Chapitre 2

Utiliser `optimtool` pour résoudre les problèmes PLNE

Dans le chapitre précédent, nous avons montré comment utiliser la boite à outils optimtool pour résoudre les problèmes de la programmation linéaire. Le solveur de PL constitue une base pour les autres solveurs, particulièrement, le solveur de la PLNE, à travers la résolution des PL-relaxé à chaque nœud de l'arbre Branch & Bound. Dans ce chapitre, nous allons montrer à travers des illustrations guidées comment utiliser cette boite à outils pour résoudre les de la programmation linéaire en nombres entiers.

2.1. Programmation linéaire en nombres entiers binaires

La programmation linéaire en nombres entiers binaires est le problème de trouver un vecteur binaire x qui minimise une fonction linéaire $f^{\mathrm{T}}x$ soumise à des contraintes linéaires. La forme générale du problème est donnée comme suit :

$\min_x f^{\mathrm{T}}x,$

Sachant que :

$$\begin{cases} A.x \leq b \\ Aeq.x = beq \\ x \in \{0,1\} \end{cases}$$

La syntaxe générale d'utilisation du solveur est la suivante :

```
[x,fval,exitflag,output] = bintprog(f,A,b,Aeq,Beq,x0,options)
```

Les tableaux suivants résument les arguments d'entrée et de sortie pour le solveur `bintprog` :

Arguments d'entrée		
`f`	Vecteur des coefficiants de la fonction objectif linéaire.	
`A`	Matrice des coefficiants pour les contraintes d'inégalité linéaires.	
`b`	Vecteur du second membre pour les contraintes d'inégalité linéaires.	
`Aeq`	Matrice des coefficiants pour les contraintes d'égalité linéaires.	
`beq`	Vecteur du second membre pour les contraintes d'égalité linéaires.	
`x0`	Point initial de l'algorithme.	
`options`	Structure contenant les options de l'algorithme.	
`problem`	`f`	Vecteur des coefficiants de la fonction objectif linéaire.
	`Aineq`	Matrice des coefficiants pour les contraintes d'inégalité linéaires.
	`bineq`	Vecteur du second membre pour les contraintes d'inégalité linéaires.
	`Aeq`	Matrice des coefficiants pour les contraintes d'égalité linéaires.
	`beq`	Vecteur du second membre pour les contraintes d'égalité linéaires.
	`x0`	Point initial pour x.
	`solveur`	'bintprog'
	`options`	Structure des options crée par `optimset`.

Tableau 2.1. Arguments d'entrée – bintprog -

Arguments de sortie		
exitflag	Entier identifiant la raison pour laquelle l'algorithme à terminé.	
	1	Fonction convergée vers une solution x.
	0	Nombre d'itérations dépassé options.MaxIter.
	-2	Aucune solution réalisable n'a été trouvée.
	-4	Nombre de nœud recherché est dépassé options.MaxNodes.
	-5	Temps de recherche est dépassé options.MaxTime.
	-6	Nombre d'itérations du solveur LP effectuée à un nœud pour résoudre le Problème de relaxation LP est dépassé options.MaxRLP.
output	Structure contenant des informations sur l'optimisation. Les domaines de la structure sont :	
	iterations	Nombre d'itérations atteint.
	nodes	Nombre de nœuds recherchés.
	time	Temps d'exécution de l'algorithme.
	algorithm	Algorithme d'optimisation utilisé.
	branchStrategy	Stratégie utilisée pour sélectionner la branche variable.
	nodeSearchStrategy	Stratégie utilisée pour sélectionner le prochain nœud dans l'arbre de recherche.
	message	Quitter le message.

Tableau 2.2. Arguments de sortie – bintprog -

Vous pouvez utiliser `optimset` pour définir ou modifier les valeurs des champs dans la structure des options :

Options	
`BranchStrategy`	Stratégie de branchement utilisée par l'algorithme pour sélectionner la branche variable dans l'arbre de recherche. Les choix sont : ○ `'mininfeas'` : choisissez la variable avec l'infaisabilité minimale entière, qui est, la variable dont la valeur est la plus proche de 0 ou 1 mais pas égal à 0 ou 1. ○ `'maxinfeas'` : Choisissez la variable avec l'infaisabilité entière maximum, c'est-à-dire la variable dont la valeur est la plus proche de 0,5 (par défaut).
`Diagnostics`	Affiche des informations de diagnostic.
`Display`	niveau d'affichage. `'off'` n'affiche aucune sortie; `'iter'` affiche la sortie à chaque itération; `'final'` (par défaut) affiche uniquement la sortie finale.
`MaxIter`	Nombre maximum d'itérations autorisé.
`MaxNodes`	Nombre maximum de solutions, ou nœuds, de la fonction de recherche.
`MaxRLPIter`	Nombre maximum d'itérations du solveur LP pour résoudre le problème de relaxation linéaire à chaque nœud.
`MaxTime`	Durée maximale en secondes pendant laquelle la fonction s'exécute.
`NodeDisplayInterval`	Intervalle d'affichage du nœud.
`NodeSearchStrategy`	Stratégie utilisée par l'algorithme pour sélectionner le nœud suivant à rechercher dans l'arborescence de recherche. Les choix sont : ○ «`df`» - Depth first search strategy. Stratégie de recherche en profondeur d'abord. À chaque nœud dans l'arborescence de recherche, s'il y a un nœud enfant à un niveau plus bas qui n'a pas encore été exploré, l'algorithme choisit cet enfant pour chercher. Sinon, l'algorithme se déplace au nœud à un niveau plus haut dans l'arbre et choisit un nœud enfant à un niveau plus bas que ce nœud. ○ «`bn`» - Best node search strategy. Meilleure stratégie de recherche de nœuds, choisit le nœud avec la borne la plus basse sur la fonction objective.
`TolFun`	Tolérance d'arrêt sur la valeur de la fonction objectif.
`TolXInteger`	Tolérance dans laquelle la valeur de la variable est considérée comme entière.
`TolRLPFun`	Tolérance d'arrêt sur la valeur de la fonction objectif d'un problème de relaxation linéaire.

Tableau 2.3. Options – bintprog -

2.2. Illustrations de la PLNB

Illustration 2.1

Cette illustration montre comment utiliser l'outil d'optimisation avec le solveur bintprog pour minimiser une fonction linéaire soumise à des contraintes linéaires.

Considérez le problème d'optimisation suivant :

$$\min f(x) = -9x_1 - 5x_2 - 6x_3 - 4x_4$$

`S.C :`

$$6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 9$$
$$x_3 + x_4 \leq 1$$
$$-x_1 + x_3 \leq 0$$
$$-2x_2 + x_4 \leq 0$$

$x_1, x_2, x_3 \; et \; x_4 \in \{0,1\}$

Étape 1: Écrivez le problème sous la forme matricielle.

- Vous remarquez que le problème est un problème de minimisation, donc vous entrez le vecteur des coefficients de la fonction objectif te qu'il est. Vous avez quatre inégalités, et vous n'avez aucune égalité ni contraintes de borne (les variables sont des binaires, elles prennent soit 0 soit 1).
- Toutes les inégalités sont sous la forme $A.x \leq b$.
- Ecrivez le problème sous la forme matricielle, et extraire les vecteurs et les matrices des coefficients.

Étape 2: configurez et exécutez le problème à l'aide de l'outil d'optimisation.

- Saisir `optimtool` dans **Command Window** (la fenêtre de commande) pour ouvrir l'outil d'optimisation (**Optimization Tool**).
- Sélectionnez « **bintprog** » dans la liste des solveurs. Vous remarquez qu'il n'existe pas de champ « **Algorithm** », parce qu'un seul algorithme est implimenté dans ce solveur qui est l'algorithme Brach & Bound.

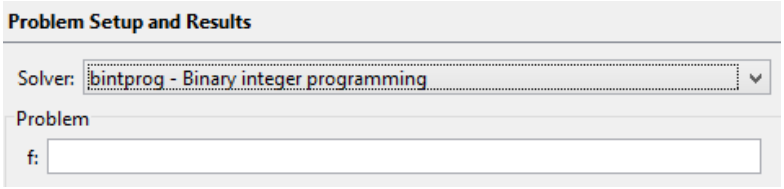

Figure 2.1. Choix du solveur bintprog.

Définissez les contraintes.

- Pour créer des variables pour la fonction objectif, faire entrez les coefficients [-9; -5; -6; -4] dans le champ f.
- Pour créer des variables pour les contraintes d'inégalités, faire entrez [6 3 5 2; 0 0 1 1; -1 0 1 0; 0 -1 0 1] dans le champ A et entrez [9; 1; 0; 0] dans le champ b.
- Pour les contraintes d'égalité, vous les laissez vides.

Figure 2.2. Saisie des données - bintprog.

- Pour un premier lancement, l'algorithme choisit le point de départ (par défaut [0;0]. Après la première exécution du solveur, vous pouvez spécifier un point de départ quelconque dans le champ « **Start point** » (Point de départ).

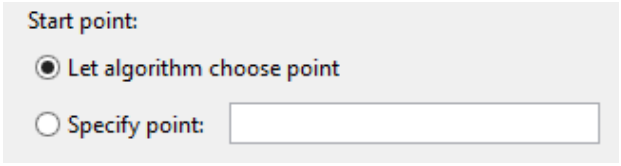

Figure 2.3. Point de départ - bintprog.

- Dans le volet Options, développez l'option « **Display to command window** » (Afficher dans la fenêtre de commande) si nécessaire, et sélectionnez « **Iterative** » (itératif) pour afficher les informations d'algorithme à « **Command Window** » (la Fenêtre de commande) pour chaque itération.
- Dans le champ « **Nodeinterval** », vous pouvez spécifiez l'intervalle d'affichage du nœud. Comme vous pouvez le laissez par défaut. Il définit le pas d'affichage de la valeur de la fonction objectif.
- De plus, pour afficher un rapport de diagnostic détaillé dans la fenêtre de commande, cochez « **Show diagnostics** ».

Figure 2.4. Options d'affichage - bintprog.

- Vous pouvez aussi spécifiez des conditions d'arrêts dans l'option « **Stopping criteria** ». Comme vous pouvez les laissez par défaut. Pour l'algorithme Branch & Bound les critères d'arrêt sont donnés par la sous-fenêtre suivante :

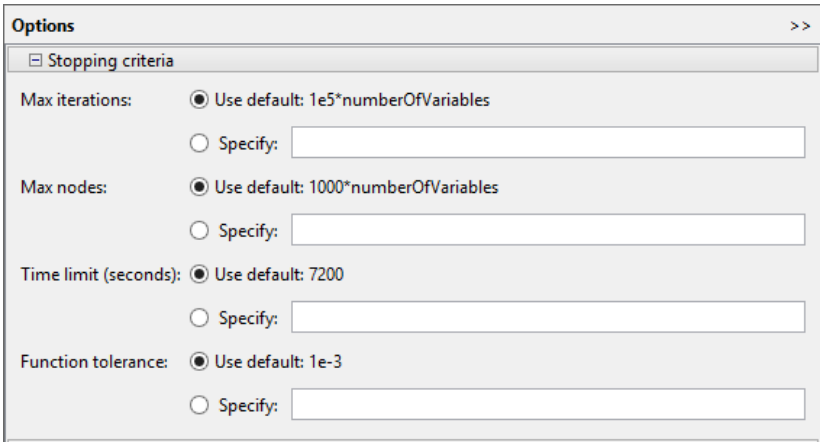

Figure 2.5. Critères d'arrêt - bintprog.

Pour l'algorithme simplex du solveur « **linprog** », qui est appelé à chaque nœud pour résoudre un PL-relaxé sont donnés par la sous-fenêtre suivante :

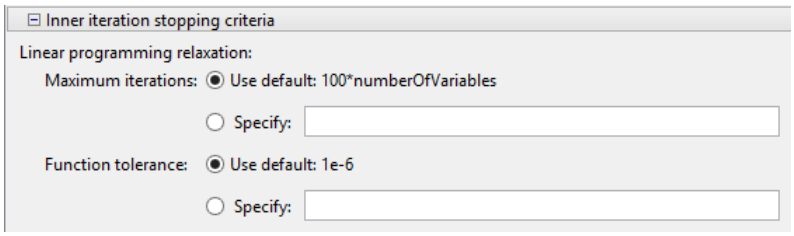

Figure 2.6. Critères d'arrêt – PL-Relaxé.

- Vous pouvez aussi configurer votre solveur, vous choisissez la stratégie de recherche des nœuds dans le graphe et la stratégie de branchement.

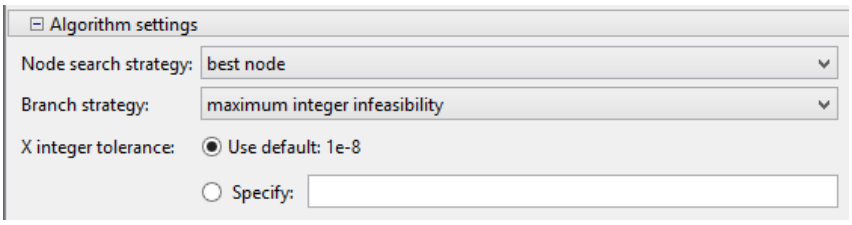

Figure 2.7. Stratégies de recherche et de branchement.

- Cliquez sur le bouton « **Start** » (Démarrer), comme illustré dans la figure suivante.

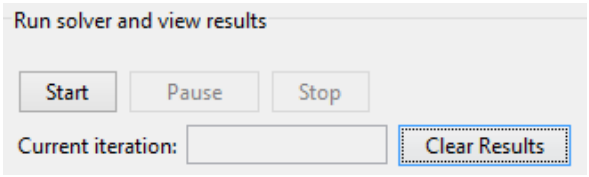

Figure 2.8. Lancement du solveur – Bintprog.

- Lorsque l'algorithme se termine, sous « **Run solver and view results** », les informations suivantes s'affichent:

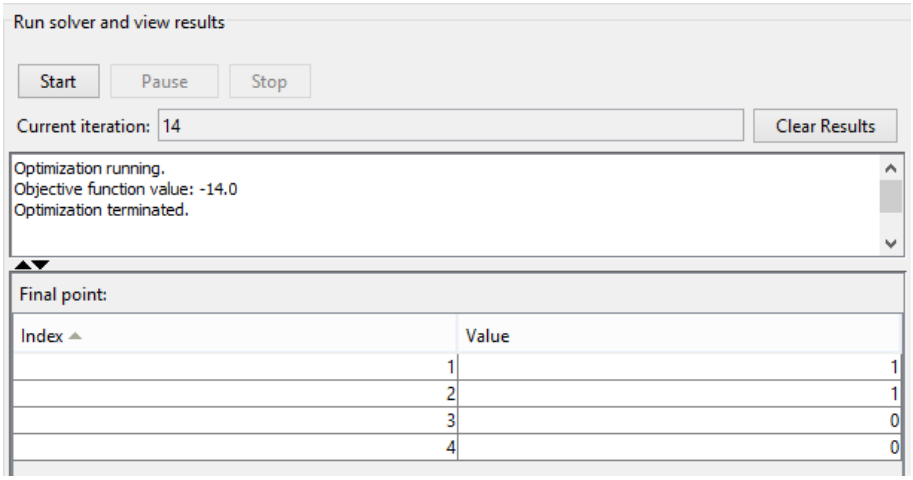

Figure 2.9. Visualisation des résultats – Bintprog.

- La valeur « **Current iteration** » (d'itération actuelle) à la fin de l'algorithme, qui est 14 pour cette illustration.
- La valeur finale de la fonction objectif à la fin de l'algorithme est :
  Valeur de la fonction objectif: 14.0
- Le message de fin d'algorithme est :

```
Optimization running.
Objective function value: -14.0
Optimization terminated.
```

- Le point final, qui est pour ce problème

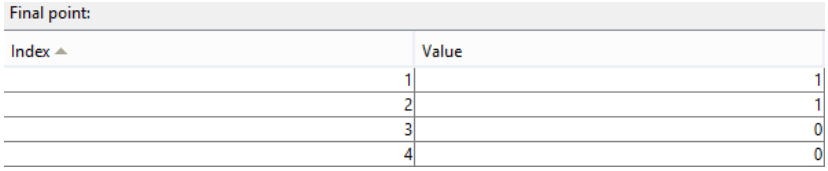

Figure 2.10. Point final – Bintprog.

- Dans « **Command Window** » (la fenêtre de commande), les informations de l'algorithme sont affichées pour chaque itération:

```
Diagnostic Information

Number of variables: 4


Number of 0-1 binary integer variables:     4
Number of linear inequality constraints:    4
Number of linear equality constraints:      0

Algorithm selected
   LP-based branch-and-bound



End diagnostic information
Explored   Obj of LP   Obj of best   Unexplored   Best lower    Relative gap
 nodes    relaxation  integer point     nodes    bound on obj  between bounds
      1          -15             -          2             -15              -
*     2           -9            -9          1             -15            60%
*     7          -14           -14          0           -14.4           2.7%
Optimization terminated.
>>
```

Si vous travaillez sur la fenêtre de commande :

Vous pouvez charger les matrices et les vecteurs `A, Aeq, b, beq et f` dans l'espace de travail MATLAB (MATLAB workspace) s'ils sont déjà créés (`file.mat`) avec la commande :

```
>> load nomfichier
```

Ou le créé directement dans l'espace de travail MATLAB.

Illustration 2.2

Prenez le problème de l'illustration précédente et vous créez ces matrices et vecteurs dans l'espace de travail MATLAB.

- dans l'espace de travail MATLAB. Vous cliquez sur nouvelle matrice (New variable) ou tapez Ctrl+N.
- une variable est créé et nommez-la.

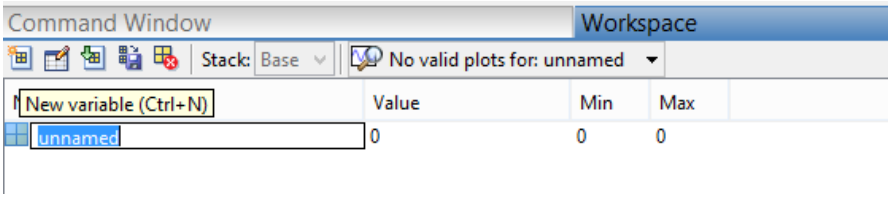

Figure 2.11. Création d'une variable dans l'espace de travail Matlab.

- remplir le vecteur ou la matrice et enregistrez.

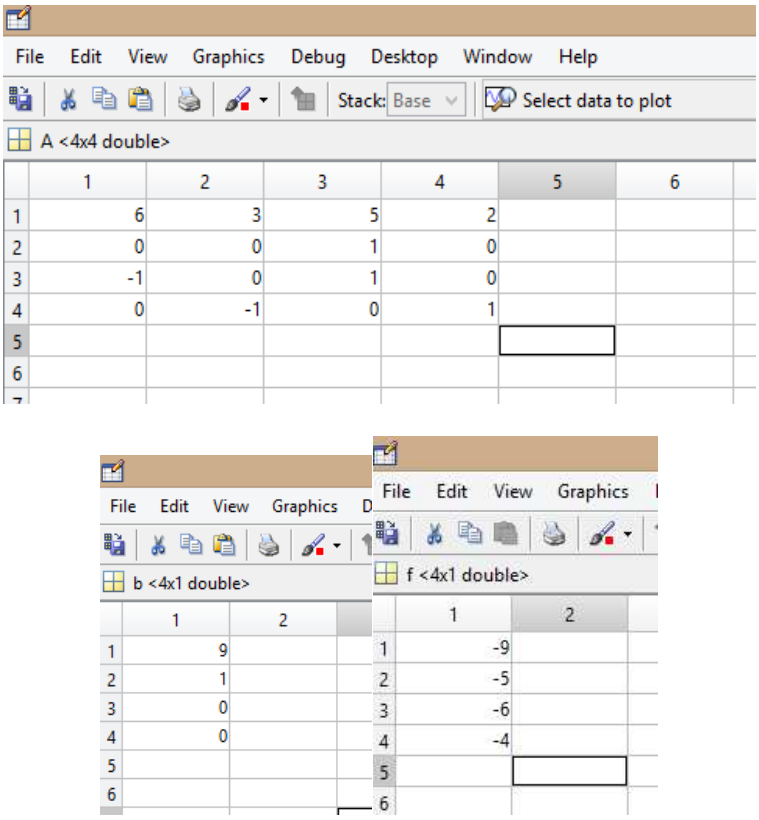

Figure 2.12. Chargement des variables.

- Donc la matrice `A` et les vecteurs `b` et `f` sont créés et stockés dans le path Matlab.

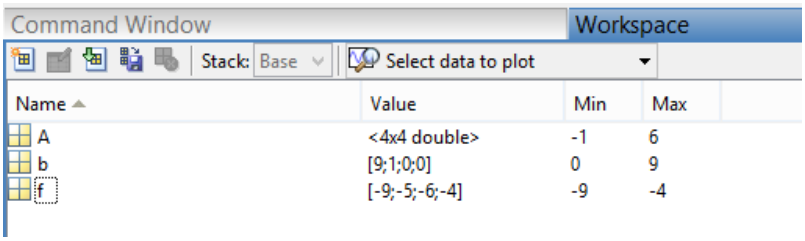

Figure 2.13. Visualisation des variables dans l'espace de travail Matlab.

Vous chargez les matrices et les vecteurs `A, b, et f` dans l'espace de travail MATLAB avec la commande `>> load nomfichier` ou ouvrir-les directement. La sous fenêtre précédente de l'espace de travail MATLAB s'affiche.

Puis, vous pouvez utiliser bint`prog` dans la fenêtre de commande pour résoudre le problème:

```
x = bintprog(f,A,b)
```

Les résultats affichés sont donnés par la figure 2.14. Il est à noter que l'affichage itératif doit être défini à l'aide de l'optimset qui n'est pas définit dans la présente illustration.

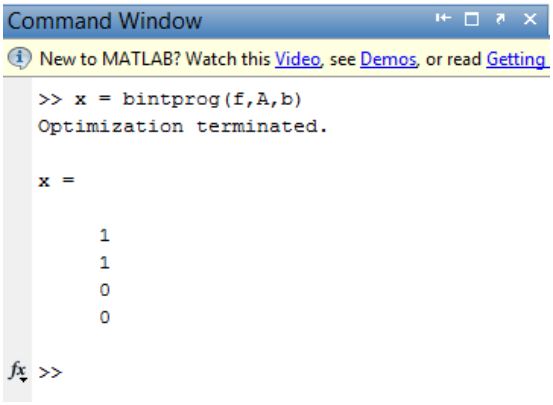

Figure 2.14. Résultats d'illustration 1.2.

Illustration 1.3

Une troisième alternative est d'écrire un M-file (programmation en Matlab) `monfichier.m` (`Resolution_bintprog.m`) et vous déclarer les matrices et les vecteurs `A, b` et `f` (vous pouvez les chargés directement).

Dans le même fichier vous écrivez

```
% Problème de programmationlinéaire en nombres entiers binaires
% Chargement des f,A,b

f = [-9; -5; -6; -4];
A = [6 3 5 2; 0 0 1 1; -1 0 1 0; 0 -1 0 1];
b = [9; 1; 0; 0];
x = bintprog(f,A,b)
```

Puis vous exécuter.

Après l'exécution du `fichier.mat` (`Resolution_bintprog.m`) des paramètres et des variables sont créé automatiquement.

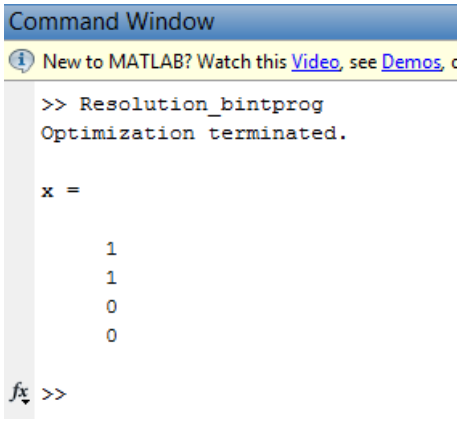

Figure 2.15. Résultats d'illustration 1.3.