

## 3 Outils de PLNE

### 3.1 MATLAB : Toolbox Optimisation

MATLAB propose différents outils de résolution de problèmes d'optimisation au sein de sa toolbox "Optimisation". Pour ce BE, nous utiliserons *linprog* pour la résolution des systèmes linéaires et *intlinprog* pour des variables de décision binaires. Ces deux outils peuvent être utilisés soit depuis leur interface graphique, soit en faisant directement appel aux solveurs tels des fonctions MATLAB.

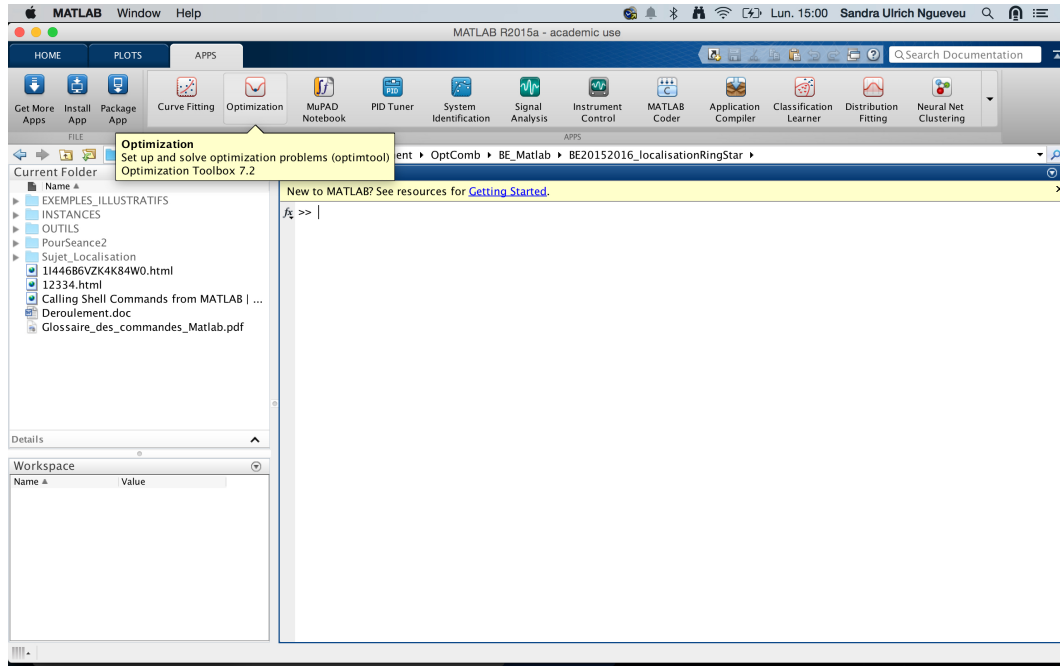


FIGURE 1 – Lancer l'interface graphique de la toolbox "Optimisation"

#### *linprog*

Linprog peut résoudre les problèmes formulés sous la forme suivante :

$$\min f^T x \quad (1)$$

s. c.

$$A.x \leq b \quad (2)$$

$$Aeq.x = beq \quad (3)$$

$$lb \leq x \leq ub \quad (4)$$

avec  $f$ ,  $x$ ,  $b$ ,  $beq$ ,  $lb$  et  $ub$  des vecteurs, tandis que  $A$  et  $Aeq$  sont des matrices.

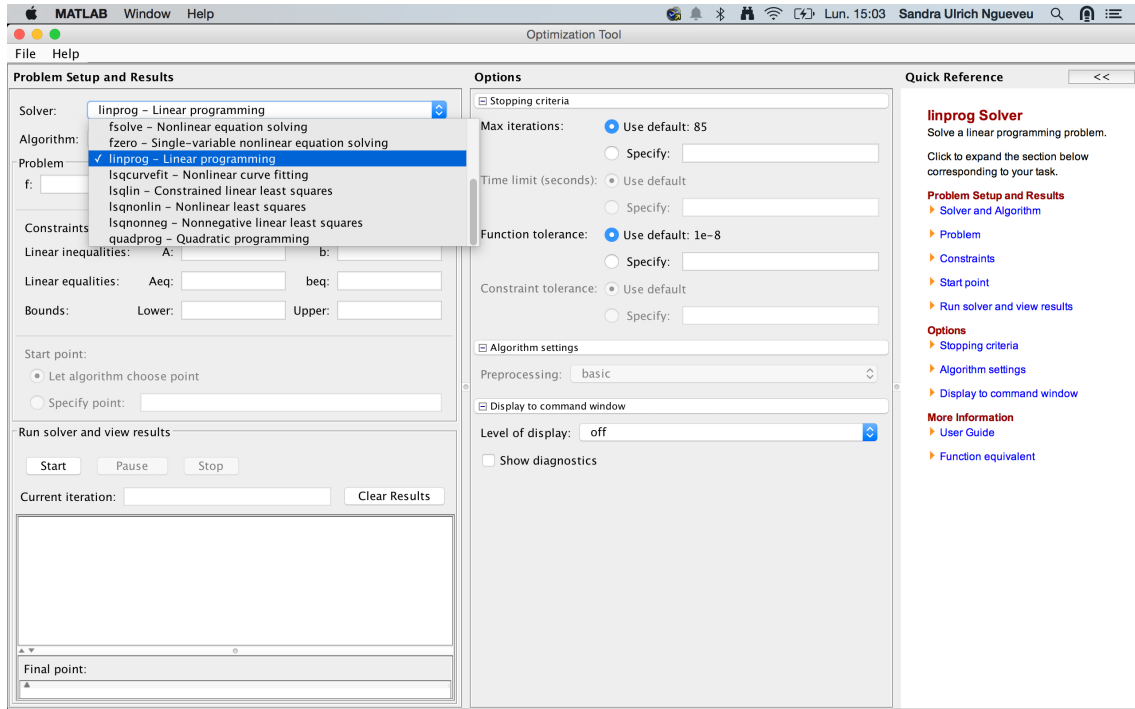


FIGURE 2 – LinProg : Interface graphique

Interface graphique : voir Figure 2

Appels directs au solveur :

```
>> x = linprog(f,A,b)
>> x = linprog(f,A,b,Aeq,beq)
>> x = linprog(f,A,b,Aeq,beq,lb,ub)
>> x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
>> x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
>> x = linprog(problem)
>> [x,fval] = linprog(...)
>> [x,fval,exitflag] = linprog(...)
>> [x,fval,exitflag,output] = linprog(...)
>> [x,fval,exitflag,output,lambda] = linprog(...)
```

- $x_0$  : point de départ de l'optimisation disponible uniquement sous certaines options (lorsque l'option LargeScale est paramétrée 'off' en utilisant optimset). L'algorithme par défaut ainsi que le simplexe ignore  $x_0$ .
- options : à fixer en utilisant optimset
- exitflag : entier spécifiant la raison de l'arrêt de l'algorithme
  - 1 : le solveur a convergé vers la solution  $x$
  - 0 : le nombre d'itérations maximum a été dépassé (option MaxIter)

- -2 : aucune solution réalisable n'a été trouvée
- -3 : le problème n'est pas borné
- -4 : la valeur NaN a été rencontrée au cours de l'exécution de l'algorithme
- -5 : le primal et le dual sont tous les deux infaisables
- -7 : aucun progrès n'a pu être fait dans la direction de recherche choisie
- output : structure contenant les informations concernant l'optimisation
  - Iterations : le nombre d'itérations de l'algorithme
  - Algorithm : spécifie l'algorithme d'optimisation utilisé
  - Cgiterations : 0 (utile pour la rétro-compatibilité)
  - Message : message de sortie de fonction

## intlinprog

Intlinprog peut résoudre les problèmes formulés sous la forme suivante :

$$\min f^T x \quad (5)$$

s. c.

$$A.x \leq b \quad (6)$$

$$Aeq.x = beq \quad (7)$$

$$lb \leq x \leq ub \quad (8)$$

$$x(intcon) \text{ sont entiers} \quad (9)$$

avec  $f$ ,  $x$ ,  $b$ ,  $beq$ ,  $lb$ ,  $ub$  et  $intcon$  des vecteurs, tandis que  $A$  et  $Aeq$  sont des matrices.

Interface graphique : A priori non disponible

Appels directs au solveur : voir Figure 3

```
>> x = intlinprog(f,intcon,A,b)
```

```
>> x = intlinprog(f,intcon,A,b,Aeq,beq)
```

```
>> x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub)
```

```
>> x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub,options)
```

```
>> x = intlinprog(problem)
```

```
>> [x,fval] = intlinprog(...)
```

```
>> [x,fval,exitflag] = intlinprog(...)
```

```
>> [x,fval,exitflag,output] = intlinprog(...)
```

— exitflag / output : voir linprog

— options

— BranchingRule : "maxpscost" (pseudocoût maximum, par défaut), "mostfractional" (plus proche de 0.5) ou "maxfun" (plus grand coef de fonction objectif)

— CutGeneration : niveau de génération de coupes ; "none" (aucune), "basic" (niveau *normal*, par défaut), "intermediate" (un peu plus de types de coupes) ou "advanced" (utilise la plupart des types de coupes)

— CutGenMaxIter : "none" = désactivé, entier de 1 à 50 = nombre de tentatives de génération de coupes avant branchements (10 par défaut)

— Display : "off" ou "none" = aucun, "final" = valeurs finales ou "iter" = à chaque itération (par défaut)

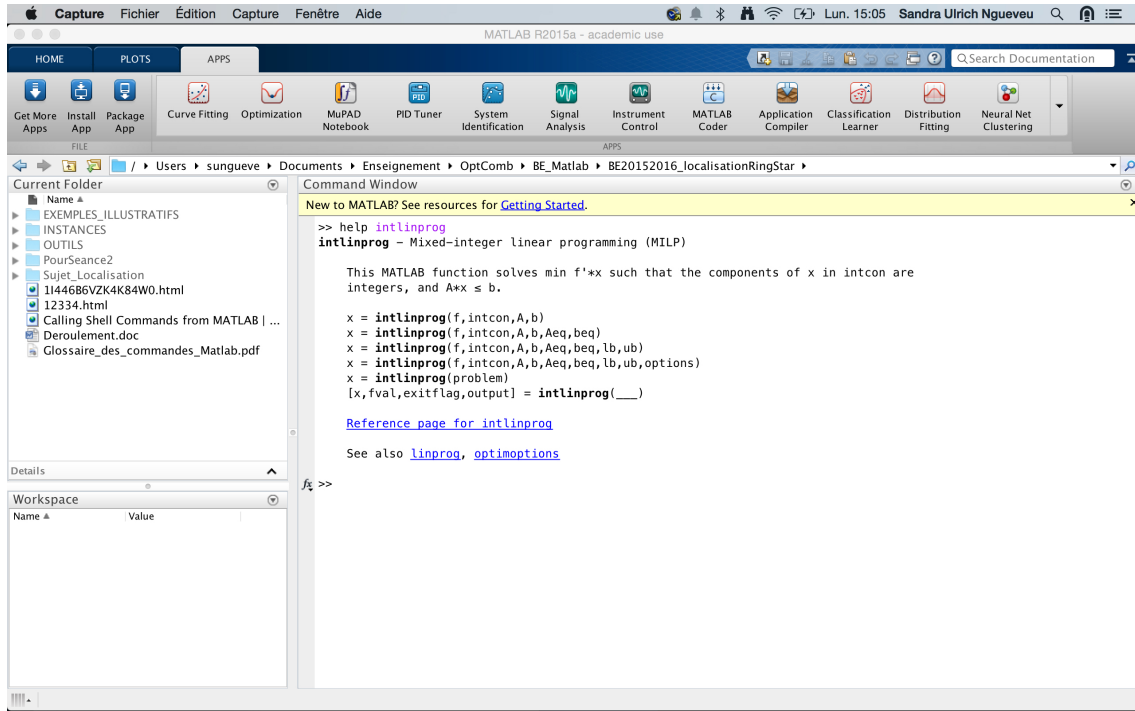


FIGURE 3 – IntlinProg : Interface graphique

- Heuristics : recherche de solutions faisables ; “none”, “rss” (par défaut), “round” ou “rins”
- HeuristicsMaxNodes : nbre max de noeuds explorés lors de la recherche de solution faisable (50 par défaut)
- IPPreprocess : type de prétraitements ; “none”, “basic” (par défaut) ou “advanced”
- LPMaxIter : Nombre max d’itérations de simplex par noeuds (3e4 par défaut)
- LPPreprocess : prétraitements lors de résolutions de relaxations linéaires ; “none” ou “basic” (par défaut)
- MaxNodes : nombre maximum de noeuds du branch-and-bound (1e7 par défaut)
- MaxNumFeasPoints : autre critère d’arrêt, le nbre max de sols faisables trouvées (Inf par défaut)
- MaxTime : temps limite en secondes (7200 par défaut)
- NodeSelection : stratégie d’exploration ; “simplebestproj” (par défaut), “minobj” ou “mininfeas”
- ObjectiveCutOff : borne sup (Inf par défaut)
- OutputFcn : spécifie une ou plusieurs fonctions personnalisées à appeler lors de certains événements. Ex : @savemilpsolutions stocke les sols entières dans xIntSol où chaque colonne correspond à une solution
- PlotFcns : trace différents indicateurs de progrès au cours l’exécution. Ex : @optimplotmilp trace l’évolution des bornes inf et sup
- RelObjThreshold : écart relatif min entre deux coûts de sols pour être considérés distincts (1e-4 par défaut)
- RootLPAlgorithm : algo de résolution des relax linéaires ; “dual-simplex” (par défaut) ou “primal-simplex”
- RootLPMaxIter : nbre max d’itérations de simplex appliquées à la relaxation linéaire du noeud racine (3e4 par défaut)
- TolCon : tolérance appliquée aux contraintes (1e-4 par défaut)
- TolFunLP : tolérance appliquée lors de l’identification des coûts réduits des variables candidates à l’entrée en base (1e-7 par défaut)
- TolGapAbs : tolérance absolue appliquée sur l’écart “borne sup - borne inf” (0 par défaut)
- TolGapRel : tolérance relative appliquée sur l’écart “borne sup - borne inf” (1e-4 par défaut)
- TolInteger : tolérance pour qu’un réel soit considéré entier (1e-5 par défaut)

## 3.2 GLPK

### 3.2.1 Solveur Executable GLPSOL

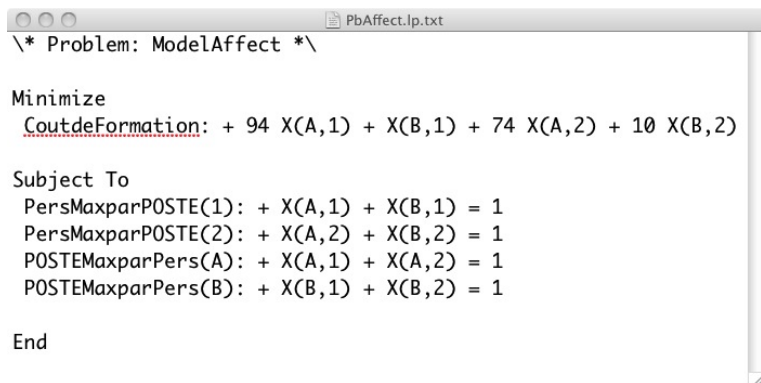
Le solveur glpsol de GLPK est un exécutable (avec sa .dll si besoin) permettant de lire et résoudre les problèmes formulés soit sous format “.lp”, soit sous format GMPL (dérivé de l’AMPL). Il se manipule en ligne de commande. Le format général des instructions est :

```
>> glpsol [options] [nom_des_fichiers]
```

La liste des instructions disponibles s’obtient avec la commande :

```
>> glpsol --help
```

#### Format “.lp”



```
\* Problem: ModelAffect *\n\nMinimize\n  CoutdeFormation: + 94 X(A,1) + X(B,1) + 74 X(A,2) + 10 X(B,2)\n\nSubject To\n  PersMaxparPOSTE(1): + X(A,1) + X(B,1) = 1\n  PersMaxparPOSTE(2): + X(A,2) + X(B,2) = 1\n  POSTEMaxparPers(A): + X(A,1) + X(A,2) = 1\n  POSTEMaxparPers(B): + X(B,1) + X(B,2) = 1\n\nEnd
```

FIGURE 4 – Exemple du fichier “PbAffect.lp.txt”

Lire et résoudre un problème PbAffect.lp.txt écrit sous format “.lp” :

```
>> glpsol --lp PbAffect.lp.txt
```

#### Format gmpl

Lire et résoudre un problème posé sous format GMPL :

— la section “modèle” et la section “données” (optionnelle) sont dans le même fichier :

```
>> glpsol --math ModelAffectwithData.lp.txt
```

```
>> glpsol -m ModelAffectwithData.lp.txt
```

— la section “données” est dans un autre fichier (ignorer la section “données” du fichier modèle, s’il y a en une) :

```
>> glpsol -m ModelAffect.lp.txt -d DataAffect.dat.txt
```

— générer le fichier “\*.lp” à partir du modèle GMPL

```
>> glpsol -m ModelAffect.lp.txt -d DataAffect.dat.txt --wlp PbAffect2.lp.txt
```

```

# GLPK model file created by SUN for OptCombAII

##### Model #####

##### Sets #####

set INTERIMAIRES;

set POSTES;

##### Variables #####

var X{i in INTERIMAIRES, j in POSTES}, >=0;
# or binary ?;

##### Constants: Data to load #####

param cout{i in INTERIMAIRES, j in POSTES}, integer;

##### Constraints #####

s.t. PersMaxparPOSTE{j in POSTES}:
    sum{i in INTERIMAIRES} X[i,j] = 1;

s.t. POSTEMaxparPers{i in INTERIMAIRES}:
    sum{j in POSTES} X[i,j] = 1;

##### Objective #####

minimize CoutdeFormation:
    sum{i in INTERIMAIRES, j in POSTES} X[i,j]*cout[i,j];

end;

```

```

# Data for exercise 4.4 from OptCombAII

data;

set INTERIMAIRES :=
A
B;

set POSTES :=
1
2;

param cout: 1 2 :=
A 94 74
B 1 10;

end;

```

FIGURE 5 – Exemple de fichiers “ModelAffect.mod” et “DataAffect.dat”

- écrire la solution dans un fichier (version lisible)
 

```
>> glpsol -m ModelAffect.lp.txt -d DataAffect.dat.txt -o SolAffect.sol.txt
```
- écrire la solution dans un fichier (version chargeable)
 

```
>> glpsol -m ModelAffect.lp.txt -d DataAffect.dat.txt -w SolAffect.sol.txt
```
- écrire dans un fichier les informations affichées pendant la résolution
 

```
>> glpsol -m ModelAffect.lp.txt -d DataAffect.dat.txt --display ResolAffect.txt
```
- vérifier (sans résoudre) que la syntaxe du problème est correcte
 

```
>> glpsol --check -m ModelAffect.lp.txt -d DataAffect.dat.txt
```

Pour les options supplémentaires, se référer à l’aide GLPK accessible avec la commande :

```
>> glpsol --help
```

D'autres exemples et illustrations sont disponibles à l'adresse :  
<http://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-135/Decouverte-du-solveur-GLPK>

### L'instruction "printf"

(voir fichier `gmpl.pdf`)

— afficher un texte à l'écran

```
>> printf 'Hello, world! \n';
```

— afficher un texte avec des variables à l'écran

```
>> printf : "x = %.3f; y = %.3f; z = %.3f \n", x, y, z;
```

```
>> printf{i in I} 'total flow from %s is %g \n', i, sum{j in J} x[i,j];
```

```
>> printf{k in K} "x[%s] = " & (if x[k] < 0 then " ?" else "%g"), k, x[k];
```

— redirection vers un fichier initialement créé vide

```
>> printf : "x = %.3f; y = %.3f; \n", x, y > "fichiervideavantredirection.txt"
```

— redirection à la fin d'un fichier

```
>> printf : "y = %.3f; z = %.3f \n", y, z >> "fichierpasvideavantredirection.txt"
```

```
>> printf{i in I, j in J} : "flow from %s to %s is %d \n", i, j, x[i,j] >> fiche & ".txt" ;
```

### 3.2.2 Bibliothèque GLPK

Il est possible (et même préférable) d'utiliser la bibliothèque de fichiers GLPK au sein d'un code C (ou C++, java, ...) plutôt que de faire appel à l'exécutable du solveur puisque cette librairie donne accès à des fonctions avancées telles que la modification des procédures de séparation et évaluation, calculs de bornes inférieures, ... ce qui permet de coder rapidement sa propre PSE.

## 3.3 Exercice de prise en main des outils

$$\min \text{ ou } \max x_1 + x_2 \quad (10)$$

s.c.

$$-2x_1 + 2x_2 \geq 1 \quad (11)$$

$$-8x_1 + 10x_2 \leq 13 \quad (12)$$

$$x_1, x_2 \geq 0 \text{ et entiers} \quad (13)$$

Utiliser MATLAB puis GLPK pour résoudre la relaxation linéaire puis le problème entier, à l'aide de l'interface graphique, puis par l'appel aux fonctions.