Université de Reims Champagne-Ardenne **Pierre Delisle**
Département de Mathématiques et Informatique **CHPS0742 - Optimisation**
2018/2019

Travaux Pratiques 1
Optimisation linéaire
Simplexe : Méthode du pivot – Utilisation du solveur GLPK

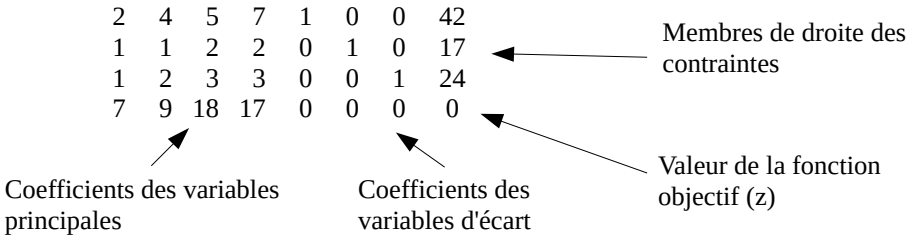
Partie 1 – Simplexe : Méthode du pivot

Présentation du TP

L'algorithme du pivot n'est rien d'autre qu'un formalisme plus automatique à manipuler, mais qui cache tout à fait le sens des opérations, pour appliquer la méthode du simplexe à un problème de programmation linéaire. Prenons l'exemple suivant :

Maximiser	$z = 7x1 + 9x2 + 18x3 + 17x4$
Sous les contraintes	$2x1 + 4x2 + 5x3 + 7x4 \leq 42$
	$x1 + x2 + 2x3 + 2x4 \leq 17$
	$x1 + 2x2 + 3x3 + 3x4 \leq 24$
	$x1, x2, x3, x4 \geq 0$

On peut le représenter de la façon suivante :



La représentation ci-haut correspond à la première étape du formalisme du dictionnaire. Nous verrons plus loin les étapes menant à la solution optimale du problème. L'objectif de ce TP est donc de programmer la méthode du pivot en C et de l'appliquer aux exemples vus en cours et en TD.

Prise en main du logiciel

La partie du programme permettant d'importer, de stocker en mémoire et d'afficher un programme linéaire vous est déjà proposée. Votre travail est de compléter le programme afin de représenter le problème sous la forme du pivot, puis d'y appliquer la méthode. Vous trouverez une archive du programme de base sur le site Web du cours (http://cosy.univ-reims.fr/~pdelisle/enseignement.php).

Cette archive contient tous les modules du programme dont ceux à compléter, ainsi que des fichiers de tests correspondant aux problèmes vus en cours et en TD. Tout d'abord, voici les fichiers que vous n'avez pas à modifier., du moins pas pour la partie principale du TP :

- **types.h** : définition de la structure de données permettant de représenter le problème ;
- **probleme.h** et **probleme.c** : fonctions d'importation et d'affichage d'un programme linéaire ;
- **makefile** : permet de compiler le programme en utilisant la comande **make** ;
- **prob_test.txt**, **prob_cours1_1.txt**, ... : fichiers de problèmes que vous utiliserez pour tester votre programme.

Voici les fichiers que vous avez à modifier

- **pivot.h** et **pivot.c** : a priori vides, vous devrez les compléter par les fonctions demandées plus loin ;
- **simplexe.h** et **simplexe.c** : contient le main avec l'appel des fonctions d'importation et d'affichage déjà écrites. Vous devrez y écrire l'algorithme principal du pivot en appelant les fonctions réalisées plus loin.

Test du programme de base

Téléchargez l'archive, décompressez-là dans un répertoire et compilez le programme avec la commande **make**. Ensuite, exécutez le programme pour voir à quoi il ressemble avec la commande ./simplexe **prob_test.txt**.

Avant de commencer à programmer, prenez quelques instants pour étudier la structure de la variable *probleme*, utilisée dans le main, que vous trouverez dans **types.h**. Elle représente toutes les données du problème. Les variables **nVar** et **nCont** sont respectivement le nombre de variables principales et le nombre de contraintes. La variable typeOpt permet d'indiquer si c'est un problème de maximisation (1) ou de minimisation (0). Le tableau de réels **fonc** représente la fonction objectif, c'est-à-dire les coefficients des variables principales, de sorte que la valeur à la case j soit le coefficient de la variable xj. Les autres membres de la structure permettent de représenter les contraintes. La matrice de réels **cont** représente les coefficients des aij, la valeur à la position ij de la matrice étant le coefficient de la variable xij pour la contrainte i. Le tableau de réels **valCont** correspond à la valeur du membre de droite de chaque contrainte, la valeur à la position i étant associée à la contrainte i. Finalement le tableau d'entiers **signeCont** représente tout simplement le signe de chaque contrainte, la valeur à la position i étant le signe de la contrainte i. On ne considère que les cas où le signe est ≤ (0) ou ≥ (1). Comme vous pouvez le constater, toutes les valeurs du problème initial sont présentes dans la variable **probleme**, celle-ci étant déjà créée et complètement initialisée. C'est à partir de cette variable que vous construirez les autres composantes du programme. C'est maintenant l'heure de programmer.

Représentation du formalisme initial du pivot

La première étape est de représenter le programme linéaire sous le formalisme initial du pivot. Pour ce faire, une matrice de réels suffit (une matrice d'entiers ne permettrait pas de représenter les fractions).

1) Déterminez le nombre de lignes que cette matrice doit contenir en fonction des valeurs initiales du problème.
2) Déterminez le nombre de colonnes que cette matrice doit contenir en fonction des valeur initiales du problème.
3) Dans les fichiers **pivot.h** et **pivot.c**, écrivez le code de la fonction **initMatPivot** qui permet d'allouer la mémoire de la matrice ainsi que de la remplir à partir des données du problème initial.
4) Dans les fichiers **pivot.h** et **pivot.c**, écrivez le code de la fonction **afficherMatPivot** permettant d'afficher la matrice.
5) À partir du main (fichier **simplexe.c**), appelez les fonctions écrites aux deux questions précédentes et vérifiez que vous obtenez bien un résultat semblable au suivant :

```
  2.00   4.00   5.00   7.00   1.00   0.00   0.00  42.00
  1.00   1.00   2.00   2.00   0.00   1.00   0.00  17.00
  1.00   2.00   3.00   3.00   0.00   0.00   1.00  24.00
  7.00   9.00  18.00  17.00   0.00   0.00   0.00   0.00
```

Cette matrice correspond au Dictionnaire I du formalisme du dictionnaire. Les 3 premières lignes sont les contraintes exprimées en fonction des variables actuellement en base, qui sont ici les variables d'écart x5, x6 et x7. La dernière ligne correspond à la fonction objectif. Sur cette ligne, les positions 0 à 3 sont les coefficients des variables hors-base (x1, x2, x3 et x4), les positions 4 à 6 sont les coefficients des variables d'écart (x5, x6 et x7) qui sont bien entendu à 0 parce que ces variables sont actuellement en base, et la case complètement en bas à droite représente la valeur de la solution réalisable actuelle. Celle-ci est évidemment à 0 car elle est associée à une solution de départ où les variables principales sont nulles. Nous sommes maintenant prêts à effectuer une première itération de l'algorithme du simplexe.

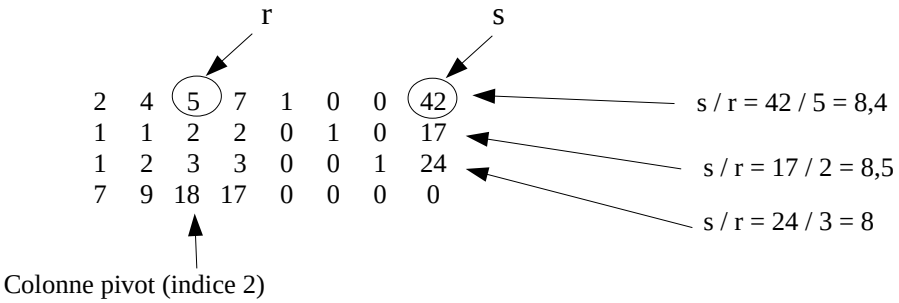
Première itération de la méthode du simplexe

Pour effectuer une itération de la méthode du simplexe, il faut tout d'abord choisir la variable hors-base qui entrera en base. Nous choisirons donc celle qui a le coefficient le plus élevé. Dans le formalisme du pivot, cela équivaut à choisir la colonne de coefficient strictement positif le plus élevé dans la dernière ligne de la matrice. Cette colonne est appelée **colonne pivot**.

6) Dans **pivot.h** et **pivot.c**, écrivez le code de la fonction **selectionnerColPivot** qui retourne l'indice de la colonne pivot pour cette itération. Affichez cette valeur en appelant la fonction à partir du main dans **simplexe.c** et vérifiez que vous obtenez bien l'indice **2**.

Ayant ainsi la variable entrante, il faut maintenant choisir la variable sortante. Dans le formalisme du pivot, cela équivaut à déterminer la **ligne pivot** qui correspondra à l'indice de la ligne de la matrice correspondant à cette variable. Pour tout élément strictement positif r de la colonne pivot, sauf celle de la dernière ligne, on calcule le rapport s/r où s est l'élément de la dernière colonne sur la même ligne que r. La ligne pour laquelle ce rapport est le plus petit (qui

correspond en fait à la variable la plus contraignante sur l'augmentation de la variable qui entre en base) devient la ligne pivot.



Dans ce cas, la ligne où le rapport s/r est le plus petit est celle d'indice 2. Elle devient donc la ligne pivot.

7) Dans **pivot.h** et **pivot.c**, écrivez le code de la fonction **selectionnerLignePivot** qui retourne l'indice de la ligne pivot pour cette itération. Affichez cette valeur en appelant la fonction à partir du main dans **simplexe.c** et vérifiez que vous obtenez bien l'indice **2**.

La valeur de la matrice à l'indice [lignePivot][colonnePivot] est appelé le **pivot**, ici la valeur 3.

Connaissant maintenant les variables entrante et sortante, il faut maintenant faire le changement de variables requis pour obtenir le nouveau dictionnaire. Dans le formalisme du pivot, ceci se fait en 2 étapes : division de tous les éléments de la ligne pivot par le pivot et mise à 0 de toutes les valeurs de la colonne pivot à l'exception du pivot.

8) Dans **pivot.h** et **pivot.c**, écrivez le code de la fonction **diviserLignePivot** qui divise tous les éléments de la ligne pivot par le pivot. Afin de vérifier son bon fonctionnement, appelez la fonction dans le main, affichez ensuite la matrice résultante et vérifiez que vous obtenez bien le résultat suivant :

```
  2.00   4.00   5.00   7.00   1.00   0.00   0.00  42.00
  1.00   1.00   2.00   2.00   0.00   1.00   0.00  17.00
  0.33   0.67   1.00   1.00   0.00   0.00   0.33   8.00
  7.00   9.00  18.00  17.00   0.00   0.00   0.00   0.00
```

Il faut maintenant mettre à 0 tous les éléments de la colonne pivot, à l'exception du pivot lui-même. Pour ce faire, il faut effectuer des combinaisons linéaires de la nouvelle ligne pivot et de chaque autre ligne. Autrement dit, pour chaque ligne, il faut additionner la ligne pivot après l'avoir multipliée par un certain coefficient qui mettra la valeur de la colonne pivot à 0.

Pour le cas de la première ligne, on peut constater que la valeur à la colonne pivot est de 5. Considérant que la nouvelle valeur du pivot est de 1, il faut soustraire (5 / 1) * 1 = 5 pour que la valeur de la première ligne soit à 0. Pour conserver l'équivalence du système d'équations, il faut donc aussi soustraire, à chaque élément de la ligne courante, la valeur de même indice de la ligne pivot multipliée par le ratio 5 / 1. De façon générale, il faut soustraire, pour chaque élément [i,j] de la ligne i, à l'exception de la ligne pivot,

(matrice[i][colPivot] / matrice[lignePivot][colPivot]) * matrice[lignePivot][j].

Pour la première ligne, la valeur 2 deviendra donc 2 - ((5 / 1) * 0.33...) = 0.33...., la valeur 4 deviendra donc 4 - ((5 / 1) * 0.66...) = 0.66..., la valeur 5 deviendra donc 5 - ((5 / 1) * 1...) = 0, et ainsi de suite.

9) Dans **pivot.h** et **pivot.c**, écrivez le code de la fonction **miseAZeroColPivot** qui met à 0 toutes les valeurs de la colonne pivot, à l'exception de celle de la ligne pivot, et qui effectue les combinaisons linéaires nécessaires. Afin de vérifier son bon fonctionnement, appelez la fonction dans le main, affichez ensuite la matrice résultante et vérifiez que vous obtenez bien le résultat suivant :

```
  0.33   0.67   0.00   2.00   1.00   0.00  -1.67   2.00
  0.33  -0.33   0.00   0.00   0.00   1.00  -0.67   1.00
  0.33   0.67   1.00   1.00   0.00   0.00   0.33   8.00
  1.00  -3.00   0.00  -1.00   0.00   0.00  -6.00 -144.00
```

Cette nouvelle matrice correspond en fait au Dictionnaire II de la méthode « traditionnelle ». Les valeurs de la dernière colonne sont les valeurs des variables de base pour cette nouvelle solution réalisable. Vous remarquerez aussi que la valeur tout en bas à droite de la matrice pivot n'est plus nulle mais négative : l'opposé de cette valeur correspond à la valeur de la fonction objectif de cette solution.

Vous constaterez que l'algorithme n'est pas terminé : il reste un coefficient positif pour une des variables hors-base du dictionnaire. Il faut donc appliquer à nouveau les fonctions développées précédemment et ce, jusqu'à ce que toutes les valeurs des coefficients hors-base soient négatifs (évidemment, sans considérer les coefficients nuls qui correspondent aux variables de base).

10) Dans le fichier **simplexe.c**, ajoutez le code permettant d'itérer les étapes jusqu'à la fin de l'algorithme. Vérifiez bien que vous obtenez alors le résultat suivant où la valeur de z (147) est optimale :

```
  0.00   1.00   0.00   2.00   1.00  -1.00  -1.00   1.00
  1.00  -1.00   0.00   0.00   0.00   3.00  -2.00   3.00
  0.00   1.00   1.00   1.00   0.00  -1.00   1.00   7.00
  0.00  -2.00   0.00  -1.00   0.00  -3.00  -4.00 -147.00
```

11) Après la fin de l'algorithme, affichez les valeurs des variables principales et des variables d'écart. Vous devriez obtenir un résultat semblable au suivant :

```
Valeur des variables principales :
x1 =   3.00
x2 =   0.00
x3 =   7.00
x4 =   0.00
Valeur des variables d'ecart :
x5 =   1.00
x6 =   0.00
x7 =   0.00
Valeur de la fonction objectif z = 147.00
```

12) Afin de vérifier votre programme, exécutez-le avec les autres fichiers de problèmes fournis, ceux-ci correspondant à des problèmes vus en cours et en TD.

Questions supplémentaires pour les rapides et les motivés

13) Vous savez qu'il n'est pas nécessaire de choisir la variable dont le coefficient est le plus grand pour entrer en base : n'importe quelle variable de coefficient strictement positif fait l'affaire. Établir un autre critère de choix amènera quand même à la solution optimale, mais peut-être en nombre d'étapes différent (plus grand ou plus petit selon le cas). Écrivez le code permettant de choisir choisir différemment la variable entrante (plus petit indice ou aléatoirement) et comparez les résultats obtenus (valeur de la solution optimale et nombre d'étapes pour y parvenir).

14) Écrivez le code permettant de transformer une contrainte ≥ en contrainte ≤.

15) Écrivez le code permettant de traiter un problème de minimisation. Il s'agit de construire la matrice pivot en ramenant le problème à une maximisation.

16) À partir d'une matrice pivot déjà construite, écrivez le code permettant de savoir si au moins un des membres de droite des équations des contraintes (bi) est négatif. Ceci indique que la solution initiale nulle n'est pas réalisable.

17) À partir d'une matrice pivot déjà construite, écrivez le code permettant de savoir si sous les coefficients des variables principales de la fonction objectif sont négatifs.

18) Si la solution initiale nulle d'un problème n'est pas réalisable et si les coefficients de la fonction objectif sont tous négatifs, ce problème est dual-réalisable. Écrivez le code permettant de construire la matrice pivot du problème dual et résolvez-le.

Partie 2 – Utilisation du solveur GLPK

L'objectif de la partie 1 de ce TP était d'apprendre et de programmer un des formalismes de résolution de programmes linéaires basé sur la méthode du simplexe. Par contre, dans de vraies applications d'optimisation, il n'est pas nécessaire (et surtout déconseillé !) de programmer vous même cet algorithme. En fait, il existe déjà différents logiciels et bibliothèques qui font ce travail probablement mieux que vous ! Un de ceux-ci est GLPK (GNU Linear Programming Kit), gratuit et disponible à l'adresse suivante : http://www.gnu.org/software/glpk/.

GLPK est un ensemble de routines réunies dans une bibliothèque que l'on peut appeler, entres autres, à partir d'un programme écrit en langage C. Le code donné à la page suivante permet de résoudre le premier programme linéaire vu en cours :

Maximiser $z = 400x1 + 200x2$

Sous les contraintes $30x1 + 20x2 \leq 6000$
$40x1 + 10x2 \leq 4000$
$x1 \geq 0, x2 \geq 0$

Vous trouverez le code source de ce programme sur votre bureau virtuel, dans le répertoire « Documents », dans le fichier **prob1.c**. La compilation de ce programme n'est pas différente de la compilation de n'importe quel programme C :

gcc -c prob1.c

Par contre, pour l'édition de liens, il faut spécifier la bibliothèque glpk :

gcc -o prob1 prob1.o -lglpk

Finalement, pour exécuter le programme, on procède comme d'habitude :

./prob1

L'exécution affichera un résultat semblable au suivant à l'écran :

```
GLPK Simplex Optimizer, v4.45
2 rows, 2 columns, 4 non-zeros
*     0: obj =   0.000000000e+00  infeas =  0.000e+00 (0)
*     2: obj =   6.400000000e+04  infeas =  0.000e+00 (0)
OPTIMAL SOLUTION FOUND

z = 64000; x1 = 40; x2 = 240;
```

La solution optimale a donc bel et bien été trouvée par GLPK (quelle surprise !).

1) Étudiez attentivement le programme donné en exemple à la page suivante et testez le soit en recopiant le code, soit en le téléchargeant directement à partir de votre bureau virtuel.

2) Utilisez GLPK pour résoudre le problème de la partie 1 de ce TP et pour vérifier les solutions des cours et des TD.

Pour plus d'informations sur l'utilisation de GLPK, vous trouverez une documentation complète dans le fichier **glpk.pdf** présent sur votre bureau virtuel, dans le répertoire « Documents ».

```c
#include <stdio.h>
#include <stdlib.h>
#include <glpk.h>                                   /*fichier d'entete de la bibliotheque GLPK*/

int main (void) {
    glp_prob *prob1;                                /*pointeur sur un objet probleme*/
    int ia[1+4];                                    /*indices de lignes de la matrice des contraintes*/
    int ja[1+4];                                    /*indices de colonnes de la matrice des contraintes*/
    double ar[1+4];                                 /*valeurs des coefficients de la matrice des contraintes*/
    double z, x1, x2;

    prob1 = glp_create_prob();                      /*creation d'un objet probleme*/
    glp_set_prob_name(prob1, "probCours1");         /*definition d'un nom symbolique pour le probleme*/

    glp_set_obj_dir(prob1, GLP_MAX);                /*definition du type d'optimisation, ici maximisation*/
    glp_add_rows(prob1, 2);                         /*addition de 2 contraintes au probleme*/
    glp_set_row_name(prob1, 1, "machine1");         /*definition d'un nom symbolique pour la contrainte 1*/
    glp_set_row_bnds(prob1, 1, GLP_UP, 0.0, 6000.0);/*definition du type et des limites de la contrainte 1 :*/
                                                    /*GLP_UP : borne superieure --> contrainte de type <= */
                                                    /*0.0 : valeur de la borne inferieure (ignore ici)*/
                                                    /*6000.0 : valeur de la borne superieure (bi)*/
    glp_set_row_name(prob1, 2, "machine2");         /*definition d'un nom symbolique pour la contrainte 2*/
    glp_set_row_bnds(prob1, 2, GLP_UP, 0.0, 4000.0);/*definition du type et des limites de la contrainte 2*/

    glp_add_cols(prob1, 2);                         /*addition de 2 variables au probleme*/
    glp_set_col_name(prob1, 1, "x1");               /*definition d'un nom symbolique pour la variable x1*/
    glp_set_col_bnds(prob1, 1, GLP_LO, 0.0, 0.0);   /*definition du type et des limites de la variable x1*/
                                                    /*GLP_LO : borne inferieure --> x1 >= 0*/
                                                    /*ici, le premier 0.0 est la borne, le 2e est ignore*/
    glp_set_obj_coef(prob1, 1, 400.0);              /*definition du coefficient de la variable x1 (cj)*/
    glp_set_col_name(prob1, 2, "x2");               /*definition d'un nom symbolique pour la variable x2*/
    glp_set_col_bnds(prob1, 2, GLP_LO, 0.0, 0.0);   /*definition du type et des limites de la variable x2*/
    glp_set_obj_coef(prob1, 2, 200.0);              /*definition du coefficient de la variable x2*/

    ia[1] = 1, ja[1] = 1, ar[1] = 30.0;             /*definition des elements non nuls de la matrice des*/
    ia[2] = 1, ja[2] = 2, ar[2] = 20.0;             /*contraintes (coefficients aij)*/
    ia[3] = 2, ja[3] = 1, ar[3] = 40.0;             /*ia : indices de lignes*/
    ia[4] = 2, ja[4] = 2, ar[4] = 10.0;             /*ja : indices de colonnes*/
                                                    /*ar : coefficients numeriques */
                                                    /*ex : 30x1 de la contrainte 1 :*/
                                                    /*ligne 1, colonne 1, valeur 30.0*/

    glp_load_matrix(prob1, 4, ia, ja, ar);          /*chargement de la matrice des contraintes definie par les
                                                    /*indices de lignes, les indices de colonnes et les*/
                                                    /*valeurs des coefficients. Le parametre 4 signifie qu'il*/
                                                    /*y a 4 valeurs a entrer dans la matrice*/

    glp_simplex(prob1, NULL);                       /*resolution du probleme par le simplexe. Le 2e */
                                                    /*parametre permet de definir des options de resolution.*/
                                                    /*Ici on utilise la configuration par defaut*/
                                                    /*Les resultats sont stockes dans l'objet probleme*/

    z = glp_get_obj_val(prob1);                     /*Recuperation de la valeur de la fonction objectif*/
    x1 = glp_get_col_prim(prob1, 1);                /*Recuperation de la valeur de la 1ere colonne (x1)*/
    x2 = glp_get_col_prim(prob1, 2);                /*Recuperation de la valeur de la 2e colonne (x2)*/

    printf("\nz = %g; x1 = %g; x2 = %g;\n", z, x1, x2);/*affichage des resultats a l'ecran*/
    glp_delete_prob(prob1);                         /*suppression de l'objet probleme*/
    return 0;
}
```