Conjugate gradient algorithm for consistent generalized Sylvester-transpose matrix equations

Kanjanaporn Tansri, Sarawanee Choomklang and Patrawut Chansangiam*

Department of Mathematics, School of Science, King Mongkut’s Institute of Technology Ladkrabang, Bangkok 10520, Thailand

* Correspondence: Email: patrawut.ch@kmitl.ac.th; Tel: +66935266600; Fax: +6602329840011 ext. 284.

Abstract: We develop an effective algorithm to find a well-approximate solution of a generalized Sylvester-transpose matrix equation where all coefficient matrices and an unknown matrix are rectangular. The algorithm aims to construct a finite sequence of approximated solutions from any given initial matrix. It turns out that the associated residual matrices are orthogonal, and thus, the desire solution comes out in the final step with a satisfactory error. We provide numerical experiments to show the capability and performance of the algorithm.

Keywords: conjugate gradient algorithm; generalized Sylvester-transpose matrix equation; Kronecker product; matrix norm; orthogonality

Mathematics Subject Classification: 15A60, 15A69, 65F45

1. Introduction

Sylvester-type matrix equations show up naturally in several branches of mathematics and engineering. Indeed, many problems in vibration and structural analysis, robotics control and spacecraft control can be represented by the following general dynamical linear model:

\[ \sum_{i=0}^{s_1} A_i x^{(i)} + \sum_{j=0}^{s_2} B_j u^{(j)} = 0 \]  

(1.1)

where \( x \in \mathbb{R}^{m \times 1} \) and \( u \in \mathbb{R}^{n \times 1} \) are the state vector and the input vector respectively, and \( A_i \in \mathbb{R}^{m \times m} \) and \( B_j \in \mathbb{R}^{m \times n} \) are the system coefficient matrices; see e.g., [1, 2]. The dynamical linear system (1.1) includes

\[ A_1 \dot{x} + A_0 x + B_0 u = 0, \quad \text{the descriptor linear system}, \]  

(1.2)
as special cases. Certain problems in control engineering, such as pole/eigenstructure assignment and observer design of the system (1.1) are closely related to the Lyapunov matrix equation $AX + XA^T = B$, the Sylvester matrix equation $AX + XB = C$, and other related equations; see e.g., [2–7]. In particular, the Sylvester matrix equation also plays a fundamental role in signal processing and model reduction; see e.g., [8–10]. These equations are special cases of a generalized Sylvester-transpose matrix equation:

$$\sum_{i=1}^{k} A_i X B_i + \sum_{j=1}^{t} C_j X^T D_j = E. \quad (1.5)$$

A traditional way to solve Eq (1.5) for an exact solution is to transform it to an equivalent linear system via the Kronecker linearization; see Section 3 for details. However, this approach is only suitable when the dimensions of coefficient matrices are small. In practice, for a large-dimension case, it is enough to find a well-approximate solution via an iterative procedure, so that it is not necessary to use as massive as traditional methods. There are several articles that consider problems that approximate the generalized Sylvester resistive matrix equations and constructs a finite sequence of approximated solutions from any given initial matrix. In the last five years, many researchers developed iterative methods for solving Sylvester-type matrix equations related to Eq (1.5). A group of Hermitian and skew-Hermitian splitting (HSS) methods aims to decompose a square matrix as the sum of its Hermitian part and skew-Hermitian part. There are several variations of HSS, namely, GMHSS [11], preconditioned HSS [12], FPPSS [13], and ADSS [14]. A group of gradient-based iterative (GI) algorithms aims to construct a sequence of approximated solutions converging to the exact solution, based on the gradients of quadratic norm-error functions. The original GI method for a generalized Sylvester matrix equation was developed by Ding et al. [15]. Then Niu et al. [16] adjusted the GI method by introducing a weighted factor. After that a half-step-update of GI method, called MGI method, introducing by Wang et al. [17]. The idea of GI algorithm can be used in conjunction with matrix diagonal-extraction to get AJGI [18] and MGI [19] algorithms. See more GI algorithms for a generalized Sylvester matrix equations in [20–22]. For a generalized Sylvester-transpose matrix equation $AXB + CXT D = E$, there are GI algorithm [23] and an accelerated gradient-based iterative (AGBI) algorithm [24] to construct approximate solutions. There are also GI techniques based on optimization, e.g., [25–27]. See more computational methods for linear matrix equations in a survey [28]. The iterative procedures can be used to find solutions of certain nonlinear matrix equations; see e.g., [29–31]. There are also applications of such techniques to parameter estimation in dynamical systems; see e.g., [32–34].

An idea of conjugate gradient (CG) is a remarkable technique constructing an orthogonal basis from the gradient of the associated quadratic function. There are several variations of CG to solve such matrix equations, e.g., BiCG [35], Bi-conjugate residual method [35], CGS [36], preconditioned nested splitting CG [37], generalized conjugate direction (GCD) method [38], conjugate gradient least-squares (CGLS) method [39], and GPBiCG [40].

In this paper, we propose a conjugate gradient algorithm to solve the generalized Sylvester-transpose matrix Eq (1.5) in the consistent case, where all given coefficient matrices and the unknown matrix are rectangular (see Section 4). The algorithm aims to construct a sequence of approximate solutions.
of (1.5) from any given initial value. It turns out that a desire solution comes out in the final step of iterations with a satisfactory error (see Section 4). To validate the theory, we provide numerical experiments to show the applicability and the performance of the algorithm (see Section 5). In particular, the performance of the algorithm is significantly better than that of the direct Kronecker linearization and recent gradient-based iterative algorithms.

2. Preliminaries

In this section, we recall useful tools and facts from matrix analysis that are used in later discussions. Throughout, we denote the set of all \( m \times n \) real matrices by \( \mathbb{R}^{m \times n} \).

Recall that the Kronecker product of \( A = [a_{ij}] \in \mathbb{R}^{m \times n} \) and \( B \in \mathbb{R}^{p \times q} \) is defined by

\[
A \otimes B = [a_{ij}B] \in \mathbb{R}^{mq \times np}.
\]

**Lemma 1** (see, e.g., [41]). The following properties hold for any compatible matrices \( A, B, C \):

1) \((A \otimes B)^T = A^T \otimes B^T\),
2) \((A + B) \otimes C = A \otimes C + B \otimes C\),
3) \(A \otimes (B + C) = A \otimes B + A \otimes C\).

The vector operator Vec(·) assigns to each matrix \( A = [a_{ij}] \in \mathbb{R}^{m \times n} \) the column vector

\[
\text{Vec} A = [a_{11} \cdots a_{m1} \cdots a_{12} \cdots a_{m2} \cdots a_{1n} \cdots a_{mn}]^T \in \mathbb{R}^{mn}.
\]

This operator is bijective, linear, and compatible with the usual matrix multiplication in the following sense.

**Lemma 2** (see, e.g., [41]). For any \( A \in \mathbb{R}^{m \times n} \), \( B \in \mathbb{R}^{p \times q} \) and \( X \in \mathbb{R}^{n \times p} \), we have

\[
\text{Vec}AXB = (B^T \otimes A) \text{Vec} X.
\]

Recall that the commutation matrix \( P(m, n) \) is a permutation matrix defined by

\[
P(m, n) = \sum_{i=1}^{m} \sum_{j=1}^{n} E_{ij} \otimes E_{ij}^T \in \mathbb{R}^{mn \times mn}
\]

where each \( E_{ij} \in \mathbb{R}^{m \times n} \) has entry 1 in \((i, j)\)-th position and all other entries are 0.

**Lemma 3** (see, e.g., [41]). For any \( A \in \mathbb{R}^{m \times n} \) and \( B \in \mathbb{R}^{p \times q} \), we have

\[
B \otimes A = P(n, p)^T (A \otimes B) P(n, q).
\]

**Lemma 4** (see, e.g., [41]). For any matrix \( X \in \mathbb{R}^{m \times n} \), we have

\[
\text{Vec}(X^T) = P(m, n) \text{Vec}(X).
\]

**Lemma 5** (see, e.g., [41]). For any matrices \( A, B, X, Y \) of compatible dimensions, we have

\[
(\text{Vec}(Y))^T (A \otimes B) \text{Vec}(X) = \text{tr}\left( A^T Y^T BX \right).
\]

Recall that the Frobenius norm of \( A \in \mathbb{R}^{m \times n} \) is defined by

\[
||A|| = \left( \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}} = \left( \text{tr}(A^T A) \right)^{\frac{1}{2}}.
\]
3. The direct Kronecker linearization for a generalized Sylvester-transpose matrix equation

From now on, let \( m, n, p, q, r, s, k, \in \mathbb{N} \) be such that \( mq = np \). Consider the generalized Sylvester-transpose matrix Eq (1.5) where \( A_i \in \mathbb{R}^{m \times n}, B_i \in \mathbb{R}^{n \times q}, C_j \in \mathbb{R}^{m \times p}, D_j \in \mathbb{R}^{n \times q}, D_j \in \mathbb{R}^{n \times q}, E \in \mathbb{R}^{m \times q} \) are given matrices, and \( X \in \mathbb{R}^{n \times p} \) is unknown. The Eq (1.5) includes the Lyapunov equation, the Sylvester equation, the equation \( AXB + CXD = E \), and the equation \( AXB + CX^TD = E \) as special cases.

A direct method to solve Eq (1.5) is to transform it to an equivalent linear system. For convenience, denote \( P = P(n, p) \). By taking the vector operator to (1.5) and utilizing Lemma 4, we get

\[
\text{Vec } E = \text{Vec} \left( \sum_{i=1}^{s} A_i XB_i + \sum_{j=1}^{t} C_j X^T D_j \right) = \sum_{i=1}^{s} (B_i^T \otimes A_i) \text{Vec } X + \sum_{j=1}^{t} (D_j^T \otimes C_j) \text{Vec } X^T = \sum_{i=1}^{s} (B_i^T \otimes A_i) \text{Vec } X + \sum_{j=1}^{t} (D_j^T \otimes C_j) P \text{Vec } X = \left( \sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j) P \right) \text{Vec } X. \tag{3.1}
\]

Let us denote \( x = \text{Vec } X, b = \text{Vec } E, \) and

\[
K = \sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j) P \in \mathbb{R}^{mq \times np}. \tag{3.2}
\]

Thus, Eq (3.1) is equivalent to a linear algebraic system \( K x = b \). Hence, Eq (3.1) is consistent if and only if the associated linear system is consistent (i.e., \( \text{rank } [K b] = \text{rank } K \)). When we solve for \( x \), we can get the unknown matrix \( X \) using the injectivity of the vector operator. However, if the matrix coefficients \( A_i, B_i, C_j, D_j \) are of large sizes, then the size of \( K \) can be very large due to the Kronecker multiplication. Thus, traditional methods such as Gaussian elimination and \( LU \) factorization require a large memory to solve the linear system for an exact solution. Thus, the direct method is suitable for matrices of small sizes. For matrices of moderate/large sizes, it is enough to find a well-approximate solution for Eq (3.1) via an iterative procedure.

4. A conjugate gradient algorithm for consistent generalized Sylvester-transpose matrix equations

The main task is to find a well-approximate solution of the matrix Eq (1.5):

\textbf{Problem 4.1.} Let \( m, n, p, q, r, s, k, \in \mathbb{N} \) be such that \( mq = np \). Consider the generalized Sylvester-transpose matrix Eq (1.5) where \( A_i \in \mathbb{R}^{m \times n}, B_i \in \mathbb{R}^{n \times q}, C_j \in \mathbb{R}^{m \times p}, D_j \in \mathbb{R}^{n \times q}, D_j \in \mathbb{R}^{n \times q}, E \in \mathbb{R}^{m \times q} \) are given matrices, and \( X \in \mathbb{R}^{n \times p} \) is unknown. Suppose that Eq (1.5) has a solution. Given an error \( \epsilon > 0 \), find \( \tilde{X} \in \mathbb{R}^{n \times p} \) such that

\[
\left\| E - \sum_{i=1}^{s} A_i \tilde{X} B_i - \sum_{j=1}^{t} C_j \tilde{X}^T D_j \right\| < \epsilon.
\]
We will solve Problem 4.1 under an additional assumption that $K$ in (3.2) is symmetric. We propose the following algorithm:

**Algorithm 1:** A CG algorithm for a generalized Sylvester-transpose matrix equation

$A_i \in \mathbb{R}^{m \times n}$, $B_i \in \mathbb{R}^{p \times q}$, $C_j \in \mathbb{R}^{m \times p}$, $D_j \in \mathbb{R}^{n \times q}$ for any $i = 1, 2, \ldots, s$, $j = 1, 2, \ldots, t$ and $E \in \mathbb{R}^{m \times q}$;

Given $\epsilon > 0$, set $k = 0$, $U_0 = 0$. Choose $X_0 \in \mathbb{R}^{n \times p}$

$$R_0 = E - \sum_{j=1}^{t} C_j X_0 D_j$$

for $k = 0$ to $np$ do

| if $\|R_k\| \leq \epsilon$ then \
| | $X_k$ is the desire solution; break \
| else \
| | if $k = 0$ then \
| | | set $U_{k+1} = R_k$ \
| | else \
| | | set $U_{k+1} = R_k + \frac{\|R_k\|^2}{\|R_{k-1}\|^2} U_k$
| end

$$V_{k+1} = \sum_{i=1}^{s} A_i U_{k+1} B_i + \sum_{j=1}^{t} C_j U_{k+1}^T D_j$$

$$\alpha_{k+1} = \text{tr} \left( U_{k+1}^T V_{k+1} \right)$$

$$X_{k+1} = X_k + \frac{\|R_k\|^2}{\alpha_{k+1}} U_{k+1}$$

$$R_{k+1} = E - \sum_{i=1}^{s} A_i X_{k+1} B_i - \sum_{j=1}^{t} C_j X_{k+1}^T D_j$$

update $k$

end

We call $X_k$ the approximate solution at the $k$-th step. The main computation

$$X_{k+1} = X_k + \frac{\|R_k\|^2}{\alpha_{k+1}} U_{k+1},$$

means that the next approximation $X_{k+1}$ is the sum between the current one $X_k$ and the search direction $U_{k+1}$ with the step size $\|R_k\|^2/\alpha_{k+1}$.

**Remark 4.2.** The stopping rule of Algorithm 1 is based on the size of the residual matrix $R_k$. One can impose another stopping criterion besides $\|R_k\| \leq \epsilon$, e.g., the norm of the difference $X_{k+1} - X_k$ between successive iterates is small enough.

**Remark 4.3.** Let us discuss the complexity analysis for Algorithm 1. For convenience, suppose that all matrices in Eq (1.5) are of sizes $n \times n$. Each step of the algorithm requires the matrix addition ($O(n^2)$), the matrix multiplication ($O(n^3)$), the matrix transposition ($O(n^2)$), the matrix norm ($O(n)$), and the matrix trace ($O(n)$). In summary, the complexity analysis for each step is $O(n^3)$, and thus the algorithm runtime complexity is cubic time.
Next, we will show that, for any given initial matrix $X_0$, Algorithm 1 produces approximate solutions so that the set of residual matrices is an orthogonal set and, thus, we get the desired solution in a finite step. We divide the proof into several lemmas.

**Lemma 6.** Consider Problem 4.1. Suppose that the sequence $\{R_i\}$ is generated by Algorithm 1. We have

$$R_{k+1} = R_k - \frac{||R_k||^2}{\alpha_{k+1}} V_{k+1} \quad \text{for } k = 1, 2, ... \quad (4.1)$$

**Proof.** From Algorithm 1, we have that for any $k$,

$$R_{k+1} = E - \sum_{i=1}^{s} A_i X_k B_i - \sum_{j=1}^{t} C_j X_k^T D_j$$

$$= E - \sum_{i=1}^{s} A_i \left( X_k + \frac{||R_k||^2}{\alpha_{k+1}} U_{k+1} \right) B_i - \sum_{j=1}^{t} C_j \left( X_k^T + \frac{||R_k||^2}{\alpha_{k+1}} U_{k+1}^T \right) D_j$$

$$= E - \sum_{i=1}^{s} A_i X_k B_i - \sum_{j=1}^{t} C_j X_k^T D_j - \frac{||R_k||^2}{\alpha_{k+1}} \left( \sum_{i=1}^{s} A_i U_k B_i + \sum_{j=1}^{t} C_j U_k^T D_j \right)$$

$$= R_k - \frac{||R_k||^2}{\alpha_{k+1}} V_{k+1}.$$

□

**Lemma 7.** Assume that the matrix $K$ in (3.2) is symmetric. The sequences $\{U_i\}$ and $\{V_i\}$ generated by Algorithm 1 satisfy

$$\text{tr} \left( U_m^T V_n \right) = \text{tr} \left( V_m^T U_n \right) \quad \text{for any } m, n. \quad (4.2)$$

**Proof.** Applying Lemmas 1–5 and the symmetry of $K$, we have

$$\text{tr} \left( V_m^T U_n \right) = (\text{Vec} V_m)^T \text{Vec} U_n$$

$$= \left[ \sum_{i=1}^{s} (B_i^T \otimes A_i) \text{Vec} U_m + \sum_{j=1}^{t} (D_j^T \otimes C_j) \text{Vec} U_m^T \right] \text{Vec} U_n$$

$$= [K \text{Vec} U_m]^T \text{Vec} U_n$$

$$= (\text{Vec} U_m)^T K \text{Vec} U_n$$

$$= (\text{Vec} U_m)^T \left[ \text{Vec} \left( \sum_{i=1}^{s} A_i U_n B_i + \sum_{j=1}^{t} C_j U_n^T D_j \right) \right]$$

$$= (\text{Vec} U_m)^T \text{Vec} V_n$$

$$= \text{tr} \left( U_m^T V_n \right)$$

for any $m, n$. □

AIMS Mathematics

Volume 7, Issue 4, 5386–5407.
Lemma 8. Assume that the matrix $K$ is symmetric. The sequences $\{R_i\}, \{U_i\}$ and $\{V_i\}$ are generated by Algorithm 1 satisfy

$$\text{tr}(R^T_i R_{m-1}) = 0 \quad \text{and} \quad \text{tr}(U^T_{m+1} V_m) = 0 \quad \text{for any} \quad m. \quad (4.3)$$

Proof. To prove this conclusion, we use induction principle. In order to compute related terms, we use Lemmas 6 and 7. For $m = 1$, we get

$$\text{tr}(R^T_1 R_0) = \text{tr}\left( R_0 - \frac{||R_0||^2}{\alpha_1} V_1 \right)^T R_0$$

$$= \text{tr}(R^T_1 R_0) - \frac{||R_0||^2}{\alpha_1} \text{tr}(V^T_1 R_0)$$

$$= ||R_0||^2 - \frac{||R_0||^2}{\alpha_1} = 0,$$

and

$$\text{tr}(U^T_2 V_1) = \text{tr}\left( R_1 + \frac{||R_1||^2}{||R_0||^2} U_1 \right)^T V_1$$

$$= \text{tr}(R^T_1 V_1) + \frac{||R_1||^2}{||R_0||^2} \text{tr}(U^T_1 V_1)$$

$$= -\frac{\alpha_1}{||R_0||^2} \text{tr}(R^T_1 R_1) + \alpha_1 \frac{||R_1||^2}{||R_0||^2} = 0.$$

These imply that (4.3) hold for $m = 1$.

In the inductive step, for $m = k$ we assume that $\text{tr}(R^T_k R_{k-1}) = 0$ and $\text{tr}(U^T_{k+1} V_k) = 0$. Then

$$\text{tr}(R^T_{k+1} R_k) = \text{tr}\left( R_k - \frac{||R_k||^2}{\alpha_{k+1}} V_{k+1} \right)^T R_k$$

$$= \text{tr}(R^T_k R_k) - \frac{||R_k||^2}{\alpha_{k+1}} \text{tr}(V^T_{k+1} R_k)$$

$$= \text{tr}(R^T_k R_k) - \frac{||R_k||^2}{\alpha_{k+1}} \text{tr}(V^T_{k+1} \left(U_{k+1} - \frac{||R_k||^2}{||R_{k-1}||^2} U_k\right))$$

$$= ||R_k||^2 - \frac{||R_k||^2}{\alpha_{k+1}} \text{tr}(V^T_{k+1} U_{k+1})$$

$$= 0,$$

and

$$\text{tr}(U^T_{k+2} V_{k+1}) = \text{tr}\left( R_{k+1} + \frac{||R_{k+1}||^2}{||R_k||^2} U_{k+1} \right)^T V_{k+1}$$

$$= \text{tr}(R^T_{k+1} V_{k+1}) + \frac{||R_{k+1}||^2}{||R_k||^2} \text{tr}(U^T_{k+1} V_{k+1})$$

$$= \text{tr}(R^T_{k+1} \left(-\frac{\alpha_{k+1}}{||R_k||^2} (R_{k+1} - R_k)\right)) + \frac{||R_{k+1}||^2}{||R_k||^2} \alpha_{k+1}$$
Hence, Eq (4.3) holds for any \( m \).

\[ \square \]

**Lemma 9.** Assume that the matrix \( K \) is symmetric. Suppose the sequences \( \{R_i\}, \{U_i\} \) and \( \{V_i\} \) are generated by Algorithm 1. Then

\[
\text{tr} \left( R^T_{m} R_0 \right) = 0, \quad \text{tr} \left( U^T_{m+1} V_1 \right) = 0 \quad \text{for any } m.
\]

**Proof.** From Lemma 8 for \( m = 1 \), we get \( \text{tr}(R^T_1 R_0) = 0 \) and \( \text{tr}(P^T_1 Q_1) = 0 \). Now, suppose that Eq (4.4) is true for all \( m = 1, \ldots, k \). From Lemmas 6 and 7, for \( m = k + 1 \) we write

\[
\text{tr} \left( R^T_{k+1} R_0 \right) = \text{tr} \left( \left[ R_k - \frac{\|R_k\|^2}{\alpha_{k+1}} V_{k+1} \right]^T R_0 \right)
\]

\[
= \text{tr} \left( R^T_k R_0 \right) - \frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}(V^T_{k+1} R_0)
\]

\[
= -\frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}(V^T_{k+1} U_1)
\]

\[
= -\frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}(U^T_{k+1} V_1) = 0,
\]

and

\[
\text{tr} \left( U^T_{k+2} V_1 \right) = \text{tr} \left( V^T_{k+2} U_1 \right)
\]

\[
= \text{tr} \left[ \left( -\frac{\alpha_{k+2}}{\|R_{k+1}\|^2} (R_{k+2} - R_{k+1}) \right)^T U_1 \right]
\]

\[
= -\frac{\alpha_{k+2}}{\|R_{k+1}\|^2} \left[ \text{tr} \left( R^T_{k+2} U_1 \right) - \text{tr} \left( R^T_{k+1} U_1 \right) \right]
\]

\[
= -\frac{\alpha_{k+2}}{\|R_{k+1}\|^2} \left[ \text{tr} \left( R^T_{k+2} R_0 \right) - \text{tr} \left( R^T_{k+1} R_0 \right) \right] = 0.
\]

Hence, Eq (4.4) holds for any \( m \). \[ \square \]

**Theorem 4.4.** Assume that \( K \) is symmetric. Suppose the sequences \( \{R_i\}, \{U_i\} \) and \( \{V_i\} \) are generated by Algorithm 1. Then for any \( m, n \) such that \( m \neq n \), we have

\[
\text{tr} \left( R^T_{m} R_{n-1} \right) = 0 \quad \text{and} \quad \text{tr} \left( U^T_{m} V_n \right) = 0.
\]

**Proof.** By Lemma 7 and the fact that \( \text{tr}(R^T_{m} R_{n-1}) = \text{tr}(R^T_{m-1} R_{n-1}) \) for any \( m, n \), it suffices to prove (4.5) for any \( m, n \) such that \( m > n \). By Lemma 8, Eq (4.5) holds for \( m = n + 1 \). For \( m = n + 2 \), we have

\[
\text{tr} \left( R^T_{m+2} R_n \right) = \text{tr} \left[ \left( R_{m+1} - \frac{\|R_{m+1}\|^2}{\alpha_{m+2}} V_{m+2} \right)^T R_n \right]
\]
\[
\begin{align*}
\text{tr} (R_{n+1}^T R_{n-1}) &= \text{tr} \left( R_n - \frac{||R_n||^2}{\alpha_{n+1}} U_{n+1} \right) R_{n-1} \\
&= -\frac{||R_n||^2}{\alpha_{n+1}} \text{tr} \left[ V_{n+1}^T \left( U_{n+1} - \frac{||R_n||^2}{||R_{n-1}||^2} U_{n-1} \right) \right] \\
&= -\frac{||R_n||^2}{\alpha_{n+1}} \text{tr} \left[ \frac{\alpha_n}{||R_{n-1}||^2} \left( R_n - R_{n-1} \right) \right] \\
&= -\frac{\alpha_n}{||R_{n-1}||^2} \text{tr} \left( U_{n+1} V_n^T \right) - \frac{||R_{n-1}||^2}{||R_{n-2}||^2} \text{tr} \left( V_{n+1}^T U_{n-1} \right) \\
&= -\frac{\alpha_n}{||R_{n-1}||^2} \frac{||R_n||^2}{\alpha_{n+1}} \text{tr} \left( U_{n+1} V_n \right),
\end{align*}
\]
Similarly, we can write \( \text{tr}(R_{n+2}^T R_n) \) and \( \text{tr}(U_{n+2}^T V_n) \) in terms of \( \text{tr}(R_{n+1}^T R_{n-1}) \) and \( \text{tr}(U_{n+1}^T V_{n-1}) \), respectively. Repeating this process until the terms \( \text{tr}(R_2^T R_0) \) and \( \text{tr}(U_2^T V_1) \) show up. By Lemma 9, we get \( \text{tr}(R_{n+2}^T R_n) = 0 \) and \( \text{tr}(U_{n+2}^T V_n) = 0 \).

Next, for \( m = n + 3 \), we have

\[
\text{tr}
\begin{pmatrix}
R_{n+2}^T & \|R_{n+2}\|^2 V_{n+2}
\end{pmatrix}
\begin{pmatrix}
\alpha_{n+3}
\end{pmatrix}
\begin{pmatrix}
R_n
\end{pmatrix}
\]

\[
= -\frac{\alpha_{n-1} \|R_{n-1}\|^2}{\|R_{n-2}\|^2} \left[ \text{tr}\left( V_n^T U_{n-1} \right) - \frac{\|R_{n-2}\|^2}{\|R_{n-3}\|^2} \text{tr}\left( V_n^T U_{n-2} \right) \right]
\]

\[
= -\frac{\alpha_{n-1} \|R_{n-1}\|^2}{\|R_{n-2}\|^2} \frac{\alpha_n}{\|R_{n-3}\|^2} \text{tr}\left( V_n^T U_{n-2} \right),
\]

\[
= \frac{\alpha_{n-1} \|R_{n-1}\|^2}{\|R_{n-2}\|^2} \frac{\alpha_n}{\|R_{n-3}\|^2} \text{tr}\left( U_n^T V_{n-2} \right),
\]

Similarly, we can write \( \text{tr}(R_{n+2}^T R_n) \) and \( \text{tr}(U_{n+2}^T V_n) \) in terms of \( \text{tr}(R_{n+1}^T R_{n-1}) \) and \( \text{tr}(U_{n+1}^T V_{n-1}) \), respectively. Repeating this process until the terms \( \text{tr}(R_2^T R_0) \) and \( \text{tr}(U_2^T V_1) \) show up. By Lemma 9, we get \( \text{tr}(R_{n+2}^T R_n) = 0 \) and \( \text{tr}(U_{n+2}^T V_n) = 0 \).
\[ \text{tr} \left( U_{n+2}^TV_{n-1} \right) = \frac{\alpha_n \|R_{n+1}\|^2 \|R_{n-1}\|^2}{\alpha_{n+2} \|R_{n-2}\|^2} \text{tr} \left( U_{n+2}^TV_{n-1} \right), \]

and

\[ \text{tr} \left( U_{n+2}^TV_{n-1} \right) = \text{tr} \left[ R_{n+1} - \frac{\|R_{n+1}\|^2}{\|R_{n+1}\|^2} U_{n+1} \right]^T V_{n-1} \]
\[ = \text{tr} \left[ R_{n+1}^T \left( -\frac{\alpha_n}{\|R_{n-2}\|^2} R_{n-1} - R_{n-2} \right) \right] \]
\[ = \frac{\alpha_n R_{n+1}}{\|R_{n+1}\|^2} \text{tr} \left[ \left( R_{n+1} - \frac{\|R_{n-2}\|^2}{\alpha_{n+1}} V_{n+1} \right)^T \right] \]
\[ = -\frac{\alpha_n R_{n+1}}{\|R_{n+1}\|^2} \text{tr} \left( V_{n+1}^T U_{n-1} \right) - \frac{\|R_{n-2}\|^2}{\|R_{n-3}\|^2} \text{tr} \left( V_{n+1}^T U_{n-1} \right) \]
\[ = \frac{\alpha_n R_{n+1}}{\|R_{n+1}\|^2} \text{tr} \left( U_{n+1}^TV_{n-2} \right). \]

Hence, we can write \( \text{tr}(R_{n+3}^T R_n) \) and \( \text{tr}(U_{n+3}^TV_n) \) in terms of \( \text{tr}(R_{n+2}^T R_{n-1}) \) and \( \text{tr}(U_{n+2}^TV_{n-1}) \), respectively. Repeating this process until the terms \( \text{tr}(R_3^T R_0) \) and \( \text{tr}(U_4^TV_1) \) by Lemma 9, we get \( \text{tr}(R_{n+3}^T R_n) = 0 \) and \( \text{tr}(U_{n+3}^TV_n) = 0. \)

Suppose that \( \text{tr}(R_{m-1}^T R_{n-1}) = \text{tr}(U_{m-1}^T V_{n-1}) = 0 \) for \( m = n + 1, \ldots, k. \) Then for \( m = k + 1 \), we have

\[ \text{tr} \left( R_k^T R_{n-1} \right) = \text{tr} \left[ R_{k-1} - \frac{\|R_{k-1}\|^2}{\alpha_k} V_k \right]^T R_{n-1} \]
\[ = \text{tr}(R_{k-1}^T R_{n-1}) - \frac{\|R_{k-1}\|^2}{\alpha_k} \text{tr}(V_k^T R_{n-1}) \]
\[ = -\frac{\|R_{k-1}\|^2}{\alpha_k} \text{tr} \left( V_k^T \left( U_n - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} U_{n-1} \right) \right) \]
\[ = -\frac{\|R_{k-1}\|^2}{\alpha_k} \text{tr} \left( V_k^T U_n \right) - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \text{tr} \left( V_k^T U_{n-1} \right) = 0. \]

and

\[ \text{tr} \left( U_{k+1}^TV_{n-1} \right) = \text{tr} \left[ R_k + \frac{\|R_k\|^2}{\|R_{k-1}\|^2} U_k \right]^T V_{n-1} \]
\[ = \text{tr}(R_k^T V_{n-1}) + \frac{\|R_k\|^2}{\|R_{k-1}\|^2} \text{tr}(U_k^T V_{n-1}) \]
\[ = \text{tr} \left[ R_k^T \left( -\frac{\alpha_{n-1}}{\|R_{n-2}\|^2} R_{n-1} - R_{n-2} \right) \right] \]
\[ = -\frac{\alpha_{n-1}}{\|R_{n-2}\|^2} \text{tr} \left( R_k^T R_{n-1} - R_k^T R_{n-2} \right) = 0. \]

Hence, \( \text{tr}(R_{m-1}^T R_{n-1}) = 0 \) and \( \text{tr}(U_{m}^TV_n) = 0 \) for any \( m, n \) such that \( m \neq n. \)

**Theorem 4.5.** Consider Problem 4.1 under the assumption that the matrix \( K \) is symmetric. Suppose that the sequence \( \{X_i\} \) is generated by Algorithm 1. Then for given initial matrix \( X_0 \in \mathbb{R}^{m \times p} \), an exact solution \( X \) can be obtained in at most \( np \) iteration steps.
Proof. Suppose that $R_i \neq 0$ for $i = 0, 1, \ldots, np - 1$. Then we compute $X_{np}$ according to Algorithm 1. Assume that $R_{np} \neq 0$. By Theorem 4.4, the set $\{R_0, R_1, \ldots, R_{np}\}$ is orthogonal in $\mathbb{R}^{nxp}$. So, $\{R_0, R_1, \ldots, R_{np}\}$ is linearly independent. Since the dimension of $\mathbb{R}^{nxp}$ is $np$, any linearly independent subset of $\mathbb{R}^{nxp}$ must have at most $np$ elements. So this is false because the set $\{R_0, R_1, \ldots, R_{np}\}$ has $np + 1$ elements. Thus, $R_{np} = 0$, hence $X_{np}$ is a solution of the equation. □

5. Numerical experiments with discussions

In this section, we report numerical results to illustrate the applicability and the effectiveness of Algorithm 1. All iterations have been carried out by MATLAB R2021a, on a macOS (M1 chip 8C CPU/8C GPU/8GB/512GB). We perform the experiments for several generalized Sylvester-transpose matrix equations, and an interesting special case, namely, the Sylvester equation. We vary given coefficient matrices so that they are square/non-square sparse/dense matrices of moderate/large sizes. The dense matrices considered here are involved a matrix whose all entries are 1, which is denoted by ones. The identity matrix of size $n \times n$ is denoted by $I_n$. For each experiment, we set the stopping rule to be $\|R_k\| \leq \epsilon$ where $\epsilon = 10^{-3}$. We discuss the performance of the algorithm through the norm of residual matrices, iteration number, and computational time (CT). The CT (in seconds) is measured by tic-toc function in MATLAB.

In the following three examples, we concern the applicability of Algorithm 1 as well as its performance comparing to the direct Kronecker linearization mentioned in Section 3.

Example 1. Consider a moderate-scaled generalized Sylvester-transpose equation

$$A_1XB_1 + A_2XB_2 + C_1X^TD_1 + C_2X^TD_2 = E$$

where all matrices are $50 \times 50$ tridiagonal matrices given by

\[
A_1 = \text{tridiag}(-1,2,-1), \quad A_2 = \text{tridiag}(1,-1,1), \quad B_1 = \text{tridiag}(-2,0,-2), \quad B_2 = \text{tridiag}(-2,-1,-2), \\
C_1 = \text{tridiag}(0,2,0), \quad C_2 = \text{tridiag}(1,2,1), \quad D_1 = \text{tridiag}(0,-4,0), \quad D_2 = \text{tridiag}(-2,-4,-2), \\
E = \text{tridiag}(-1,1,9).
\]

We run Algorithm 1 using an initial matrix $X_0 = 0.25 \times \text{ones} \in \mathbb{R}^{50\times50}$. According to Theorem 4.5, Algorithm 1 will produce a solution of the equation within $10^4$ iterations. The resulting simulation illustrated in Figure 1 shows the norms of residual matrices $R_k$ at each iteration.
Although the errors $\|R_k\|$ grow up and down during iterations, they generally climb down to zero. The algorithm takes 138 iterations to get a desire solution (so that $\|R_k\| \leq 10^{-3}$), which is significantly less than the theoretical one ($10^4$ iterations). For the computational time, Algorithm 1 spends totally 0.131079 seconds to get a desire solution, while the direct Kronecker linearization consuming 1.581769 seconds to obtain the exact solution. Thus, the performance of Algorithm 1 is significantly better than the direct method. Moreover, for sparse coefficient matrices, Algorithm 1 can produce a desire solution in a fewer iterations (that is, 138 iterations) than the theoretical one (that is, $10^4$ iterations in this case).

**Example 2.** Consider a generalized Sylvester-transpose matrix equation

$$A_1XB_1 + A_2XB_2 + A_3XB_3 + C_1X^TD_1 = E$$

with rectangular coefficient matrices of moderate-scaled as follows:

- $A_1 = \text{tridiag}(1, 3, 1), \ A_2 = \text{tridiag}(-1, 2, -1), \ A_3 = \text{tridiag}(-1, 1, -1) \in \mathbb{R}^{40 \times 40}$,
- $B_1 = \text{tridiag}(-2, 1, -2), \ B_2 = \text{tridiag}(1, -3, 1), \ B_3 = \text{tridiag}(2, -3, 2) \in \mathbb{R}^{50 \times 50}$,
- $C_1 = 3 \times \text{ones} \in \mathbb{R}^{40 \times 50}, \ D_1 = -3 \times \text{ones} \in \mathbb{R}^{40 \times 50}, \ E = -0.9 \times \text{ones} \in \mathbb{R}^{40 \times 50}$.

Taking an initial matrix $X_0 \in \mathbb{R}^{40 \times 50}$, we get an approximate solution $X_k \in \mathbb{R}^{40 \times 50}$ with a satisfactory error $\|R_k\| \leq 10^{-3}$ in 164 steps, using 0.196250 seconds. We see in Figure 2 that during iterations, although the errors $\|R_k\|$ grow up and down, they generally climb down to zero. On the other hand, the direct Kronecker linearization consumes 0.811170 seconds to get an exact solution. Thus, Algorithm 1 is applicable and effective.
Example 3. Consider a large-scaled generalized Sylvester-transpose equation

\[ A_1XB_1 + C_1X^TD_1 + C_2X^TD_2 = E \]

where all matrices are 100 \times 100 tridiagonal matrices given by

\[
    A_1 = \text{tridiag}(-2, -6, -2), \quad B_1 = \text{tridiag}(2, -1, 2), \quad C_1 = \text{tridiag}(0, -1, 0), \quad C_2 = \text{tridiag}(-1, 2, -1), \\
    D_1 = \text{tridiag}(0, 2, 0), \quad D_2 = \text{tridiag}(2, -4, 2), \quad E = \text{tridiag}(1, -8, 1).
\]

The resulting simulation of Algorithm 1 using an initial matrix \( X_0 = 0.5 \times \text{ones} \in \mathbb{R}^{100 \times 100} \) is shown in the next figure.

Figure 3 shows the error gradually decreasing into \( \epsilon = 10^{-3} \) in 774 steps, consuming around 2 seconds.
Next, we investigate the effect of changing initial points. So we make experiments for the initial matrices $X_0 = 5 \times \text{ones}$, $X_0 = 0$, and $X_0 = -5 \times \text{ones}$. Table 1 shows that, no matter the initial point, we get a desire solution in around 2 seconds. On the other hand, the direct method consumes around 70 seconds to get an exact solution. Thus, Algorithm 1 significantly outperforms the direct method.

| Initial matrix | Iterations | CT     | Relative error |
|----------------|------------|--------|----------------|
| Direct         | -          | 69.500953 | 0              |
| $X_0 = 5 \times \text{ones}$ | 830        | 2.247507      | $8.9145 \times 10^{-4}$ |
| $X_0 = 0.5 \times \text{ones}$ | 774        | 1.986782      | $7.5755 \times 10^{-4}$ |
| $X_0 = 0$       | 16         | 0.106482      | $5.3862 \times 10^{-4}$ |
| $X_0 = -5 \times \text{ones}$ | 830        | 2.190269      | $9.1232 \times 10^{-4}$ |

In the rest of numerical examples, we compare the performance of Algorithm 1 to the direct method as well as recent gradient-based iterative algorithms mentioned in Introduction.

**Example 4.** Consider a large-scaled generalized Sylvester-transpose matrix equation

$$AXB + CX^TD = E,$$

where $A, B, C, D, E$ are $100 \times 100$ matrices as follows:

$$A = \text{tridiag}(-1, 3, -1), \quad B = \text{tridiag}(1, 7, 1), \quad C = 6 \times \text{ones}, \quad D = -3 \times \text{ones}, \quad E = 0.7 \times I_{100}.$$
In fact, this equation has a unique solution. Despite the direct method, we compare the performance of Algorithm 1 to GI [23] and AGBI [24] algorithms. All iterative algorithms are implemented using the initial $X_0 = -0.001 \times I_{100} \in \mathbb{R}^{100\times100}$.

According to [23], the GI algorithm is applicable as long as a convergent factor $\mu$ satisfies

$$0 < \mu < \frac{2}{\lambda_{\text{max}}(A^T A) \lambda_{\text{max}}(B^T B) + \lambda_{\text{max}}(C^T C) \lambda_{\text{max}}(D^T D)},$$

where $\lambda_{\text{max}}(A^T A)$ is the largest eigenvalue of $A^T A$. We run GI algorithm under 3 different convergent factors, namely, $m_1 = 6.1728 \times 10^{-12}$, $m_2 = 8.8183 \times 10^{-12}$ and $m_3 = 3.0864 \times 10^{-11}$. We implement AGBI algorithm with a convergent factor $0.000988$ and a weighted factor $10^{-8}$. Figure 4 shows that the CG algorithm (Algorithm 1) converges faster than GI with $m_1$, GI with $m_2$, GI with $m_3$, and AGBI algorithms. Table 2 shows that, in 30 iterations, GI, AGBI and the direct method consume a big amount of time to get the exact solution, while Algorithm 1 produces a small-error solution in a small time (0.073613 seconds).

Table 2. Relative error and CTs for Example 4.

| Method     | Iterations | CT      | Relative error |
|------------|------------|---------|----------------|
| CG         | 30         | 0.073613| 0.000001       |
| GI with $m_1$ | 30         | 0.109370| 18.788266      |
| GI with $m_2$ | 30         | 0.115890| 16.853503      |
| GI with $m_3$ | 30         | 0.109668| 16.724640      |
| AGBI       | 30         | 0.118294| 16.724926      |
| Direct     | -          | 66.928143| 0              |
Example 5. Consider a consistent generalized Sylvester-transpose matrix equation

\[ AXB + CX^TD = E, \]

with 100 × 100 coefficient matrices:

\[ A = \text{tridiag}(-1, 2, -1), \quad B = \frac{1}{3} \times \text{ones}, \quad C = -3 \times \text{ones}, \quad D = \text{tridiag}(3, -6, 3), \quad E = -1.2 \times \text{ones}. \]

In fact, this matrix equation has a solution, which is not unique. We will seek for a solution of the equation using Algorithm 1, GI and AGBI algorithms with the same initial matrix \( X_0 = -0.4 \times \text{ones} \in \mathbb{R}^{100 \times 100} \).

![Figure 5. Relative error for Example 5.](image)

### Table 3. Relative error and CTs for Example 5.

| Method       | Iterations | CT       | Relative error |
|--------------|------------|----------|----------------|
| CG           | 200        | 0.395984 | 0.361597       |
| GI with m1   | 200        | 0.654457 | 1473.481117    |
| GI with m2   | 200        | 0.591405 | 1186.764341    |
| GI with m3   | 200        | 0.595052 | 645.799529     |
| AGBI         | 200        | 0.599059 | 1718.220885    |

We carry out GI algorithm with three different convergent factors, namely, \( m_1 = 1.7132 \times 10^{-8}, \ m_2 = 3.0837 \times 10^{-8} \ and \ m_3 = 1.5418 \times 10^{-7} \). We implement AGBI algorithm with the convergent factor 0.000112 and the weighted factor 0.00005. Table 3 and Figure 5 express the computational time and the errors for 200 iterations of the simulations. We see that the computational time of CG algorithm...
is slightly less than those of GI (with parameters \(m_1, m_2, m_3\)) and AGBI algorithms. However, the outcome error produced by CG algorithm is significantly less than those of other algorithms.

**Example 6.** Consider the following Sylvester matrix equation

\[
AX + XB = C,
\]

where all coefficient matrices are \(100 \times 100\) tridiagonal matrices given by

\[
A = \text{tridiag}(1, -6, 1), \quad B = \text{tridiag}(3, 0, 3), \quad C = \text{tridiag}(1, 1, 9).
\]

![Figure 6. Relative error for Example 6.](image)

**Table 4.** Relative error and CTs for Example 6.

| Method | Parameters | Iterations | CT   | Relative error |
|--------|------------|------------|------|----------------|
|        |            |            |      |                |
| CG     | -          | -          | 10   | 0.018412       | 0.000000         |
| GI     | \(\mu = 0.034482\) | - | 10   | 0.019581       | 7.365534         |
| RGI    | \(\mu = 0.266489\) | \(\omega = 0.05\) | 10   | 0.015338       | 8.786233         |
| MGI    | \(\mu = 0.025316\) | -          | 10   | 0.019361       | 2.544848         |
| AGBI   | \(\mu = 0.233918\) | \(\omega = 0.05\) | 10   | 0.022275       | 8.791699         |

We compare the performance of CG algorithm (Algorithm 1) to GI [23], RGI [16], MGI [17] and AGBI [24] algorithms with parameters as shown in Table 4. We implement the algorithms with the same initial matrix \(X_0 = -5 \times \text{ones} \in \mathbb{R}^{100 \times 100}\). Table 4 shows that the computational times for
implementing 30 iterations of CG and other algorithms are close together. However, the relative errors in Figure 6 and Table 4 express that CG algorithm produces a sequence of well-approximate solutions in a few iterations with the lowest error comparing to other GI algorithms.

6. Conclusions

We propose an iterative procedure (Algorithm 1) to construct a sequence of approximate solutions for the generalized Sylvester-transpose matrix Eq (1.5) with rectangular coefficient matrices. The algorithm is applicable whenever the matrix $K$, defined by Eq (3.2), is symmetric. In fact, the residual matrices $R_k$, produced by the algorithm, form an orthogonal set with respect to the usual inner product for matrices. Thus, we obtain the desire solution within a finite step, says, $np$ steps. Numerical simulations have verified the applicability of the algorithm for square/non-square sparse/dense matrices of moderate/large sizes. The algorithm is always applicable no matter how we choose an initial matrix. Moreover, for sparse coefficient matrices of large size, the iteration number to get a desire solution can be dramatically less than $np$ iterations. The performance of the algorithm is significantly better than the direct Kronecker linearization and recent gradient-based iterative algorithms when the matrix coefficients are of moderate/large sizes.

Acknowledgments

This research project is supported by National Research Council of Thailand (NRCT): (N41A640234).

Conflict of interest

All authors declare that they have no conflict of interest.

References

1. Y. Kim, H. S. Kim, J. Junkins, Eigenstructure assignment algorithm for second order systems, J. Guid. Control Dyn., 22 (1999), 729–731. http://dx.doi.org/10.2514/2.4444
2. B. Zhou, G. R. Duan, On the generalized Sylvester mapping and matrix equations, Syst. Control Lett., 57 (2008), 200–208. http://dx.doi.org/10.1016/j.sysconle.2007.08.010
3. L. Dai, Singular control systems, Berlin: Springer, 1989.
4. G. R. Duan, Eigenstructure assignment in descriptor systems via output feedback: A new complete parametric approach, Int. J. Control., 72 (1999), 345–364. http://dx.doi.org/10.1080/002071799221154
5. F. Lewis, A survey of linear singular systems, Circ. Syst. Signal Process., 5 (1986), 3–36. http://dx.doi.org/10.1007/BF01600184
6. G. R. Duan, Parametric approaches for eigenstructure assignment in high-order linear systems, Int. J. Control Autom. Syst., 3 (2005), 419–429.
7. K. Nouri, S. Beik, L. Torkzadeh, D Baleanu, An iterative algorithm for robust simulation of the Sylvester matrix differential equations, Adv. Differ. Equ., 2020 (2020), http://dx.doi.org/10.1186/s13662-020-02757-z

8. M. Epton, Methods for the solution of $AXD - BXC = E$ and its applications in the numerical solution of implicit ordinary differential equations, BIT, 20 (1980), 341–345. http://dx.doi.org/10.1007/BF01932775

9. D. Hyland, D. Bernstein, The optimal projection equations for fixed order dynamic compensation, IEEE Trans. Control., 29 (1984), 1034–1037. http://dx.doi.org/10.1109/TAC.1984.1103418

10. D. Calvetti, L. Reichel, Application of ADI iterative methods to the restoration of noisy images, SIAM J. Matrix Anal. Appl., 17 (1996), 165–186. http://dx.doi.org/10.1137/S0895479894273687

11. M. Dehghan, A. Shirilord, A generalized modified Hermitian and skew-Hermitian splitting (GMHSS) method for solving complex Sylvester matrix equation, Appl. Math. Comput., 348 (2019), 632–651. http://dx.doi.org/10.1016/j.amc.2018.11.064

12. S. Y. Li, H. L. Shen, X. H. Shao, PHSS Iterative method for solving generalized Lyapunov equations, Mathematics, 7 (2019), 38. http://dx.doi.org/10.3390/math7010038

13. H. L. Shen, Y. R. Li, X. H. Shao, The four-parameter PSS method for solving the Sylvester equation, Mathematics, 7 (2019), 105. http://dx.doi.org/10.3390/math7010105

14. M. Dehghan, A. Shirilord, Solving complex Sylvester matrix equation by accelerated double-step scale splitting (ADSS) method, Engineering with Computers, 37 (2021), 489–508. http://dx.doi.org/10.1007/s00366-019-00838-6

15. F. Ding, T. Chen, Gradient based iterative algorithms for solving a class of matrix equations, IEEE Trans. Automat. Contr., 50 (2005), 1216–1221. http://dx.doi.org/10.1109/TAC.2005.852558

16. Q. Niu, X. Wang, L. Z. Lu, A relaxed gradient based algorithm for solving Sylvester equation, Asian J. Control, 13 (2011), 461–464. http://dx.doi.org/10.1002/asjc.328

17. X. Wang, L. Dai, D. Liao, A modified gradient based algorithm for solving Sylvester equation, Appl. Math. Comput., 218 (2012), 5620–5628. http://dx.doi.org/10.1016/j.amc.2011.11.055

18. Z. Tian, M. Tian, C. Gu, X. Hao, An accelerated Jacobi-gradient based iterative algorithm for solving Sylvester matrix equations, Filomat, 31 (2017), 2381–2390. http://dx.doi.org/10.2298/FIL1708381T

19. N. Sasaki, P. Chansangiam, Modified Jacobi-gradient iterative method for generalized Sylvester matrix equation, Symmetry, 12 (2020), 1831. http://dx.doi.org/10.3390/sym12111831

20. X. Zhang, X. Sheng, The relaxed gradient based iterative algorithm for the symmetric (skew symmetric) solution of the Sylvester equation $AX + XB = C$, Math. Probl. Eng., 2017 (2017), 1624969. http://dx.doi.org/10.1155/2017/1624969

21. A. Kittisopaporn, P. Chansangiam, W. Lewkeeratiyukul, Convergence analysis of gradient-based iterative algorithms for a class of rectangular Sylvester matrix equation based on Banach contraction principle, Adv. Differ. Equ., 2021 (2021), 17. http://dx.doi.org/10.1186/s13662-020-03185-9
22. N. Boonruangkan, P. Chansangiam, Convergence analysis of a gradient iterative algorithm with optimal convergence factor for a generalized Sylvester-transpose matrix equation, *AIMS Mathematics*, 6 (2021), 8477–8496. http://dx.doi.org/10.3934/math.2021492

23. L. Xie, J. Ding, F. Ding, Gradient based iterative solutions for general linear matrix equations, *Comput. Math. Appl.*, 58 (2009), 1441–1448. http://dx.doi.org/10.1016/j.camwa.2009.06.047

24. Y. J. Xie, C. F. Ma, The accelerated gradient based iterative algorithm for solving a class of generalized Sylvester transpose matrix equation, *Appl. Math. Comp.*, 273 (2016), 1257–1269. http://dx.doi.org/10.1016/j.amc.2015.07.022

25. A. Kittisopaporn, P. Chansangiam, Gradient-descent iterative algorithm for solving a class of linear matrix equations with applications to heat and Poisson equations, *Adv. Differ. Equ.*, 2020 (2020), 324. http://dx.doi.org/10.1186/s13662-020-02785-9

26. A. Kittisopaporn, P. Chansangiam, The steepest descent of gradient-based iterative method for solving rectangular linear system with an application to Poisson’s equation, *Adv. Differ. Equ.*, 2020 (2020), 259. http://dx.doi.org/10.1186/s13662-020-02715-9

27. Y. Qi, L. Jin, H. Li, Y. Li, M. Liu, Discrete computational neural dynamics models for solving time-dependent Sylvester equation with applications to robotics and MIMO systems, *IEEE Trans. Ind. Inform.*, 16 (2020), 6231–6241. http://dx.doi.org/10.1109/TII.2020.2966544

28. V. Simoncini, Computational methods for linear matrix equations, *SIAM Rev.*, 58 (2016), 377–441. http://dx.doi.org/10.1137/130912839

29. H. Zhang, H. Yin, Refinements of the Hadamard and Cauchy Schwarz inequalities with two inequalities of the principal angles, *J. Math. Inequal.*, 13 (2019), 423–435. http://dx.doi.org/10.7153/jmi-2019-13-28

30. H. Zhang, Quasi gradient-based inversion-free iterative algorithm for solving a class of the nonlinear matrix equations, *Comput. Math. Appl.*, 77 (2019), 1233–1244. http://dx.doi.org/10.1016/j.camwa.2018.11.006

31. H. Zhang, L. Wan, Zeroing neural network methods for solving the Yang-Baxter-like matrix equation, *Neurocomputing*, 383 (2020), 409–418. http://dx.doi.org/10.1016/j.neucom.2019.11.101

32. F. Ding, G. Liu, X. Liu, Parameter estimation with scarce measurements, *Automatica*, 47 (2011), 1646–1655. http://dx.doi.org/10.1016/j.automatica.2011.05.007

33. F. Ding, Y. Liu, B. Bao, Gradient based and least squares based iterative estimation algorithms for multi-input multi-output systems, *P. I. Mech. Eng. I-J. Sys.*, 226 (2012), 43–55. http://dx.doi.org/10.1177/0959651811409491

34. F. Ding, Combined state and least squares parameter estimation algorithms for dynamic systems, *Appl. Math. Model.*, 38 (2014), 403–412. http://dx.doi.org/10.1016/j.apm.2013.06.007

35. M. Hajarian, Developing BiCG and BiCR methods to solve generalized Sylvester-transpose matrix equations, *Int. J. Autom. Comput.*, 11 (2014), 25–29. http://dx.doi.org/10.1007/s11633-014-0762-0

36. M. Hajarian, Matrix form of the CGS method for solving general coupled matrix equations, *Appl. Math. Lett.*, 34 (2014), 37–42. http://dx.doi.org/10.1016/j.aml.2014.03.013
37. Y. F. Ke, C. F. Ma, A preconditioned nested splitting conjugate gradient iterative method for the large sparse generalized Sylvester equation, *Appl. Math. Comput.*, 68 (2014), 1409–1420. http://dx.doi.org/10.1016/j.camwa.2014.09.009

38. M. Hajarian, Generalized conjugate direction algorithm for solving the general coupled matrix equations over symmetric matrices, *Numer. Algor.*, 73 (2016), 591–609. http://dx.doi.org/10.1007/s11075-016-0109-8

39. M. Hajarian, Extending the CGLS algorithm for least squares solutions of the generalized Sylvester-transpose matrix equations, *J. Franklin Inst.*, 353 (2016), 1168–1185. http://dx.doi.org/10.1016/j.jfranklin.2015.05.024

40. M. Dehghan, R. Mohammadi-Arani, Generalized product-type methods based on Bi-conjugate gradient (GPBiCG) for solving shifted linear systems, *Comput. Appl. Math.*, 36 (2017), 1591–1606. http://dx.doi.org/10.1007/s40314-016-0315-y

41. R. Horn, C. Johnson, *Topics in matrix analysis*, Cambridge: Cambridge University Press, 1991. http://dx.doi.org/10.1017/CBO9780511840371

©2022 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0)