Résolution d'un problème de programmation quadratique avec une M-matrice

Katia HASSAINI et Mohand Ouamer BIBI

Laboratoire de Modélisation et d'Optimisation des Systèmes (LAMOS)
Université de Béjaia 06000, Algérie,
Hassaini@Gmail.com mohandbibi@yahoo.fr

Résumé Dans ce papier, une méthode de résolution d'un problème de programmation quadratique avec une M-matrice et des contraintes simples est présentée. Elle se base sur les algorithmes de R. Chandrasekaran [1] et de F.T. Luk et M. Pagano [2]. Ces méthodes utilisent le fait qu'une M-matrice possède une inverse non négative qui permet d'avoir une suite monotone de solutions réalisables [3,4]. En introduisant le concept de support pour une fonction objectif [16], notre approche se différencie par une condition plus générale qui permet d'avoir une solution réalisable initiale plus proche de la solution optimale. La programmation sous MATLAB de notre méthode et de celle de Luk et Pagano nous a permis de faire une comparaison entre ces dernières, de les illustrer par deux exemples pratiques, et ce, en variant le nombre de variables.

Mots clés : Programmation quadratique convexe, M-matrices, méthode de Chandrasekaran , méthode de Newton projetée, méthode de support.

1 Introduction

Dans la littérature, plusieurs approches ont été proposées pour la résolution des problèmes de programmation quadratique quand la matrice D associée est définie positive. Cependant, il est possible d'exploiter les propriétés spéciales d'une M-matrice pour obtenir des algorithmes spéciaux plus efficaces. D'ailleurs, de tels problèmes avec des M-matrices trouvent des applications dans la résolution numérique des équations aux dérivées partielles elliptiques. Ces problèmes incluent divers types de problèmes de Dirichlet avec obstacles [11,12], les modèles d'application de torsion à une barre [19], etc. Les M-matrices sont aussi connues pour avoir de nombreuses applications dans la modélisation des systèmes dynamiques, dans les sciences économiques et écologiques [13,14,15]. Plusieurs de leurs propriétés sont utilisées pour établir des résultats de stabilité pour les systèmes dynamiques en général [8,9].

Dans cet article, on propose une nouvelle approche qui se distingue de celles de R. Chandrasekaran, F.T. Luk et M. Pagano, par une condition plus générale qui permet d'avoir une solution réalisable initiale plus proche de la solution optimale. La programmation sous MATLAB de notre méthode et de celle de Luk et Pagano nous a permis de faire une comparaison entre ces dernières, de les illustrer par deux exemples pratiques, et ce, en variant le nombre de variables.

2 Position du problème et définitions

Considérons le problème suivant de programmation quadratique avec contraintes simples :

$$\begin{cases} F(x) = \frac{1}{2}x^T Dx + c^T x \longrightarrow \min, \\ \\ x \geq 0, \end{cases} \tag{1}$$

où $c = c(J) = (c_j, \quad j \in J)$ et $x = x(J) = (x_j, \quad j \in J)$ sont des n-vecteurs réels, $J = \{1, 2, \cdots, n\}$. La matrice $D = D(J, J)$ est une M-matrice carrée symétrique $(D = D^T)$ d'ordre n. Le symbole (T) est celui de la transposition.

Rappelons qu'une M-matrice $D = (d_{ij}, \quad 1 \leq i, j \leq n)$ satisfait les conditions suivantes :

$$d_{ii} > 0, \qquad d_{ij} \leq 0, \quad i \neq j, \qquad D^{-1} \geq 0,$$

où le symbole $D^{-1} \geq 0$ veut dire que tous les éléments de la matrice D^{-1} sont positifs ou nuls.

Remarque 1. Une M-matrice symétrique est toujours définie positive $(x^T Dx > 0, \quad \forall \ x \neq 0)$.

Définition 1.
Un vecteur $x \geq 0$ est appelé *solution réalisable* du problème (1). Une solution réalisable x^0 est dite *optimale* si elle réalise le minimum de la fonction objectif du problème (1).
Ainsi, une solution réalisable x^0 du problème (1) est optimale si et seulement si pour tout $j \in J$, les conditions d'optimalité suivantes sont satisfaites :

$$\begin{cases} x_j^0 = 0 \quad \Rightarrow \quad g_j(x^0) \geq 0, \\ \\ x_j^0 > 0 \quad \Rightarrow \quad g_j(x^0) = 0, \quad j \in J, \end{cases} \tag{2}$$

où $g(x) = g(J) = Dx + c$ est le gradient de la fonction objectif F au point x.

Considérons le problème de programmation quadratique sans contraintes :

$$\begin{cases} F(x) = \frac{1}{2}x^T Dx + c^T x \longrightarrow \min, \\ \\ x \in \mathbb{R}^n. \end{cases} \tag{3}$$

La solution optimale $\widehat{x}$ du problème (3) vérifie

$$D\widehat{x} + c = 0 \iff \widehat{x} = -D^{-1}c.$$

Soient J_S et J_N une partition de J :

$$J_S \cup J_N = J, \quad J_S \cap J_N = \emptyset.$$

Le gradient de la fonction F au point x peut alors s'écrire sous la forme suivante :

$$g = \begin{pmatrix} g_S \\ g_N \end{pmatrix}, \quad g_S = g(J_S) = D_S x_S + D_{SN} x_N + c_S, \quad g_N = g(J_N) = D_{NS} x_S + D_N x_N + c_N,$$

où

$$x = \begin{pmatrix} x_S \\ x_N \end{pmatrix}, \; c = \begin{pmatrix} c_S \\ c_N \end{pmatrix}, \; D_S = D(J_S, J_S), \; D_N = D(J_N, J_N), \; D_{SN} = D(J_S, J_N).$$

Définition 2.

- Le sous-ensemble J_S est appelé *support* de la fonction objectif, car il vérifie toujours la condition :

$$\det D_S = \det D(J_S, J_S) \neq 0.$$

  La paire $J_p = \{J_S, J_N\}$ est alors dite support du problème (1).
- Le couple $\{x, J_p\}$, formé d'une solution réalisable x et du support J_p est appelé *solution réalisable de support*.

Définition 3.

- Un vecteur $\kappa = \kappa(J) = (\kappa(J_S), \kappa(J_N))$ vérifiant

$$\begin{cases} \kappa_N = 0, \\ \\ \kappa_S = -D_S^{-1} c_S \; . \end{cases}$$

  est dit *pseudo-solution* du problème (1).
  Une pseudo-solution vérifie toujours $g_S(\kappa) = 0$.
- Le support $J_p = \{J_S, J_N\}$ est appelé support coordinateur s'il existe une pseudo-solution κ telle que :

$$g_j(\kappa) \geq 0, \qquad j \in J_N. \tag{4}$$

  On dit dans ce cas que la pseudo-solution κ est associée au support coordinateur J_P.

Théorème 1. *Une pseudo-solution κ associée à un support coordinateur J_p est optimale dans le problème (1) si et seulement si*

$$\kappa_j \geq 0, \qquad j \in J_S. \tag{5}$$

Remarque 2. Toute pseudo-solution κ, associée à un support coordinateur J_p, est solution réalisable du dual du problème primal (1) :

$$\begin{cases} F(\kappa) = -\frac{1}{2} \kappa^T D \kappa \longrightarrow \max, \\ \\ \qquad D\kappa + c \geq 0. \end{cases} \tag{6}$$

Remarque 3. Comme $g(\widehat{x}) = 0$, alors la solution optimale $\widehat{x}$ du problème sans contraintes (3) est une pseudo-solution du problème (1), associée à un support coordinateur $J_p = \{J_S, J_N\}$, où $J_S = J$ et $J_N = \emptyset$. D'aprés le théorème 1, si $\widehat{x} \geq 0$, alors $x^0 = \widehat{x}$ est une solution optimale du problème (1).

Rappelons le lemme suivant :

Lemme 1. *[2].*

(a) *Si* $c \geq 0$, *alors* $x^0 = 0$ *résoud le problème (1),*

(b) *Si* $c \leq 0$, *alors* $x^{\,0} = -D^{-1}c$ *résoud le problème (1).*

Pour éviter les deux cas triviaux précédents, considérons le cas général où c contient des composantes positives et négatives. Construisons alors les deux ensembles suivants tels que :

$$J_S = \{j \in J : \widehat{x}_j \geq 0\}, \quad J_N = \{j \in J : \widehat{x}_j < 0\}, \qquad J_S \cup J_N = J.$$

– Si $J_S = J$, alors $x^0 = \widehat{x} = -D^{-1}c$ est une solution optimale du problème (1).

– Sinon, soit y la projection de $\widehat{x}$ sur le domaine admissible du problème (1), où $y = (y_j, \ j \in J)$, $y_j = \max\{0, \widehat{x}_j\}$. Donc on aura

$$\begin{cases} y_N = 0 > \widehat{x}_N \ . \\ \\ y_S = \widehat{x}_S, \end{cases}$$

Lemme 2. *L'inégalité suivante est vérifiée*

$$g_S(y) \leq g_S(\widehat{x}).$$

Preuve 1. *On a*

$$\begin{aligned} g_S(\widehat{x}) &= D_S \ \widehat{x}_S + D(J_S, J_N) \ \widehat{x}_N \ + \ c_S \\ &= D_S \ y_S + \ c_S + \ D(J_S, J_N) \ \widehat{x}_N \\ &= g_S(y) + \ D(J_S, J_N) \ \widehat{x}_N \\ &\geq \ g_S(y), \end{aligned}$$

car $D(J_S, J_N) \leq 0$ *et* $\widehat{x}_N < 0$. □

Puisque $g_S(\widehat{x}) = 0$, alors du lemme 2 on déduit que $g_S(y) \leq 0$. On construit alors un vecteur x tel que

$$\begin{cases} x_N = \ y_N = \ 0 \ , \\ \\ x_S = -D_S^{-1} \ c_S \ . \end{cases} \tag{7}$$

On a donc

$$g_S(y) \ \leq \ 0 \quad et \qquad g_S(x) = 0.$$

Lemme 3. *Les vecteurs* y *et* x *vérifient l'inégalité suivante :*

$$x_S \ \geq \ y_S \ \geq \ 0.$$

Preuve 2. *On a*

$$D_S \ (x_S - y_S) = D_S \ x_S + c_S - [D_S \ y_S + c_S].$$

Comme $x_N \ = \ y_N \ = \ 0$, $g_S(y) \ \leq \ 0$ *et* $g_S(x) = 0$, *alors on déduit*

$$D_S \ (x_S - y_S) = g_S(x) - g_S(y) \geq 0.$$

La sous-matrice D_S *étant une M-matrice [2], il en résulte que* $D_S^{-1} \geq 0$. *Par conséquent, en multipliant à gauche par* D_S^{-1} *l'inégalité ci-dessus, on obtient*

$$x_S \ \ \geq \ \ y_S \ \geq \ \ 0. \ \square$$

En vertu du lemme 3, le vecteur x construit (7) est une solution réalisable du problème (1), et d'après [4], il vérifie encore l'inégalité $x \ \leq \ x^0$, où x^0 est la solution optimale du problème (1). On a alors le lemme suivant :

Lemme 4.

$$F(x) \ \leq \ F(y).$$

Preuve 3. *Soit*

$$2F(x) = x_S^T \ D_S x_S + 2 \ c_S^T \ x_S.$$

Comme $x_S = -D_S^{-1} c_S$, *on aura*

$$2F(x) = c_S^T \ D_S^{-1} c_S - 2 \ c_S^T \ D_S^{-1} c_S = -c_S^T \ D_S^{-1} c_S.$$

La sous-matrice D_S *étant définie positive, alors on peut écrire*

$$\begin{aligned} 2F(x) \leq \ & (y_S - x_S)^T \ D_S (y_S - x_S) - c_S^T \ D_S^{-1} c_S \\ \leq \ & y_S^T \ D_S \ y_S + x_S^T \ D_S \ x_S - 2 y_S^T \ D_S \ x_S - c_S^T \ D_S^{-1} \ c_S \\ \leq \ & y_S^T \ D_S \ y_S + c_S^T \ D_S^{-1} \ D_S \ D_S^{-1} \ c_S + 2 \ y_S^T \ D_S \ D_S^{-1} \ c_S - c_S^T \ D_S^{-1} \ c_S \\ \leq \ & y_S^T \ D_S \ y_S + 2 \ c_S^T \ y_S \\ \leq \ & 2 \ F(y) \end{aligned}$$

D'où

$$F(x) \ \leq \ F(y). \square$$

Remarque 4. Le vecteur x construit (7) vérifie ainsi l'inégalité suivante :

$$F(\widehat{x}) < F(x^0) \leq F(x) \leq F(y), \tag{8}$$

où y est la projection de $\widehat{x}$ sur le domaine admissible du problème (1).

Théorème 2. *[2]. Supposons que $D_S^{-1}c_S \leq 0$ pour un certain sous-ensemble non vide $J_S \subset J$. On définit un vecteur x avec $x_S = -D_S^{-1}c_S$ et $x_N = 0$. Soient J_N^- et J_N^+ deux ensembles qui partitionnent J_N tels que*

$$J_N^- = \{j \in J_N : \ g_j(x) < 0\},$$
$$J_N^+ = \{j \in J_N : \ g_j(x) \geq 0\}.$$

Si l'ensemble J_N^- est vide, alors

1. *x résoud le problème (1), sinon*

2. *Soit $\overline{J_S} := J_S \cup J_N^-$. Construisons $\overline{x}$ avec $\overline{x}(\overline{J_S}) = -D^{-1}(\overline{J_S}, \overline{J_S})\, c(\overline{J_S})$ et $\overline{x}(J_N^+) = 0$. On obtient*

(a) $\overline{x}(J_S) \geq x(J_S) \geq 0, \quad \overline{x}(J_N^-) \geq 0, \quad \overline{x}(J_N^+) = 0,$

(b) $g_j(\overline{x}) \leq g_j(x) \quad j \in J_N^+,$

(c) $F(\overline{x}) < F(x)$.

3 Algorithme de la méthode

Calculer $\widehat{x}$, solution optimale du problème (3) :

$$g(\widehat{x}) = D\widehat{x} + c = 0 \Longrightarrow \widehat{x} = -D^{-1}c.$$

– Si $\widehat{x} \geq 0$, alors terminer et le vecteur $x^0 = \widehat{x}$ est une solution optimale du problème (1).

– Sinon, construire les ensembles :

$$J_S = \{j \in J : \ \widehat{x}_j \geq 0\}, \quad J_N = \{j \in J : \ \widehat{x}_j < 0\}.$$

– Construire x de la manière suivante :

$$x_N = 0, \quad x_S = -D_S^{-1}c_S.$$

Soient J_N^- et J_N^+ deux ensembles qui partitionnent J_N tels que

$$J_N^- = \{j \in J_N : \ g_j(x) < 0\}, \quad J_N^+ = \{j \in J_N : \ g_j(x) \geq 0\}.$$

Répéter jusqu'à ce que l'ensemble J_N^- soit vide

(1) Calculer $g_N(x) = D(J_N, J_S)\, x_S + c_N$,

(2) Poser $J_N^- = \{j \in J_N : \ g_j(x) < 0\}$,

(3) Si l'ensemble J_N^- est non vide, alors

(a) Poser $J_S := J_S \cup J_N^-$ et $J_N := J_N \setminus J_N^-$,

(b) Reconstruire x tel que $x_N = 0$ *et* $x_S = -D_S^{-1} c_S$.

4 Exemples numériques et comparaisons

Nous avons choisi deux problèmes représentatifs. Le but est d'examiner l'efficacité de notre algorithme et de faire une comparaison avec l'algorithme de F.T. Luk et M. Pagano [2]. Les deux méthodes ont été programmées sous le langage MATLAB 7.0 R14, sous le système d'exploitation Windows XP, avec une RAM de 512 Mo, CPU 5.00GHz. On définit par

- **Algorithme1** : Méthode de Chandrasekaran, de F.T. Luk et M. Pagano [2].
- **Algorithme2** : Notre algorithme.
- NJ_S : Le cardinal de l'ensemble J_S, juste après avoir calculé $\widehat{x} = -D^{-1}c$.
- $\overline{NJ_S}$: Le cardinal de l'ensemble J_S, à l'optimum.
- NP : Le cardinal de l'ensemble P, à l'initialisation de x.
- $\overline{NP}$: Le cardinal de l'ensemble P, à l'optimum.
- **Itération** : Le nombre d'itérations effectuées par chaque algorithme.
- **Temps** : Le temps machine (CPU times) d'exécution en secondes.

4.1 Exemple 1.

Considérons le programme quadratique (1) :

$$\begin{cases} F(x) = \frac{1}{2} x^T D x + c^T x \longrightarrow \min, \\ \\ \qquad\qquad\qquad\qquad\qquad x \geq 0. \end{cases}$$

La matrice D choisie est telle que

$$D = \begin{pmatrix} 2 & -1 & & & \bigcirc \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ \bigcirc & & & -1 & 2 \end{pmatrix}_{n \times n} . \tag{9}$$

On choisit de générer le vecteur c de façon à avoir trois cas de l'ensemble J_S. On pose r_i un nombre aléatoire qui suit une loi uniforme $r_i \in U\,[0,1]$. Les temps

machine d'exécution de chaque programme pour la résolution d'un programme quadratique sous la forme (1) sont représentés dans les tableaux ci-dessous :

Cas 1. On génère le vecteur c de manière à avoir $NJ_S = n$:

$$c_i = 11 - 23\ r_i \qquad pour \quad i = 1, 2, \ldots, n. \tag{10}$$

Dimension	Algorithme1				Algorithme2			
n	Itération	NP	$\overline{NP}$	Temps	Itération	NJ_S	$\overline{NJ_S}$	Temps
500	09	259	500	0.7969	0	500	500	0.1250
1000	11	539	1000	4.3125	0	1000	1000	0.1875
1500	13	783	1500	14.2188	0	1500	1500	0.4219
2000	15	1028	2000	34.7500	0	2000	2000	1.3125

Tab. 1. CPU Times (en secondes) avec $NJ_S = n$.

Cas 2. Le vecteur c est généré par

$$c_i = 11 - 22\ r_i \qquad pour \quad i = 1, 2, \ldots, n. \tag{11}$$

Dimension	Algorithme1				Algorithme2			
n	Itération	NP	$\overline{NP}$	Temps	Itération	NJ_S	$\overline{NJ_S}$	temps
500	16	262	498	1.1875	10	373	498	0.5000
1000	24	508	995	14.1250	08	922	995	1.4036
1500	23	752	1487	23.8125	21	727	1487	6.1719
2000	26	975	1921	70.0156	28	970	1921	13.7031

Tab. 2. CPU Times (en secondes) avec $NJ_S < n$.

Cas 3. On génère le vecteur c par

$$c_i = 11 - 20\ r_i \qquad pour \quad i = 1, 2, \ldots, n. \tag{12}$$

Dimension	Algorithme1				Algorithme2			
n	Itération	NP	$\overline{NP}$	Temps	Itération	NJ_S	$\overline{NJ_S}$	temps
500	06	219	360	0.3438	08	0	360	0.2969
1000	09	467	764	2.5469	10	0	764	1.0156
1500	12	660	1118	6.2500	13	0	1118	2.6094
2000	13	930	1591	16.4063	14	0	1591	5.1875

Tab. 3. CPU Times (en secondes) avec $NJ_S = 0$.

Dans cet exemple, on remarque bien que notre approche est plus efficace en temps machine que l'approche de Chandrasekaran, de F.T. Luk et M. Pagano. Et cela, quelque soit le cardinal de J_S à l'étape initiale de l'algorithme.

4.2 Exemple 2.

Dans cet exemple, on choisit la matrice D comme étant une matrice déduite de l'approximation du Laplacien par des différences finies en 5-points :

$$D = \begin{pmatrix} B & -1 & & \bigcirc \\ -1 & B & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & B & -1 \\ \bigcirc & & & -1 & B \end{pmatrix}_{m^2 \times m^2}, \tag{13}$$

où

$$B = \begin{pmatrix} 4 & -1 & & \bigcirc \\ -1 & 4 & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & 4 & -1 \\ \bigcirc & & & -1 & 4 \end{pmatrix}_{m \times m}. \tag{14}$$

Soit $n = m^2$. On considère la matrice de "mise à jour partielle" E telle que :

$$E = \begin{pmatrix} B & & & \bigcirc \\ & B & & \\ & \ddots & \ddots & \ddots \\ & & & B & \\ \bigcirc & & & & B \end{pmatrix}_{n \times n}. \tag{15}$$

La variable r_i étant toujours un nombre aléatoire qui suit une loi uniforme $r_i \in U\,[0,1]$, on considère les 3 cas suivants :

Cas 1. Le vecteur c est généré par

$$c_i = 8 - 20\ r_i \qquad pour \quad i = 1, 2, \ldots, n. \tag{16}$$

Dimension	Algorithme1				Algorithme2			
n	Itération	NP	$\overline{NP}$	Temps	Itération	NJ_S	$\overline{NJ_S}$	Temps
500	03	299	500	0.3438	0	500	500	0.1563
1000	04	594	1000	1.9844	0	1000	1000	0.7188
1500	04	921	1500	4.5000	0	1500	1500	2.2188
2000	04	1210	2000	9.8438	0	2000	2000	4.3281

Tab. 4. CPU Times (en secondes) avec $NJ_S = n$.

Cas 2. Le vecteur c est généré par

$$c_i = 8 - 16\ r_i \qquad pour \quad i = 1, 2, \ldots, n. \tag{17}$$

Dimension	Algorithme1				Algorithme2			
n	Itération	NP	$\overline{NP}$	Temps	Itération	NJ_S	$\overline{NJ_S}$	Temps
500	10	259	472	0.7656	13	2	472	1.1094
1000	14	495	948	4.7188	16	388	948	7.7813
1500	16	729	1442	15.7969	18	153	1442	26.7656

Tab. 5. CPU Times (en secondes) avec $NJ_S < n$.

Cas 3. Le vecteur c est généré par

$$c_i = 8 - 10\ r_i \qquad pour \quad i = 1, 2, \ldots, n. \tag{18}$$

Dimension	Algorithme1				Algorithme2			
n	Itération	NP	$\overline{NP}$	Temps	Itération	NJ_S	$\overline{NJ_S}$	Temps
500	01	91	100	0.0938	4	0	100	0.2188
1000	03	219	246	0.2500	5	0	246	0.8125
1500	02	259	337	0.2969	05	0	337	2.2344
2000	02	392	444	0.9531	05	0	444	4.5781

Tab. 6. CPU Times (en secondes) avec $NJ_S = 0$.

5 Conclusion

Les lemmes 2, 3 et 4 nous ont permis de démarrer l'algorithme avec une solution réalisable x vérifiant les conditions du théorème 2 et l'inégalité (8). Dans le cas où la matrice de notre fonction objectif $D = I$, où I est l'identité, on aura

$$J_S = \{j \in J : c_j \leq 0\}, \quad J_N = \{j \in J : c_j > 0\},$$

et on retrouve les conditions d'initialisation des algorithmes de R. Chandrasekaran, de F.T. Luk et M. Pagano [1,2]. Remarquons encore que les algorithmes cités terminent avec J_N ou $J_{\overline{N}}$ vide, alors que le nôtre termine toujours avec $J_{\overline{N}} = \emptyset$ et $J_N \neq \emptyset$, et ce, à cause de notre initialisation. En effet, le cas $J_N = \emptyset$ correspond à la solution optimale $x^0 = \widehat{x}$.

Dans l'exemple 1, on remarque bien que notre approche est plus efficace en temps machine que l'approche de Chandrasekaran, de F.T. Luk et M. Pagano. Et cela, quelque soit le cardinal de J_S à l'étape initiale de l'algorithme.

Dans le deuxième exemple, on remarque dans le tableau (4) que notre approche est meilleure que l'autre et cela résulte du fait que la solution réalisable initiale est elle-même solution optimale du problème. Par contre, on voit bien que dans les tableaux (5) et (6), l'algorithme 1 se comporte mieux que le nôtre.

Références

1. Chandrasekaran, R. : A special case of the complementary pivot problem. Opsearch, **7** (1970), 263–268.
2. Luk, F. T. and Pagano, M. : Quadratic programming with M-Matrices. Linear Algebra And Its Applications, **33** (1980), 15–40.
3. Stachurski, A. : Monotone sequences of feasible solutions for quadratic programming problems with M-matrices and box constraints. In System Modeling and Optimization. Book series : Lecture Notes in Control and Information Sciences, (1986), Vol 84, 896–902.
4. Stachurski, A. : An Equivalence between two algorithms for a class of quadratic programming problems with M-matrices. Optimization, **21(6)** (1990), 871–878.
5. Li, L. and Kobayashi, Y. : A block recursive algorithm for the linear complementarity problem with an M-matrix. International Journal of Innovative, Computing, Information and Control, Vol 2, Number 6, (2006), 1327–1335.
6. Kunish, K. and Rendl, F. : An infeasible active set method for convex problems with simple bounds. SIAM Journal on optimization, **14(1)**(2003), 35–52.
7. Voglis, C. and Lagaris, I. E. : BOXCQP : An Algorithm for Bound Constrained Convex Quadratic Problems. 1st International Conference "From Scientific Computing To Computational Engineering", Vol 1, (2004), 261–268, Athens, Greece, September 8-10.
8. Stipanovic, D. M and Šiljak, D. D. : Stability of polytopic systems via convex M-matrices and parameter-dependent Liapunov functions. Nonlinear Analysis, **40** (2000), 589–609.
9. Stipanovic, D. M., Shankaran, S. and Tomlin, C.J. : Multi-agent avoidance control using an M-matrix property. Electronic Journal of Linear Algebra. A publication of the International Linear Algebra Society, **12** (2005), 64–72.
10. Pang, J. S. : On a class of least-element complementarity problems.(1976), Report SOL 76-10, Systems Optimization Lab, Stanford Univ.
11. Levati, G., Scarpini, F. and Volpi, G. : Sul trattemento numerico di alcuni problemi variazionali di tipo unilaterale. L.A.N. Pub.82, Univ. of Pavia (1974).
12. Scarpini, F. : Some algorithms solving the unilateral Dirichlet problems with two constraints. Calcolo, **12**(1975), 113–149.
13. Šiljak, D. D. : Large-scale Dynamic Systems : Stability and Structure. North-Holland, New York, (1978).
14. Varga, R. S. : Matrix Iterative Analysis. Prentice-Hall, Englewood Cliffs, N.J. (1962).
15. Berman, A. and Plemmons, R. J. : Nonnegative Matrices in Mathematical Sciences. Academic Press, New York, (1979).
16. Gabasov, R., Kirillova, F. M., Kostyukova, O. I. and Raketsky, V.M. : Constructive methods of optimization, volume 4 : Convex Problems. University Press, Minsk (1987).
17. Brahmi, B. and Bibi, M. O. : Méthode duale de support pour la résolution des problèmes quadratiques convexes à variables bornées. In Actes du Colloque sur l'Optimisation et les Systèmes d'Information, 11-13 juin 2007, USTO, Oran, pp.365-376.
18. Bibi, M. O. and Radjef, S. : Solution of a convex quadratic program with mixed variables via the direct support method. In Actes du Colloque MOAD : Méthodes et Outils d'Aide à la Décision, 18-20 Novembre 2007, Université de Béjaia, p.39-44.

19. Céa, J. and Glowinski, R. : Sur des méthodes d'optimisation par relaxation. RAIRO, **R-3** (1973) 5–32.

20. Bertsekas, D. P. : Projection Newton Method for Optimization Problems with Simple Constraints. SIAM Journal on Control and Optimization, **20**(1982)2, 221–246.