print('----------------------------------------------------------')
    print("\nLa valeur optimale est : %d"%t.obj[-1])
    print()

    #2ème exemple
    print('--------------2ème exemple(PL1 du rapport)---------------')
    print("                      MAXIMISATION                       ")
    """
    x1 + 4x2 <= 8
    x1 + 2x2 <= 4
    max z = 3x1+9x2
    x1,x2 >= 0
    """
```

```python
    t = Simplex([-3,-9])#On met la fonction objectif(coefficients multipliés par␣
↪-1)
    t.minmax="MAX"#Il s'agit d'un problème de maximisation
    t.add_constraint([1, 4], 8)#Première contrainte
    t.add_constraint([1, 2], 4)#Deuxième contrainte
    t.accept_fraction = True#On spécifie qu'on souhaite l'utilisation des␣
↪fractions
    t.solve()
    print('-----------------------------------------------------------')
    print("\nLa valeur optimale est : %d"%t.obj[-1])
    print()

    #3ème exemple
    print('--------------3ème exemple(PL2 du rapport)---------------')
    print("                      MAXIMISATION                       ")
    """
    x1 + 7x2 <= 1
    3x1 + 14x2 <= 3
    max z = 3x1+3x2
    x1,x2 >= 0
    """

    t = Simplex([-3,-3])#On met la fonction objectif(coefficients multipliés par␣
↪-1)
    t.minmax="MAX"#Il s'agit d'un problème de maximisation
    t.add_constraint([1, 7], 1)#Première contrainte
    t.add_constraint([3, 14], 3)#Deuxième contrainte
    t.accept_fraction = True#On spécifie qu'on souhaite l'utilisation des␣
↪fractions
    t.solve()
    print('-----------------------------------------------------------')
    print("\nLa valeur optimale est : %d"%t.obj[-1])
    print()

    #4ème exemple
    print('--------------4ème exemple(PL3 du rapport)---------------')
    print("                      MAXIMISATION                       ")

    """
    x1 + 2x2 + 3x3 <= 4
    2x1 +5x2 + 6x3 <= 8
    3x1 +4x2 + 9x3 <= 12
    max z = 3x1 -5x2 + 4x3
    x1,x2,x3 >= 0
    """
```

```
    t = Simplex([-3,5,-4])#On met la fonction objectif(coefficients multipliés␣
 ↪par -1)
    t.minmax="MAX"#Il s'agit d'un problème de maximisation
    t.add_constraint([1, 2, 3], 4)#Première contrainte
    t.add_constraint([2, 5, 6], 8)#Deuxième contrainte
    t.add_constraint([3, 4, 9], 12)#Troisième contrainte
    t.accept_fraction = True#On spécifie qu'on souhaite l'utilisation des␣
 ↪fractions
    t.solve()
    print('---------------------------------------------------------')
    print("\nLa valeur optimale est : %d"%t.obj[-1])
```

```
-----------------------1er exemple-----------------------
                      MINIMISATION
---------------------------------------------------------
J       x1      x2      x3      x4      x5      x6      b
x4      2       1       1       1       0       0       4
x5      1       2       1       0       1       0       8
x6      0       0       1       0       0       1       5
Z       -2      -3      -1      0       0       0       0
Colonne du pivot: 1
Ligne du pivot: 1
Variable entrante : x1
Variable sortante : x4
---------------------------------------------------------
J       x1      x2      x3      x4      x5      x6      b
x1      1       1/2     1/2     1/2     0       0       2
x5      0       3/2     1/2     -1/2    1       0       6
x6      0       0       1       0       0       1       5
Z       0       -2      0       1       0       0       4
Colonne du pivot: 2
Ligne du pivot: 1
Variable entrante : x2
Variable sortante : x1
---------------------------------------------------------
J       x1      x2      x3      x4      x5      x6      b
x2      2       1       1       1       0       0       4
x5      -3      0       -1      -2      1       0       0
x6      0       0       1       0       0       1       5
Z       4       0       2       3       0       0       12
---------------------------------------------------------

La valeur optimale est : -12

--------------2ème exemple(PL1 du rapport)---------------
                      MAXIMISATION
---------------------------------------------------------
```

```
J       x1      x2      x3      x4      b
x3      1       4       1       0       8
x4      1       2       0       1       4
Z       3       9       0       0       0
Colonne du pivot: 1
Ligne du pivot: 2
Variable entrante : x1
Variable sortante : x4
---------------------------------------------------------
J       x1      x2      x3      x4      b
x3      0       2       1       -1      4
x1      1       2       0       1       4
Z       0       3       0       -3      -12
Colonne du pivot: 2
Ligne du pivot: 2
Variable entrante : x2
Variable sortante : x1
---------------------------------------------------------
J       x1      x2      x3      x4      b
x3      -1      0       1       -2      0
x2      1/2     1       0       1/2     2
Z       -3/2    0       0       -9/2    -18
---------------------------------------------------------

La valeur optimale est : 18

--------------3ème exemple(PL2 du rapport)---------------
                      MAXIMISATION
---------------------------------------------------------
J       x1      x2      x3      x4      b
x3      1       7       1       0       1
x4      3       14      0       1       3
Z       3       3       0       0       0
Colonne du pivot: 1
Ligne du pivot: 1
Variable entrante : x1
Variable sortante : x3
---------------------------------------------------------
J       x1      x2      x3      x4      b
x1      1       7       1       0       1
x4      0       -7      -3      1       0
Z       0       -18     -3      0       -3
---------------------------------------------------------

La valeur optimale est : 3

--------------4ème exemple(PL3 du rapport)---------------
                      MAXIMISATION
```

```
---------------------------------------------------------
J       x1      x2      x3      x4      x5      x6      b
x4      1       2       3       1       0       0       4
x5      2       5       6       0       1       0       8
x6      3       4       9       0       0       1       12
Z       3       -5      4       0       0       0       0
Colonne du pivot: 1
Ligne du pivot: 1
Variable entrante : x1
Variable sortante : x4
---------------------------------------------------------
J       x1      x2      x3      x4      x5      x6      b
x1      1       2       3       1       0       0       4
x5      0       1       0       -2      1       0       0
x6      0       -2      0       -3      0       1       0
Z       0       -11     -5      -3      0       0       -12
---------------------------------------------------------

La valeur optimale est : 12
```