CentraleSupélec

RAPPORT TP D'OPTIMISATION

Réalisé par

SEKKAT Mohammed Amine

BELMAMOUN Yassine

L’ensemble du code est présent à l’adresse :

https://github.com/yassinepample/TP_optimisation

1/ Séance 1 : optimisation continue et optimisation approchée

L'executable de cette partie se trouve dans MAIN.m

1.1 Optimisation sans contraintes

1.1.1 Méthode du gradient

a) La fonction **GradResults** permet de minimiser une fonction par la méthode du gradient à pas fixe.

Nous la testons ici avec différentes valeurs de Rho. On obtient les résultats suivants :

- Rho = 0.01: Minimum = -1.8369

```
      initial_x: [5×1 double]
        minimum: [5×1 double]
      f_minimum: -1.8369
     iterations: 542
      converged: 1

Elapsed time is 0.070075 seconds.
gradient_rho_constant : minimum=-1.8369
```

- Pour Rho = 0.1, il n'y pas de convergence
- Rho = 0.5, il n'y pas de convergence

On s'apercoit donc que la convergence n'est pas toujours assurée en prenant une méthode du gradient à Rho constant.

b) L'algorithme se trouve dans le fichier gradient_rho_adaptatif.m

c) Avec 'coût' représentant dans le cas adaptatif le nombre d'appel de la fonction, et en partant d'un pas initial Rho0 de 0.01, on a les résultats suivants :

Algorithme Rho Constant = 0.01
Algorithme Rho Adaptatif : Rho0 = 0.01

```
gradient_rho_constant : minimum=-1.8369
gradient_rho_adaptatif : minimum=-1.8345
```

```
GradResults =

  struct with fields:

      initial_x: [5×1 double]
        minimum: [5×1 double]
      f_minimum: -1.8369
     iterations: 542
      converged: 1
```

```
GradResultsAdaptatif =

  struct with fields:

      initial_x: [5×1 double]
        minimum: [5×1 double]
      f_minimum: -1.8345
           cout: 118
      converged: 1
```

```
Elapsed time is 0.061785 seconds. Elapsed time is 0.039538 seconds.
```

Algorithme Rho Constant = 0.1
Algorithme Rho Adaptatif : Rho0 = 0.1

```
gradient_rho_constant : minimum=NaN
gradient_rho_adaptatif : minimum=-1.8356
```

```
GradResults =

  struct with fields:

      initial_x: [5×1 double]
        minimum: [5×1 double]
      f_minimum: NaN
     iterations: 10000
      converged: 0
```

```
GradResultsAdaptatif =

  struct with fields:

      initial_x: [5×1 double]
        minimum: [5×1 double]
      f_minimum: -1.8356
           cout: 117
      converged: 1
```

```
Elapsed time is 0.499307 seconds.
```

```
Elapsed time is 0.039119 seconds.
```

Algorithme Rho Constant = 0.5
Algorithme Rho Adaptatif : Rho0 = 0.5

```
gradient_rho_constant : minimum=NaN
gradient_rho_adaptatif : minimum=-1.8368
```

```
GradResults =

  struct with fields:

      initial_x: [5×1 double]
        minimum: [5×1 double]
      f_minimum: NaN
     iterations: 10000
      converged: 0
```

```
GradResultsAdaptatif =

  struct with fields:

      initial_x: [5×1 double]
        minimum: [5×1 double]
      f_minimum: -1.8368
           cout: 196
      converged: 1
```

```
Elapsed time is 0.483533 seconds.
```

```
Elapsed time is 0.045326 seconds.
```

On remarque alors que la convergence est toujours assurée dans le cas de rho adaptatif, avec un temps de calcul plus faible.

1.1.2 Utilisation des routines d'optimisation de la Toolbox Optimization. Méthode de Quasi-Newton (version BFGS).

En utilisant la méthode de quasi Newton en laissant Matlab calculer un gradient approché (option par défaut), on obtient les résultats suivants:

Méthode Rho constant =0.01	Méthode Rho Adaptatif Rho0=0.01	Quasi-Newton sans fournir de gradient	Quasi-Newton en fournissant le gradient
-1.8369	-1.8345	-1.8370	-1.8370

Methodes de Quasi-Newton	
Sans renseigner le gradient	En renseignant le gradient
iterations: 13 funcCount: 96 stepsize: 3.9877e-05 lssteplength: 1 firstorderopt: 4.2170e-06 algorithm: 'quasi-newton' message: 'Local minimum Elapsed time is 4.367695 seconds.	iterations: 13 funcCount: 16 stepsize: 3.9748e-05 lssteplength: 1 firstorderopt: 4.2848e-06 algorithm: 'quasi-newton' message: 'Local minimum found. Elapsed time is 0.202605 seconds.

Les méthodes de quasi-Newton présentent moins d'itérations et d'appels à la fonction, mais présentent néanmoins un temps de calcul plus important, étant donné le calcul de l'inverse d'une matrice dans le processus d'optimisation.

La fonction f1 est quadratique : il serait possible de calculer explicitement l'expression dans R5 et à partir de là se ramener à un problème de minimisation où on peut expliciter la solution par Fritz-Jones.

1.2 Optimisation sous contraintes

1.2.1 Optimisation à l'aide de routines Matlab

En utilisant l'algorithme SQP de Matlab, nous obtenons les résultats suivants:

F1

```
X =                  iterations: 16
                      funcCount: 104
    0.0000    constrviolation: 0
    0.1269           stepsize: 3.7224e-05
    0.0000          algorithm: 'interior-point'
    0.0194      firstorderopt: 7.2239e-07
    0.0000       cgiterations: 0

   Elapsed time is 4.833395 seconds.
```

Valeur_Min = -0.1385

F2

```
X =                  iterations: 20
                      funcCount: 127
    0.0000    constrviolation: 0
    0.1494           stepsize: 4.4515e-07
    0.0000          algorithm: 'interior-point'
    0.0163      firstorderopt: 8.4818e-07
    0.0000       cgiterations: 0
```

Elapsed time is 4.294305 seconds.

Valeur_Min = 2 x 10^-6

1.2.2 Optimisation sous contraintes et pénalisation

1) On prend une fonction C1 pour être sûr qu'elle soit semi continue inférieur.

$$\beta([x_1, x_2, x_3, x_4, x_5]) = \sum_{i=1}^{5} [\ (Max(0, -x_i))^2 \ + \ (Max(0,\ x_i - \ 1))^2 \ ]$$

Cette fonction de pénalisation est positive et telle que B(U)=0 si U appartient à Uad.

La fonction est implémentée dans le fichier Beta.m, et le script lançant l'optimisation de la fonction de pénalisation pour f1 et f2 se trouve dans MAIN2.m .

1.2.3 Méthodes duales pour l'optimisation sous contraintes

1) Le lagrangien de la fonction est défini dans le fichier lagrangien.m.

$$L(X,\beta) = f1(X,B,S) + \sum_{i=1}^{10} \beta_i g_i(X)$$

qui peut s'écrire sous forme matricielle par :

$$L(X,\beta) = f1(X,B,S) + (\beta)^T (C1 * X - C2)$$

2) L'algorithme d'Uzawa est implémenté dans le fichier MAIN2.m .

On trouve avec cette méthode

F1min =-0.1381

1.3 Optimisation non convexe - Recuit simulé

a) La fonction f4 est implémentée dans le fichier f4.m .

b) On utilise ici BFGC avec fminunc, en prenant différentes valeurs initiales.

- U0 = [0,0,0,0,0]

```
X =

   -0.2904
    0.5645
   -0.9885
    0.3437
    0.4298

FVAL =  -7.6563
```

- `U0 = [0.5;0.5;0.5;0.5;0.5];`

```
X =

    0.5567
    0.1942
   -0.2749
   -0.1579
   -0.0220

FVAL =  -1.3731
```

- `U0 = [0.5;0.5;1;1;1];`

```
X =

    0.4945
    0.5057
    1.0185
    1.0132
    1.0135

FVAL =  105.4410
```

c) Etant donné que plus T est faible, plus la probabilité p = exp(−(J(y)−J(xi))/T) de choisir un point "Moins bon" est faible, il convient de choisir un T petit. On fixe ici T = 50.
Par ailleurs, on diminue le nombre d'itérations à la même température afin de converger plus rapidement : On fixe L = 10.
Avec ces paramètres, nous verifions aisément que nous convergeons toujours vers le même point, à savoir le minimum global

X = -1.5041; 0.4583; -0.6201; 0.9569; -0.8089
FVAL = -10.7979

1.4 Application Synthèse d'un filtre à réponse impulsionnelle finie

1) En minimisant le critère J comme problème d'optimisation sans contrainte par la méthode de Quasi Newton, et en prenant comme H initial le vecteur 2*ones(30,1), on obtient :

Jmin = 0.7669

Ce résultat dépend de la valeur initial de H: en prenant Hinitial = ones(30,1), on trouve Jmin = 1.

2) Le problème peut être reformulé de la manière suivante :

$Min \quad Max|H_0(u) - H(u)|$ Pour $u \in [0, 0.1]U[0.15, 0.5]$

2/ Séances 2 et 3 : optimisation discrète et optimisation multi-objectif

2.1 Rangement d'objets (optimisation combinatoire)

On note n le nombre de boîtes qui est égal au nombre d'objets par hypothèse du problème.
On considère ici la matrice X(i, j)= 1 si l'objet j est dans la boîte i.

1. La condition pour que la boite i contienne un objet et un seul s'écrit:

$$\sum_{i=1}^{n} x_{i,j} = 1$$

Par ailleurs, la condition pour que l'objet j ne soit que dans une seule boite s'écrit:

$$\sum_{j=1}^{n} x_{i,j} = 1$$

2. Le problème s'écrit ici comme un problème linéaire en nombre entier qui s'écrit de manière à minimiser la fonction de déplacement total:

$$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j} \left|\left|O_j - B_i\right|\right|$$

$\left|\left|O_j - B_i\right|\right|$ constante définie du problème, notée $d_{i,j}$.

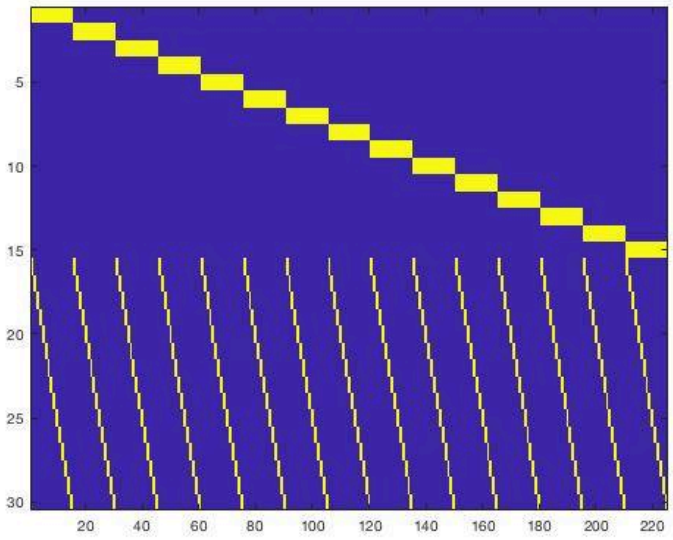
Dans un premier temps, il convient de transformer la matrice X en vecteur de taille n².
Celui ci s'écrit :

$$X = (\underbrace{x_{1,1} \quad \dots \quad x_{1,n}}_{n} \; \underbrace{x_{2,1} \quad \dots \quad x_{2,n}}_{n} \; \underbrace{x_{3,1} \quad \dots \quad x_{3,n}}_{n} \quad \dots . \quad \underbrace{x_{i,1} \quad \dots \quad x_{i,n}}_{n} \;)$$

Les contraintes s'écrivent de la forme A_eq.X = B_eq avec A_eq qui s'écrit de la manière suivante dans le cas 3x3

$$A = \overbrace{\begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} \end{pmatrix}}^{n^2} \Bigg\} 2n$$

La représentation à l'aide de la fonction « imagesc » de la matrice des contraintes d'égalité est de la forme suivante dans le cas n=15.



B_eq est le vecteur de taille n^2 et ne contenant que des 1.

En transformant la matrice ($\left|\left|O_j - B_i\right|\right|$) i,j en vecteur, le problème devient :

$$Min \sum_{i=1}^{n^2} di\ xi$$

sous contrainte A_eq.X = B_eq.

Le résultat que nous obtenons (fval) est le suivant :

15.3776

Le programme est implémenté dans le fichier **TP2/partie_1/question_2.m**

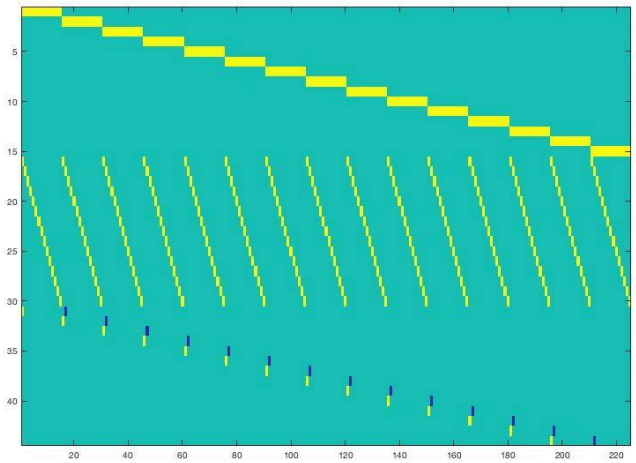
3- La condition pour que l'objet n°1 se situe dans la boîte située juste à gauche de la boîte contenant l'objet n°2.

$$\forall k \quad x_{k,1} = x_{k+1,2}$$

Ajouter cette contrainte revient à enrichir la matrice de contraintes afin de prendre en compte la condition que $\forall k \quad x_{k,1} - x_{k+1,2} = 0$

On ajoute donc à la matrice construite précédemment la matrice suivante dans le cas n=3 par concaténation des lignes :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$



Le programme est implémenté dans le fichier **TP2/partie_1/question_3.m**

On obtient : **Dmin =15.56**

4- Si $x_{i,3} = 1$ et $x_{i+k,4} = 0$ cela signifie que l'objet 3 se trouve dans une boite i et l'objet 4 ne se trouve pas dans une boite à droite de l'objet 3: l'objet 3 est donc à droite de l'objet 4.

Si $x_{i,3} = 0$ et $x_{i+k,4} = 1$ cela signifie que l'objet 3 ne se trouve pas dans une boite à gauche de l'objet 4: l'objet 3 est donc à droite de l'objet 4.

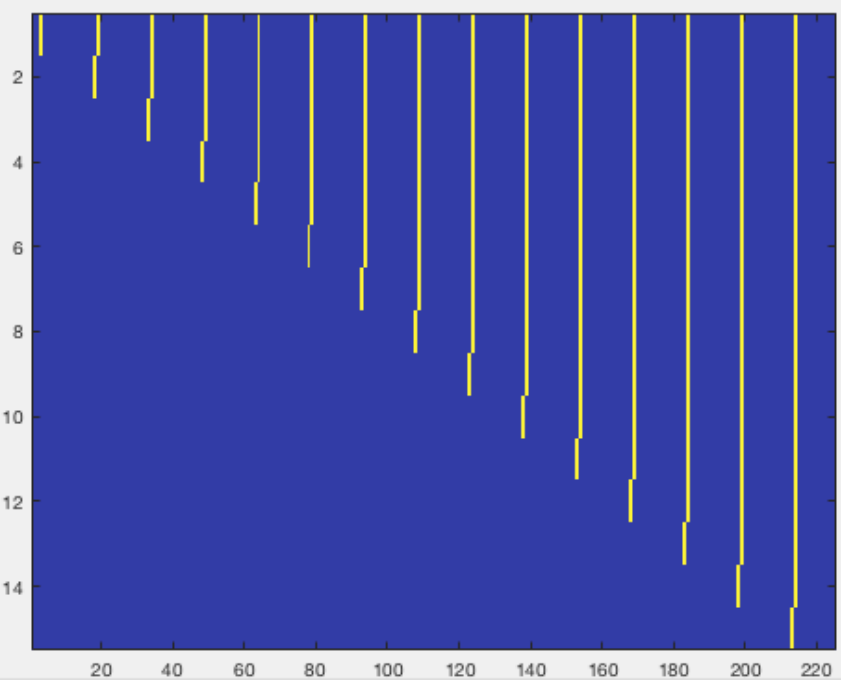
Si $x_{i,3} = 0$ et $x_{i+k,4} = 0$ Alors ni l'objet 3 ni l'objet 4 ne sont dans les boites i et (i+k): Si l'objet 3 n'est pas dans la boite i, alors l'objet 4 n'est pas a droite du 3.

Donc la condition traduisant la contrainte que la boîte contenant l'objet n°3 se situe à droite de la boîte contenant l'objet n°4 est bien xi,3 + xi+k,4 ≤ 1 ∀i, ∀k > 0.

Ainsi, la condition s'écrit :

$$x_{i,3} + \sum_{k=1}^{n} x_{i+k,4} \leq 1 \quad \forall i$$

La matrice de contrainte qui intègre cette contrainte est de la forme suivante :



Le programme est implémenté dans le fichier **TP2/partie_1/question_4.m**

On obtient : **Dmin = 15.901379**

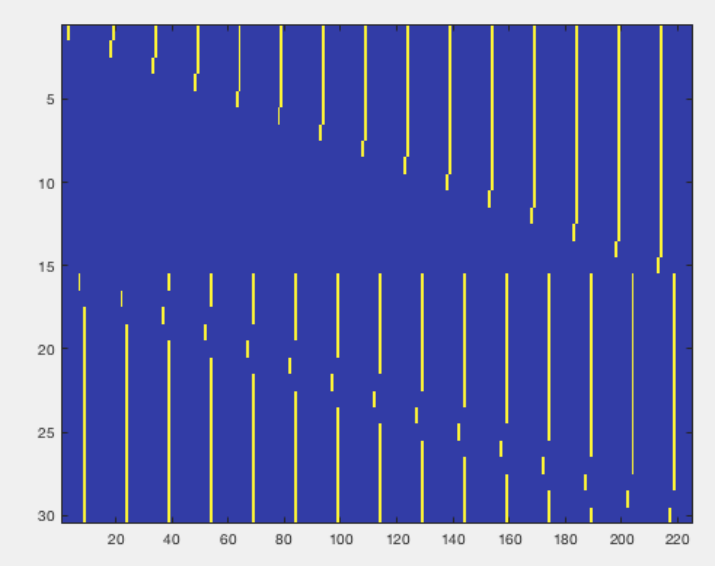
5- Modélisation pour traduire la contrainte que la boîte contenant l'objet n°7 se situe à côté de la boîte contenant l'objet n°9:

$$x_{k,7} = x_{k+1,9} \text{ ou } x_{k+1,7} = x_{k,9}$$

Celle ci s'écrit :

$$x_{i,7} + \sum_{|k|\geq 2}^{n} x_{i+k,9} \leq 1 \quad \forall i$$

La matrice de contrainte qui intègre cette contrainte est de la forme suivante :



Le programme est implémenté dans le fichier **TP2/partie_1/question_5.m**

On obtient : **Dmin = 15.9048**

2.2 Communication entre espions (optimisation combinatoire)

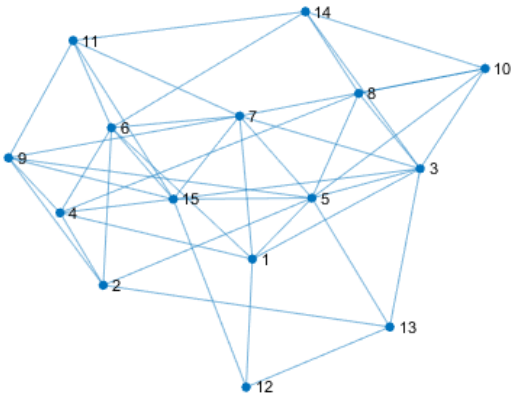
On modélise le lien entre les espions par x de taille n^2 tel que $x_{i,j} = 1$ si les espions i et j communiquent, et 0 sinon.

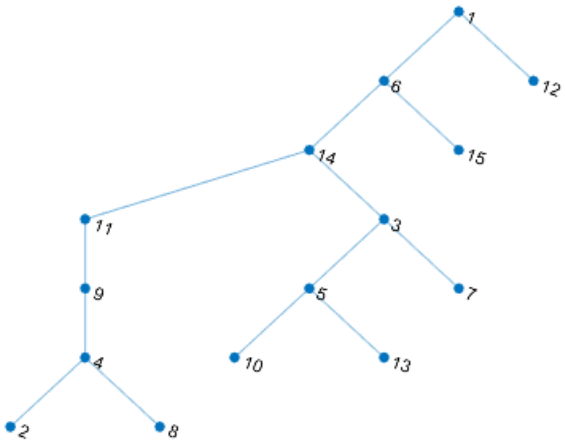
Notons que $x_{i,j} = x_{j,i}$.

La fonction a minimiser est donc $\prod_{i=1}^{n} p_{i,j}^{x_{i,j}}$.

Afin d'avoir un problème linéaire, nous passons au logarithme: la fonction à minimiser devient :

$$\sum_{i=1}^{n} x_{i,j} \mathrm{Log}(p_{i,j})$$





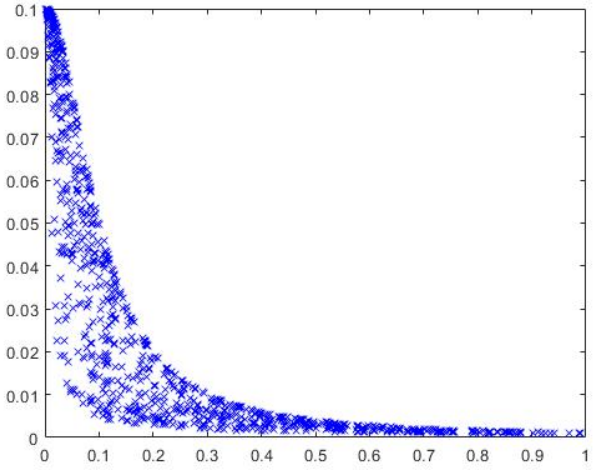
En raisonnant par événement contraire, on obtient comme valeur minimale :

$$\mathrm{Pmin} = 1 - \prod_{i=1}^{n} (1 - \exp(\mathrm{edge}_i)) = \mathbf{0.58092}$$

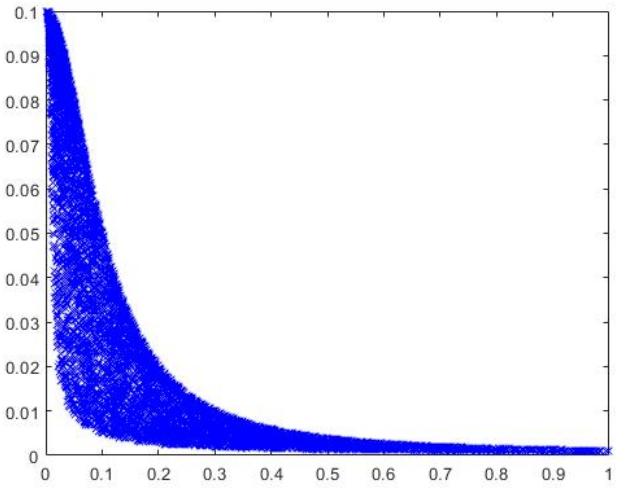
2.3 Dimensionnement d'une poutre (optimisation multiobjectif)

Il convient ici de générer N couples (a,b) de solutions réalisables, puis d'approximer le front de pareto par la courbe obtenue à partir des solutions de rang 1. Dès lors, une fois les solutions réalisables obtenues, on détermine le front de pareto par domination des solutions.

N=1000



N=10 000



2.4 Approvisionnement de changer (optimisation combinatoire)

Nous cherchons à minimiser les dépenses de l'entreprise en terme d'utilisation d'engins (location, acheminement, remise).

Le problème s'étale sur une durée de n=99 semaines.

Nous modélisons le problème de la manière suivante :

$$X = \begin{pmatrix} a & b & c \\ d & e & f \\ & \dots & \end{pmatrix}$$

Les lignes représentant les semaines et les 3 colonnes sont définies comme suit :

- La première colonne étant le nombre d'engins présent au début de la semaine i sur le chantier.
- La deuxième colonne étant le nombre d'engins acheminer au début de la semaine i.
- La troisième colonne étant le nombre d'engins remis à la société de location au début de la semaine i.

Nous transformations la matrice X en un vecteur de la façon suivante :

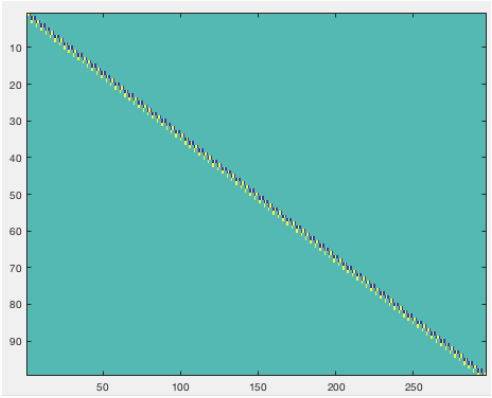
$$\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ \dots \end{pmatrix}$$

Le coût a payé par l'entreprise à l'agence de location est défini comme le produit suivant:

$$\underbrace{\left(\underbrace{200 \quad 800 \quad 1200}_{} \quad \dots \quad \underbrace{200 \quad 800 \quad 1200}_{}\right)}_{} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ \dots \end{pmatrix}$$

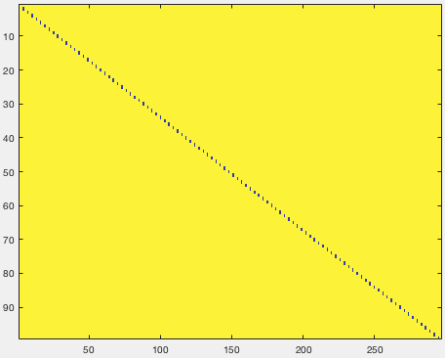
Il existe deux contraintes :

- Contrainte de conservation du nombre d'engins (Egalité) :



Cette contrainte d'égalité consiste à dire que le nombre d'engins présents à la semaines (n+1) est égale au nombre d'engins présents à la semaine n + le nombre d'engins apportés à la semaine (n+1) moins le nombre d'engins remis à la semaine (n+1).

- Contrainte de respect du nombre d'engins nécessaire par semaine (Inégalité) :



Cette contrainte consiste à vérifier que le nombre d'engins présents à la semaine n est supérieur ou égale.

Remarque importante : Il faut noter que le cout total de cet algorithme ne prend pas en compte les frais d'acheminement à la semaine 1 (36*800=28,800) et les frais de remise à dernière semaine (173*1200=207,600).

On obtient donc :

fval = 3,112,400

Le cout tôtal à payer par l'entreprise à l'agence de location des engins est donc de :

Coût = fval + 207,600 + 28,800 = 3,348,800

L'implementation du code se situe au niveau du fichier :
TP2/partie_3/approvisionnement_chantier.m

Voici le résultat qui permet de voir qu'il est plus intéressant d'un point de vue finanier de garder des engins sur le chantier quitte à ne pas les utiliser (temporairement) plutôt que de les rendre et les re-louer lorsqu'il y en a besoin à nouveau.

Les semaines 14 à 18 sont un bon exemple pour comprendre ce qui se passe.

Semaine	Engins présents	Engins nécessaires	Engins non-utilisés
1	36	36	0
2	61	61	0
3	84	84	0
4	87	87	0
5	121	121	0
6	121	88	33
7	121	66	55
8	121	69	52
9	121	71	50
10	121	78	43
11	121	116	5
12	149	149	0
13	149	130	19
14	**158**	**158**	**0**
15	**158**	**143**	**15**
16	**158**	**151**	**7**
17	**158**	**148**	**10**
18	**180**	**180**	**0**
19	170	170	0
20	165	165	0
21	158	158	0
22	131	130	1
23	131	96	35
24	131	82	49
25	131	80	51
26	131	81	50
27	131	117	14
28	131	113	18
29	131	124	7
30	131	131	0
31	131	110	21
32	131	103	28
33	131	131	0
34	126	126	0
35	126	121	5
36	126	101	25

37	126	118	8
38	126	95	31
39	126	85	41
40	126	82	44
41	126	91	35
42	126	67	59
43	126	75	51
44	126	118	8
45	145	145	0
46	145	138	7
47	150	150	0
48	150	147	3
49	150	120	30
50	150	102	48
51	150	146	4
52	150	130	20
53	166	166	0
54	166	160	6
55	167	167	0
56	164	153	11
57	164	160	4
58	164	145	19
59	164	164	0
60	137	137	0
61	134	110	24
62	134	133	1
63	134	107	27
64	134	75	59
65	134	95	39
66	134	132	2
67	134	116	18
68	134	117	17
69	134	134	0
70	134	129	5
71	134	134	0
72	134	108	26
73	141	141	0
74	155	155	0
75	151	151	0
76	125	111	14
77	125	125	0
78	112	106	6

79	112	80	32
80	112	91	21
81	112	92	20
82	112	90	22
83	112	95	17
84	112	102	10
85	112	87	25
86	112	106	6
87	112	100	12
88	112	112	0
89	149	149	0
90	171	171	0
91	171	157	14
92	173	173	0
93	173	137	36
94	173	134	39
95	173	141	32
96	173	165	8
97	173	149	24
98	173	173	0
99	173	129	44