Review Article

Alternating Minimization Methods for Solving Multilinear Systems

Maolin Liang and Lifang Dai

School of Mathematics and Statistics, Tianshui Normal University, Tianshui 741001, China

Correspondence should be addressed to Maolin Liang; liangml2005@163.com

Received 7 November 2020; Revised 6 June 2021; Accepted 23 June 2021; Published 13 July 2021

Academic Editor: Rahib Abiyev

Copyright © 2021 Maolin Liang and Lifang Dai. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recent works on the multilinear system \( A^{m-1}x - b \) with an order-\( m \) and dimension-\( n \) tensor \( A \) and a vector \( b \) of dimension-\( n \) have been motivated by their applications in data mining, numerical PDEs, tensor complementary problems, and so on. In this paper, we propose an alternating minimization method for the solution of the system mentioned above and present several randomized versions of this algorithm in order to improve its performance. The provided numerical experiments show that our methods are feasible for any tensor \( A \) and outperform some existing ones in the same case.

1. Introduction

Many higher-dimensional problems in scientific computing come down to solve the following system of equations:

\[
\mathcal{A}^{m-1} = b, \tag{1}
\]

where \( \mathcal{A} = (a_{i_1\ldots i_m}) \) is a tensor in \( \mathbb{R}^{[m,n]} \), the set of all \( m \)th-order \( n \)-dimensional tensors over the real field, and the right-hand side \( b \) a vector in \( \mathbb{R}^n \), and \( \mathcal{A}^{m-1} \) is a vector in \( \mathbb{R}^n \), whose \( i \)th-entry is defined by

\[
\left( \mathcal{A}^{m-1} \right)_i = \sum_{i_2,\ldots,i_m} a_{i_2\ldots i_m} x_i x_{i_2} \cdots x_{i_m}. \tag{2}
\]

We call (1) multilinear system since it is obviously linear over \( x \) for each mode. Similar to linear systems case, we also call (1) a nonhomogeneous multilinear system if the right-hand side is nonzero; otherwise, it is a homogeneous one.

Many problems in scientific computing including data mining [1], the numerical solution of nonlinear PDEs [2–6], tensor complementary problems [7], and higher-dimensional statistics [8], can be characterized via the multilinear systems mentioned above. To be precise, by using the finite difference method, the discretization of nonlinear PDE,

\[
\begin{aligned}
\mathcal{A}^{m-2} \sum_{i=1}^d \frac{\partial^2 u}{\partial x_i^2} (x) &= -f(x), \quad \text{in } \Omega = [0,1]^m, \\
u(x) &= g(x), \quad \text{on } \partial \Omega,
\end{aligned} \tag{3}
\]

can be reformulated as the form of (1) [3]; see Section 4 for details. Particularly, if \( \mathcal{A} \) is a Z-tensor, it has been shown in [7] that the sparsest solution of the tensor complementarity problem,

\[
\begin{aligned}
\min & \quad \|x\|_0 \\
\text{s.t.} & \quad \mathcal{A}^{m-1} - b \geq 0, \quad x \geq 0, \quad x^T (\mathcal{A}^{m-1} - b) = 0,
\end{aligned} \tag{4}
\]

can be obtained by solving the following relaxed form:

\[
\begin{aligned}
\min & \quad \|x\|_1 \\
\text{s.t.} & \quad \mathcal{A}^{m-1} = b, \quad x \geq 0.
\end{aligned} \tag{5}
\]

In addition, using the “optimize then discretize” approach [8, 9], the differential equation,
\[
\begin{align*}
\max_{y, \lambda} \left\{ \frac{1}{2} \sigma(x, \lambda)^2 U^T x + \beta y \right\}, & \quad \text{on } \Omega, \\
\frac{1}{2} \sigma(x, \lambda)^2 U^T x + \beta y, & \quad \text{on } \partial \Omega,
\end{align*}
\]

with the elliptic operator \( L^N(x) = (1/2) \sigma(x, \lambda)^2 U^T x + \mu(x, \lambda) U^T x \). \( \Omega = (0, 1) \), \( \Gamma = [0, \infty) \) and compact metric space \( \Lambda \) can be discretized as the well-known 3rd-order Bellman equation, which has the form of (1); see Section 4 for details.

Recently, several works concerning the multilinear system (1) have been obtained when the coefficient tensor \( \mathcal{A} \) is a structured tensor. In [3], Ding and Wei have shown that this system has a unique positive solution when the coefficient tensor \( \mathcal{A} \) is a nonsingular M-tensor with a positive vector \( b \) and then generalized the classical iteration methods for linear systems and the Newton method for nonlinear equations to the multilinear case. Specifically, a homotopy method was also presented by Han [10] for finding the unique positive solution. Very recently, for symmetric M-tensor system, Li et al. [11] and Xie et al. [12], respectively, proposed splitting methods based on the ones in matrix theory and tensor methods for (1). Besides, an algorithm based on the fast Fourier transform was proposed by Xie et al. [13] for the tensor system (1) when the tensor \( \mathcal{A} \) is a circulant one.

In this paper, we continue to study the multilinear system (1) in the case that the contained tensor \( \mathcal{A} \) is a general one. In contrast to the structured case mentioned above, there is little work on (1) with a general tensor \( \mathcal{A} \) as far as we are informed. Generally, it is difficult to derive solvability conditions for a general multilinear system as in [3], so we alternatively study the least-squares problem associated with (1), that is,

\[
\min_{x} \| \mathcal{A}^T x - b \|. \tag{7}
\]

Throughout this paper, the symbol \( \| \cdot \| \) denotes the 2-norm of a vector or matrix. We call (7) the tensor least-square (TLS) problem corresponding to the multilinear system (1). The approaches presented in this paper are quite different from the ones mentioned above. By the definition of tensor-vector multiplication, we rewrite the TLS problem (7) as the following optimization problem (see Section 3 for details):

\[
\begin{align*}
\text{minimize} & \quad f(x_2, \ldots, x_m) = \frac{1}{2} \| \mathcal{A}^T x_2 \cdot \cdots \cdot x_m - b \|^2 \\
\text{s.t.} & \quad x_2 = \cdots = x_m \\
& \quad x_p \in \mathbb{R}^n, \quad p = 2, 3, \ldots, m.
\end{align*}
\]

This is a multiblock optimization problem with consensus constrained conditions. Alternating minimization iteration is popular for solving large-scale optimization problems; see, for example, [14, 15], which realizes the reduction of the problem from a nonlinear case to some linear subproblems via updating only one variable while the others are fixed. We apply the alternating minimization strategy to (8), followed by a projection concerning with the consensus constraints. We call it the alternating minimization method for solving the TLS problem, denoted by AMM (i.e., Algorithm 1). Although the alternating minimization scheme and its various variants are widely exploited, their convergence analyses are commonly difficult [16]. We leave the convergence analysis of the AMM an open problem. It should be pointed out that we proposed an alternating iterative method for (1) [17]; this kind of iterative method is ADMM-type one, so the method proposed here is very different from the ADMM-type method.

Notably, a revived interest in the design and analysis of algorithms for solving large-scale optimization problems is to add randomness to the update order of the multiblock updates, which is beneficial to alleviate the iteration complexity and improve the convergence; see, for example, [18–20] and the references therein. As variants of the AMM, we present two random iterations for solving the TLS problem (7); see Algorithms 2 and 3 for details. Besides, Algorithm 4 is another modified version of the AMM. At present, we have not found a similar iterative method for solving multilinear systems.

The contribution of this paper is as follows: (1) We considered the least-square problem of the multilinear system (1) by transforming it as an optimization problem. (2) We proposed an alternative iterative method for the optimization problem mentioned above, which is easy to carry out in a parallel version. (3) As modifications of the alternative iterative method, we apply the randomized technique to accelerate it, that is, Algorithms 2 and 3. The performed numerical results show that the randomized methods may have better performance under suitable initial iterative vectors.

The outline of this paper is as follows: in Section 2, we recall some notations and operations and then give a short review of the classical alternating minimization method. In Section 3, we establish some alternating minimization methods for the TLS problem (7), which are the variants of the classical one. In Section 4, some numerical experiments are performed to illustrate the efficiency and feasibility of the proposed algorithms. Finally, we conclude this paper.

2. Preliminaries

2.1. Notation and Operations. We will briefly describe notation for tensors and tensor operations used in this paper. Scalars are denoted by lowercase letters, for example, \( a \); vectors are denoted by boldface lowercase letters, for example, \( \mathbf{a} \); matrices are denoted by boldface capital letters, for example, \( \mathbf{I} \); identity matrix with proper size; tensors are denoted by calligraphic script letters, for example, \( \mathcal{A} \). The set of all \( N \)-order tensor is denoted by \( \mathbb{R}^{I_1 \times \cdots \times I_N} \); particularly, the set of all \( n \)-mode-dimensional tensors over real field is denoted by \( \mathbb{R}^{[n]} \); particularly, the set of all \( n \)-mode-dimensional tensors over real field is denoted by \( \mathbb{R}^{[m,n]} \).

We need the following concepts related to tensors; see, for example, [21] for details.

**Definition 1** (Tensor-vector multiplication). Let a tensor \( \mathcal{F} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} \) and a vector \( \mathbf{v} \in \mathbb{R}^{I_n} \); then, the \( n \)-mode (vector) product of the tensor \( \mathcal{F} \) with the vector \( \mathbf{v} \), denoted by \( \mathcal{F} \otimes \mathbf{v} \), is an \( I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \) tensor, elementwise,
\[(\mathcal{T}_2v)_{i_1\ldots i_n} = \sum_{i_n=1}^{I_n} t_{i_1\ldots i_n} v_{i_n}. \tag{9}\]

Particularly, if \(\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_k}\), and \(v \in \mathbb{R}^{I_2}\), then \(\mathcal{T}_2v = \sum_{i_2=1}^{I_2} \mathcal{T}(\cdot, i_2, \cdot) v_{i_2}\), where \(\mathcal{T}(\cdot, i_2, \cdot)\) stands for the \(i_2\)th lateral slice of this tensor. In other words, the 2-mode (vector) product between a tensor of order 3 and a vector with the appropriate size is a matrix, the linear combination of the lateral slices taking the associated elements of the vector as the coefficient.

By the definition of tensor-by-vector multiplication, one can rewrite (1) as the form of a least-squares problem:
\[
\min_{x} \|Ax - b\|^2.
\tag{10}\]

**Definition 2** (tensor-matrix multiplication). For the tensor \(\mathcal{T}\) in Definition 1 and the matrix \(A \in \mathbb{R}^{I_1 \times I_n}\), the \(n\)-mode (matrix) product, denoted by \(\mathcal{T} \times_n A\), is an \(I_1 \times I_2 \times \cdots \times I_N\) matrix, whose \((i_1, i_2)\)-entry is defined by
\[
(\mathcal{T} \times_n A)_{i_1, i_2} = \sum_{i_3=1}^{I_3} \cdots \sum_{i_N=1}^{I_N} t_{i_1, i_3, \ldots, i_N} a_{i_2, i_{N+1}}.
\tag{11}\]

**Definition 3** (matricization/unfolding). For the tensor \(\mathcal{T}\) in Definition 1, the mode-\(n\) matricization (unfolding) of the tensor \(\mathcal{T}\), denoted by \(T_{(n)}\), is an \(I_1 \times I_2 \times \cdots \times I_N\) matrix, whose \((i_1, i_2)\)-entry is defined by
\[
T_{(n)}(i_1, i_2) = t_{i_1, i_2, \ldots, i_N}, \quad j = 1, 2, \ldots, n.
\]

2.2. The Classical Alternating Minimization Method. Consider the following multiblock minimization problem:
\[
\begin{align*}
\text{minimize} & \quad F(x_1, x_2, \ldots, x_d) \\
\text{s.t.} & \quad x_p \in \mathcal{V}_p \subseteq \mathbb{R}^{I_p}, \quad p = 1, 2, \ldots, d, \\
\end{align*}
\tag{12}\]

where \(F: \mathbb{R}^{I_1} \times \mathbb{R}^{I_2} \times \cdots \times \mathbb{R}^{I_d} \to \mathbb{R}\) is a multivariate function. This problem can be solved by the so-called block nonlinear Gauss–Seidel (B-NGS) iteration [15, 22], namely,
\[
x_p^{(k+1)} = \arg\min_{x_p} F(x_1^{(k+1)}, \ldots, x_{p-1}^{(k+1)}, x_p, x_{p+1}^{(k)}, \ldots, x_d^{(k)}), \quad p = 1, 2, \ldots, d,
\tag{13}\]

which reduces to the ALS method if each subproblem embedded is a least-squares problem; see [16, 21, 23, 24] for some recent developments and applications.

3. Alternating Minimization Methods

This section is devoted to the derivation of the alternating minimization method (AMM) as well as some variants for the solution of the system of multilinear equation (1).

To facilitate implementation of the algorithms, we convert equivalently (10) into the following form:
\[
\frac{\partial f}{\partial x_i} = A_i^T(A_i x - b)_i, \quad i = 1, 2, \ldots, d,
\tag{14}\]

And, correspondingly, the optimization problem (8) can be written as
\[
\begin{align*}
\text{minimize} & \quad f(x_2, \ldots, x_m) = \frac{1}{2n} \|A_p x_p - b\|^2 \\
\text{s.t.} & \quad x_2 = \cdots = x_m,
\end{align*}
\tag{15}\]

where \(A_p = \mathcal{A}_2 x_2 \times_1 \cdots \times_{p-1} x_{p-1} \times_{p+1} \cdots \times_m x_m \in \mathbb{R}^{n \times n}\). For any \(p \in \{2, 3, \ldots, m\}\), updating one block \(x_p\) while fixing the remaining ones, we have
\[
x_p^{(k+1)} = \arg\min_{x_p} f(k+1)(x_p) = \frac{1}{2n} \|A_p x_p - b\|^2,
\tag{16}\]

where
\[
f(k+1)(x_p) = f(x_2^{(k+1)}, \ldots, x_{p-1}^{(k+1)}, x_p, x^{(k)}, \ldots, x_m^{(k)}),
\tag{17}\]

and \(A_p = \mathcal{A}_2 x_2 \times_1 \cdots \times_{p-1} x_{p-1} \times_{p+1} \cdots \times_m x_m\) with \(x^{(k)} = \sum_{p=2}^m x_p^{(k)} / (m-1)\).

The above procedure reformulates the alternating minimization method for solving the TLS problem (7), which can be precisely represented in the following algorithm:

We have the following comments in order.

Remark 1

(a) This algorithm enables that the large-scale TLS problem (7) is solved by several small-size subproblems, which may be computed in an efficient and perhaps highly parallel manner. In other words, Algorithm 1 is more convenient to be parallelized than the existing methods proposed in, for instance, [3, 11, 13].

(b) Obviously, every subproblem contained in Algorithm 1 is a linear least-squares problem, and the least-norm solution to the subproblem in (16) has closed form \(x_p^{(k+1)} = (A_p^T A_p)^{-1} A_p^T b\), where the subscript \(\dagger\) stands for the Moore–Penrose inverse of a matrix [25]. Additionally, for the large-scale linear (least-squares) subproblems contained in this algorithm, one can apply the iterative methods (e.g., LSQR, GMRES [25]) to them.

(c) The update in step 2 corresponds to the consensus restrictions in (15). In other words, \(x^{(k+1)}\) is the minimizer of the function \(g(y) = \sum_{p=2}^m \|y - x_p^{(k+1)}\|^2\).

(d) From Algorithm 1, one can observe that the main amount of calculations hinges on forming the matrices \(A_p^{(k+1)}\) in (17) for \(p = 2, 3, \ldots, m\). So, it is easy to know that the computation complexity of this algorithm is of order \(O(m^3)\). Notably, if the tensor \(\mathcal{A}\) in (1) has the CP decomposition with CP-rank \(r\), the computation complexity of the above algorithm decreases to \(O((m-1)^3mr)\).
Just as mentioned in the Introduction, a revived interest in the design and analysis of algorithms for solving large-scale optimization problems is to add randomness to the updating order of the multiblock variants. The above randomly updating technique can be utilized in Algorithm 1. In fact, updating $x_p^{(k+1)}$ for stochastically chosen $p \in \{2, 3, \ldots, m\}$, we obtain the first randomized AMM.

In Algorithm 2, we randomly update $x_p$ by (16) for each $p \in \{2, 3, \ldots, m\}$. In this procedure, all the block coordinates are updated after one loop. Furthermore, if we randomly update $x_p$ in a subset of $\{2, 3, \ldots, m\}$, we obtain the second randomized AMM.

It follows from Algorithm 3 that some variants in this algorithm may not be updated within one loop, which is the difference between the above two algorithms. In numerical experiments, we will see that the randomized algorithms presented above indeed have better performance than the original AMM, that is, Algorithm 1.

In addition, we note that, in Algorithm 1, $x^{(k+1)}$ is the average value of all the coordinates $x_p^{(k+1)}$ with $p = 2, 3, \ldots, m$. That is to say, the average value is obtained only after all the coordinates are updated. On the other hand, if we can take the average value at once after one coordinate is updated until all the coordinates are contained, we then obtain another modification of Algorithm 1.

Certainly, we can also add the randomness in Algorithm 4 as in Algorithm 2.

We should mention that the alternating minimization methods presented above are numerically feasible and efficient for solving the multilinear system (1) or the associated least-squares problem, which are very different from some existing ones; see, for example, [3, 11, 12], but we have not found a feasible approach to complete their convergence analyses, which will be further studied in future work.

As is well-known, the Newton method is an efficient approach in solving nonlinear equations [22]. In [3], the authors established the Newton iteration for solving the multilinear system (1) in the case that the coefficient tensor $\mathcal{A}$ is a symmetrical nonsingular M-tensor. When the tensor $\mathcal{A}$ in (1) is not symmetric, we can also derive a similar Newton-type iteration method (say NS-Newton for convenience) as presented in [3].

Suppose that the tensor system (1) is solvability, we rewrite it as

$$g(x) := \mathcal{A}x^{m-1} - b = 0. \quad (18)$$

This is a nonlinear equation. Notice that the first-order gradient of $g(x)$ is $\nabla g(x) = \sum_{p=2}^{m} A_{sp} \in \mathbb{R}^{m \times n}$, where $A_{sp} = \mathcal{A} x_{s+1} \ldots, p-1 x_{p-1} x_{p} \ldots, m x$. Then, the iterations are as follows.

If the tensor $\mathcal{A}$ is symmetric over the last $m - 1$ modes, we obtain $\nabla g(x) = (m - 1) A_{sp}$ for any $p \in \{2, 3, \ldots, m\}$. In particular, the above algorithm reduces to the Newton iteration proposed in [3] when the tensor $\mathcal{A}$ was a symmetrically nonsingular M-tensor. In this case, the matrix $\nabla g(x^{(k)})$ is nonsingular as well. It should be emphasized that the matrix $\nabla g(x^{(k)})$ may not be nonsingular for a general tensor; in this case, one can replace the inverse with its Moore–Penrose inverse.

Our numerical tests show that this is an efficient algorithm. In contrast to Algorithm 5, the first three algorithms proposed previously can be implemented in parallel when updating $x_p^{(k+1)}$, which indicates that these methods are promising for solving large-scale multilinear problems.

4. Numerical Experiments

In this section, we provide some numerical experiments to illustrate the feasibility and efficiency of our algorithms proposed in the previous section. All codes were implemented in MATLAB (version 2016a) with a machine precision 1.0e–16 on a personal computer with Inter(R) Core(TM) i5-4200M@2.5GHz, and 4.00 G memory. All tensor-vector multiplications were computed turning to the Tensor Toolbox Version 2.6 (http://www.sandia.gov/tgkolda/TensorToolbox/index-2.6.html).

For the convenience of expression, we use “IT” and “CPU” to denote the number of iterations and the elapsed CPU time (in seconds), respectively. Let “ERR” and “RES” represent, respectively, two symbols defined by

$$\text{ERR} = \left\| x^{(k+1)} - x^{(k)} \right\|,$$

$$\text{RES} = \left\| \mathcal{A}(x^{(k)})^{m-1} - b \right\|,$$

where $x^{(k)}$ stands for the $k$th iteration value. For consistent multilinear systems, the stopping rule is that $\text{RES} < \epsilon = 1.0e-03$, or that the number of iterations exceeds the maximum $k_{\text{max}} = 10000$. On the other hand, for the inconsistent case, the terminated criterion is that the error $\text{ERR} < \epsilon$ or that the number of iterations exceeds the above maximum $k_{\text{max}}$. In addition, the number of iteration steps, the CPU time, and the residual contained in the tables below are, respectively, the average of 5 runs from different initial iteration vectors unless otherwise stated. In particular, the symbol “– –” in the tables means that the iteration did not derive the desired solution before the terminated rule is satisfied.

Example 1. Let the tensor $\mathcal{A}$ in (1) be given by

$$\mathcal{A} = s I - \mathcal{B} = (b_{i1\ldots i_m}) \in \mathbb{R}^{[m,n]} \text{with } b_{i1\ldots i_m} = |\tan(i_1 + i_2 + \cdots + i_m)|,$$

where $s = (1 + a) \cdot \max_{1 \leq i \leq n} (\mathcal{B} e^{m-1})$ with $e = \text{ones}(n, 1)$ and $a = 0.01$, and the right-hand side $b$ is chosen such that $\mathcal{A}(x^*)^{m-1} = b$, where $x^* = 8 \cdot e$.

It is obvious that the tensor $\mathcal{A}$ in this example is a symmetrically nonsingular M-tensor, which follows from Theorem 3.2 of [3] that the multilinear system (1) has the unique solution $x^*$. We compared our algorithms (i.e., Algorithms 1–5) with the Newton method presented in [3] for the different choices of the order $m$ and dimension $n$ of
Mathematical Problems in Engineering

Input: \( \mathcal{A} \in \mathbb{R}^{[m,n]}, b \in \mathbb{R}^n. \)
Input: \( x_0 \in \mathbb{R}^n \) with \( p = 2, 3, \ldots, m. \)
 Initialize \( k (k = 0, 1, 2, \ldots); \)
Step 1. For \( p = 2, 3, \ldots, m, \) update \( x_p^{(k+1)} \) by (16), and then
Step 2. Compute \( x^{(k+1)} = \sum_{m-2}^m x_p^{(k+1)} / (m-1) \)

Algorithm 1: AMM: alternating minimization method for the TLS problem (7).

Input: \( \mathcal{A} \in \mathbb{R}^{[m,n]}, b \in \mathbb{R}^n. \)
Input: \( x_0 \in \mathbb{R}^n \) with \( p = 2, 3, \ldots, m. \)
 Initialize \( k (k = 0, 1, 2, \ldots); \)
Pick a permutation \( \sigma \) of 2, 3, \ldots, \( m \) uniformly at random.
Step 1. For \( i = 1, 2, \ldots, m-1, \) update \( x_i^{(k+1)} \) by (16), and then
Step 2. Compute \( x^{(k+1)} = \sum_{m-2}^m x_p^{(k+1)} / (m-1) \)

Algorithm 2: Random AMM (R-AMM-I) for problem (8).

Input: \( \mathcal{A} \in \mathbb{R}^{[m,n]}, b \in \mathbb{R}^n. \)
Initialize: \( x_0 \in \mathbb{R}^n \) with \( p = 2, 3, \ldots, m. \)
 Round \( k (k = 0, 1, 2, \ldots); \)
Let \( S \) be a nonempty set with \( m-1 \) repeatable elements derived from \([2, \ldots, m]\).
Step 1. For every \( i \in S, \) update \( x_i^{(k+1)} \) by (16), and then
Step 2. Compute \( x^{(k+1)} = \sum_{m-2}^m x_p^{(k+1)} / (m-1) \)

Algorithm 3: Random AMM (R-AMM-II) for problem (8).

Input: \( \mathcal{A} \in \mathbb{R}^{[m,n]}, b \in \mathbb{R}^n. \)
Initialize: \( z_1(0) \in \mathbb{R}^n. \)
 Round \( k (k = 0, 1, 2, \ldots); \)
For \( p = 2, \ldots, m, \) compute
\[ x_p^{(k+1)} = \arg\min_{x_p} J (x_1^{(k+1)}, \ldots, x_p^{(k+1)}, \ldots, x_{p-1}^{(k)}, x_p, \mathcal{A}, x_1^{(k)}, \ldots, x_{p-1}^{(k)}). \]
\[ z_p^{(k)} = \left( \sum_{p-2}^m x_p^{(k+1)} / p \right) / p. \]

Algorithm 4: A modified AMM (AMM-III) for the TLS problem (7).

Input: \( \mathcal{A} \in \mathbb{R}^{[m,n]}, b \in \mathbb{R}^n. \)
Initialize: \( x_0 \in \mathbb{R}^n. \)
 Round \( k (k = 0, 1, 2, \ldots); \)
Step 1. Compute \( \nabla g(x^{(k)}). \)
Step 2. Compute \( r_k = b - \mathcal{A} (x^{(k)})^m. \)
Step 3. Compute \( d_k = \nabla g(x^{(k)})^{-1} r_k. \)
Step 4. Update \( x^{(k+1)} = x^{(k)} + d_k. \)

Algorithm 5: Newton-type method (NS-Newton) for the multilinear system (1).

the tensor \( \mathcal{A} \). Notably, the latter is a feasible method only for symmetrically nonsingular M-tensor systems and denoted by S-Newton for simplicity.

For randomly chosen initial vectors \( x^{(0)} = z_1^{(0)} = \text{rand}(n, 1) \), the corresponding numerical results including the number of iterations, the elapsed CPU time, and the residuals are listed in Tables 1–3. From Table 1, one can observe that the AMM-III takes more iteration steps and CPU time than the others, whose numbers of iteration steps exceed the maximum when \( m = 5: 12 \). As two
versus the iteration step $k$ in Figure 1 for $m = 4, 8$ and $n = 2$ with initial iterative vectors $x^{(0)} = [1, 2]^T$. From these figures, one can see that the convergence of these algorithms is nonlinear.

Next, the proposed iterative methods of this paper will be applied to two numerical examples deriving from realistic applications.

Example 2. Consider the numerical solution of the Klein–Gordon equation [2, 27]:

$$
\begin{align*}
\Delta u(x) &= -f(x), & & \text{in } \Omega, \\
\frac{\partial u(x)}{\partial n} &= g(x), & & \text{on } \partial \Omega,
\end{align*}
$$

(21)

where $\Delta = \sum_{i=0}^{d} (\partial^2/\partial x_i^2)$, $\Omega = [0, 1]^d$, and $m = 3, 4, \ldots$.

When $d = 1$ and the above Klein–Gordon equation can be discretized as the following multilinear system [3]:

$$
\mathcal{L}_h u^{m-1} = f,
$$

(22)

in which $h = (1/(n-1))$ and $\mathcal{L}_h = (\mathcal{L}_h)_{i_1, \ldots, i_m} \in \mathbb{R}^{[m,n]}$ with the entries defined by

- modifications of the AMM method, the R-AMM-I and R-AMM-II methods generally have fewer numbers of iterations than the original one; the paid price is that both of them cost more CPU time. For the S-Newton method, the number of iteration steps is not necessarily less than those of the AMM method, the R-AMM-I method, and the R-AMM-II method, but the clapsed CPU time of which is always less than the others. The numerical results in Table 2 illustrate that one can improve the performance of the R-AMM-III method by adding randomized acceleration. Nevertheless, the improved version still needs more iteration steps and CPU time. Hence, in the sequel, we only discuss the performance of Algorithms 1–3. Since the ADMM-type method proposed by Liang et al. in [17] is an alternating iterative method and the SOR method presented in [26] is an extension of the well-known SOR method for solving linear systems, we compared these two methods with the proposed ones here; see Table 3 for details. This table shows that the AMM and its two randomized versions as well as the S-Newton method have better performance than the SOR method in the number of iteration steps and the elapsed CPU time, while the ADMM-type method takes less iteration steps sometimes for larger $m$. In order to show the feature of Algorithms 1–3, and 5, we, respectively, plotted the figure regarding the residual

$\text{Table 1: Numerical results for the multilinear systems in Example 1.}$

| Algorithms | AMM | R-AMM-I | R-AMM-II | AMM-III | S-Newton |
|------------|-----|---------|---------|---------|----------|
| IT         | 10  | 10      | 10      | 10      | 10       |
| CPU        | 0.0595 | 0.0587 | 0.0691 | 0.0588 | 0.0098   |
| RES        | $1.3859e-06$ | $1.3859e-06$ | $1.3859e-06$ | $1.3859e-06$ | $1.3859e-06$ |
| IT         | 101 | 160     | 90      | 69      | 158      |
| CPU        | 1.3740 | 2.1948 | 1.3568 | 0.9404 | 0.2814   |
| RES        | $1.2806e-04$ | $1.2828e-04$ | $1.2785e-05$ | $2.4082e-04$ | $6.6883e-06$ |
| IT         | 35  | 35      | 35      | —       | 35       |
| CPU        | 0.8129 | 0.8111 | 0.8509 | —       | 0.1036   |
| RES        | $5.5163e-07$ | $5.5165e-07$ | $5.5167e-07$ | —       | $5.5165e-07$ |
| IT         | 149 | 199     | 204     | —       | 195      |
| CPU        | 5.4332 | 7.2626 | 7.6269 | —       | 0.7267   |
| RES        | $7.0794e-05$ | $3.6912e-06$ | $2.4473e-05$ | —       | $7.2131e-06$ |
| IT         | 75  | 75      | 75      | —       | 75       |
| CPU        | 3.8404 | 3.8413 | 3.8947 | —       | 0.3609   |
| RES        | $2.1842e-07$ | $2.6061e-07$ | $1.7296e-07$ | —       | $1.7407e-07$ |
| IT         | 172 | 216     | 404     | —       | 263      |
| CPU        | 12.2829 | 15.5679 | 29.4014 | —       | 1.7491   |
| RES        | $2.5257e-04$ | $7.2879e-05$ | $7.0054e-05$ | —       | $6.3380e-05$ |
| IT         | 136 | 136     | 136     | —       | 136      |
| CPU        | 12.4856 | 12.3464 | 12.4625 | —       | 1.0673   |
| RES        | $5.1420e-05$ | $5.2800e-05$ | $4.6484e-05$ | —       | $2.7792e-05$ |
| IT         | 458 | 299     | 374     | —       | 288      |
| CPU        | 52.8898 | 34.5459 | 43.4561 | —       | 2.6043   |
| RES        | $5.0163e-04$ | $5.9289e-04$ | $4.2793e-04$ | —       | $2.4541e-04$ |
| IT         | 238 | 238     | 238     | —       | 238      |
| CPU        | 33.9051 | 33.9277 | 34.1376 | —       | 2.4006   |
| RES        | $1.3987e-04$ | $2.5207e-04$ | $4.9489e-04$ | —       | $1.2302e-04$ |
| IT         | 2229 | 1373    | 3103    | —       | —        |
| CPU        | 403  | 253     | 558     | —       | —        |
| RES        | $0.0000e+00$ | $0.0000e+00$ | $0.0000e+00$ | —       | —        |
### Table 2: Numerical results for the multilinear systems in Example 1.

| Algorithms | [m, n] | [4, 10] | [4, 20] | [4, 40] | [4, 60] |
|------------|--------|--------|--------|--------|--------|
| AMM        | IT     | 115    | 192    | 228    | 725    |
|            | CPU    | 1.1197 | 2.8279 | 17.4153| 290.0682|
|            | RES    | 6.4585E-06 | 1.1342E-07 | 6.4105E-05 | 9.7471E-05 |
| R-AMM-I    | IT     | 123    | 401    | 227    | 494    |
|            | CPU    | 1.2166 | 5.9627 | 17.4263| 196.4768|
|            | RES    | 3.1021E-05 | 3.7321E-05 | 3.3234E-05 | 3.5282E-05 |
| R-AMM-II   | IT     | 60     | 307    | 474    | 1027   |
|            | CPU    | 0.7606 | 6.2364 | 37.8383| 433.1855|
|            | RES    | 4.7718E-05 | 1.9313E-04 | 1.7141E-04 | 5.0952E-05 |
| AMM-III    | IT     |        |        |        |        |
|            | CPU    |        |        |        |        |
|            | RES    |        |        |        |        |
| AMM-III-I  | IT     | 158    | 376    | 1896   | 553    |
|            | CPU    | 1.6260 | 5.5448 | 145.3878| 258.5192|
|            | RES    | 5.6223E-06 | 6.7505E-05 | 1.1723E-04 | 1.9261E-06 |
| AMM-III-II | IT     | 98     | 944    | 1431   | 929    |
|            | CPU    | 1.1797 | 19.9625| 114.3256| 386.6988|
|            | RES    | 4.0573E-05 | 3.9176E-04 | 2.8023E-06 | 7.1267E-06 |
| S-Newton   | IT     | 95     | 412    | 490    | 653    |
|            | CPU    | 0.2210 | 1.4215 | 9.7721 | 68.4803|
|            | RES    | 1.9016E-04 | 3.1456E-05 | 3.3576E-07 | 1.9708E-04 |

### Table 3: Numerical results for the multilinear systems in Example 1.

| Algorithms | AMM | R-AMM-I | R-AMM-II | S-Newton | SOR | ADMM-type |
|------------|-----|---------|----------|----------|-----|-----------|
| [3, 10]    | IT  | 10      | 10       | 10       | 60  | —         |
|            | CPU | 0.0798  | 0.0837   | 0.1109   | 0.0653 | 0.9776  | 63.2002   |
|            | RES | 2.0627e-10 | 2.0627e-10 | 1.9949e-10 | 2.0249e-10 | 9.1981e-04 | 6.5422e-03 |
| [3, 30]    | IT  | 14      | 14       | 14       | 43  | —         |
|            | CPU | 0.0543  | 0.0581   | 0.1047   | 0.0528 | 0.5670  | 55.3239   |
|            | RES | 1.4074e-09 | 1.4074e-09 | 1.3932e-09 | 1.3854e-09 | 7.6229e-04 | 2.9197e-03 |
| [3, 50]    | IT  | 14      | 14       | 14       | 45  | —         |
|            | CPU | 0.0688  | 0.0696   | 0.1118   | 0.0631 | 1.1789  | 59.9732   |
|            | RES | 5.5115e-06 | 5.5115e-06 | 5.5115e-06 | 5.5114e-06 | 9.9423e-04 | 2.8297e-03 |
| [4, 10]    | IT  | 65      | 60       | 61       | 97  | —         |
|            | CPU | 0.4238  | 0.3671   | 0.4555   | 0.3384 | 0.4563  | 0.3600    |
|            | RES | 6.4552e-09 | 1.0080e-06 | 4.0921e-06 | 5.8740e-07 | 9.4153e-04 | 1.9697e-04 |
| [4, 30]    | IT  | 102     | 243      | 483      | 87  | —         |
|            | CPU | 2.1458  | 4.9919   | 1.9280   | 9.6193 | 1.4587  | 219.0979  |
|            | RES | 1.4629e-07 | 6.6867e-04 | 1.2782e-07 | 8.0313e-08 | 8.2593e-04 | 1.7089e-02 |
| [4, 40]    | IT  | 41      | 114      | 1815     | 375 | 190       | 478       |
|            | CPU | 3.8281  | 10.1056  | 165.9918 | 31.2988 | 9.7806  | 40.5234   |
|            | RES | 5.7258e-07 | 2.2322e-05 | 6.8631e-07 | 8.2008e-07 | 9.1916e-04 | 2.3125e-05 |
| [5, 10]    | IT  | 37      | 37       | 37       | 75  | 36        |
|            | CPU | 0.4777  | 0.4737   | 0.5419   | 0.4426 | 0.5611  | 0.4755    |
|            | RES | 5.9909e-07 | 6.0299e-07 | 5.3983e-07 | 5.5564e-07 | 9.9197e-04 | 6.6949e-06 |
| [5, 20]    | IT  | 49      | 49       | 49       | 49  | 48        |
|            | CPU | 3.3240  | 3.3223   | 3.3805   | 3.3177 | 4.1611  | 3.5067    |
|            | RES | 5.8495e-07 | 6.7395e-07 | 7.9920e-07 | 6.1547e-07 | 9.7748e-04 | 1.4438e-06 |
| [5, 30]    | IT  | 41      | 41       | 41       | 41  | 40        |
|            | CPU | 17.5376 | 17.2470  | 17.3708  | 17.3022 | 30.4449 | 17.4773   |
|            | RES | 3.8089e-06 | 4.1881e-06 | 3.9757e-06 | 2.3715e-06 | 9.4562e-04 | 2.9135e-06 |
| [6, 10]    | IT  | 594     | 895      | 5895     | 624  | 420       | 5652      |
|            | CPU | 28.5333 | 42.4125  | 303.6122 | 29.6389 | 7.7784  | 273.8786  |
|            | RES | 3.1993e-05 | 5.2957e-04 | 3.0382e-05 | 2.4495e-05 | 9.5355e-04 | 5.0279e-05 |
Let $f$ be the vector such that $\mathcal{L}_h(u^*)^{m-1} = f$ with $u^* = 8 \cdot \text{ones}(n,1)$. Since the coefficient tensor $\mathcal{A}$ (22) in this example is a nonsymmetric and nonsingular M-tensor, we compared the three proposed iterative algorithms with the NS-Newton method, the SOR method, and the ADMM-type method for initial iterative vector $x(0)$ chosen randomly and reported the numerical results in Table 4. This table reflects that the numbers of iterations steps of the three AMM-type methods proposed in this article and the NS-Newton method are identical and the CPU time is almost the same. Relatively, the SOR method has better performance than the ADMM-type method.

Example 3. Consider the differential equation

\[
\begin{align*}
- \max_{\gamma, \lambda \in (\Gamma, \Lambda)} \left\{ L^A U - \eta U - \frac{1}{2} ay^2 U + \beta y \right\} &= 0, \quad \text{on } \Omega, \\
U &= g, \quad \text{on } \partial \Omega,
\end{align*}
\]

(24)

where $L^A U(x) = (1/2)\sigma(x, \lambda)^2 U''(x) + \mu(x, \lambda)U'(x)$, $\Omega = (0,1)$, $\Gamma = [0, \infty)$ and $\Lambda$ is a compact metric space.

By applying the “optimize then discretize” approach to the differential equation (24), it can be discretized as a 3\textsuperscript{rd}-order Bellman equation [8]:

\[
\max_{\lambda \in \Lambda} \mathcal{A}^\lambda \mathbf{u}^2 = \mathbf{b},
\]

(25)
in which $\mathbf{u} = (u_i) \in \mathbb{R}^n$, $u_i \equiv U(i\Delta x)$ with $\Delta x = (1/n)$ for $i = 0, 1, 2, \ldots, n$, $\mathcal{A}^\lambda$ is a 3\textsuperscript{rd}-order and $(n+1)$-dimensional parameterized tensor, and the entries of them are defined as follows:
Table 4: Numerical results for the multilinear systems in Example 2.

| Algorithms | AMM | R-AMM-I | R-AMM-II | NS-Newton | SOR | ADMM-type |
|------------|-----|---------|----------|-----------|-----|-----------|
| [3, 10]    | IT  | 8       | 8        | 8         | 8   | 423       |
|            | CPU | 0.1517  | 1.3087   | 0.1981    | 0.0869 | 1.2052    |
|            | RES | 1.9308e-04 | 1.9308e-04 | 1.9308e-04 | 1.9308e-04 | 9.7660e-04 | 1.1703e-02 |
| [3, 30]    | IT  | 15      | 15       | 15        | 15   | 2955      |
|            | CPU | 0.0611  | 0.0594   | 0.0773    | 0.0524 | 12.3794   |
|            | RES | 5.9708e-04 | 5.9708e-04 | 6.1631e-04 | 6.0411e-04 | 9.9562e-04 | 1.2772e-03 |
| [3, 50]    | IT  | 16      | 16       | 16        | 16   | 2081      |
|            | CPU | 0.0714  | 0.0687   | 0.0997    | 0.0640 | 50.0584   |
|            | RES | 5.3146e-04 | 5.3146e-04 | 5.2894e-04 | 5.3783e-04 | 2.6927e-02 | 1.0036e-03 |
| [4, 10]    | IT  | 19      | 19       | 19        | 19   | 705       |
|            | CPU | 0.1641  | 0.1249   | 0.1530    | 0.1204 | 3.2752    |
|            | RES | 7.7367e-04 | 7.7367e-04 | 7.7367e-04 | 7.7367e-04 | 9.8150e-04 | 3.9236e-02 |
| [4, 30]    | IT  | 19      | 19       | 19        | 19   | 4635      |
|            | CPU | 0.3644  | 0.3575   | 0.3748    | 0.3416 | 75.4072   |
|            | RES | 6.7312e-04 | 6.7312e-04 | 6.7312e-04 | 6.7312e-04 | 9.9835e-04 | 9.9831e-04 |
| [4, 50]    | IT  | 25      | 25       | 25        | 25   | 7856      |
|            | CPU | 1.2006  | 1.1478   | 1.1888    | 1.1115 | 293.3731  |
|            | RES | 8.8715e-04 | 8.8715e-04 | 8.8715e-04 | 8.8715e-04 | 9.9992e-04 | 9.9991e-04 |
| [5, 10]    | IT  | 19      | 19       | 19        | 19   | 1155      |
|            | CPU | 0.2283  | 0.2200   | 0.2505    | 0.1999 | 7.2257    |
|            | RES | 4.0658e-04 | 4.0658e-04 | 4.0658e-04 | 4.0658e-04 | 9.8803e-04 | 9.9987e-04 |
| [5, 20]    | IT  | 20      | 20       | 20        | 20   | 3314      |
|            | CPU | 1.3222  | 1.3061   | 1.3944    | 1.2267 | 116.2169  |
|            | RES | 8.7162e-04 | 8.7162e-04 | 8.7162e-04 | 8.7162e-04 | 9.9932e-04 | 2.2657e-03 |
| [5, 30]    | IT  | 22      | 22       | 22        | 22   | 8897      |
|            | CPU | 9.0624  | 9.4346   | 9.6649    | 9.3347 | 1549.5668 |
|            | RES | 4.7169e-04 | 4.7169e-04 | 4.7169e-04 | 4.7169e-04 | 9.9987e-04 | 9.9994e-04 |
| [6, 10]    | IT  | 22      | 22       | 22        | 22   | 1513      |
|            | CPU | 0.8528  | 0.8472   | 0.8541    | 0.8229 | 24.7529   |
|            | RES | 6.8396e-04 | 6.8396e-04 | 6.8396e-04 | 6.8396e-04 | 9.9128e-04 | 9.9992e-04 |

\[
\mathcal{A}_{ij} (\lambda) = \mathcal{A}_{ij} (\lambda),
\]

\[
2 \mathcal{A}_{ij} (\lambda) = \frac{1}{2} \sigma^2_i (\lambda) \frac{1}{(\Delta x)^2} + \frac{\mu_i (\lambda)}{\Delta x^2 (\cos, 0)} (\mu_i (\lambda)),
\]

\[
\mathcal{A}_{ii} (\lambda) = \frac{1}{2} \sigma^2_i (\lambda) \frac{1}{(\Delta x)^2} + |\mu_i (\lambda)| \frac{1}{\Delta x} + \eta_i,
\]

\[
2 \mathcal{A}_{ij} (\sigma) = \frac{1}{2} \sigma^2_i (\lambda) \frac{1}{(\Delta x)^2} - \mu_i (\lambda) \frac{1}{\Delta x} (\cos, 0) (\mu_i (\lambda)),
\]

\[
\mathcal{A}_{ii} (\lambda) = \mathcal{A}_{ii} (\lambda), \quad i = 1, 2, \ldots, n - 1,
\]

and \( \mathcal{A}_{ij} (\lambda) = 1 \), if \( i = 0, n \). The vector \( \mathbf{b} = (b_i) \) is defined by

\[
b_i = \begin{cases} 
\frac{1}{2} \beta_i^2, & \text{if } i = 1, 2, \ldots, n - 1, \\
\gamma_i^2, & \text{if } i = 0, n.
\end{cases}
\]

In this example, we let \( \sigma (x, \lambda) = 0.2, \alpha (x) = 2 - x, \eta (x) = 0.04, u (x, \lambda) = 0.044, \beta (x) = 1 + x, g (x) = 1, \) and \( \lambda = -1 \). The coefficient tensor \( \mathcal{A} \) here is a nonsymmetric and nonsingular M-tensor. We compared the three proposed iterative algorithms with the NS-Newton method, the SOR method, and the ADMM-type method for the initial iterative vector \( \mathbf{x}^{(0)} \) chosen randomly and reported the numerical results in Table 5. From this table, one can see that the three proposed iteration methods in the present paper as well as the NS-Newton method have almost the same convergence; they take fewer iteration steps and CPU time than the SOR method and the ADMM-type method. Notably, the SOR method needs more iteration steps than the ADMM-type method.

Example 4. Let the tensor \( \mathcal{A} \) in (1) be given by \( \mathcal{A} = \text{tenrand} (\mathbf{v}) \in \mathbb{R}^{[n,m]} \) with \( \mathbf{v} = \text{ones} (1, m) * n \), and let the right-hand side \( \mathbf{b} \) be chosen such that \( \mathcal{A} (\mathbf{x})^{m-1} = \mathbf{b} \), where \( \text{tenrand} (\mathbf{v}) \) returns an \( m \)-th order \( n \)-dimensional tensor containing pseudorandom values drawn from the uniform distribution and \( \mathbf{x}^* \) is the same as that defined in Example 1.

In this example, the tensor \( \mathcal{A} \) is a randomly generated tensor; then, the solution of the multilinear system (1) may not be unique. We compared the proposed algorithms (i.e., Algorithms 1–3) with the NS-Newton method and the ADMM-type method for randomly produced initial vectors \( \mathbf{x}^{(0)} = \mathbf{z}^{(0)} = \text{rand} (n, 1) \). The obtained results were displayed in Table 6, which indicates that all the tested methods converge before the number of iteration steps arrives at the
Table 5: Numerical results for the multilinear systems in Example 3.

| Algorithms  | [m, n] | [3,10] | [3,30] | [3,50] | [3,100] | [3,200] |
|-------------|--------|--------|--------|--------|---------|---------|
| AMM         | IT     | 7      | 10     | 14     | 11      |         |
|             | CPU    | 0.0194 | 0.0289 | 0.0415 | 0.1281  | 0.5046  |
|             | RES    | 2.8217e-05 | 4.1736e-05 | 5.7067e-06 | 3.6582e-06 | 1.8151e-04 |
| R-AMM-I     | IT     | 7      | 10     | 14     | 11      |         |
|             | CPU    | 0.0197 | 0.0250 | 0.0381 | 0.1465  | 0.4947  |
|             | RES    | 2.8217e-05 | 4.1736e-05 | 5.7067e-06 | 3.6582e-06 | 1.8151e-04 |
| R-AMM-II    | IT     | 7      | 10     | 14     | 11      |         |
|             | CPU    | 0.0274 | 0.0350 | 0.0504 | 0.1514  | 0.5053  |
|             | RES    | 2.8217e-05 | 4.1736e-05 | 5.7067e-06 | 3.6582e-06 | 1.8151e-04 |
| NS-Newton   | IT     | 7      | 10     | 14     | 11      |         |
|             | CPU    | 0.0152 | 0.0221 | 0.0338 | 0.1332  | 0.4939  |
|             | RES    | 2.8217e-05 | 4.1736e-05 | 5.7067e-06 | 3.6582e-06 | 1.8151e-04 |
| SOR         | IT     | 74     | 843    | 2433   | 9643    |         |
|             | CPU    | 1.4385 | 3.1727 | 10.7665 | 118.9747 |         |
|             | RES    | 9.1750e-04 | 9.9189e-04 | 9.9983e-04 | 9.991e-04 |         |
| ADMM-type   | IT     | 30     | 118    | 627    | 7400    |         |
|             | CPU    | 0.0908 | 0.4398 | 3.0127 | 89.5569 |         |
|             | RES    | 8.6000e-04 | 9.9739e-04 | 9.9547e-04 | 9.9987e-04 |         |

Table 6: Numerical results for the multilinear systems in Example 4.

| Algorithms  | AMM       | R-AMM-I   | R-AMM-II  | NS-Newton |
|-------------|-----------|-----------|-----------|-----------|
|             | IT        | 11        | 11        | 10        | 8         |
|             | CPU       | 0.0674    | 0.0636    | 0.0694    | 0.0231    |
|             | RES       | 1.2805e-04 | 3.8136e-04 | 5.2323e-04 | 1.7398e-04 |
| [3,2]       | IT        | 34        | 74        | 88        | 32        |
|             | CPU       | 0.4561    | 1.0108    | 1.3156    | 0.2187    |
|             | RES       | 7.0644e-04 | 6.6799e-04 | 6.9935e-04 | 3.096e-05 |
| [4,2]       | IT        | 75        | 61        | 76        | 27        |
|             | CPU       | 1.7356    | 1.4059    | 1.8333    | 0.3656    |
|             | RES       | 4.8703e-04 | 4.7376e-04 | 6.6511e-04 | 1.051e-05 |
| [5,2]       | IT        | 107       | 149       | 205       | 50        |
|             | CPU       | 3.8485    | 5.3445    | 7.5521    | 1.0975    |
|             | RES       | 8.9606e-04 | 4.9813e-04 | 5.5719e-04 | 6.6814e-06 |
| [6,2]       | IT        | 79        | 116       | 115       | 67        |
|             | CPU       | 4.1114    | 6.0589    | 6.1478    | 2.1800    |
|             | RES       | 3.5437e-04 | 5.9663e-04 | 3.7370e-04 | 1.4862e-04 |
| [7,2]       | IT        | 192       | 175       | 215       | 108       |
|             | CPU       | 13.6602   | 12.4068   | 15.5309   | 4.8704    |
|             | RES       | 5.9745e-04 | 7.8879e-04 | 6.9943e-04 | 8.1033e-05 |
| [8,2]       | IT        | 331       | 494       | 467       | 150       |
|             | CPU       | 31.4421   | 48.2269   | 45.9329   | 8.9267    |
|             | RES       | 8.2688e-04 | 6.4713e-04 | 8.5859e-04 | 7.9815e-08 |
| [9,2]       | IT        | 247       | 446       | 430       | 157       |
|             | CPU       | 29.2164   | 52.7115   | 51.3435   | 11.9918   |
|             | RES       | 8.6464e-04 | 7.6783e-04 | 7.1800e-04 | 5.6479e-05 |
| [10,2]      | IT        | 223       | 478       | 509       | 181       |
|             | CPU       | 223.2902  | 71.5219   | 76.9164   | 17.1811   |
|             | RES       | 6.0747e-04 | 6.3177e-04 | 4.8267e-04 | 1.2165e-05 |
| [11,2]      | IT        | 617       | 1817      | 1010      | 2234      |
|             | CPU       | 112.0114  | 333.9998  | 188.5232  | 261.1745  |
|             | RES       | 4.8158e-04 | 6.4279e-04 | 5.7918e-04 | 6.9879e-04 |
maximum $k_{\text{max}}$. Particularly, the randomized technique may improve the performance of the AMM method. Besides, in Figure 2, we also displayed the convergence behavior of Algorithms 1–3 and 5 when $m = 4, 8$ with $n = 2$ and initial iterative vector $x^{(0)} = [1, 2]^T$.

From the above numerical tests, we claim that Algorithm 1 is an efficient method for solving some consistent multilinear system (1); what is more, the first two modified versions of this algorithm are also feasible. In particular, one of the advantages of these algorithms is that they can be carried out in a parallel version, which indicates that our algorithms may be implementable for large-scale tensor systems, while the Newton type methods are not easy to run in a parallel version.

Additionally, we may always meet the inconsistent multilinear system because of errors. In the next example, we consider the least-squares problem associated with the multilinear system (1); there, the coefficient tensor $\mathcal{A}$ and the right-hand side $b$ are chosen randomly.

**Example 5.** Let the tensor $\mathcal{A}$ in (1) be given by $\mathcal{A} = \text{tenrand}(v) \in \mathbb{R}^{[m,n]}$ with $v = \text{ones}(1,m) \ast n$, and the right-hand side $b = \text{randn}(n,1) \in \mathbb{R}^n$, where $\text{tenran d}(v)$ returns an $m$th-order $n$-dimensional tensor containing pseudorandom values drawn from the standard normal distribution.

We consider the least-squares problem of the multilinear system (1), since the tensor $\mathcal{A}$ and the right-hand side $b$ are random, which may lead to an inconsistent multilinear system. In view of the bad performance of Algorithm 4, we only compare Algorithms 1–3 and 5 with the SOR method [26] and the ADMM-type method [17].

For the randomly chosen initial vector $x^{(0)} = \text{rand}(n,1)$, we still run 5 trials of the algorithms in the cases that $m = 3$: 3: 18 with $n = 2$, respectively; the numerical results are listed in Table 7. In the tests, the SOR method is not convergent before the number of iteration steps arrives at $k_{\text{max}}$, and so it is not contained in Table 7. From this table, we can observe that all the tested algorithms are convergent. Relatively, the NS-Newton method often possesses the least number of iteration steps for the lower $m$, while the ADMM-type method has better performance than the others for the higher case $m$. Moreover, the ADMM-type method takes less CPU time than the former when their iteration steps are identical. On the whole, the original AMM method performs better than the two random versions.

Next, we present an example in which the entries of the coefficient tensor vary greatly, which means that it leads to ill-posed multilinear systems.

**Example 6.** Let the tensor $\mathcal{A} = \text{tensor}(1: n^m, v)$ with $v = \text{ones}(1,m) \ast n$, and let the vector $b$ be chosen in the same way as in Example 2, where $\text{tensor}(1: n^m, v)$ reshapes the numbers 1, 2, $\ldots, n^m$ as an $m$th-order $n$-dimensional tensor.

For $m = 5$ and $n = 2$, we, respectively, ran Algorithms 2 and 3 for randomly chosen 1000 initial vectors $x^{(0)} = \text{rand}(n,1)$ and displayed the numerical results in Figure 3, in which the $x$-axis refers to the $k$th initial vector and the $y$-axis represents the difference of the numbers of iteration steps (left) or the elapsed CPU time (right) of the two algorithms. Specifically, the red symbol “+” means that the number of iteration steps or the CPU time of Algorithm 2 is more than that of Algorithm 3, and the green symbol “*” means that their iteration steps are identical. Otherwise, we use the blue cycle “o.” In Figure 3, the red symbols account for 49.6% of the total (left) and 39.8% of the total (right). Moreover, for the blue symbols “o”, there are
48.1% of cases (left) and 60.2% of cases (right). We should mention that the number of iteration steps of Algorithm 1 always exceeds the $k_{\text{max}}$ for the chosen initial vectors.

The obtained numerical results indicate that all the algorithms introduced in this paper are feasible and efficient for the tested multilinear systems. In particular, the randomized versions always have better performance than the original one. Notably, many others trials not listed here also reflect that our algorithms are commonly comparable to the existing ones when solving some structured multilinear systems. However, there are still many problems to be studied; for instance, although the proposed algorithms are numerically effective, unfortunately, we could not give their proofs of convergence at present. Moreover, for very large $m$, our algorithms may be intractable due to the curse-of-dimensionality [28]; thus, how to break the curse is an interesting but important work.

5. Conclusions

In this paper, we introduced the alternating minimal method as well as its randomized versions for solving the least-squares problem corresponding to the multilinear systems (1), that is, Algorithms 3.1–3.4. The performed numerical
experiments reflect that the proposed algorithms are feasible and efficient. However, we cannot provide the convergence proofs of the two iterative algorithms and leave it as an open problem, which will be considered in the future work.

**Data Availability**

The underlying data are included in the article.

**Conflicts of Interest**

The authors declare that they have no conflicts of interest.

**Acknowledgments**

This work was supported by the National Natural Science Foundation of China (no. 11961057), the Science and Technology Project of Gansu Province (nos. 201B-221), the Fuxi Scientific Research Innovation Team of Tianshui Normal University (no. FXD2020-03), and the Science Foundation (nos. CXT2019-36 and CXJ2020-11) as well as the Education and Teaching Reform Project of Tianshui Normal University (nos. JY202004 and JY203008).

**References**

[1] X. Li and M. K. Ng, “Solving sparse non-negative tensor equations: algorithms and applications,” *Frontiers of Mathematics in China*, vol. 10, no. 3, pp. 649–680, 2015.

[2] Y. Matsuno, “Exact solutions for the nonlinear Klein-Gordon and Liouville equations in four-dimensional Euclidean space,” *Journal of Mathematical Physics*, vol. 28, no. 10, pp. 2317–2322, 1987.

[3] W. Ding and Y. Wei, “Solving multi-linear systems with M-tensors,” *Journal of Scientific Computing*, vol. 68, no. 2, pp. 689–715, 2016.

[4] M. Shahriari and J. Manafian, “An efficient algorithm for solving the fractional Dirac differential operator,” *Advanced Mathematical Models & Applications*, vol. 5, no. 3, pp. 289–297, 2020.

[5] J. Manafian, O. Ilhan, S. Y. Mohyalddeen et al., “New strategic method for fractional mitigating internet bottleneck with quadratic-cubic nonlinearity,” *Mathematical Sciences*, 2021.

[6] S. Pourghahban, J. Manafian, M. Ranjar, A. Aliyeva, and Y. S. Gasimov, “An efficient alternating direction explicit method for solving a nonlinear partial differential equation,” *Mathematical Problems in Engineering*, vol. 2020, Article ID 9647416, 12 pages, 2020.

[7] Z. Luo, L. Qi, and N. Xiu, “The sparsest solutions to Z-tensor complementarity problems,” *Optimization Letters*, vol. 11, no. 3, pp. 471–482, 2017.

[8] P. Azimzadeh and E. Bayraktar, “High order Bellman equations and weakly chained diagonally dominant tensors,” *SIAM Journal on Matrix Analysis and Applications*, vol. 40, no. 1, pp. 276–298, 2019.

[9] X. Wang, M. Che, and Y. Wei, “Neural networks based approach solving multi-linear systems with M-tensors,” *Neurocomputing*, vol. 351, pp. 33–42, 2019.

[10] L. Han, “A homotopy method for solving multilinear systems with M-tensors,” *Applied Mathematics Letters*, vol. 69, pp. 49–54, 2017.

[11] D. Li, S. Xie, and H. Xu, “Splitting methods for tensor equations,” *Numerical Linear Algebra*, vol. 33, no. 1, pp. 75–94, 2018.

[12] Z. Xie, X. Jin, and Y. Wei, “Tensor methods for solving symmetric M-tensor systems,” *Journal of Scientific Computing*, vol. 74, no. 1, pp. 412–425, 2018.

[13] Z. Xie, X. Jin, and Y. Wei, “A fast algorithm for solving circulant tensor systems,” *Linear Multilinear Algebra*, vol. 65, no. 9, pp. 1894–1904, 2017.

[14] P. Comon, X. Luciani, and A. L. F. De Almeida, “Tensor decompositions, alternating least squares and other tales,” *Journal of Chemometrics*, vol. 23, no. 7–8, pp. 393–405, 2009.

[15] L. Grippo and M. Sciandrone, “On the convergence of the block nonlinear Gauss-Seidel method under convex constraints,” *Operations Research Letters*, vol. 26, no. 3, pp. 127–136, 2000.

[16] A. Uschmajew, “Local convergence of the alternating least squares algorithm for canonical tensor approximation,” *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 2, pp. 639–652, 2012.

[17] M. Liang, B. Zheng, and R. Zhao, “Alternating iterative methods for solving tensor equations with applications,” *Numerical Algorithms*, vol. 80, no. 4, pp. 1437–1465, 2019.

[18] Y. Nesterov, “Efficiency of coordinate descent methods on huge-scale optimization problems,” *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.

[19] R. Sun, Z. Luo, and Y. Ye, “On the expected convergence of randomly permuted ADMM,” 2015, https://arxiv.org/abs/1503.06387.

[20] M. Schmidt, N. L. Roux, and F. Bach, “Minimizing finite sums with the stochastic average gradient,” *Mathematical Programming*, vol. 162, no. 1–2, pp. 83–112, 2017.

[21] T. Kolda and B. Bader, “Tensor decompositions and applications,” *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[22] J. Ortega and W. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[23] S. Holtz, T. Rohwedder, and R. Schneider, “The alternating linear scheme for tensor optimization in the tensor train format,” *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. A683–A713, 2012.

[24] M. Mohlenkamp, “Musings on multilinear fitting,” *Linear Algebra and Its Applications*, vol. 438, no. 2, pp. 834–852, 2013.

[25] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, 4th edition, 2013.

[26] D. Liu, W. Li, and S. Vong, “P+he tensor splitting with applications to solve multi-linear systems,” *Frontiers of Mathe- matics in China*, vol. 23, no. 3, pp. 677–702, 2018.

[27] D. Li, S. Xie, and H. Xu, “Splitting methods for tensor equations,” *Journal of Computational and Applied Mathematics*, vol. 330, no. 1, pp. 75–94, 2018.

[28] D. Li, S. Xie, and H. Xu, “Splitting methods for solving multi-linear systems with M-tensors,” *Journal of Computational and Applied Mathematics*, vol. 330, no. 1, pp. 75–94, 2018.

[29] D. Li, S. Xie, and H. Xu, “Splitting methods for solving multi-linear systems with M-tensors,” *SIAM Journal on Matrix Analysis and Applications*, vol. 40, no. 1, pp. 276–298, 2019.

[30] X. Wang, M. Che, and Y. Wei, “Neural networks based approach solving multi-linear systems with M-tensors,” *Neurocomputing*, vol. 351, pp. 33–42, 2019.

[31] L. Han, “A homotopy method for solving multilinear systems with M-tensors,” *Applied Mathematics Letters*, vol. 69, pp. 49–54, 2017.