The Application of Fuzzification for Solving Quadratic Functions

Lunshan Gao (lgao@wlu.ca)
Wilfrid Laurier University Faculty of Science  https://orcid.org/0000-0002-5890-7735

Research Article

Keywords: standard quadratic optimization problem, ε - δ condition, fuzzification, normalized possibility distribution, Hessian matrix, CPLEX

Posted Date: December 6th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-1121957/v1

License: ☑️ This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License
The Application of Fuzzification for Solving Quadratic Functions

Lunshan Gao

Received: date / Accepted: date

Abstract This paper describes an approximation algorithm for solving standard quadratic optimization problems (StQPs) over the standard simplex by using fuzzification technique. We show that the approximate solution of the algorithm is an $\epsilon$-critical point and satisfies $\epsilon - \delta$ condition. The algorithm is compared with IBM ILOG CPLEX (short for CPLEX). The computational results indicate that the new algorithm is faster than CPLEX. Especially for infeasible problems. Furthermore, we calculate 100 instances for different size StQP problems. The numerical experiments show that the average computational time of the new algorithm for calculating the first local minimizer is in $O(n)$ when the size of the problems is less or equal to 450.

Keywords standard quadratic optimization problem · $\epsilon - \delta$ condition · fuzzification · normalized possibility distribution · Hessian matrix · CPLEX

1 Introduction

1.1 The problem

A standard quadratic optimization problem (StQP) is described as follows.

$$\min_{x \in \Delta} f(x) = x^T Qx,$$

$$\Delta = \{x \in \mathbb{R}^n \mid x_i \geq 0; e^T x = 1\}.$$  

where $Q$ is an arbitrary symmetric $n \times n$ matrix and $e$ is the $n$-vector of all ones.

The research of approximate algorithms is important for solving NP-hard problems. A StQP is a NP-hard problem when the matrix is indefinite (Bomze

---

L. Gao, Ph.D., P.Eng.,
Physics and Computer Science, Wilfrid Laurier University, ON. Canada E-mail: lgao@wlu.ca, https://orcid.org/0000-0002-5890-7735
Ahmadi and Zhang show that it is NP-hard to decide if a quadratic function has a local minimizer over a polyhedron (Ahmadi and Zhang, 2020). Furthermore, they indicate that if there is a polynomial-time algorithm that finds a point within Euclidean distance $c^n$ (for any constant $c \geq 0$) of a local minimizer of an $n$-variate quadratic function over a polytope, then $P = NP$ (Ahmadi and Zhang, 2020).

On the other hand, researchers have proved that there are polynomial time approximate scheme (PTAS) for solving StQPs over the standard simplex (Bomze and de Klerk, 2002; De Klerk, 2008; De Klerk et al., 2008).

Calculating local optimal solution is a major topic in nonconvex optimization problems (Agarwal et al., 2016; Ahmadi and Zhang, 2020). Agarwal et al. present that finding approximate local minima for nonconvex optimization problems can be done in linear time (Agarwal et al., 2016) and prove that the total time of computing local minima over nonconvex domains is $O\left(\frac{dn^d}{\epsilon^d}\right)$, where $d$ is the degree and $n$ is the number of variables. It is clear that the computational time will become larger when $\epsilon$ becomes smaller such that if $\epsilon \to 0$ then time $\to \infty$.

Fuzzy sets theory and possibility theory were invented half century ago (Zadeh, 1978). Fuzzy sets theory has been applied for solving optimization problems (Gao-1, 2020; Gao-2, 2020; Gao and Kawarada, 1995; Kaufmann and Gupta, 1998; Liu, 2007; Momot and Momot, 2009). A fuzzy number is a fuzzy set whose membership function is defined in $R$. Triangular fuzzy number (TFN) is a special case of a fuzzy number whose membership function forms a triangle (Dijkman et al., 1983). The concept of fuzzification is commonly used in fuzzy sets theory and fuzzy control theory (Gao, 1999; Zimmermann, 2001). A fuzzification is the process of transforming system variables into fuzzy sets which are usually fuzzy numbers or TFNs. For multiple variables, fuzzification can map $x \in R^n$ to a set of TFNs. A set of TFNs is usually called a possibility distribution. In practice, normalized possibility distributions are used (Gao-1, 2020).

This paper is the continuation research of the previous paper (Gao-1, 2020). We take advantage of piecewise linearity of TFNs and apply the concept of normalized possibility distribution to find optimal values for StQP problems. This paper addresses the following issues.

1. The approximation solution of our algorithm is an $\epsilon$-critical point.
2. The approximate solution satisfies $\epsilon$-$\delta$ condition.
3. Our algorithm is compared with CPLEX.

This paper is organized as follows. Section 2 gives a preliminary to basic concepts that are used in this paper. Section 3 describes the algorithm in detail. Section 4 provides the analysis of the algorithm and shows that the approximate solution satisfies $\epsilon - \delta$ condition. Section 5 gives numerical experiment results. Section 6 gives the conclusion.

2 Preliminary

**Definition 1** ($\epsilon$-approximate local minimum (Agarwal et al., 2016))
Suppose that function \( f(x)(x \in \mathbb{R}^n) \) is twice differentiable. A point \( x^* \in \mathbb{R}^n \) is called an \( \epsilon \)-approximate local minimum if the following is satisfied.

\[
\| \nabla f(x) \|_{x=x^*} \leq \epsilon, \quad (\epsilon > 0), \quad (2.1)
\]

\[
\nabla^2 f(x) \big|_{x=x^*} \geq -(\epsilon)^{1/2} I
\]

where \( \| \cdot \| \) denotes the Euclidean norm of a vector and \( I \) is the identity matrix.

We say that a point \( x^* \) is an \( \epsilon \)-critical point if inequality (2.1) is satisfied.

In this paper, we only use the concept of \( \epsilon \)-critical point such that the inequality (2.1) is satisfied.

**Definition 2** (epsilon-delta condition)

A function \( f(x) : \mathbb{R}^n \to \mathbb{R} \) is continuous at point \( x^* \in \mathbb{R}^n \). If for given \( \epsilon > 0 \), there exists \( \delta > 0 \) such that if \( \| x - x^* \| < \delta \), then \( | f(x) - f(x^*) | < \epsilon \), where \( \| \cdot \| \) denotes the Euclidean norm of a vector.

Note that Hessian matrix plays an important role in optimization problems. The property of Hessian matrix determines if a solution is an optimal solution.

**Definition 3** For a function \( f : \mathbb{R}^n \to \mathbb{R} \) and \( x \in \mathbb{R}^n \), if \( f(x) \) is twice differentiable, then the Hessian matrix of \( f(x) \) at \( x = x_0 \) is defined as follows.

\[
Hf \big|_{x=x_0} = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} \big|_{x=x_0} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \big|_{x=x_0} & \ldots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \big|_{x=x_0} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} \big|_{x=x_0} & \frac{\partial^2 f}{\partial x_2^2} \big|_{x=x_0} & \ldots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \big|_{x=x_0} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_n \partial x_1} \big|_{x=x_0} & \frac{\partial^2 f}{\partial x_n \partial x_2} \big|_{x=x_0} & \ldots & \frac{\partial^2 f}{\partial x_n^2} \big|_{x=x_0}
\end{bmatrix}
\]

In this paper we will use the following results.

If \( f(x_0) \) is a local minimum value, then the Hessian matrix is positive semidefinite.

If \( f(x_0) \) is a local maximum value, then the Hessian matrix is negative semidefinite.

### 3 The Algorithm

This section consists of three parts. The first part is to describe the fuzzification that transfers the space \( x \in \mathbb{R}^n \) into the space of a normalized possibility distribution. The second part is to prove that the approximate solution is an \( \epsilon \)-critical point. The final part is to describe the transformed Hessian matrix at the \( \epsilon \)-critical point after the fuzzification.

#### 3.1 The Fuzzification

Fuzzification is the process of assigning input variables of a system to fuzzy sets by using membership functions. In this paper, the variable \( x \in \mathbb{R}^n \) is assigned to a set of normalized TFNs.
Suppose that \( \lambda_i = (d_i^l, d_i^m, d_i^r) \) is a TFN, and \( \lambda(t) = (\lambda_1(t), \cdots, \lambda_n(t)) \) is a set of TFNs, that is considered as a possibility distribution. Since \( x \in \Delta \) in problem (1.1), it is required to normalize a given possibility distribution. A normalized TFN \( \mu_i(t) \) is defined as follows.

\[
\mu_i(t) = \frac{\lambda_i(t)}{\sum_{j=1}^{n} \lambda_j(t)}, \quad (i = 1, 2, \cdots, n).
\]

The fuzzification is to map \( x_i \) to a normalized TFN \( \mu_i(t) \). As a result, variable \( x \in \mathbb{R}^n \) is mapped to a set of normalized TFNs. The mathematical form of the fuzzification is as follows.

\[
x = \mu(t) = \frac{\lambda(t)}{\sum_{j=1}^{n} \lambda_j(t)}, \quad (t \in D = [0, 1]).
\]

where \( x \in \mathbb{R}^n, \mu(t) \in \mathbb{R}^n, \) and \( \lambda(t) \in \mathbb{R}^n \).

When a set of TFNs \( \lambda(t) \) is given in domain \( D = [0, 1] \), the mean value \( d_i^m \) of TFN \( \lambda_i \) will distribute in domain \( D = [0, 1] \). Without loss of generality, we suppose that

\[
0 \leq d_1^m \leq d_2^m \leq \cdots \leq d_n^m \leq 1.
\]

These mean values decompose the domain \( D = [0, 1] \) into \( M \) intervals such that

\[
D = \bigcup_{i=1}^{M} D_i,
\]

where \( 1 \leq M \leq n \) and \( D_1 = [d_1^l, d_1^m], D_M = [d_{n-1}^m, d_n^r], D_i = [d_{i-1}^m, d_i^m] (i = 2, \cdots, M-1) \). TFN \( \lambda_i(t) \) can be denoted as follows in \( D_j \subseteq [0, 1] \).

\[
\lambda_i(t) = a_i t + b_i, \quad a_i \in \mathbb{R}, \quad b_i \in \mathbb{R}
\]

\[
(i = 1, 2, \cdots, n), \quad t \in D_j (j = 1, 2, \cdots, M)
\]

A possibility distribution \( \lambda(t) \) is as follows.

\[
(\lambda(t))^T = (a_1 t + b_1, a_2 t + b_2, \cdots, a_n t + b_n), \quad t \in D_i (i = 1, 2, \cdots, M).
\]

The coefficients \( a_i \) and \( b_i \) are denoted as follows.

\[
A = (a_1, a_2, \cdots, a_n), \quad B = (-b_1, -b_2, \cdots, -b_n).
\]

The fuzzification can be denoted as follows.

\[
x = \mu(t), \quad \mu(t) \in \Delta, \quad t \in D_i (i = 1, 2, \cdots, M).
\]

where \( \mu(t) \) is denoted as follows in \( D_i \).

\[
(\mu(t))^T = \left( \frac{a_1 t + b_1}{\sum_{j=1}^{n} a_j t + \sum_{j=1}^{n} b_j}, \cdots, \frac{a_n t + b_n}{\sum_{j=1}^{n} a_j t + \sum_{j=1}^{n} b_j} \right), \quad t \in D_i.
\]
The derivative of $\mu(t)(t \in D_i)$ is as follows.

$$\left(\frac{d\mu(t)}{dt}\right)^T = \left(\frac{a_1 \sum_{j=1}^n b_j - b_1 \sum_{j=1}^n a_j}{(\sum_{j=1}^n (a_j + b_j))^2}, \cdots, \frac{a_n \sum_{j=1}^n b_j - b_n \sum_{j=1}^n a_j}{(\sum_{j=1}^n (a_j + b_j))^2}\right), t \in D_i.$$

By substituting $x \in \triangle$ with $\mu(t) \in \triangle$, the problem (1.1) can be transformed as follows.

$$\min_{\mu(t) \in \triangle} f(\mu(t)) = (\mu(t))^T Q\mu(t), t \in D_i.$$  \hspace{1cm} (3.5)

The derivative of $f(\mu(t))$ is as follows.

$$\frac{df(\mu(t))}{dt} = \left(\frac{d\mu(t)}{dt}\right)^T Q\mu(t) + (\mu(t))^T Q \frac{d\mu(t)}{dt} = 0, t \in D_i$$ \hspace{1cm} (3.6)

Since $Q$ is a symmetric matrix, equation (3.6) becomes the following.

$$\frac{df(\mu(t))}{dt} = 2 \left(\frac{d\mu(t)}{dt}\right)^T Q\mu(t) = 2K(t)V^T \lambda(t) = 0, t \in D_i$$ \hspace{1cm} (3.7)

where $K(t)$ and $V$ are denoted in (3.8) and (3.9).

$$K(t) = \frac{1}{(\sum_{j=1}^n (a_j^t + b_j^t))^3} = \frac{1}{(\sum_{j=1}^n \lambda_j(t))^3}, t \in D_i.$$ \hspace{1cm} (3.8)

$$V^T = (a_1 \sum_{j=1}^n b_j - b_1 \sum_{j=1}^n a_j, a_2 \sum_{j=1}^n b_j - b_2 \sum_{j=1}^n a_j, \cdots, a_n \sum_{j=1}^n b_j - b_n \sum_{j=1}^n a_j).$$ \hspace{1cm} (3.9)

The necessary and sufficient condition of $t_i^*$ being a solution of equation (3.7) is as follows(Gao-1 , 2020).

$$d_{i-1}^m \leq t_i^* \leq \frac{V^T QB}{V^T QA} \leq d_i^m, (i = 1, 2, \cdots, M).$$ \hspace{1cm} (3.10)

where $d_{i-1}^m, d_i^m$ is the mean value of TFN $\lambda_{i-1}(t)$, $\lambda_i(t)$, respectively, $Q$ is the symmetric matrix, and $A$ and $B$ are defined in (3.3).

3.2 $\mu(t^*)$ is an $\epsilon$-critical points

In this section, we prove that if $t^*$ is a solution of (3.6), then $\mu(t^*)$ is an $\epsilon$-critical point of $f(\mu(t))$. According to the definition 1 in section 2, for a given $\epsilon > 0$, we need to prove the following.

$$\|\nabla f(x)\|_{\mu(t)=\mu(t^*)} \leq \epsilon.$$  \hspace{1cm} (3.11)

Since we know that

$$\left(\nabla f(\mu(t))\right)^T |_{\mu(t)=\mu(t^*)} = \left(\frac{\partial f}{\partial \mu_1}, \frac{\partial f}{\partial \mu_2}, \cdots, \frac{\partial f}{\partial \mu_n}\right) |_{t=t^*} = 0, t^* \in D_i.$$
Let us calculate \( \frac{\partial f}{\partial \mu_i}(i = 1, 2, \ldots, n) \). The first order partial derivative is as follows.

\[
\frac{\partial f}{\partial \mu_i(t)} = \frac{\partial}{\partial \mu_i(t)} \left((\mu(t))^T Q \mu(t), (i = 1, 2, \ldots, n)\right) = \left(\frac{\partial \mu(t)}{\partial \mu_i(t)}\right)^T Q \mu(t) + (\mu(t))^T Q \frac{\partial \mu(t)}{\partial \mu_i(t)}
\]

(3.12)

Since \( x_i \) and \( x_j \) are independent, we assume that \( \mu_i(t) \) and \( \mu_j(t) \) are independent too. Therefore, \( \frac{\partial \mu(t)}{\partial \mu_i(t)} \) is as follows:

\[
\frac{\partial \mu(t)}{\partial \mu_i(t)} = (0, 0, \ldots, d\mu_i(t) \frac{dt}{dt}, \ldots, 0)
\]

Since \( Q \) is a symmetric matrix, equation (3.12) becomes as follows:

\[
\frac{\partial f}{\partial \mu_i(t)} = 2(0, 0, \ldots, d\mu_i(t) \frac{dt}{dt}, \ldots, 0)Q \mu(t), (i = 1, 2, \ldots, n)
\]

(3.13)

where \( Q_T = (q_{i1}, q_{i2}, \ldots, q_{in}) \). If we use the notations \( K(t) \) and \( V \) in (3.8) and (3.9), then (3.13) becomes as follows.

\[
\frac{\partial f}{\partial \mu_i(t)} = 2K(t) V \sum_{j=1}^{n} q_{ij} \lambda_j(t), (i = 1, 2, \ldots, n).
\]

We have the following theorem.

**Theorem 1** Suppose that \( t^* \in D_i \) is a solution of (3.6). For a given \( \epsilon > 0 \), If

\[
|\frac{\partial f}{\partial \mu_i(t)}|_{t=t^*} = |2K(t^*) V \sum_{j=1}^{n} q_{ij} \lambda_j(t^*)| \leq \left(\frac{\epsilon^2}{n}\right)^{1/2},
\]

(3.14)

then \( \mu(t^*) \) is an \( \epsilon \)-critical point.

**Proof** Based on the definition of \( \epsilon \)-critical point, we need to prove the following.

\[
\| \nabla f \|_{\mu(t)=\mu(t^*)} \leq \epsilon.
\]

Since

\[
\| \nabla f \|_{\mu(t)=\mu(t^*)} = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial f}{\partial \mu_i(t)}\right)^2}.
\]

According to (3.14), we have the following

\[
\| \nabla f \|_{\mu(t)} = \sqrt{\sum_{i=1}^{n} (2K(t^*) V \sum_{j=1}^{n} q_{ij} \lambda_j(t^*) )^2} \leq \sqrt{\sum_{i=1}^{n} \frac{\epsilon^2}{n}} = \epsilon.
\]

Thus, \( \mu(t^*) \) is an \( \epsilon \)-critical point of \( f(\mu(t)) \).
3.3 Hessian matrix after the fuzzification

In this section, we will discuss how Hessian matrix is transformed into a different form of Hessian matrix after the fuzzification.

Fig. 1: The transformation of Hessian matrix

**Theorem 2** If \( t^* \in D_i \) is a solution of equation (3.7), then the transformed Hessian matrix is as follows.

\[
Hf(\mu(t)) \mid_{\mu(t)=\mu(t^*)} = C \begin{bmatrix}
V_1 q_{11} & V_1 q_{12} V_2 & \cdots & V_1 q_{1n} V_n \\
V_2 q_{11} V_1 & V_2 q_{12} V_2 & \cdots & V_2 q_{1n} V_n \\
\vdots & \vdots & \ddots & \vdots \\
V_n q_{11} V_1 & V_n q_{12} V_2 & \cdots & V_n^2 q_{nn}
\end{bmatrix} = Q' \quad (3.15)
\]

where \( q_{ij} \) is the element of matrix \( Q \), \( V_i \) is an element of \( V \) that is defined in (3.9), and \( C \) is a constant which is denoted as follows.

\[
C = \frac{2}{\left(\sum_{j=1}^{n} \lambda_j(t)\right)^4} \mid_{t=t^*}.
\]

**Proof** According to equation (3.13), the second order partial derivative is as follows.

\[
\frac{\partial}{\partial \mu_j(t)} \left( \frac{\partial f}{\partial \mu_i(t)} \right) = \frac{\partial}{\partial \mu_j(t)} \left( 2 a_i \sum_{j=1}^{n} b_j - b_i \sum_{j=1}^{n} a_j Q_i^T \mu(t) \right)
\]

\[
= 2 \frac{a_i \sum_{j=1}^{n} b_j - b_i \sum_{j=1}^{n} a_j q_{ij} \sum_{i=1}^{n} b_i - b_j \sum_{i=1}^{n} a_i}{(\sum_{j=1}^{n} \lambda_j(t))^2}
\]

If we denote that \( C = \frac{2}{\left(\sum_{j=1}^{n} \lambda_j(t)\right)^4} \mid_{t=t^*} \) and use the elements of \( V \) in (3.9), then the second order derivative becomes as follows.

\[
\frac{\partial}{\partial \mu_j(t)} \left( \frac{\partial f}{\partial \mu_i(t)} \right) = CV_i q_{ij} V_j.
\]

Therefore, the transformed Hessian matrix in domain \( D_i \) is as follows.

\[
Hf(\mu(t)) \mid_{\mu(t)=\mu(t^*)} = H' = C \begin{bmatrix}
V_1^2 q_{11} & V_1 q_{12} V_2 & \cdots & V_1 q_{1n} V_n \\
V_2 q_{11} V_1 & V_2^2 q_{22} & \cdots & V_2 q_{1n} V_n \\
\vdots & \vdots & \ddots & \vdots \\
V_n q_{11} V_1 & V_n q_{12} V_2 & \cdots & V_n^2 q_{nn}
\end{bmatrix}
\]
We know that $\mu(t^*)$ is an $\epsilon$-critical point if $t^*$ is a solution of (3.6) and the inequalities in (3.14) is satisfied. It is necessary to investigate the transformed Hessian matrix after the fuzzification. This leads to the following theorem.

**Theorem 3** If $t^* \in D_i$ is a solution of equation (3.6) and $S = V^TQA > 0$, then the transformed Hessian matrix is positive semi-definite in $D_i$.

**Proof** According to the Theorem 3.2 in the paper (Gao-1, 2020), when $S = V^TQA > 0$, then $f(\mu(t^*))$ is a local minimum value such that $f(\mu(t)) > f(\mu(t^*))$ for all $t \in D_i$. That is, the transformed Hessian matrix $H'$ in $D_i$ is positive semi-definite.

Let us see an example. The following Hessian matrix $H$ is indefinite because it has one negative eigenvalue and two positive eigenvalues.

$$H = \begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & 3 \\ -1 & 3 & 1 \end{pmatrix}$$

If we define $\lambda_3(t) = 0$ such that $a_3 = 0$ and $b_3 = 0$ and we establish a possibility distribution such as $(\lambda_1(t), \lambda_2(t), 0)$. After the fuzzification, the Hessian matrix $H$ becomes $H'$ that is positive semi-definite because all of its eigenvalues are non-negative.

$$H = \begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & 3 \\ -1 & 3 & 1 \end{pmatrix} \xrightarrow{\text{Fuzzification (} V_3 = 0 \text{)}} H' = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Theorem 3 provides a method to verify if an $\epsilon$-critical point is an approximate minimum solution or not. The algorithm is depicted in Algorithm 1.

**Algorithm 1**

Given integer number $M \geq 1$

while $M > 0$

Step 1. Using random number generator generates vectors $A$ and $B$ to fuzzyfy variable $x \in \mathbb{R}^n$ with a normalized possibility distribution $\mu(t) \in \mathbb{R}^n$. Determine sub domain $D_i = [d_{i-1}^m, d_i^m](i = 1, \cdots, M)$.

Step 2. Calculate $V$ in $D_i$ based on (3.9).

Step 3. Calculate $t^*$ based on equation (3.10).

Step 4. Verify if $t^*$ satisfies the inequalities (3.10). If $t^* \notin D_i$, go to next sub domain.

Step 5. Calculate $S = V^TQA$, and decide $f(\mu(t^*))$ is a local minimum value or a local maximum value in $D_i$.

Step 6. Compute the inner critical point $\mu(t^*)$ and the approximate optimal value $f(\mu(t^*))$.

end while

Step 7. Find the maximum(minimum) value of $f(\mu(t^*))$ by comparing all the local maximum(minimum) values.

return $\mu(t^*)$ and $f(\mu(t^*))$
4 Algorithm Analysis

In this section, we show that the approximate solution \( \mu(t^*) \) satisfies \( \delta - \epsilon \) proof conditions when \( \mu(t^*) \) is an \( \epsilon \)-approximate local minimizer.

**Theorem 4** Suppose that \( t^* \in D_i \) is a solution of (3.7). If \( \mu(t^*) \) is an \( \epsilon \)-critical point of \( f(\mu(t)) \), then \( \forall \, \epsilon > 0 \), there \( \exists \, \delta > 0 \) such that

\[
\| \mu(t) - \mu(t^*) \| < \delta, \, t \in D_i,
\]

then

\[
\| f(\mu(t)) - f(\mu(t^*)) \| < \epsilon, \, t \in D_i.
\]

**Proof** The Taylor expansion of \( f(\mu(t)) \) at \( \mu(t^*) \) in domain \( D_i \) is as follows.

\[
f(\mu(t)) = f(\mu(t^*)) + (\nabla f(\mu(t^*)))^T (\mu(t) - \mu(t^*))
\]

\[
+ \frac{1}{2} (\mu(t) - \mu(t^*))^T \tilde{H}(\mu(t^*)) (\mu(t) - \mu(t^*)) + H.O.T
\]

(4.2)

where \( \nabla f(\mu(t^*)) \) is the gradient of \( f \) at \( \mu(t^*) \), \( H(\mu(t^*)) \) is the transformed Hessian matrix at \( t^* \in D_i \). We have the following.

\[
| f(\mu(t)) - f(\mu(t^*)) | = | (\nabla f(\mu(t^*)))^T (\mu(t) - \mu(t^*))
\]

\[
+ \frac{1}{2} (\mu(t) - \mu(t^*))^T \tilde{H}(\mu(t^*)) (\mu(t) - \mu(t^*)) + H.O.T |
\]

\[
\leq | (\nabla f(\mu(t^*)))^T (\mu(t) - \mu(t^*)) | + | \frac{1}{2} (\mu(t) - \mu(t^*))^T \tilde{H}(\mu(t^*)) (\mu(t) - \mu(t^*)) |
\]

(4.3)

Since

\[
| (\nabla f(\mu(t^*)))^T (\mu(t) - \mu(t^*)) | \leq \frac{1}{2} (\| \nabla f(\mu(t^*)) \| + \| \mu(t) - \mu(t^*) \|)
\]

The inequality in (4.3) becomes as follows.

\[
| f(\mu(t)) - f(\mu(t^*)) | \leq \frac{1}{2} (\| \nabla f(\mu(t^*)) \| + \| \mu(t) - \mu(t^*) \|)
\]

\[
+ | \frac{1}{2} (\mu(t) - \mu(t^*))^T \tilde{H}(\mu(t^*)) (\mu(t) - \mu(t^*)) |
\]

(4.4)

Since \( \mu(t^*) \) is an \( \epsilon_1 \)-critical point in \( D_i \), \( \| \nabla f(\mu(t^*)) \| < \epsilon_1 \).

According to Theorem 3, the transformed Hessian matrix \( \tilde{H}(\mu(t^*)) \) is positive semi-definite in \( D_i \) such that all eigenvalues of \( H(\mu(t^*)) \) are non-negative. We suppose that the maximum eigenvalue of \( \tilde{H}(\mu(t^*)) \) is denoted as \( \gamma_{\max} \).

The inequality (4.4) becomes as follows.

\[
| f(\mu(t)) - f(\mu(t^*)) | \leq \frac{1}{2} \epsilon_1 + \frac{1}{2} \| \mu(t) - \mu(t^*) \| + \frac{1}{2} \gamma_{\max} \| \mu(t) - \mu(t^*) \|
\]

(4.5)
For a given $\epsilon > \epsilon_1 > 0$, we suppose the following.

$$\frac{1}{2}\epsilon_1 + \frac{1}{2} \| \mu(t) - \mu(t^*) \| + \frac{1}{2} \gamma_{\text{max}} \| \mu(t) - \mu(t^*) \| \leq \epsilon, t \in D_i.$$ 

Then, we have the following.

$$\| \mu(t) - \mu(t^*) \| \leq \frac{\epsilon - \epsilon_1}{1 + \gamma_{\text{max}}}, t \in D_i,$$

Let us denote $\delta$ as follows.

$$\delta = \frac{\epsilon - \epsilon_1}{1 + \gamma_{\text{max}}} > 0.,$$

(4.6)

That is, $\forall \epsilon > \epsilon_1 > 0, \exists \delta > 0$, which is defined in (4.6), if $\| \mu(t) - \mu(t^*) \| < \delta$, then $| f(\mu(t)) - f(\mu(t^*)) | < \epsilon$.

5 Numerical Results

In this section, first, we will show the comparison between the new algorithm and CPLEX that can be downloaded from IBM website (IBM , 2020). Secondly, we calculate 100 instances for each size from $n=50$ to $n=450$ to find out the first local minimizer, and the averages of computational times are measured. The numerical experiment results will be discussed.

CPLEX is a well-known tool for computing optimal values of non-linear functions such as quadratic polynomial functions (IBM , 2020). The C++ methods of CPLEX are called with Microsoft Visual Studio 2015.

5.1 Comparison with CPLEX

In this section, numerical experiments are performed with a computer with Intel i5-7200U processor and 8 GB RAM is used.

When we use CPLEX, the parameters Param::MIP::Tolerances::AbsMIPGap and Param::MIP::Tolerances::MIPGap are set with 0.001 and 0.01. The parameter Param::MIP::Tolerances::AbsMIPGap indicates the absolute MIP gap tolerance. It sets an absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. The default value of this parameter is 1e-06 (IBM , 2020). The parameter Param::MIP::Tolerances::MIPGap indicates a relative MIP gap tolerance. It sets a relative tolerance on the gap between the best integer objective and the objective of the best node remaining. The default value of this parameter is 1e-04 (IBM , 2020).

For CPLEX, The smaller the two parameters, the longer the computational time. If we use the default values for the two parameters, the computation of CPLEX will take longer times.

According to Table 1, when $n = 10$, for the 20 instances, CPLEX solution is same as the new algorithm solution for each instance. However, the new algorithm is faster than CPLEX. Based on the average time for the 20 instances, the new algorithm is ten times faster than CPLEX. Especially, for
some instances, for example, instance number 7, 14, 18, and 20, CPLEX takes quite long time to calculate the solutions.

According to Table 2, when \( n = 20 \), for the 20 instances, compare with CPLEX, some of the new algorithm solutions have discrepancy. Especially for the instance number 5, 12, and 19, the new algorithm solutions are not global optimal solutions but local optimal solutions. If we ignore these discrepancies, the new algorithm is faster than CPLEX. For example, the instance number 2, 3, 6, 8, and 13. For some instances, for example, instance number 2, 8, 13, and 18, CPLEX takes quite long time to calculate the solutions.

The reason why CPLEX takes long time is because these instances are infeasible. A optimization problem is said to be infeasible if no solution which satisfies all the constraints exists (J. W. Chinneck, 2008). CPLEX is an effective algorithm for solving infeasible problems. The technique that CPLEX uses is called infeasibility isolation (IBM, 2020).

Table 1: Comparison between CPLEX and our algorithm (n=10)

| Instances | Solutions | Computational time in Sec. |
|-----------|-----------|----------------------------|
|           | CPLEX     | Our algorithm               |
| 1         | 0.3333    | 0.3333                      | 2.48 | 0.153 |
| 2         | 0.3333    | 0.3333                      | 1.83 | 0.101 |
| 3         | 0.25      | 0.25                        | 0.41 | 0.159 |
| 4         | 0.3333    | 0.3333                      | 0.69 | 0.156 |
| 5         | 0.25      | 0.25                        | 0.25 | 0.046 |
| 6         | 0.3333    | 0.3335                      | 0.41 | 0.154 |
| 7         | 0.3333    | 0.3333                      | 2.14 | 0.061 |
| 8         | 0.3333    | 0.3333                      | 1.06 | 0.156 |
| 9         | 0.3333    | 0.3333                      | 0.94 | 0.154 |
| 10        | 0.25      | 0.25                        | 0.14 | 0.15  |
| 11        | 0.3333    | 0.3333                      | 0.81 | 0.067 |
| 12        | 0.3333    | 0.3333                      | 1.14 | 0.061 |
| 13        | 0.25      | 0.25                        | 0.22 | 0.16  |
| 14        | 0.3333    | 0.3333                      | 2.77 | 0.153 |
| 15        | 0.25      | 0.25                        | 0.28 | 0.036 |
| 16        | 0.25      | 0.25                        | 0.44 | 0.149 |
| 17        | 0.25      | 0.25                        | 0.75 | 0.156 |
| 18        | 0.3333    | 0.3333                      | 2.38 | 0.031 |
| 19        | 0.3333    | 0.3333                      | 0.73 | 0.16  |
| 20        | 0.3333    | 0.3333                      | 3.28 | 0.02  |
Table 2: Comparison between CPLEX and our algorithm (n=20)

| Instances | Solutions CPLEX | Solutions Our algorithm | Computational time in Sec. CPLEX | Computational time in Sec. Our algorithm |
|-----------|----------------|------------------------|-------------------------------|----------------------------------------|
| 1         | 0.1667         | 0.1667                 | 6.66                          | 3.655                                  |
| 2         | 0.2            | 0.2                    | 23.34                         | 3.282                                  |
| 3         | 0.2            | 0.2                    | 11.78                         | 3.562                                  |
| 4         | 0.1667         | 0.1667                 | 4.53                          | 3.189                                  |
| 5         | 0.2            | 0.25                   | 23.95                         | 3.204                                  |
| 6         | 0.2            | 0.2                    | 10.77                         | 3.549                                  |
| 7         | 0.2            | 0.2006                 | 14.42                         | 3.31                                   |
| 8         | 0.2            | 0.2                    | 44.70                         | 3.337                                  |
| 9         | 0.1667         | 0.1734                 | 13.19                         | 3.245                                  |
| 10        | 0.2            | 0.2043                 | 11.97                         | 3.335                                  |
| 11        | 0.1667         | 0.1667                 | 11.98                         | 3.671                                  |
| 12        | 0.2            | 0.25                   | 15.13                         | 4.333                                  |
| 13        | 0.2            | 0.2                    | 42.065                        | 4.093                                  |
| 14        | 0.1667         | 0.1709                 | 3.27                          | 3.347                                  |
| 15        | 0.1667         | 0.1724                 | 5.03                          | 3.738                                  |
| 16        | 0.2            | 0.2039                 | 31.88                         | 3.73                                   |
| 17        | 0.2            | 0.2023                 | 9.41                          | 3.393                                  |
| 18        | 0.25           | 0.2597                 | 97.22                         | 3.574                                  |
| 19        | 0.2            | 0.2501                 | 11.25                         | 3.307                                  |
| 20        | 0.1667         | 0.1677                 | 6.56                          | 3.519                                  |

Compare with CPLEX, the infeasibility does not affect the new algorithm. The reason why the new algorithm is not affected by the infeasibility is that the new algorithm uses random number generators for the coefficients of the possibility distributions. A further research is required for clarifying the reason why infeasibility issue does not affect the new algorithm.

5.2 Numerical experiments

The numerical experiments in this section are performed with a computer with 6 core Intel i7-8700 CPUs and 16 GB RAM. The algorithm is implemented in C++ with Microsoft Visual Studio 2015.

First, we generate 100 instances of symmetric matrices for each size of 50, 100, 150, 200, 250, 300, 350, 400, and 450 StQPs by using random number generator. Secondly, we measure the computational time when the first minimizer of each instance has been found by running the new algorithm. Finally, we calculate the average computational time for each size. The average computation times for each size of StQPs are shown in Table 3. The data is visualized in Figure 1. Figure 1 indicates that the average computational time for calculating the first minimizer of StQP in in $\mathcal{O}(n)$. 
Table 3: The computational time of finding the first local minimizer

| Size (n) | Number of instances | Computational time in Sec. |   |   |
|---------|--------------------|---------------------------|---|---|
|         |                    | minimum | maximum | average |
| 50      | 100                | 0.004   | 0.507   | 0.046   |
| 100     | 100                | 0.007   | 9.897   | 0.545   |
| 150     | 100                | 0.04    | 70.817  | 4.058   |
| 200     | 100                | 0.046   | 384.428 | 21.430  |
| 250     | 100                | 0.14    | 1262.58 | 68.322  |
| 300     | 100                | 0.133   | 3270.97 | 87.338  |
| 350     | 100                | 0.149   | 4693.98 | 166.255 |
| 400     | 100                | 0.331   | 8658.38 | 352.36  |
| 450     | 100                | 0.331   | 11913.4 | 411.054 |

Fig. 2: Average computational time for finding the first minimizer

6 Conclusion

We propose a new algorithm for finding approximate solutions for StQP problems. This paper proves that (1) the approximate solution of the new algorithm is a $\epsilon$-critical point and (2) The approximate solution satisfies $\epsilon \delta$ condition. Furthermore, we show that when the approximate solution is local minimum, then the transformed Hessian matrix is positive semi-definite.

Table 1 and Table 2 indicate that the new algorithm has better performance than CPLEX in most cases. Note that CPLEX uses the Branch and Bound technique that takes more time when it deals with the infeasibility issue (J. W. Chinneck, 2008). On the other hand, our algorithm uses normalized possibility distributions that are generated with randomized algorithm technique which is capable to circumvent the infeasibility and achieve better performances.
Future research will involve the application of the new algorithm to solve practical problems such as portfolio optimization problems and the analysis of the new algorithm from the perspective of computational complexity theory.

Acknowledgements
The author would like to thank IBM to provide the free of charge CPLEX optimization software for the comparison.

Declarations:
Ethical approval - This research did not contain any studies involving animal or human participants, nor did it take place on any private or protected areas. No specific permissions were required for corresponding locations.
Funding details (In case of Funding) - Not applicable.
Conflict of interest - Not applicable.
Informed Consent - Informed consent was obtained from all individual participants included in the study.
Authorship contributions - L. Gao conceived of the presented idea and developed the algorithm and performed the computations.

References

N. Agarwal, Z. Zhu, B. Bullins, E. Hazan and T. Ma (2016) Finding approximation local minima for nonconvex optimization in linear time. arXiv:1611.01146v2.

A. Ahmadi and J. Zhang (2020) On the Complexity of Finding a Local Minimizer of a Quadratic Function over a Polytope. Optimization Online 2020 at arXiv:2008.05558v2.

I. M. Bomze and E. de Klerk (2002) Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. J. Global Optim. 24(2), 163-185

I. M. Bomze (1998) On standard quadratic optimization problems. J. Global Optim. 13, 369-387

I. M. Bomze (2002) Branch-and-Bound approaches to standard quadratic optimization problems. J. Global Optim. 22, 17-37

J. W. Chinneck (2008) FEASIBILITY AND INFEASIBILITY IN OPTIMIZATION: Algorithms and Computational Methods. Springer

E. De Klerk (2008) The complexity of optimizing over a simplex, hypercube or sphere: a short survey. Central European Journal of Operations Research, 16, 111-125.

E. De Klerk, D. den Hertog, and G. Elabwabi (2008) On the complexity of optimization over the standard simplex. European Journal of Operational Research 191, 773-785.

J.G. Dijkman, H. van Haeringen, and S.J. DE Lange (1983) Fuzzy numbers. Journal of mathematical analysis and applications, 92, 301-341.

L. Gao-1 (2020) An approximation algorithm for solving standard quadratic optimization problems. Journal of Intelligent and Fuzzy Systems(2020), vol. 39, no. 3, pp. 4383-4392

L. Gao-2 (2020) An algorithm for finding approximate Nash equilibria in bi-matrix games. Soft Computing (2020), https://doi.org/10.1007/s00500-
L. Gao (1999) The fuzzy arithmetic mean. Fuzzy Sets and Systems, 107, 335-348.

L. Gao and H. Kawarada (1995) Application of fuzzy average to curve and surface fitting. Proc. FUZZ-IEEE/IFES’95.

IBM (2020) IBM ILOG. CPLEX Optimization Studio. Accesses October 21, 2020, http://www.cplex.com.

A. Kaufmann and M. M. Gupta (1998) Fuzzy Mathematical Models in Engineering and Management Science, Elsevier, Amsterdam.

S.T. Liu (2007) Solving quadratic programming with fuzzy parameters based on extension principle. Proceeding of IEEE International Conference Fuzzy Systems, FUZZ-IEEE 2007, 1-5.

A. Momot and M. Momot (2009) Fuzzy weighted averaging using criterion function minimization. ICMMI 2009: 273-280.

I. Nowak (1999) A new semidefinite programming bound for indefinite quadratic forms over a simplex. J. Global Optim. 14, 357-364.

L. A. Zadeh (1978) Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems, 1, 3-28.

H.-J. Zimmermann (2001) Fuzzy Set Theory and Its Application. 4th Edition. Kluwer Academic Publishers