A Smoothing Newton Algorithm for Tensor Complementarity Problem Based on the Modulus-Based Reformulation

XIE Xuezhi (谢学智), GU Weizhe (谷伟哲)
(School of Mathematics, Tianjin University, Tianjin 300350, China)

Abstract: In the recent past few years, tensor, as the extension of the matrix, is well studied. Among many tensor related problems, the tensor complementarity problem (TCP) is a useful area for many researchers to study, and many methods to solve TCP have been proposed. In this article, under the condition of a strong $P$-tensor, we propose a smoothing Newton method to solve TCP based on the modulus-based reformulation, and we prove that the smoothing Newton method is globally convergent. Some numerical examples are given to demonstrate the efficiency of the smoothing Newton algorithm.

Key words: Tensor complementarity problem; Smoothing Newton method; Modulus-based reformulation

1. Introduction

The classical complementarity problem is to find a vector $x \in \mathbb{R}^n$ such that

$$x \geq 0, \quad F(x) + q \geq 0, \quad \text{and} \quad x^T(F(x) + q) = 0,$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a function and $q \in \mathbb{R}^n$. If $F = Ax$ is a linear function where $A \in \mathbb{R}^{n \times n}$, then (1.1) becomes

$$x \geq 0, \quad Ax + q \geq 0, \quad \text{and} \quad x^T(Ax + q) = 0,$$

which is called a linear complementarity problem, denoted by LCP($A; q$).\[^1\] If $F$ is a nonlinear mapping, then (1.1) is called a nonlinear complementarity problem, denoted by NCP.\[^2-4\] In this paper, we consider $F(x) = \mathcal{A}x^{m-1}$, where $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is a tensor, and $\mathcal{A}x^{m-1}$ is a vector defined by

$$(\mathcal{A}x^{m-1})_i = \sum_{i_2, \ldots, i_m=1}^n a_{i_1 i_2 \ldots i_m} x_{i_2} \ldots x_{i_m} \quad i = 1, 2, \ldots, n,$$

with $\mathbb{R}^{[m,n]}$ being the set of all real $m$-th order $n$-dimensional tensor. In this case, (1.1) becomes a special NCP such that

$$x \geq 0, \quad \mathcal{A}x^{m-1} + q \geq 0, \quad \text{and} \quad x^T(\mathcal{A}x^{m-1} + q) = 0,$$

Received date: 2019-12-20
Foundation item: Supported by the National Natural Science Foundation of China(11871051)
Biography: XIE Xuezhi, male, Han, Hebei, major in optimization theory and algorithm.
which is called a tensor complementarity problem, denoted by \( TCP(\mathcal{A}, q) \). TCP as a generalization of LCP and NCP was firstly introduced by SONG and QI in [5], and get further studied in [6-10], such as XIE, LI and XU [8] proposed an iterative method to solve a lower-dimensional tensor complementarity problem. Because of the good properties of structural tensors, many algorithms for solving the tensor complementarity problem are obtained in recent years.

Recently, LIU, LI and VONG [11] proposed a nonsmooth Newton method to solve TCP based on a modulus-based reformulation. Before this, many good algorithms about modulus-based have been established for LCP. On the one hand, BAI [12-14] proposed a modulus-based matrix splitting and multisplitting iteration method for solving LCP, and presented convergence analysis for the established methods. Based on the splitting of system matrix \( A \), the LCP is transformed into an implicit fixed-point equation, and a kind of matrix splitting iterative method based on module is established to solve large sparse LCPs. [15] On the other hand, [16] proposed a modulus-based nonsmooth Newton’s method for solving TCP. With the help of the methods for LCP, in this paper we propose a smoothing Newton method to solve the TCP based on a strong P-tensor. Firstly, we choose the strong P-tensor because a tensor complementarity problem has the property of global uniqueness and solvability (GUS-property) if and only if the tensor involved in the concerned problem is a strong P-tensor. [10] Then the smoothing Newton method has been widely used in NCP. [4,17-18] In order to use this method to solve the problem, we define a smooth approximation function. Under the above conditions of nonsingular, this method is well defined. It can be suitable for tensor complementarity problem based on the modulus-based reformulation and obtain good convergence.

The rest of this paper is organized as follows. In Section 2, we give some notations, definitions and preliminary results which are useful in the sequent analysis. In Section 3, under some sufficient conditions, we establish a smoothing Newton method to solve the TCP based on the modulus-based reformulation, and show the convergence of the method. In Section 4, we give some numerical examples to show the efficiency of the proposed algorithm. Conclusions are given in Section 5.

2. Preliminaries

In this section, we give some necessary notations, and review five definitions and four propositions, which will be used in the sequent analysis.

In this paper, for any differentiable function \( F : \mathbb{R}^n \to \mathbb{R} \), \( F'(x) \) denotes the Jacobian matrix of mapping \( F \) at \( x \). The norm \( \| \cdot \| \) denotes the Euclidean norm. For two matrices \( A = (a_{ij}), B = (b_{ij}) \in \mathbb{R}^{m \times n} \), we write \( A \succeq B(A > B) \) iff \( a_{ij} \geq b_{ij} \) (\( a_{ij} > b_{ij} \)) holds for all \( 1 \leq i \leq m \) and \( 1 \leq j \leq n \), and denote \( \mathbb{R}^n_+ := \{ x \in \mathbb{R}^n : x \geq 0 \} \) and \( \mathbb{R}^n_{++} := \{ x \in \mathbb{R}^n : x > 0 \} \).

A \( m \)-order \( n \)-dimensional tensor is denoted by \( \mathcal{A} = (a_{i_1i_2\ldots i_m})_{1\leq i_1\ldots i_m \leq n} \). Denote the set of all real \( m \)-th order \( n \)-dimensional tensor by \( \mathbb{R}^{[m,n]} \). Compare with \( \mathcal{A}x^{m-1} \in \mathbb{R}^n \), and \( \mathcal{A}x^{m-2} \in \mathbb{R}^{n \times n} \) can be expressed as

\[
(\mathcal{A}x^{m-2})_{ij} = \sum_{i_3\ldots i_m=1}^{n} a_{i_1i_2\ldots i_m} x_{i_3} \ldots x_{i_m} \quad i, j = 1, 2, \ldots, n.
\]

When \( m = 2 \), \( \mathbb{R}^{[m,n]} \) is the set of all real matrices of \( n \times n \), which is usually denoted by \( \mathbb{R}^{n \times n} \). For any \( \mathcal{A} = (a_{i_1i_2\ldots i_m}) \in \mathbb{R}^{[m,n]} \) is called a symmetric tensor, iff the entries \( a_{i_1i_2\ldots i_m} \)
are invariant under any permutation of their indices. Denote the set of all real \( m \)-th order \( n \)-dimensional symmetric tensor by \( S^{[m,n]} \). For any \( \mathcal{A} = (a_{i_1i_2\cdots i_m}) \in \mathbb{R}^{[m,n]} \), the diagonal entry of \( \mathcal{A} \) is \( a_{i_1i_2\cdots i_m} \) with \( i_1 = i_2 = \cdots = i_m \); others are called off-diagonal entries. \( \mathcal{A} \) is called a diagonal tensor, iff all its off-diagonal entries are zero (we use \( \mathcal{I} \) to denote the unit tensor, which is a diagonal tensor with each diagonal entry being 1). In particular, \( \mathcal{I} \) represents the unit matrix.

Then, we give some definitions and propositions of the strong \( P(P_0) \)-tensor.

**Definition 2.1** [19] A tensor \( \mathcal{A} \in \mathbb{R}^{[m,n]} \) is said to be a strong \( P(P_0) \)-tensor, iff \( \mathcal{A} x^{m-1} + q \) is a \( P(P_0) \)-function.

**Definition 2.2** [19] A mapping \( F : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^n \) is said to be \( P(P_0) \) function on \( S \), if

\[
\max_{i,j} (x_i - y_i)(F_i(x) - F_i(y)) > (\geq) 0 \quad \text{for all} \quad x,y \in S \quad \text{with} \quad x \neq y.
\]

**Definition 2.3** [19] A matrix \( M \in \mathbb{R}^{n \times n} \) is said to be a \( P(P_0) \) matrix, iff for each \( x \in \mathbb{R}^n \setminus \{0\} \), there exists an index \( i \in [n] \) such that \( x_i \neq 0 \) and \( x_i (Mx)_i > (\geq) 0 \).

**Proposition 2.1** [19] Let \( \mathcal{A} \in S^{[m,n]} \), \( \mathcal{A} x^{m-1} + q \) is a \( P(P_0) \)-function, iff its Jacobian matrix is a \( P(P_0) \)-matrix.

**Proposition 2.2** [20] If \( A \in \mathbb{R}^{n \times n} \) is a \( P \)-matrix, \( B \in \mathbb{R}^{n \times n} \) is a positive diagonal matrix with \( 0 < B < I \), then \( I - B + AB \) is a \( P \)-matrix.

**Proposition 2.3** [10] Suppose that \( \mathcal{A} \in \mathbb{R}^{[m,n]} \) is a strong \( P(P_0) \)-tensor, then TCP(\( \mathcal{A} \), \( q \)) (1.2) has the GUS-property.

The following theorem detects the relationship between a modulus-based equation and the TCP(\( \mathcal{A} \), \( q \))(1.2). The proof can be obtained from [11].

**Theorem 2.1** [11] A vector \(|x|x + x \in \mathbb{R}^n \) is a solution of the TCP(\( \mathcal{A} \), \( q \))(1.2) if and only if the vector \( x \in \mathbb{R}^n \) is a solution of the following system of equations:

\[
|x| - x = \mathcal{A}(|x| + x)^{m-1} + q,
\]

where \( |x| = (|x_1|, |x_2|, \ldots, |x_n|)^T \in \mathbb{R}^n \).

Thus, to search the solution of TCP(\( \mathcal{A} \), \( q \))(1.2) can be equivalent solving the modulus-based equation (2.1).

**Definition 2.4** It is well-known that

\[
\max \{x, 0\} = \frac{|x| + x}{2}.
\]

From the equation, we can see \( |x| = 2 \max \{x, 0\} - x \). We define a function \( \psi(\mu, x) : \mathbb{R}^2 \rightarrow \mathbb{R} \) satisfying \( \max \{x, 0\} = \lim_{\mu \rightarrow 0^+} \psi(\mu, x) \). The function is as follow:

\[
\psi(\mu, x) = \mu \ln(1 + e^x).
\]

The function \( \psi(\mu, x) \) is infinitely differentiable function of \( x \), which is a smooth function on \( \mathbb{R} \). So we define

\[
|x| = 2 \max \{x, 0\} - x = \lim_{\mu \rightarrow 0^+} 2 \psi(\mu, x) - x
\]

**Definition 2.5** Through Definition 2.4, in order to solve the equation in \( \mathbb{R}^n \), we define \( \psi : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n \) by

\[
2\Psi(\mu, x) - x := 2 \begin{pmatrix} \psi(\mu, x_1) \\ \vdots \\ \psi(\mu, x_n) \end{pmatrix} - x,
\]
where \( \mu > 0, x \in \mathbb{R}^n \).

From the above definitions, to solve the equation
\[
F(x) = x - |x| + \mathcal{A}(|x| + x)^{m-1} + q = 0, F(x) \in \mathbb{R}^n,
\]
is equivalent to solve the equation
\[
f(\mu, x) = 0, \mu \to 0, \tag{2.2}
\]
where the function \( f(\mu, x) : \mathbb{R}^{n+1} \to \mathbb{R}^n \) is defined by
\[
f(\mu, x) := 2x - 2\mathcal{A}(\mu, x) + \mathcal{A}(2\mathcal{A}(\mu, x))^{m-1} + q. \tag{2.3}
\]

**Proposition 2.4**

(i) We introduce a merit function \( H(x) = \frac{1}{2}||f(\mu, x)||^2 \). When \( \mu \) is a sufficient small positive number, we have that \( H(x) = 0 \) at \( x \) is equivalent to \( f(\mu, x) = 0 \) at \( x \).

(ii) Since \( f(\mu, x) \) is continuously differentiable at \( x \) with \( \mu > 0 \), and its Jacobian matrix
\[
f'_x(\mu, x) := 2I - 2\mathcal{A}'(\mu, x) + 2(m - 1)\mathcal{A}'(2\mathcal{A}(\mu, x))^{m-2}\mathcal{A}'(\mu, x)
\]
where \( f'_x(\mu, x) \in \mathbb{R}^{n \times n} \) is a matrix and the specific forms of the diagonal matrix \( \mathcal{A}' \in \mathbb{R}^{n \times n} \) as following:
\[
\mathcal{A}'(\mu, x) := \begin{pmatrix}
\frac{e^{\frac{x}{1+e^{\frac{1}{\mu}}}}}{} & 0 & \cdots & 0 \\
0 & \frac{e^{\frac{x}{1+e^{\frac{2}{\mu}}}}}{} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \frac{e^{\frac{x}{1+e^{\frac{n}{\mu}}}}}{}
\end{pmatrix}.
\tag{2.4}
\]

(iii) Since \( H(x) = \frac{1}{2}||f(\mu, x)||^2 \), we can see \( \nabla H(x) = f'_x(\mu, x)^T f(\mu, x) \).

It is obvious that \( x \) is a solution of \( f(\mu, x) = 0 \Leftrightarrow x \) is a solution of the modulus-based equation \( F(x) = x - |x| + \mathcal{A}(|x| + x)^{m-1} + q = 0 \Leftrightarrow |x| + x \) is a solution of \( TCP(\mathcal{A}, q)(1.2) \).

Next, the main work is to solve \( f(\mu, x) = 0 \). Since the function \( f(\mu, x) \) is continuously differentiable for any \( x \in \mathbb{R}^n \) with \( \mu > 0 \), we can apply a smoothing Newton method to solve the system of smooth equations \( f(\mu, x) = 0 \) at each iteration and let \( \mu \) be a sufficient small positive number. So the solution of the problem (2.1) can be found. The smoothing Newton’s method is a classic method for solving the nonlinear equation \( G(x) = 0 \) with \( x^{(i+1)} = x^{(i)} - [G'(x^{(i)})]^{-1}G(x^{(i)}) \),

where \( G : \mathbb{R}^n \to \mathbb{R}^n \) is a continuously differentiable function. This is the fundamental idea of the algorithm. In the next section, we will introduce the algorithm in detail.

3. A Smoothing Newton Algorithm

In this section, in order to solve the equation (2.1) we apply a smoothing Newton algorithm to solve the \( f(\mu, x) = 0 \). Without affecting the final experimental results, we fix \( \mu \) to a sufficient small positive number, and we use the following assumption to ensure that the algorithm is well-defined and globally convergent.

**Assumption 3.1** Assume \( \mathcal{A} \in \mathbb{R}^{[m,n]} \) is a strong \( P \) tensor, \( q \in \mathbb{R}^n \), and the function \( \mathcal{A}x^{m-1} + q \) is a \( P \) function.

**Remark 3.1** If the function \( \mathcal{A}x^{m-1} + q \) is a \( P \) function, then the corresponding \((m - 1)\mathcal{A}x^{m-2} \) is a \( P \) matrix, which is a good condition for the smoothing Newton algorithm.

Now, we give the algorithmic framework.
Algorithm 3.1 (A Modulus-Based Smoothing Newton Algorithm)

Step 0 Choose $\sigma \in (0,1)$. $\mu > 0$ is a sufficient small positive number, and $x^0 \in \mathbb{R}^n$. Set $k := 0$.

Step 1 If $H(x^k) = 0$, stop. Otherwise, go to Step 2.

Step 2 Compute $\Delta x^k \in \mathbb{R}^n$ by

$$f (\mu, x^k) + f'_s (\mu, x^k) \Delta x^k = 0.$$ 

Step 3 Let $i_k$ be the minimum of the values $0, 1, 2, \cdots$ such that

$$H(x^k + 2^{-i_k} \Delta x^k) \leq H(x^k) + \sigma 2^{-i_k} \nabla H(x^k)^T \Delta x^k.$$ 

Step 4 Set $x^{k+1} := x^k + 2^{-i_k} \Delta x^k$ and $k := k + 1$. Go to Step 1.

Under the assumptions, we discuss Algorithm 3.1 is well defined and analyze the convergence of Algorithm 3.1. Firstly we discuss Algorithm 3.1 is well-defined.

Proposition 3.1 If the tensor $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is a strong P-tensor, the matrix $f'_s (\mu, x)$ obtained from above is nonsingular.

Proof Suppose that $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is a strong P-tensor, from Definition 2.1 and Proposition 2.1, we know that $2(m-1)\mathcal{A}(2\Psi(\mu, x))^{m-2}\Psi'_2 (\mu, x)$ is a P-matrix. Letting $J(\mu, x) = \mathcal{A}(2\Psi(\mu, x))^{m-2}$, from Definition 2.3, we know that there exists an index $i \in [n]$ such that $x \neq 0$ and $x_i (J(\mu, x)\Psi'_2 (\mu, x)x)_i > 0$.

Notice that the diagonal matrix $\Psi'_2 (\mu, x)$ (see (2.4)) with $\mu > 0$ satisfies

$$0 < \frac{e^{\frac{\sigma}{\mu}}}{1 + e^{\frac{\sigma}{\mu}}} < 1, \quad i = 1, 2, \cdots n.$$ 

So $\Psi'_2 (\mu, x)$ satisfies $0 < \Psi'_2 (\mu, x) < I$. From Proposition 2.2, it is well-known that $f'_s (\mu, x) = 2(I - \Psi'_2 (\mu, x)) + 2(m-1)J(\mu, x)\Psi'_2 (\mu, x)$ is a P-matrix. So the matrix $f'_s (\mu, x)$ is nonsingular.

Since $f'_s (\mu, x)$ is nonsingular, the equation (3.1) of Algorithm 3.1 is well-defined. Next, we discuss the convergence of Algorithm 3.1. For this purpose, we need to establish the following two lemmas.

Lemma 3.1 [17] Under the condition of Assumption 3.1 and $\mu > 0$, for any $\{x^k\}$, when $\|x^k\| \to \infty$, we have

$$\lim_{k \to \infty} H(x^k) = +\infty.$$ 

Lemma 3.2 [17] Under the condition of Assumption 3.1, the infinite sequence $\{x^k\}$ generated by Algorithm 3.1, the sequences $\{H(x^k)\}$ is monotonically decreasing and

$$\lim_{k \to \infty} H(x^k) = 0.$$ 

Lastly, we are in a position to show the convergence of Algorithm 3.1.

Theorem 3.1 Suppose that Lemmas 3.1-3.2 and Assumption 3.1 are satisfied. Any accumulation point of $\{x^k\}$ generated by Algorithm 3.1 is a solution of $H(x) = 0$.

Proof By Lemma 3.2, the sequence $\{H(x^k)\}$ is monotonically decreasing, hence it is bounded. With the coercive property of $H(x)$ in Lemma 3.1, the sequence $\{x^k\}$ must be bounded. Thus, the sequence $\{x^k\}$ has a convergent subsequence which we write without loss of generality as $\{x^*_k\}$. Let $x^*$ be the limiting point of the sequence. By Lemma 3.2, we obtain that $H(x^*) = 0$. Then $x^*$ is a solution of the equation (2.1). This completes the proof.
4. Numerical Examples

In this section, we give the numerical experiments to operate Algorithm 3.1. Firstly, we introduce some notations in the following tables. IT denotes the number of iterations, CPU denotes the running time (in seconds) required to complete the iteration process, \(x^0\) denotes the initial point of the algorithm, \(x^k\) denotes the points satisfying termination conditions, and \(s^k = \|x^k\| + x^k\) is the approximate solution to the TCP(\(\mathcal{A}\), \(q\))(1.2). The parameters used in the algorithm are chosen as \(\sigma = 0.3\). And we choose \(H(x^k) \leq 10^{-9}\) as the terminated condition.

After observing the modulus-based equation (2.1), we decide to choose different \(\mu\) and randomly generated initial vector \(x^0\), \(q\) to test the algorithm, and choose Example 3.10 and Example 3.30 in [11] as the system tensor.

Example 4.1 Firstly, within the calculation range allowed by Matlab, we take \(\mu_1 = 0.01\) and \(\mu_2 = 0.005\) as smooth parameters respectively, fix the \(\mathcal{A} \in \mathbb{R}^{3,2}\) (see Example 3.30 in [11]) with \(a_{111} = a_{222} = a_{121} = a_{212} = 1\), \(a_{122} = a_{211} = -1\), and other entries are zeros, and choose randomly generated initial vector \(x^0\), \(q\).

After our experiment of Example 4.1, we get eight pieces of data with \(\mu = 0.01\). From the first four data, we can see the solutions are same as Tab. 4 in [11]. As for the last four data, the algorithm can get different solutions in an acceptable accuracy with randomly generated \(q\) and \(x^0\). We also make \(\mu = 0.005\), and the results are same as \(\mu = 0.01\). All the numerical results are shown in the Tab. 4.1.

Tab. 4.1 Numerical results for Example 4.1 with \(\mu_1 = 0.01\) and \(\mu_2 = 0.005\)

<table>
<thead>
<tr>
<th>(\mu)</th>
<th>(x^0)</th>
<th>(q)</th>
<th>IT</th>
<th>CPU</th>
<th>(x^k)</th>
<th>(s^k)</th>
<th>(H(x^k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_1)</td>
<td>((1.0668, 0.0593)^T)</td>
<td>((1, 2)^T)</td>
<td>19</td>
<td>0.1529</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((-0.1461, 0.2481)^T)</td>
<td>((1, 2)^T)</td>
<td>17</td>
<td>0.1245</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((-0.9945, -1.3862)^T)</td>
<td>((1, -2)^T)</td>
<td>24</td>
<td>0.2744</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((0.0967, 1.2159)^T)</td>
<td>((1, -2)^T)</td>
<td>24</td>
<td>0.1969</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((-0.6691, 0.8580)^T)</td>
<td>((0.7143, 1.6236)^T)</td>
<td>19</td>
<td>0.1637</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((1.9908, -1.2025)^T)</td>
<td>((1.2902, 0.6686)^T)</td>
<td>20</td>
<td>0.1610</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((-1.6041, -0.2573)^T)</td>
<td>((-0.0198, -0.1567)^T)</td>
<td>18</td>
<td>0.1391</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((-2.1707, -0.0592)^T)</td>
<td>((0.2193, -0.9219)^T)</td>
<td>17</td>
<td>0.1279</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\mu_2)</td>
<td>((0.8252, 0.2308)^T)</td>
<td>((1, 2)^T)</td>
<td>19</td>
<td>0.1659</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((0.5415, 0.9321)^T)</td>
<td>((1, 2)^T)</td>
<td>18</td>
<td>0.1489</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((1.5357, 0.4344)^T)</td>
<td>((1, -2)^T)</td>
<td>29</td>
<td>0.3586</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((-0.1721, -0.3360)^T)</td>
<td>((1, -2)^T)</td>
<td>20</td>
<td>0.1690</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((1.2744, 0.6385)^T)</td>
<td>((-1.9171, 0.4699)^T)</td>
<td>24</td>
<td>0.1956</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((-0.9094, -2.3050)^T)</td>
<td>((1.3808, 1.3198)^T)</td>
<td>17</td>
<td>0.2006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((0.0203, -0.4960)^T)</td>
<td>((1.7887, 0.3908)^T)</td>
<td>15</td>
<td>0.2134</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>((0.3757, -1.3454)^T)</td>
<td>((-0.2086, 0.7559)^T)</td>
<td>16</td>
<td>0.1228</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 4.2 In this example we choose the system tensor \(\mathcal{A} \in \mathbb{R}^{3,2}\) (see Example 3.10 in [11]) with \(a_{111} = a_{222} = a_{122} = a_{211} = 1\), \(a_{122} = a_{211} = -1\), and other entries are zeros. \(\mu_1\) and \(\mu_2\) are same as Example 4.1. \(x^0\) and \(q\) are randomly generated.

In Example 4.2, we can see the condition is same as Example 4.1, and no matter how we choose the initial condition, the algorithm works well in an acceptable accuracy. The results are shown in Tab. 4.2.

In this section, all tests of the examples were done in Matlab R2014b and Tensor Toolbox 2.6. The codes were done on a DELL desktop with Inter(R) Core(TM) i5-5200U CPU 2.20
GHz and 4GB RAM running on Windows 7 system. After the tests for the smoothing Newton algorithm 3.1 with different parameter $\mu$, randomly generated initial points $x^0$ and $q$, we can see that Algorithm 3.1 is efficient and stable, and the solution $x^k$ can solve the equation (2.1). From Theorem 2.1, $|x^k|+x^k$ is a solution of the TCP($A^T$, $q$) (1.2).

Tab. 4.2 Numerical results for Example 4.2 with $\mu_1 = 0.01$ and $\mu_2 = 0.005$

<table>
<thead>
<tr>
<th>$\mu$</th>
<th>$x^0$</th>
<th>$q$</th>
<th>IT</th>
<th>CPU</th>
<th>$x^k$</th>
<th>$s^k$</th>
<th>$H(x^k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$</td>
<td>$(-1.0811, -1.1245)^T$</td>
<td>$(1, 2)^T$</td>
<td>15</td>
<td>0.1120</td>
<td>$(-0.5, -1)^T$</td>
<td>$(0.0)^T$</td>
<td>6.58e-10</td>
</tr>
<tr>
<td></td>
<td>$(0.3296, 0.5965)^T$</td>
<td>$(1, 2)^T$</td>
<td>18</td>
<td>0.1477</td>
<td>$(-0.5, -1)^T$</td>
<td>$(0.0)^T$</td>
<td>3.06e-10</td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>$(-0.0477, 0.3793)^T$</td>
<td>$(1, -2)^T$</td>
<td>28</td>
<td>0.2059</td>
<td>$(0.2401, 0.6114)^T$</td>
<td>$(0.4802, 1.2229)^T$</td>
<td>6.14e-10</td>
</tr>
<tr>
<td></td>
<td>$(-0.6934, -1.9314)^T$</td>
<td>$(1, -2)^T$</td>
<td>26</td>
<td>0.2155</td>
<td>$(0.2401, 0.6114)^T$</td>
<td>$(0.4802, 1.2229)^T$</td>
<td>4.68e-10</td>
</tr>
<tr>
<td></td>
<td>$(-1.2276, -0.6699)^T$</td>
<td>$(1.8625, 1.1069)^T$</td>
<td>14</td>
<td>0.1357</td>
<td>$(-0.9313, -0.5534)^T$</td>
<td>$(0.0)^T$</td>
<td>7.55e-10</td>
</tr>
<tr>
<td></td>
<td>$(-0.1072, -0.9771)^T$</td>
<td>$(-0.7826, -0.7673)^T$</td>
<td>20</td>
<td>0.1372</td>
<td>$(0.4412, 0.4391)^T$</td>
<td>$(0.8825, 0.8781)^T$</td>
<td>3.52e-10</td>
</tr>
<tr>
<td></td>
<td>$(-0.6368, -1.0026)^T$</td>
<td>$(0.2279, 0.6856)^T$</td>
<td>16</td>
<td>0.1908</td>
<td>$(-0.1139, -0.3428)^T$</td>
<td>$(0.0)^T$</td>
<td>3.30e-10</td>
</tr>
<tr>
<td></td>
<td>$(-0.8382, 0.2573)^T$</td>
<td>$(-0.9640, -2.3792)^T$</td>
<td>17</td>
<td>0.1182</td>
<td>$(0.5820, 0.7181)^T$</td>
<td>$(1.1640, 1.4361)^T$</td>
<td>4.20e-10</td>
</tr>
</tbody>
</table>

5. Conclusion

In this article, we proposed a smoothing Newton method for solving the TCP($A^T$, $q$)(1.2), and convert the TCP($A^T$, $q$)(1.2) to the modulus-based equation (2.1). LIU, LI and VONG proposed a nonsmooth Newton method to solve the modulus-based equation (2.1). Now we define a smooth approximation function and propose a smoothing Newton method to solve the modulus-based equation (2.1). Then we test some examples for Algorithm 3.1, and the results showed the algorithm is efficient. However, it is well-known that many people restructure the modulus-based equation as an implicit fixed-point equation and use the fixed point iterative algorithms to solve the equation in [8-9]. But no one has established the fixed point iterative algorithms for the modulus-based equation (2.1) with tensor. So this is our work to do in the future.

References:


基于模量重构的张量互补问题的光滑牛顿算法

谢学智, 谷伟哲
(天津大学数学学院, 天津 300350)

摘要：近年来，张量作为矩阵的推广，得到了广泛的研究。在众多张量相关的问题中，张量互补问题(TCP)是许多学者研究的一个重要领域，人们提出了许多解决TCP的方法。本文在强P-张量张量和光滑逼近函数的基础上，提出一种基于模量的重构的TCP光滑牛顿算法，证明光滑牛顿方法是全局收敛的，数值算例验证了光滑牛顿算法的有效性。

关键词：张量互补问题; 光滑牛顿算法; 基于模量的重构