Quadratic programming method with an M-matrix

Katia Hassaini*, Mohand Ouamer Bibi

Research Unit LaMOS, Department of Operations Research, Faculty of the Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria

Abstract In this study, we propose an new approach for solving a quadratic programming problem with an M-matrix and simple constraints. This approach is based on the algorithms of Chandrasekaran, Luk and Pagano. These methods use the fact that an M-matrix possesses a nonnegative inverse which allows to have a sequence of feasible points monotonically increasing. Introducing the concept of support for an objective function developed by Gabasov et al. in 1987, our approach leads to a more general condition which allows to have an initial feasible solution, related to a coordinator support and close to the optimal solution. The programming of our method under MATLAB and that of Luk and Pagano have allowed us to make a comparison between them, in the illustration of two practical examples. The numerical results indicate that our approach is more efficient than the approach proposed by Chandrasekaran, Luk and Pagano.

Keywords Convex Quadratic Programming, M-Matrices, Nonnegative Matrices, Projection Newton Method, Support Method.

AMS 2010 subject classifications 65K05, 90C2, 90C25, 90C30.

DOI: 10.19139/soic-2310-5070-1399

1. Introduction

In the literature, several approaches were suggested for solving a quadratic programming problem when the associated matrix $D$ is positive definite or semidefinite [35, 5, 36, 7, 13, 6, 10, 26, 3, 11, 4, 24]. However, it is possible to exploit the special properties of an M-matrix to obtain more efficient special algorithms. The M-matrices are known to have many applications in the modeling of the dynamic systems, in economic sciences and ecology [2, 37]. Such problems include various types of Dirichlet problems with obstacles [28], and models of the application of torsion to a bar [27]. Several of their properties are used in general to establish results of stability for the dynamic systems [31, 32, 29]. Quadratic minimization with an M-matrix arises directly in a variety of applications including portfolio optimization with transaction costs [23], and image segmentation [14]. Convex quadratic programming with an M-matrix is also studied on its own right [1, 12, 13, 19, 20, 33, 34, 22]. The M-matrices are also present in the obstacle problems [15], and active set methods are used to solve them [19], a direct algorithm for the solution to the affine two-sided obstacle problem with an M-matrix is presented in [12], another method for strictly convex quadratic problems is suggested in [13], this method presents an extension of the external points method [18].

The main contribution of this paper lies in the proposal of a new and efficient algorithm for solving quadratic programming problems with an M-matrix and simple constraints. This method takes advantage of the fact that an M-matrix has a nonnegative inverse (all the elements of the matrix $D^{-1}$ are nonnegative), which gives a monotonically increasing sequence of feasible solutions [33, 34]. By introducing the concept of support for an...
objective function [16], our approach differs from the method presented by Chandrasekaran [8], Luk and Pagano [20] by a more general condition that allows us to have an initial feasible solution, close to the optimal solution. This characteristic facilitates faster convergence to the optimal solution, reducing thus the number of iterations required compared with the Chandrasekaran, Luk and Pagano approaches.

The organization of the paper is as follows: in the second section, we present the problem and give some definitions related to our approach. In section 3, the algorithm for solving the quadratic programming problem with an M-matrix and simple constraints is presented. In Section 4, the programming of our method and that of Luk and Pagano under MATLAB have allowed us to make a comparison between them, in the illustration of two practical examples randomly generated, and this, by varying the number of variables. We finish the article by a conclusion.

2. Position of the problem and definitions

Let us consider the following problem of quadratic programming with simple constraints:

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad F(x) = \frac{1}{2} x^T D x + c^T x, \\
\text{subject to} & \quad x \geq 0,
\end{align*}
\]  

where \( c = c(J) = (c_j, j \in J) \) and \( x = x(J) = (x_j, j \in J) \) are real \( n \)-vectors, with \( J = \{1, 2, \cdots, n\} \). The matrix \( D = D(J, J) \) is a nonsingular symmetric square M-matrix of order \( n \).

**Definition 1.** [37]

A matrix \( D = (d_{ij}, 1 \leq i, j \leq n) \) is said to be an M-matrix if it satisfies the following properties:

\[
\begin{align*}
d_{ii} > 0, & \quad d_{ij} \leq 0, \quad i \neq j, & \quad D^{-1} \geq 0,
\end{align*}
\]

where the symbol \( D^{-1} \geq 0 \) denotes that all the elements of the matrix \( D^{-1} \) are nonnegative.

**Remark 1.** A symmetric M-matrix is always positive definite \( (x^T D x > 0, \forall x \neq 0) \). Moreover, any submatrix of an M-matrix is itself an M-matrix.

**Definition 2.**

A vector \( x \geq 0 \) is called a feasible solution of the problem (1). A feasible solution \( x^0 \) is said optimal if it gives to the objective function of the problem (1) his minimum value.

Thus, we have

**Theorem 1.** A feasible solution \( x^0 \) of the problem (1) is optimal if and only if for all \( j \in J \), the following conditions of optimality are satisfied [16]:

\[
\begin{align*}
x_j^0 = 0 & \quad \Rightarrow \quad g_j(x^0) \geq 0, \\
x_j^0 > 0 & \quad \Rightarrow \quad g_j(x^0) = 0, \quad j \in J,
\end{align*}
\]  

where \( g(x) = g(J) = D x + c \) is the gradient of the objective function \( F \) at the point \( x \).
Let us consider the quadratic program without constraints

$$\min_{x \in \mathbb{R}^n} F(x) = \frac{1}{2} x^T D x + c^T x,$$  \hspace{1cm} (3)$$

whose the optimal solution \( \hat{x} \) satisfies the equation

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

Let \( J_S \) and \( J_N \) be a partition of \( J \): \( J_S \cup J_N = J \), \( J_S \cap J_N = \emptyset \). Then the gradient of the function \( F \) at point \( x \) can be written in the following form:

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

where

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

For all subset \( J_S \) in \( J \), the following condition holds:

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

**Definition 3.**

- The subset \( J_S \) is called a *support* of the objective function and the pair \( J_p = \{J_S, J_N\} \) is a support of the problem (1).
- The couple \( \{x, J_S\} \), comprising the feasible solution \( x \) and the support \( J_S \) is called a *support feasible solution*.
- A vector \( \kappa = \kappa(J) = (\kappa(J_S), \kappa(J_N)) \) satisfying

$$\begin{cases} k_N = 0, \\
k_S = -D_S^{-1} c_S, \end{cases}$$

is called a *pseudosolution* of the problem (1). A pseudosolution verifies \( g_S(\kappa) = 0 \).

- The support \( J_P = \{J_S, J_N\} \) is called a *coordinator support* if there is a pseudosolution \( \kappa \) such that:

$$g_j(\kappa) \geq 0, \quad j \in J_N.$$  \hspace{1cm} (4)$$

In this case, we say that the pseudosolution \( \kappa \) is associated to the coordinator support \( J_P \).

**Theorem 2.** A pseudosolution \( \kappa \) associated to a coordinator support \( J_P \) is optimal in the problem (1) if and only if

$$\kappa_j \geq 0, \quad j \in J_S.$$  \hspace{1cm} (5)$$

**Remark 2.** Any pseudosolution \( \kappa \), associated to a coordinator support \( J_P \), is a feasible solution for the dual of the primal problem (1):  

$$\begin{cases} F(\kappa) = -\frac{1}{2} \kappa^T D\kappa \to \max, \\
D\kappa + c \geq 0. \end{cases}$$  \hspace{1cm} (6)$$
Remark 3. As \( g(\hat{x}) = 0 \), then the optimal solution \( \hat{x} \) of the problem without constraints (3) is a pseudosolution of the problem (1), associated to the coordinator support \( J_p = \{ J_S, J_N \} \), where \( J_S = J \) and \( J_N = \emptyset \). According to the theorem 2, if \( \hat{x} \geq 0 \), then \( x^0 = \hat{x} \) is the optimal solution of the problem (1).

Let us recall the following lemma:

Lemma 1. \([20]\).

(a) If \( c \geq 0 \), then \( x^0 = 0 \) solves the problem (1),

(b) If \( c \leq 0 \), then \( x^0 = -D^{-1}c \) solves the problem (1).

To eliminate these two trivial cases, let us consider the general one where the vector \( c \) contains both positive and negative components, and construct the two following sets of indices:

\[ J_S = \{ j \in J : \hat{x}_j \geq 0 \}, \quad J_N = \{ j \in J : \hat{x}_j < 0 \}, \quad J_S \cup J_N = J. \]

- If \( J_S = J \), then \( x^0 = \hat{x} = -D^{-1}c \) is the optimal solution of the problem (1).

- Else, let \( y \) be the projection of \( \hat{x} \) on the admissible set of the problem (1), where \( y = (y_j, \ j \in J), \ y_j = \max\{0, \hat{x}_j\} \). Therefore we will have

\[
\begin{cases}
  y_N = 0 > \hat{x}_N, \\
  y_S = \hat{x}_S.
\end{cases}
\]

For the construction of our algorithm, we have established the following lemmas:

Lemma 2. The following inequality holds:

\[ g_S(y) \leq g_S(\hat{x}). \]

Proof. We have

\[
g_S(\hat{x}) = D_S \hat{x}_S + D(J_S, J_N) \hat{x}_N + c_S \\
= D_S y_S + c_S + D(J_S, J_N) \hat{x}_N \\
= g_S(y) + D(J_S, J_N) \hat{x}_N \\
\geq g_S(y),
\]

because \( D(J_S, J_N) \leq 0 \) and \( \hat{x}_N < 0 \). \( \square \)

Since \( g_S(\hat{x}) = 0 \), then by lemma 2 we deduce that \( g_S(y) \leq 0 \). Then we construct a vector \( x \) such that

\[
\begin{cases}
  x_N = y_N = 0, \\
  x_S = -D^{-1}S c_S.
\end{cases}
\]

Thus we have

\[ g_S(y) \leq 0 \text{ and } g_S(x) = 0. \]

Lemma 3. The vectors \( y \) and \( x \) satisfy the following inequality:

\[ x_S \geq y_S \geq 0. \]
**Proof.** We have

$$D_S (x_S - y_S) = D_S x_S + c_S - (D_S y_S + c_S).$$

As $x_N = y_N = 0$, $g_S(y) \leq 0$ and $g_S(x) = 0$, then we deduce

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

The submatrix $D_S$ is an M-matrix [20], that yields $D_S^{-1} \geq 0$. Consequently, from the above inequality, we obtain

$$x_S \geq y_S \geq 0. \quad \Box$$

By lemma 3, the constructed vector $x (7)$ is a feasible solution of the problem (1). We have then the following lemma:

**Lemma 4.**

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

**Proof.** Let

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

As $x_S = -D_S^{-1} c_S$, we will have

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

Because the submatrix $D_S$ is positive definite, then we can write

$$2F(x) \leq (y_S - x_S)^T D_S (y_S - x_S) - c_S^T D_S^{-1} c_S$$

$$\leq (y_S^T D_S y_S + x_S^T D_S x_S - 2 y_S^T D_S x_S - c_S^T D_S^{-1} c_S$$

$$\leq y_S^T D_S y_S + c_S^T D_S^{-1} D_S c_S + 2 y_S^T D_S D_S^{-1} c_S - c_S^T D_S^{-1} c_S$$

$$\leq y_S^T D_S y_S + 2 c_S^T y_S$$

$$\leq 2 F(y).$$

Hence

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

**Remark 4.** The constructed vector $x (7)$ satisfies thus the following inequality:

$$F(\hat{x}) < F(x^0) \leq F(x) \leq F(y),$$

where $y$ is the projection of $\hat{x}$ on the admissible set of the problem (1).

We recall the following theorem
**Theorem 3.** [20] Assume $D_S^{-1}c_S \leq 0$ for some nonempty subset $J_S \subset J$. Define a vector $x$ with $x_S = -D_S^{-1}c_S$ and $x_N = 0$. Let $J_N^-$ and $J_N^+$ be two sets partitioning $J_N$ such that

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

If the set $J_N^-$ is empty, then

1. the vector $x$ solves the problem (1), else
2. let $J_S := J_S \cup J_N^-$. Construct $\pi$ with $\pi(J_S) = -D^{-1}(J_S, J_S) c(J_S)$ and $\pi(J_N^+) = 0$. We get

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

   (b) $g_j(\pi) \leq g_j(x)$, \quad $j \in J_N^+$,

   (c) $F(\pi) < F(x)$.

3. **Algorithm of the method**

   Based on the previous theorem, we propose the following algorithm:

   **Begin**
   
   1. Compute the optimal solution $\hat{x}$ of the problem (3):

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

   2. If $\hat{x} \geq 0$, then stop and the vector $x^0 = \hat{x}$ is the optimal solution of the problem (1).

   3. Else, define the sets:

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

   4. Construct $x$ as follows:

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

   5. Let $J_N^-$ and $J_N^+$ be two sets partitioning $J_N$ such that

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

   6. **Repeat**

   1. Compute $g_N(x) = D(J_N, J_S) x_S + c_N$,

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

   3. If $J_N^-$ is nonempty, then

      (a) Let $J_S := J_S \cup J_N^-$ and $J_N := J_N \setminus J_N^-$,

      (b) Reconstruct $x$ such that $x_N = 0$ and $x_S = -D_S^{-1}c_S$,

   until the set $J_N^- = \emptyset$.

   **End.**
4. Experimental results

In this section, we have chosen two representative problems. The goal is to show the effectiveness of our proposed algorithm in making a numerical comparison with the algorithm of Luk and Pagano [20]. All experiments were conducted on a computer with Intel(R) Core(TM) i3-2350 CPU @ 2.30 GHz with 4.00Go of RAM, working under Windows 7 operating system with MATLAB R2015a programming language. The criterion of the comparison between the two methods is the average CPU time (Avr-CPU) in seconds and the average number of iterations (Avr-Iter) necessary provided to obtain the optimal solution of the problem. All these tests were conducted on the same computer. The values presented in the tables represent the averages of 5 test problems for each value \( n \). We define by

- **Algorithm1**: Chandrasekaran, Luk and Pagano method[20],
- **Algorithm2**: proposed algorithm,
- \( NJ_S \): the number of elements in the support \( J_S \) of the objective function, just after having calculated \( \hat{x} = -D^{-1}c \),
- \( \overline{NJ_S} \): the number of elements in the support \( J_S \) of the objective function, at the optimum,
- \( NP \): the number of elements in the set \( P \), at the initialization of \( x \), with \( P = \{ j \in J : c_j \leq 0 \} \) and \( \overline{P} = J \setminus P \),
- \( \overline{NP} \): the number of elements in the set \( P \), at the optimum,
- **Avr-Iter**: the average number of iterations performed by each algorithm,
- **Avr-CPU**: the average CPU time in seconds necessary provided to obtain the optimal solution of the problem.

4.1. Example 1.

Let us consider the quadratic program (1):

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad F(x) = \frac{1}{2}x^T Dx + c^T x, \\
\text{subject to} & \quad x \geq 0,
\end{align*}
\]

and, the matrix \( D \) is the matrix corresponding to the finite difference discretization of the one-dimensional Dirichlet problem [38].

\[
D = \begin{pmatrix}
2 & -1 & 0 & \cdots & 0 \\
-1 & 2 & -1 & \ddots & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & \ddots & -1 & 2 & -1 \\
0 & \cdots & 0 & -1 & 2
\end{pmatrix}_{n \times n}
\]

We choose to generate the vector \( c \) in order to have three cases of the set \( J_S \). Let \( r_i \) be a random number from an uniform distribution \( r_i \in U[0, 1] \). The comparison criterion between the two methods is based on the average CPU time (Avr-CPU) in seconds and the average number of iterations (Avr-Iter) necessary provided to obtain the optimal solution of the problem. The results are presented in the tables below:
4.1.1. Case 1. The vector \( c \) is generated so that to have \( N J_S = n \)

\[
c_i = 11 - 20 r_i \quad \text{for} \quad i = 1, 2, \ldots, n. \tag{11}
\]

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Algorithm1</th>
<th>Algorithm2</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Avr-Iter</td>
<td>NP</td>
</tr>
<tr>
<td>500</td>
<td>07</td>
<td>280</td>
</tr>
<tr>
<td>1000</td>
<td>06</td>
<td>559</td>
</tr>
<tr>
<td>1500</td>
<td>07</td>
<td>846</td>
</tr>
<tr>
<td>2000</td>
<td>08</td>
<td>1077</td>
</tr>
<tr>
<td>2500</td>
<td>08</td>
<td>1356</td>
</tr>
<tr>
<td>3000</td>
<td>08</td>
<td>1625</td>
</tr>
<tr>
<td>4000</td>
<td>09</td>
<td>2187</td>
</tr>
<tr>
<td>5000</td>
<td>11</td>
<td>2742</td>
</tr>
</tbody>
</table>

Table 1. The average CPU time in seconds and the average number of iterations performed by each algorithm with \( N J_S = n \).

4.1.2. Case 2. For \( N J_S < n \), the vector \( c \) is generated by

\[
c_i = 11 - 22 r_i \quad \text{for} \quad i = 1, 2, \ldots, n. \tag{12}
\]

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Algorithm1</th>
<th>Algorithm2</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Avr-Iter</td>
<td>NP</td>
</tr>
<tr>
<td>500</td>
<td>14</td>
<td>249</td>
</tr>
<tr>
<td>1000</td>
<td>21</td>
<td>499</td>
</tr>
<tr>
<td>1500</td>
<td>23</td>
<td>727</td>
</tr>
<tr>
<td>2000</td>
<td>26</td>
<td>1010</td>
</tr>
<tr>
<td>2500</td>
<td>22</td>
<td>1242</td>
</tr>
<tr>
<td>3000</td>
<td>26</td>
<td>1482</td>
</tr>
<tr>
<td>4000</td>
<td>26</td>
<td>2001</td>
</tr>
<tr>
<td>5000</td>
<td>26</td>
<td>2500</td>
</tr>
</tbody>
</table>

Table 2. The average CPU time in seconds and the average number of iterations performed by each algorithm with \( N J_S < n \).

4.1.3. Case 3. For \( N J_S = 0 \), one generates the vector \( c \) by

\[
c_i = 11 - 25 r_i \quad \text{for} \quad i = 1, 2, \ldots, n. \tag{13}
\]
Table 3. The average CPU time in seconds and the average number of iterations performed by each algorithm with $NJ_S = 0$.

A comparison of the average CPU time for the proposed algorithm and the Luk and Pagano algorithm [20], is showed in Figure 1.

Figure 1. The average CPU time (Avr-CPU) in seconds performed by each algorithm for example 1

In this example, we see that our approach is more efficient in machine time than the approach of Chandrasekaran, Luk and Pagano. And that, whatever the number of elements in the support $J_S$ of the objective function at the initial step of the algorithm.

4.2. Example 2.

In this second example, the matrix $D$ of the problem is chosen as a $5$–point finite difference Laplacian operator[38]:

$$ D = \begin{pmatrix} B & -I & 0 & \cdots & 0 \\ -I & B & -I & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -I & B & -I \\ 0 & \cdots & 0 & -I & B \end{pmatrix}_{m^2 \times m^2} , \quad B = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & 4 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 4 & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{pmatrix}_{m \times m} , \quad (14) $$

with $n = m^2$. The matrix $I$ is the identity matrix.

We choose to generate the vector $c$ in order to have three cases of the set $J_S$ as in the previous example. The comparison criterion between the two methods is based on the average CPU time (Avr-CPU) in seconds and the
average number of iterations (Avr-Iter) necessary provided to obtain the optimal solution of the problem. The results obtained are presented in the tables below:

4.2.1. Case 1. The vector $c$ is generated by

$$c_i = 8 - 10 \, r_i \quad \text{for} \quad i = 1, 2, \ldots, n.$$  \hspace{1cm} (15)

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Algorithm1</th>
<th>Algorithm2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = m \times m$</td>
<td>Avr-Iter</td>
<td>NP</td>
</tr>
<tr>
<td>20 $\times$ 20</td>
<td>01</td>
<td>393</td>
</tr>
<tr>
<td>30 $\times$ 30</td>
<td>01</td>
<td>813</td>
</tr>
<tr>
<td>40 $\times$ 40</td>
<td>01</td>
<td>1222</td>
</tr>
<tr>
<td>50 $\times$ 50</td>
<td>01</td>
<td>1603</td>
</tr>
<tr>
<td>60 $\times$ 60</td>
<td>01</td>
<td>2387</td>
</tr>
<tr>
<td>70 $\times$ 70</td>
<td>01</td>
<td>3207</td>
</tr>
</tbody>
</table>

Table 4. The average CPU time in seconds and the average number of iterations performed by each algorithm with $N_J_S = n$.

4.2.2. Case 2. The vector $c$ is generated by

$$c_i = 8 - 16 \, r_i \quad \text{for} \quad i = 1, 2, \ldots, n.$$  \hspace{1cm} (16)

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Algorithm1</th>
<th>Algorithm2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \times n$</td>
<td>Avr-Iter</td>
<td>NP</td>
</tr>
<tr>
<td>20 $\times$ 20</td>
<td>12</td>
<td>198</td>
</tr>
<tr>
<td>30 $\times$ 30</td>
<td>12</td>
<td>441</td>
</tr>
<tr>
<td>40 $\times$ 40</td>
<td>15</td>
<td>849</td>
</tr>
<tr>
<td>50 $\times$ 50</td>
<td>16</td>
<td>1253</td>
</tr>
<tr>
<td>60 $\times$ 60</td>
<td>19</td>
<td>1792</td>
</tr>
<tr>
<td>70 $\times$ 70</td>
<td>24</td>
<td>2360</td>
</tr>
</tbody>
</table>

Table 5. The average CPU time in seconds and the average number of iterations performed by each algorithm with $N_J_S < n$.

4.2.3. Case 3. The vector $c$ is generated by

$$c_i = 8 - 20 \, r_i \quad \text{for} \quad i = 1, 2, \ldots, n.$$  \hspace{1cm} (17)
A comparison of the average CPU time for the proposed algorithm, the Luk and Pagano algorithm [20], is showed in Figure 2. And that, whatever the number of elements in the support $J_S$ of the objective function at the initial step of the algorithm.

From the numerical examples above, our approach often requires less average CPU time than Chandrasekaran, Luk and Pagano’s approach. This is true whatever the number of elements in the support $J_S$ of the objective function at the initial step of the algorithm.

5. Conclusion

The lemmas 2, 3 and 4 allowed us to start the algorithm with a feasible solution $x$ checking the conditions of the theorem 3 and the inequality (8). If the matrix of our objective function $D = I$, where $I$ is the identity matrix, then we have

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

and we find the conditions of the initialization of the algorithms of Chandrasekaran, Luk and Pagano [8, 20]. Let us notice that their algorithms finish with $J_N$ or $J_N^c$ empty, while ours always finishes with $J_N^c = \emptyset$ and $J_N \neq \emptyset$, and this, because of our initialization. Indeed, the case $J_N = \emptyset$ corresponds to the optimal solution $x^0 = \tilde{x}$.
QUADRATIC PROGRAMMING METHOD WITH AN M-MATRIX

From the two numerical examples, we see that our approach is more efficient in machine time than the approach of Chandrasekaran, Luk and Pagano. This is true for both examples, whatever the number of elements in the support $J_S$ of the objective function at the initial step of the algorithm.

REFERENCES
