EXACT AND LEAST-SQUARES SOLUTIONS OF A GENERALIZED SYLVESTER-TRANSPOSE MATRIX EQUATION VIA CONJUGATE GRADIENT ALGORITHMS

KANJANAPORN TANSRI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN APPLIED MATHEMATICS
DEPARTMENT OF MATHEMATICS SCHOOL OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2024

KMITL-2024-SC-D-001-009

This material is reserved for educational use. Not approved for commercial use.
Forbidden to modify the content, and cite the document when use.

COPYRIGHT 2024
SCHOOL OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
This thesis has the author's intention for non-commercial use.
Forbidden to modify the content, and cite the document when use.

Thesis Title	Exact and Least-Squares Solutions of a Generalized Sylvester-Transpose Matrix Equation via Conjugate Gradient Algorithms
Student Name	Miss Kanjanaporn Tansri
Student ID	63605009
Degree	Doctor of Philosophy (Applied Mathematics)
Department	Mathematics
Year	2024
Thesis Advisor	Assoc. Prof. Dr. Pattrawut Chansangiam

Abstract

In this research, we propose effective conjugate gradient algorithms for solving a generalized Sylvester-transpose matrix equation, where all coefficient matrices and an unknown matrix are rectangular. When the matrix equation is consistent, we derive an iterative procedure to find its exact solution. For an inconsistent case, the algorithm seeks for a least-squares solution. Moreover, if the matrix equation has many least-squares solutions, the algorithm can search for the one with minimal Frobenius-norm or find the one closest to any given matrix. For any given initial matrix, the iterative algorithms produce a finite sequence of well-approximate solutions, so that the desire solution comes out within a finite number of iterations in the absence of roundoff error. Finally, we provide numerical experiments to illustrate the capabilities and efficiency the algorithms for square/non-square dense/sparse matrices of medium/large sizes. It turns out that the proposed algorithms perform well with the direct Kronecker linearization and well-known iterative methods in both errors and computational times.

Keywords : conjugate gradient agorithm, generalized Sylvester-transpose matrix equation, Kronecker product, least-squares solution, matrix norm, orthogonality.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Acknowledgements

I would like to express my special thanks of gratitude to all of those whom made this thesis becomes a reality with their kind supports and helps.

Foremost, I would like to express my special thanks of gratitude to my advisor, Assoc.Prof.Dr. Pattrawut Chansangiam, who gives me a lot of knowledge, motivation, understanding, constant guidance and support to do this thesis on the topic Exact and Least-Squares solutions of a Generalized Sylvester-Transpose Matrix Equation via Conjugate Gradient Algorithms, which also helped me in all the time of study and doing a lots of this thesis.

In addition, I would also like to acknowledge my thesis committee members, Assoc.Prof.Dr. Sekson Sirisubtawee, Assoc.Prof.Dr. Atid Kangtunyakarn, Assoc.Prof.Dr. Chartchai Leenawong, and Assoc.Prof.Dr. Nopparat Pochai for their helpful comments and suggestion on countless revisions of this manuscript.

I am sincerely grateful to financial supports during 2020-2021 from School of Science, King Mongkut's Institute of Technology Ladkrabang, Grant No. RA/TA-2563-D-004. I also acknowledge financial supports from Thailand Research Fund. My completion of this thesis could not have been accomplished without the financial support the Royal Golden Jubilee (RGJ) Ph.D. Scholarship from the National Research Council of Thailand (NRCT), Grant No. N41A640234. Additionally, this scholarship gives me a special occasion to study research at the University of Manitoba, Canada, for 8 months under the supervision of Professor Yang Zhang.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Kanjanaporn Tansri

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Table of Contents

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

List of Tables

Table **Page**

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 1

Introduction

1.1 Research Motivation

The inspiration for this work came from the fact that we found Sylvester-type matrix equations that appeared many disciplines, including mathematics and engineering. The following Sylvester-type equations include

$$AX + XA^T = B \text{ (the Lyapunov matrix equation),}$$
$$AX + XB = C \text{ (the Sylvester matrix equation),}$$
$$X + AXB = E \text{ (the Stein matrix equation),}$$

and other realted equations; see e.g. [1, 2, 3, 4, 5, 6]. Such matrix equations play an important role in many problems in computational mathematics, signal processing and model reduction, control and system theory, robust simulation, and neural network; see e.g. [7, 8, 9]. The matrix equations mentioned above are all special cases that can be written in the form of a generalized Sylvester-transpose matrix equation:

$$AXB + CX^TD = E, \tag{1.1}$$

or more generally

$$\sum_{i=1}^{s} A_i X B_i + \sum_{j=1}^{t} C_j X^T D_j = E. \tag{1.2}$$

where $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$ for any $i = 1, 2, \ldots, s$, $j = 1, 2, \ldots, t$ and $E \in \mathbb{R}^{m\times q}$ are given matrices and $X \in \mathbb{R}^{n\times p}$ is matrix to be determined.

Normally, we solve for an unknown matrix of the equation (1.2) by reducing it to an equivalent linear system, using the vector operator and the Kronecker linearization. Well-known traditional direct methods (e.g., Gaussian Elimination, Inversion Matrix, Cramer's Rule, and LU Factorization) are used to solve the linear system. We obtain the unknown matrix of the matrix equation simply by using the injectivity of the vector operators. However, this is a numerically poor way to determine the solution of Eq. (1.2), as the size of coefficient matrices are moderate or large and may be ill-conditioned. So this approach is only applicable for small sized matrix equations. When the coefficient matrices are moderate or large size, to save the time and reduce computer memory it would be more appropriate to use an iterative method to find a well-approximate solution.

In the literature review, It was found that three popular iterative techniques used to solve for linear matrix equations. The first group of techniques is called Hermitian and skew-Hermitian splitting (HSS) methods. In last 5 years, HSS methods have

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

evolved in various several, such as preconditioned HSS [10], GMHSS [11], FPPSS [12], and ADSS [13]. The second technique is one that creates a sequence of estimated solutions from the gradient of the associated norm-error function. The generated sequence will converges on the desire solution, when we cautiously set the parameters of the gradient-based iteration algorithm. This technique is called gradient-based iterative (GI) algorithms. The original gradient-based iterative method was introduced in [14] for solving Eq. (1.1). Next, the GI method was modified by introducing weighted factor [15]. After that, a half-step update of GI method, called MGI method [16]. For another idea of gradient-based iterative algorithm have been introduced such as accelerated Jacobi GI or AJGI method [17] and modified Jacobi GI or MJGI method [18]. There are many algorithms in the group of GI techniques to find solutions of Eq. (1.1), such as GI algorithm [19], accelerated gradient-based iterative (AGBI) algorithm [20], and other GI techniques based on optimization [21, 22, 23]. In addition, there are GI algorithm to solve for least-squares solutions of Sylvester-type matrix equations, e.g. [24].

The last famous technique, known as conjugate gradient (CG) technique, aims to construct associated residual matrices orthogonal. Thus, we get an approximate sequence of exact/least-squares solutions within finite steps. In the last decade, many researchers developed CG algorithms for solve Eq. (1.2) and its special cases, e.g., Bi-conjugate gradient (BiCG) method [25], Bi-conjugate residual (BiCR) method [25], conjugate gradients squared (CGS) method [26], preconditioned nested splitting CG [27], generalized conjugate direction (GCD) method [28], conjugate gradient least-squares (CGLS) method [29], and generalized product-type based on Bi-conjugate gradient (GP-BiCG) method [30].

In this research, we propose conjugate gradient iterative algorithms for solving the generalized Sylvester-transpose matrix equation (1.2). We will divide our consideration into two cases. When Eq. (1.2) is consistent and has a unique solution, we derive an iterative algorithm to find a well-approximate exact solution as follows:

Problem 1. Assume that Eq. (1.2) has a solution. Given a small error ϵ, find $\tilde{X} \in \mathbb{R}^{n\times p}$ such that

$$\left\| E - \sum_{i=1}^{s} A_i \tilde{X} B_i - \sum_{j=1}^{t} C_j \tilde{X}^T D_j \right\|_F \; < \; \epsilon.$$

On the other hand, when Eq. (1.2) is inconsistent, we propose an algorithm to approximate least-squares solution(s). When Eq. (1.2) has a unique least-squares solution, the algorithm will solve the following problem:

Problem 2. Find a matrix $\hat{X} \in \mathbb{R}^{n\times p}$ that

$$\text{minimizes} \quad \|E - \sum_{i=1}^{s} A_i \hat{X} B_i - \sum_{j=1}^{t} C_j \hat{X}^T D_j\|_F.$$

When Eq. (1.2) has many least-squares solutions, let us denote by $\mathcal{L}$ the set of least-squares solutions of Eq. (1.2). The algorithm also solves for two following problems.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

The first one is to find a least-squares solution with the minimal norm:

Problem 3. Find the matrix X^* such that

$$\|X^*\|_F = \min_{\hat{X}\in\mathcal{L}} \|\hat{X}\|_F.$$

And the second one is to find a least-squares solution closest to a given matrix:

Problem 4. Let $Y \in \mathbb{R}^{n\times p}$. Find the matrix $\check{X}$ such that

$$\|\check{X} - Y\|_F = \min_{\hat{X}\in\mathcal{L}} \|\hat{X} - Y\|_F.$$

Moreover, We also examine the results the algorithm the prove that such each algorithms can obtain a desire solution for any initial matrix within finite steps. Numerical experiments are provided to illustrate the capability and performance of the proposed algorithms comparison with well-known algorithms and traditional direct method.

1.2 Objectives of the Study

1) Propose a CG-type algorithm to solve for the exact solution of the generalized Sylvester-transpose matrix equation.
2) Propose a CG-type algorithm to solve for a least-squares solution of the generalized Sylvester-transpose matrix equation.
3) Investigate the convergent property of the proposed algorithms.
4) Provide numerical simulations comparing to well-known algorithms and recent algorithms.

1.3 Scope of the Study

The main system we considered here is the following matrix equation

$$\sum_{i=1}^{s} A_i X B_i + \sum_{j=1}^{t} C_j X^T D_j = E. \tag{1.2}$$

Here, coefficient matrices $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$, $E \in \mathbb{R}^{m\times q}$ are given, and a rectangular matrix $X \in \mathbb{R}^{n\times p}$ is unknown. We consider two cases of Eq. (1.2)

1) Eq. (1.2) is consistent, we look for an exact solution X^*.
2) Eq. (1.2) is inconsistent, we look for
   2.1) Eq. (1.2) has a unique least-squares solution
   2.2) Eq. (1.2) has many least-squares solution, we solve
   - least-squares solution with minimal Frobenius-norm,
   - least-squares solution closest to a given matrix.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

1.4 Benefits of the Study

Attain new effective algorithms to solve the generalized Sylvester-transpose matrix equation.

1.5 Research Methodology

1) Study advanced topics in matrix theory and numerical linear algebra.
2) Study Sylvester-transpose matrix from textbooks and research papers.
3) Study iterative methods for solving linear matrix equations such as conjugate gradient algorithms, gradient-based iterative algorithms from research papers.
4) Propose a new conjugate gradient iterative algorithm to solve for the exact solution of the generalized Sylvester-transpose matrix.
5) Analyze a convergence of the first algorithm and provide numerical simulations.
6) Propose a new conjugate gradient iterative algorithm to solve for a least-squares solution of the generalized Sylvester-transpose matrix.
7) Analyze a convergence of the second algorithm and provide numerical simulations.
8) Combine and summarize all findings then write the thesis and make suggestions for further studies.

table1 - The research schedule

Table 1.1: The research schedule

Activity	Time frame						
	2020	2021		2022		2023	2024
	Aug-Dec.	Jan.-Jun.	Jul.-Dec.	Jan.-Jun.	Jul.-Dec.	Jan.-Jun.	Mar.-Jun.
Step 1							
Step 2							
Step 3							
Step 4							
Step 5							
Step 6							
Step 7							
Step 8							

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 2

Preliminaries

In this chapter, we introduce important tools that will be useful later in deriving and analyzing iterative algorithms for solving matrix equations. Throughout, we denote the set of all m-by-n real matrices by $\mathbb{R}^{m\times n}$.

2.1 Auxiliary results from matrix theory

Definition 2.1. Let $A=[a_{ij}]\in\mathbb{R}^{m\times n}$ and $B=[b_{ij}]\in\mathbb{R}^{p\times q}$. The Kronecker product of A and B is defined by

$$A\otimes B=\begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}\in\mathbb{R}^{mp\times nq}.$$

Lemma 2.2 (see, e.g., [31])**.** 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) $A\otimes(B+C) = A\otimes B+A\otimes C$.

Definition 2.3. The vector operator $\mathrm{Vec}(\cdot)$ assigns to each matrix $A=[a_{ij}]\in\mathbb{R}^{m\times n}$ the column vector

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

Lemma 2.4 (see, e.g., [31])**.** Let A and B be compatibly constant matrices. The properties of the vector operator are as follows:

1) $\mathrm{Vec}(\alpha A)=\alpha\,\mathrm{Vec}\,A$, α is a constant.

2) $\mathrm{Vec}(A+B)=\mathrm{Vec}\,A+\mathrm{Vec}\,B$.

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

Lemma 2.5 (see, e.g., [31])**.** 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

$$\mathrm{Vec}\,AXB = (B^T\otimes A)\,\mathrm{Vec}\,X.$$

Definition 2.6. Let $A=[a_{ij}]\in\mathbb{R}^{m\times m}$. The trace of matrix A is denoted by $\mathrm{tr}(A)$ and defined to be

$$\mathrm{tr}(A)=\sum_{i=1}^{m}a_{ii}.$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Lemma 2.7 (see, e.g., [31]). Let A and B be compatibly constant matrices. The properties of trace of matrix are as follows:

1) $\mathrm{tr}(A+B) = \mathrm{tr}(A) + \mathrm{tr}(B)$,

2) $\mathrm{tr}(\alpha A) = \alpha\,\mathrm{tr}(A)$, α is a constant.

3) $\mathrm{tr}(AB) = \mathrm{tr}(BA)$.

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

$$P(m,n) \;=\; \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij} \otimes E_{ij}^{T} \;\in\; \mathbb{R}^{mn\times mn} \tag{2.1}$$

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 2.8 (see, e.g., [31]). For any matrices $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, $X \in \mathbb{R}^{q\times n}$ and $Y \in \mathbb{R}^{p\times m}$ of compatible dimensions, we have

1) $B \otimes A \;=\; P(n,p)^{T}\,(A\otimes B)\,P(n,q)$,

2) $\mathrm{Vec}(A^{T}) \;=\; P(m,n)\,\mathrm{Vec}(A)$,

3) $(\mathrm{Vec}\,(Y))^{T}\,(A\otimes B)\,\mathrm{Vec}\,(X) \;=\; \mathrm{tr}\,(A^{T}Y^{T}BX)$.

Definition 2.9. The standard (Frobenius) inner product of $A, B \in \mathbb{R}^{m\times n}$ is defined by

$$\langle A, B\rangle \;:=\; \mathrm{tr}(B^{T}A) \;=\; \mathrm{tr}(AB^{T}).$$

If $\langle A, B\rangle = 0$, we say that A is orthogonal to B.

Lemma 2.10 (see, e.g., [31]). For any matrices A, B, C, D are the corresponding dimensions. A well-known property of the inner product is that

$$\langle A, BCD\rangle \;=\; \langle B^{T}AD^{T}, C\rangle.$$

Definition 2.11. The Frobenius norm of matrix $A \in \mathbb{R}^{m\times n}$ is defined by

$$\|A\| \;=\; \left(\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^{2}\right)^{\frac{1}{2}} \;=\; \left(\mathrm{tr}\,(A^{T}A)\right)^{\frac{1}{2}}.$$

2.2 Theory of linear systems

We consider the linear system

$$Ax = b \tag{2.2}$$

where $A \in \mathbb{R}^{m\times n}$ is a known constant matrix, $b \in \mathbb{R}^{m\times 1}$ is a known constant vector, and $x \in \mathbb{R}^{n\times 1}$ is an un known vector to be solved.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

The normal equation

In order to solve the linear system Eq. (2.2) when A is nonsymmetric, we can solve the equivalent linear system

$$A^T Ax \;=\; A^T b. \tag{2.3}$$

This system is known as the system of the normal equations associated with the least-squares problem:

$$\min_{x\in\mathbb{R}^n} \; \|b - Ax\|_2. \tag{2.4}$$

Note that Eq. (2.3) is typically used to solve the least-squares problem Eq. (2.4) for over determined systems, i.e., when A is a rectangular matrix of size $\mathbb{R}^{m\times n}$ and $n < m$.

Pseudoinverse

Theorem 2.12 (see, e.g., [32])**.** Let $A \in \mathbb{R}^{m\times n}$ be a matrix of rank $r > 0$. Then there exist

- positive real numbers $s_1, s_2, \ldots, s_r$,
- an orthonormal set $\{u_1, u_2, \ldots, u_r\} \subseteq \mathbb{R}^{m\times 1}$,
- an orthonormal set $\{v_1, v_2, \ldots, v_r\} \subseteq \mathbb{R}^{n\times 1}$

such that

$$A = \sum_{j=1}^{r} s_j u_j v_j^T. \tag{2.5}$$

The numbers $s_1, s_2, \ldots, s_r$ are called singular values of A. The decomposition (2.5) is known as the dyadic form of the Singular Value Decomposition (SVD) of A.

Definition 2.13. Let $A \in \mathbb{R}^{m\times n}$ be a matrix of rank r with singular value decomposition (2.5). Then the pseudoinverse (or Moore-Penrose pseudoinverse) of matrix A is given by

$$A^\dagger = \sum_{j=1}^{r} \frac{1}{s_j} v_j u_j^T. \tag{2.6}$$

Lemma 2.14 (see, e.g., [32])**.** We emphasize that $x = A^\dagger b$ solves all three the problems of linear system Eq. (2.2)

1) If A is invertible, then $m = n = r$ and

$$x = A^{-1}b = \left(\sum_{j=1}^{n} \frac{1}{s_j} v_j u_j^T\right) b.$$

In particular, when ever A is invertible, $A^\dagger = A^{-1}$.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

2) If $b \in \mathcal{R}(A)$, then $b_R = b$, and

$$x = A^{\dagger}b = \left(\sum_{j=1}^{n} \frac{1}{s_j} v_j u_j^T \right) b.$$

solves $Ax = b$. If $r < n$, then there exist infinitely many solutions to $Ax = b$ that all have the form $x = A^{\dagger}b + n$ for $n \in \mathcal{N}(A) = \text{span}\{v_{r+1}, \ldots, v_n\}$. Among all these solutions, $x = A^{\dagger}b$ is the unique solution of smallest norm.

3) When $b \notin \mathcal{R}(A)$, no x will solve $Ax = b$, but $x = A^{\dagger}b$ will minimize $\|Ax - b\|_2$. If $r < n$, there will be infinitely many x that minimize $\|Ax - b\|_2$, each having the form $x = A^{\dagger}b + n$ for $n \in \mathcal{N}(A)$. Among all these solutions $x = A^{\dagger}b$ is the unique minimizer of $\|Ax - b\|_2$ having smallest norm.

2.3 Iterative methods for linear systems

A conjugate gradient algorithm for linear systems

The conjugate gradient method was created by Magnus Hestenes and Eduard Stiefel in 1952 [33]. It is the iterative method to find the solution of linear system Eq. (2.2) under condition that the coefficient matrix $A \in \mathbb{R}^{n \times n}$ is positive definite.

The algorithm starts with an initial approximation $x^{(0)}$ and takes the first direction to be the residual from steepest descent $d^{(1)} = r^{(0)} = b - Ax^{(0)}$. Each approximation is computed by $x^{(k)} = x^{(k-1)} + \alpha_k d^{(k)}$. Applying the A-orthogonality condition from conjugate directions in order to speed up convergence, $\langle d^{(i)}, Ad^{(j)} \rangle = 0$ and $\langle r^{(i)}, r^{(j)} \rangle = 0$ when $i \neq j$ There are two stopping conditions. The first is calculating a residual to be zero, which means

$$\begin{aligned} r^{(k)} &= b - Ax^{(k)} \\ 0 &= b - Ax^{(k)} \\ Ax^{(k)} &= b \end{aligned}$$

this implies that the solution is $x^{(k)}$.

The second stopping condition is if the residual is sufficiently close to 0, within some specified tolerance. The next search direction $d^{(k+1)}$, is computed using the residual from the previous iteration $r^{(k)}$, by

$$d^{(k+1)} = r^{(k)} + \beta_k d^{(k)}$$

In order for the A-orthogonal condition to hold, β_k must be chosen in such a way that $\langle d^{(k+1)}, Ad^{(k)} \rangle = 0$. In order to figure out what the choice of β_k should be, multiply by A and take the inner product with the previous search direction $d^{(k)}$, to the left side to obtain:

$$\left\langle d^{(k)}, Ad^{(k+1)} \right\rangle = \left\langle d^{(k)}, Ar^{(k)} \right\rangle + \beta_k \left\langle d^{(k)}, Ad^{(k)} \right\rangle$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

by the A-orthogonality condition, $\left\langle d^{(k)}, Ad^{(k+1)} \right\rangle = 0$, so

$$0 \;=\; \left\langle d^{(k)}, Ar^{(k)} \right\rangle + \beta_k \left\langle d^{(k)}, Ad^{(k)} \right\rangle$$

then solving for β_k,

$$\beta_k \;=\; -\frac{\left\langle d^{(k)}, Ar^{(k)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle}.$$

Next solving for the scalar α_k, from conjugate directions,

$$\alpha_k \;=\; \frac{\left\langle d^{(k)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle}.$$

Applying the chosen definition of $d^{(k+1)}$,

$$\begin{aligned}
\alpha_k \;&=\; \frac{\left\langle d^{(k)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle} \\
&=\; \frac{\left\langle r^{(k-1)} + \beta_{k-1} d^{(k-1)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle} \\
&=\; \frac{\left\langle r^{(k-1)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle} + \beta_{k-1} \frac{\left\langle d^{(k-1)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle}
\end{aligned}$$

substituting $d^{(k-1)}$ using the definition $d^{(k+1)} = r^{(k)} + \beta_k d^{(k)}$ to obtain:

$$\alpha_k \;=\; \frac{\left\langle r^{(k-1)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle} + \beta_{k-1} \frac{\left\langle r^{(k-2)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle} + \beta_{k-1}\beta_{k-2} \frac{\left\langle d^{(k-2)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle}$$

because of the orthogonality condition of residuals, $\left\langle r^{(k-2)}, r^{(k-1)} \right\rangle = 0$. Clearly, by continuing to expand the $d^{(i)}$ terms none of the terms will contain $r^{(k-1)}$. So, every term except for the $\left\langle r^{(k-1)}, r^{(k-1)} \right\rangle$ term is zero. So,

$$\alpha_k \;=\; \frac{\left\langle r^{(k-1)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle}.$$

Lastly, calculating the residuals, starting with the definition of $x^{(k)}$,

$$x^k \;=\; x^{k-1} + \alpha_k d^{(k)}$$

multiplying by $-A$, and adding b the equation becomes,

$$b - Ax^k \;=\; b - Ax^{k-1} - \alpha_k Ad^{(k)}.$$

Now utilizing the definition of $r^{(k)}$,

$$r^k \;=\; r^{k-1} - \alpha_k Ad^{(k)}.$$

By eliminating our dependence on these repetitive calculations, the algorithm becomes more efficient. The matrix multiplications in the β_k formula can both be removed by the following:

Starting with the $r^{(k)}$ formula and taking the inner product with itself on the right,

$$\begin{aligned}
r^k \;&=\; r^{k-1} - \alpha_k Ad^{(k)} \\
\left\langle r^{(k)}, r^{(k)} \right\rangle \;&=\; \left\langle r^{(k-1)}, r^{(k)} \right\rangle - \alpha_k \left\langle Ad^{(k)}, r^{(k)} \right\rangle \\
\left\langle r^{(k)}, r^{(k)} \right\rangle \;&=\; -\alpha_k \left\langle Ad^{(k)}, r^{(k)} \right\rangle \\
-\frac{1}{\alpha_k} \left\langle r^{(k)}, r^{(k)} \right\rangle \;&=\; \left\langle d^{(k)}, Ar^{(k)} \right\rangle.
\end{aligned}$$

This material is reserved for educational purposes. Not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Next considering the α_k formula,

$$\begin{aligned} \alpha_k &= \frac{\left\langle r^{(k-1)}, r^{(k-1)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle} \\ \left\langle d^{(k)}, Ad^{(k)} \right\rangle &= \frac{1}{\alpha_k} \left\langle r^{(k-1)}, r^{(k-1)} \right\rangle. \end{aligned}$$

Substituting into the β_k formula,

$$\begin{aligned} \beta_k &= -\frac{\left\langle d^{(k)}, Ar^{(k)} \right\rangle}{\left\langle d^{(k)}, Ad^{(k)} \right\rangle} \\ &= -\frac{-\frac{1}{\alpha_k} \left\langle r^{(k)}, r^{(k)} \right\rangle}{\frac{1}{\alpha_k} \left\langle r^{(k-1)}, r^{(k-1)} \right\rangle} \\ &= \frac{\left\langle r^{(k)}, r^{(k)} \right\rangle}{\left\langle r^{(k-1)}, r^{(k-1)} \right\rangle} \end{aligned}$$

This eliminated the redundancy of the algorithm, which increases the efficiency substantially. The following is the code for the CG-method [33, 34].

Algorithm 1: A conjugate gradient algorithm for the linear system Eq. (2.2)

$r^{(0)} = b - Ax^{(0)}$, $d^{(1)} = r^{(0)}$
Set $k = 1$
while $k \leq n$ **do**
  $\alpha_{k+1} = \frac{\|r^{(k)}\|^2}{\left\langle d^{(k+1)}, Ad^{(k+1)} \right\rangle}$
  $x^{(k+1)} = x^{(k)} + \alpha_{k+1} d^{(k+1)}$
  $r^{(k)} = r^{(k-1)} - \alpha_k Ad^{(k)}$
  if $r^{(k)} = 0$ **then**
    $x^{(k)}$ is the exact solution ; break
  else
    $\beta_k = \frac{\|r^{(k)}\|^2}{\|r^{(k-1)}\|^2}$
    $d^{(k+1)} = r^{(k)} + \beta_k d^k$
  end
  update k
end

The CGLS algorithm for linear systems

We have been looking at the problem of approximately solving an overconstrained system: when Eq. (2.2) does not have a solution, finding an x that is the lease-squares solution of Eq. (2.2) is the solution corresponding normal equation Eq. (2.3).

In 1982, Paige and Saunders [35] propose a CGLS algorithm for computing the solution of the least-squares problem Eq. (2.4). An algorithm with better numerical properties is easily derived by a slight algebraic rearrangement, making use of the intermediate vector $Ad^{(k)}$ [33]. It is usually stated in notation similar to the following:

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Algorithm 2: A CGLS algorithm for the linear system Eq. (2.2)

Set $r^{(0)} = b$, $s^{(0)} = A^T b$, $d^{(1)} = s^{(0)}$, $\gamma_0 = \|s^{(0)}\|$ and $x^{(0)} = 0$

while $k = 1$ **to** n **do**

set $q^{(k)} = Ad^{(k)}$

$\alpha_k = \frac{\gamma_{k-1}}{\|q^{(k)}\|^2}$

$x^{(k)} = x^{(k-1)} + \alpha_k d^{(k)}$

$r^{(k)} = r^{(k-1)} - \alpha_k q^{(k)}$

$s^{(k)} = A^T r^{(k)}$

if $s^{(k)} = 0$ **then**

$x^{(k)}$ is the exact solution ; break

else

$\gamma_k = \|s^{(k)}\|^2$

$\beta_k = \frac{\gamma_k}{\gamma_{k-1}}$

$d^{(k+1)} = s^{(k)} + \beta_k d^k$

end

update k

end

2.4 Iterative methods for Sylvester-type matrix equations to obtain an exact solution

2.4.1 The Sylvester matrix equation

Consider the iterative solution of the following Sylvester matrix equation

$$AX + XB = C, \tag{2.7}$$

where $A \in \mathbb{R}^{m\times m}$, $B \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{m\times n}$ are known matrices, and $X \in \mathbb{R}^{m\times n}$ is the matrix to be determined. Several researchers have developed iterative methods to solve sylvester matrix equations.

The gradient based iterative (GI) algorithm

The GI algorithm was proposed by Ding et al. [14] for solving Eq. (2.7). Here, we regard Eq. (2.7) as the following two matrix equations:

$$AX = C - XB, \qquad XB = C - AX. \tag{2.8}$$

From Eq. (2.8) Ding et al. presented the iterative solution $X(k) = (X_1(k) + X_2(k))/2$ given by the GI algorithm with

$$X_1(k) = X(k-1) + \mu A^T \left[C - AX(k-1) - X(k-1)B\right],$$
$$X_2(k) = X(k-1) + \mu \left[C - AX(k-1) - X(k-1)B\right] B^T$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

It is shown in [14] that the GI algorithm converges as long as

$$0 < \mu < \frac{2}{\lambda_{\max}(AA^T) + \lambda_{\max}(B^TB)},$$

where $\lambda_{\max}(AA^T)$ is the largest eigenvalue of AA^T.

The relaxed gradient based iterative (RGI) algorithm

In [15], Niu at al. proposed a relaxed gradient based iterative algorithm for solving Eq. (2.7) by introducing a relaxed factor $\hat{\omega}$. Compute

$$\begin{aligned} X(k-1) &= \hat{\omega}X_1(k-1) + (1-\hat{\omega})X_2(k-1), \\ X_1(k) &= X(k-1) + (1-\hat{\omega})\mu A^T\left[C - AX(k-1) - X(k-1)B\right], \\ X_2(k) &= X(k-1) + \hat{\omega}\mu\left[C - AX(k-1) - X(k-1)B\right]B^T \end{aligned}$$

The RGI algorithm has been proved to be convergent when

$$0 < \mu < \frac{1}{\hat{\omega}(1-\hat{\omega})(\lambda_1 + \lambda_2 + \lambda_3)},$$

where $\lambda_1 = \lambda_{\max}(AA^T)$, $\lambda_2 = \lambda_{\max}(B^TB)$, and $\lambda_3 = \sigma_{\max}(BA^T)$ denotes the largest singular value of matrix BA^T.

The modified gradient based iterative (MGI) algorithm

Recently, in [16] Wang et al. proposed a modified gradient based iterative (MGI) algorithm to solve Eq. (2.7). Choose the initial matrices $X_1(0)$, $X_2(0)$ and compute $X(0) = (X_1(0) + X_2(0))/2$. In the step of computing

$$\begin{aligned} X_1(k) &= X(k-1) + \mu A^T\left[AX(k-1) + X(k-1)B - C\right], \\ X(k-1) &= \frac{X_1(k) + X_2(k-1)}{2}, \\ X_2(k) &= X(k-1) + \mu\left[AX(k-1) + X(k-1)B - C\right]B^T, \\ X(k) &= \frac{X_1(k) + X_2(k)}{2}. \end{aligned}$$

The modified gradient based iterative (MGI) algorithm converges to a unique solution for any initial value.

The accelerated gradient based iterative (AGBI) algorithm

Xie et al. [20] proposed the accelerated gradient based iterative (AGBI) algorithm for solving Eq. (2.7) by using the information of

$$\begin{aligned} X_1(k) &= X(k-1) + \bar{\omega}\mu A^T\left[C - AX(k-1) - X(k-1)B\right] \\ X_2(k) &= X(k-1) + (1-\bar{\omega})\mu\left[C - AX(k-1) - X(k-1)B\right]B^T \end{aligned}$$

to update $X(k-1)$. From Theorem 3.4 [20], we conclude that the AGBI algorithm will be convergent when

$$0 < \mu < \min\left\{\frac{2}{\bar{\omega}\|A\|^2}, \frac{2}{(1-\bar{\omega})\|B\|^2}\right\}.$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

2.4.2 The Stein matrix equation

Ding et al. [14] studied the solution to the following matrix equation:

$$X + AXB = E, \tag{2.9}$$

where $A \in \mathbb{R}^{m\times m}$,$B \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{m\times n}$, are given constant matrices, $X \in \mathbb{R}^{m\times n}$, is the unknown matrix to be solved. For the equation in (2.9), they present the gradient iterative algorithm to compute the solution X^*:

$$\begin{aligned} X_1(k) &= X(k-1) + \mu A^T \left[C - AX(k-1)B - X(k-1)\right] B^T, \\ X_2(k) &= X(k-1) + \mu \left[C - AX(k-1)B - X(k-1)\right], \\ X(k) &= \frac{1}{2}(X_1(k) + X_2(k)). \end{aligned}$$

when

$$\mu = \frac{1}{\lambda_{\max}(AA^T)\lambda_{\max}(B^TB) + 1} \quad \text{or} \quad \mu = \frac{1}{\|A\|_2^2\|B\|_2^2 + 1}.$$

If the matrix equation in (2.9) has a unique solution X^*, then the iterative solution $X(k)$ given by the algorithm converges to the solution X^*, i.e., $\lim_{k\to\infty} X(k) = X^*$.

2.4.3 A generalized Sylvester matrix equation

In [14], Ding et al. extended the study to a more general matrix equation:

$$\sum_{j=1}^{p} A_j X B_j = C, \tag{2.10}$$

where $A_j \in \mathbb{R}^{m\times m}$,$B_j \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{m\times n}$, are given constant matrices, $X \in \mathbb{R}^{m\times n}$, is the unknown matrix to be solved. If the matrix equation in (2.10) has a unique solution X^*, then the iterative solution $X(k)$ given by

$$\begin{aligned} X_i(k) &= X(k-1) + \mu A_i^T \left[C - \sum_{j=1}^{p} A_j X(k-1) B_j\right] B_i^T, \qquad i = 1, 2, \ldots, p, \\ X(k) &= \frac{1}{p}(X_1(k) + X_2(k) + \cdots + X_p(k)). \end{aligned}$$

when

$$\mu = \frac{1}{\sum_{j=1}^{p} \lambda_{\max}(A_jA_j^T)\lambda_{\max}(B_j^TB_j)} \quad \text{or} \quad \mu = \frac{1}{\sum_{j=1}^{p} \|A_j\|_2^2\|B_j\|_2^2}$$

converges to the solution X^*.

2.4.4 A generalized Sylvester-transpose matrix equation

A generalized Sylvester-transpose matrix equation takes the from

$$\sum_{i=1}^{s} A_i X B_i + \sum_{j=1}^{t} C_j X^T D_j = E. \tag{1.6}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

where $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$ for any $i = 1, 2, \ldots, s$, $j = 1, 2, \ldots, t$ and $E \in \mathbb{R}^{m\times q}$ are given matrices and $X \in \mathbb{R}^{p\times q}$ is unknown matrix. Kittisopaporn et al. [36] studied Eq. (1.2) when an associated matrix is of full column-rank. The techniques of gradient and steepest descent let us obtain the search direction and the step sizes. Indeed, for any initial matrix X_0, the sequence $X(k)$ of approximated solutions generated by

$$X(k) = X(k-1) + \tau_k\left(\sum_{i=1}^{s} A_i^T R(k-1)B_i^T + \sum_{j=1}^{t} D_j R(k-1)^T C_j\right)$$

converges to the exact solution X^*.

2.5 Iterative methods for Sylvester-type matrix equations to obtain a least-squares solution

When Eq. (1.2) has no solution, we will not be able to exactly solve overdetermined equations. The best we can seek for a least-squares solution, i.e. a matrix X such that minimizes the squared Frobenius norm

$$\left\|\sum_{i=1}^{s} A_i X B_i + \sum_{j=1}^{t} C_j X^T D_j - E\right\|_F.$$

The Least-squares iterative (LSI) method

In 2009, Ding et al. extended the concept of a GI method to the least-squares based iterative (LSI) algorithm. Let $0 < \mu < 2(s+t)$. Then the iterative solution $X(k)$ given by the least-squares based iterative (LSI) algorithm,

$$X_i(k) = X(k-1) + \mu(A_i^T A_i)^{-1} A_i^T\left(E - \sum_{i=1}^{s} A_i X(k-1) B_i - \sum_{j=1}^{t} C_j X(k-1)^T D_j\right) B_i^T (B_i B_i^T)^{-1},$$

$$X_j(k) = X(k-1) + \mu(D_j D_j^T)^{-1} D_j\left(E - \sum_{i=1}^{s} A_i X(k-1) B_i - \sum_{j=1}^{t} C_j X(k-1)^T D_j\right) C_j (C_j^T C_j)^{-1},$$

$$X(k) = \frac{1}{s+t}\left(\sum_{i=1}^{s} X_i + \sum_{j=1}^{t} X_j\right),$$

leads to $\lim_{k\to\infty} X(k) = X$ for any initial value $X(0)$.

The conjugate gradient least-squares iterative (CGLS) method

Hajarian [29] proposed an iterative matrix algorithm based on the conjugate gradient least-squares iterative (CGLS) method for solving the matrix X within a finite number of iterations for any arbitrary initial matrix $X(0)$. By applying knowledge of the Kronecker product and vectorization operator, the solution X can be obtained by

$$X(k) = X(k-1) + \delta(k)P(k),$$

in the absence of roundoff errors.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 3

Conjugate gradient algorithms for a generalized Sylvester-transpose matrix equation

In this chapter, we discuss how to solve the generalized Sylvester-transpose matrix equation (1.2), where $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$, $D \in \mathbb{R}^{m\times q}$, $E \in \mathbb{R}^{m\times q}$ are know coefficient matrices, and $X \in \mathbb{R}^{n\times p}$ is unknown matrix. We recall the direct Kronecker linearization to solve Eq. (1.2) for exact/least-squares solutions. We devide the consideration of Eq. (1.2) into two cases. When Eq. (1.2) has a solution (consistent), we will derive a conjugate gradient algorithm to produce an exact solution. For inconsistent case, we will propose a CG algorithm to obtain a least-squares solution, the minimal-norm least-squares solution, and the least-squares solution closet to a given matrix.

Assume that the generalized Sylvester-transpose matrix equation (1.2) is consistent. From now on, let $m, n, p, q, s, t, \in \mathbb{N}$ be such that $mq = np$, we offer solution of Problem 1:

3.1 Exact solution via a direct Kronecker linearization

A direct solution for Eq. (1.2) can be obtained directly by transforming the matrix equation into an equivalent linear system. Applying the vector operator to (1.2) and utilizing Lemmas 2.5 and 2.8, we get

$$
\begin{aligned}
\operatorname{Vec} E &= \operatorname{Vec}\left(\sum_{i=1}^{s} A_i X B_i + \sum_{j=1}^{t} C_j X^T D_j\right) \\
&= \sum_{i=1}^{s} (B_i^T \otimes A_i) \operatorname{Vec} X + \sum_{j=1}^{t} (D_j^T \otimes C_j) \operatorname{Vec} X^T \\
&= \sum_{i=1}^{s} (B_i^T \otimes A_i) \operatorname{Vec} X + \sum_{j=1}^{t} (D_j^T \otimes C_j) P(n,p) \operatorname{Vec} X \\
&= \left(\sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j)\right) \operatorname{Vec} X. \qquad (3.1)
\end{aligned}
$$

Let us denote $x = \operatorname{Vec} X$, $b = \operatorname{Vec} E$, and

$$
K = \sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j) P(n,p) \in \mathbb{R}^{mq\times np}. \qquad (3.2)
$$

That implies the equivalent of Eq. (1.2) and the linear algebraic system $Kx = b$. There are well-known direct method in general linear algebra for solve the vector x

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

such as Gaussian Elimination, Inversion Matrix, Cramer's Rule, and LU Factorization. Due to the injectivity of Vec$(\cdot)$, we obtain the unknown matrix X from Vec X.

3.2 Exact solution via a conjugate gradient algorithm

We focus our attention when the matrix coefficients A_i, B_i, C_j, D_j are moderate or large sizes. The traditional direct methods are not suitable because solving for matrices moderate or large sizes requires a large amount of time and memory to find a exact solution for Problem 1. For medium or large matrices It is enough to find an approximate solution to the equation. From now on, assume that K in (3.2) is symmetric and full-column rank, under such conditions we propose the following iterative algorithm to find a well-approximate solution for Problem 1:

Algorithm 3: A CG algorithm for a generalized Sylvester-transpose matrix equation Eq. (1.2)

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

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

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

for $k = 0$ **to** np **do**

  if $\|R_k\| \leqslant \epsilon$ **then**

    X_k is the disired 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

  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

end

We will show that Algorithm 3 produces approximate solutions within finite step, so that the set of residual matrices R_k is an orthogonal set for any given initial matrix X_0. We divides the proof into several lemmas.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

Lemma 3.1. Suppose that the iterative sequence $\{R_i\}$ is generated by Algorithm 3. We have

$$R_{k+1} \;=\; R_k - \frac{\|R_k\|^2}{\alpha_{k+1}} V_{k+1} \qquad \textit{for} \quad k \;=\; 0, 1, 2, \ldots \tag{3.3}$$

Proof. From Algorithm 3, we have that for any k,

$$\begin{aligned}
R_{k+1} \;&=\; E - \sum_{i=1}^{s} A_i X_{k+1} B_i - \sum_{j=1}^{t} C_j X_{k+1}^T D_j \\
&=\; E - \sum_{i=1}^{s} A_i \left(X_k + \frac{\|R_k\|^2}{\alpha_{k+1}} U_{k+1} \right) B_i - \sum_{j=1}^{t} C_j \left(X_k^T + \frac{\|R_k\|^2}{\alpha_{k+1}} U_{k+1}^T \right) D_j \\
&=\; E - \sum_{i=1}^{s} A_i X_k B_i - \frac{\|R_k\|^2}{\alpha_{k+1}} \sum_{i=1}^{s} A_i U_{k+1} B_i - \sum_{j=1}^{t} C_j X_k^T D_j - \frac{\|R_k\|^2}{\alpha_{k+1}} \sum_{j=1}^{t} C_j U_{k+1}^T D_j \\
&=\; E - \sum_{i=1}^{s} A_i X_k B_i - \sum_{j=1}^{t} C_j X_k^T D_j - \frac{\|R_k\|^2}{\alpha_{k+1}} \left(\sum_{i=1}^{s} A_i U_k B_i + \sum_{j=1}^{t} C_j U_k^T D_j \right) \\
&=\; R_k - \frac{\|R_k\|^2}{\alpha_{k+1}} V_{k+1}.
\end{aligned}$$

□

Lemma 3.2. Suppose that the iterative sequences $\{U_i\}$ and $\{V_i\}$, generated by Algorithm 3, under an assumption that the matrix K in (3.2) is symmetric, satisfy

$$\mathrm{tr}\,(U_m^T V_n) \;=\; \mathrm{tr}\,(V_m^T U_n) \qquad \textit{for any} \quad m, n. \tag{3.4}$$

Proof. Applying Lemmas 2.2–2.8 and the symmetry of K, we have

$$\begin{aligned}
\mathrm{tr}\,(V_m^T U_n) \;&=\; (\mathrm{Vec}\, V_m)^T \,\mathrm{Vec}\, U_n \\
&=\; \left[\mathrm{Vec} \left(\sum_{i=1}^{s} A_i U_m B_i + \sum_{j=1}^{t} C_j U_m^T D_j \right) \right]^T \mathrm{Vec}\, U_n \\
&=\; \left[\sum_{i=1}^{s} (B_i^T \otimes A_i) \,\mathrm{Vec}\, U_m + \sum_{j=1}^{t} (D_j^T \otimes C_j) \,\mathrm{Vec}\, U_m^T \right]^T \mathrm{Vec}\, U_n \\
&=\; \left[\left(\sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j) P(m,n) \right) \mathrm{Vec}\, U_m \right]^T \mathrm{Vec}\, U_n \\
&=\; [K \,\mathrm{Vec}\, U_m]^T \,\mathrm{Vec}\, U_n \\
&=\; (\mathrm{Vec}\, U_m)^T \; K \,\mathrm{Vec}\, U_n \\
&=\; (\mathrm{Vec}\, U_m)^T \left[\left(\sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j) P(m,n) \right) \mathrm{Vec}\, U_n \right] \\
&=\; (\mathrm{Vec}\, U_m)^T \left[\mathrm{Vec} \left(\sum_{i=1}^{s} A_i U_n B_i + \sum_{j=1}^{t} C_j U_n^T D_j \right) \right] \\
&=\; (\mathrm{Vec}\, U_m)^T \,\mathrm{Vec}\, V_n \\
&=\; \mathrm{tr}\,(U_m^T V_n)
\end{aligned}$$

for any m, n. □

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Lemma 3.3. Suppose that the iterative sequences $\{R_i\}$, $\{U_i\}$ and $\{V_i\}$, generated by Algorithm 3, satisfy

$$\text{tr}\left(R_m^T R_{m-1}\right) = 0 \quad \textit{and} \quad \text{tr}\left(U_{m+1}^T V_m\right) = 0 \qquad \textit{for any} \quad m. \tag{3.5}$$

Proof. We use induction principle to prove this lemma. From Lemmas 3.1 and 3.2, in the initial step $m = 1$, we obtain

$$\begin{aligned}
\text{tr}\left(R_1^T R_0\right) &= \text{tr}\left[\left(R_0 - \frac{\|R_0\|^2}{\alpha_1} V_1\right)^T R_0\right] \\
&= \text{tr}\left(R_0^T R_0\right) - \frac{\|R_0\|^2}{\alpha_1} \text{tr}\left(V_1^T R_0\right) \\
&= \|R_0\|^2 - \|R_0\|^2 \\
&= 0,
\end{aligned}$$

and

$$\begin{aligned}
\text{tr}\left(U_2^T V_1\right) &= \text{tr}\left[\left(R_1 + \frac{\|R_1\|^2}{\|R_0\|^2} U_1\right)^T V_1\right] \\
&= \text{tr}\left(R_1^T V_1\right) + \frac{\|R_1\|^2}{\|R_0\|^2} \text{tr}\left(U_1^T V_1\right) \\
&= \text{tr}\left(R_1^T \left(\frac{-\alpha_1}{\|R_0\|^2}\left(R_1 - R_0\right)\right)\right) + \alpha_1 \frac{\|R_1\|^2}{\|R_0\|^2} \\
&= -\frac{\alpha_1}{\|R_0\|^2} \text{tr}\left(R_1^T R_1\right) + \frac{\alpha_1}{\|R_0\|^2} \text{tr}\left(R_1^T R_0\right) + \alpha_1 \frac{\|R_1\|^2}{\|R_0\|^2} \\
&= -\frac{\alpha_1}{\|R_0\|^2} \|R_1\|^2 + \alpha_1 \frac{\|R_1\|^2}{\|R_0\|^2} \\
&= 0.
\end{aligned}$$

That means Eq. (3.5) hold for $m = 1$. In the procedure of inductive step, assume that for all $m = 1, \ldots, k$ Eq. (3.5) is true. Then for $m = k + 1$, we get

$$\begin{aligned}
\text{tr}\left(R_{k+1}^T R_k\right) &= \text{tr}\left[\left(R_k - \frac{\|R_k\|^2}{\alpha_{k+1}} V_{k+1}\right)^T R_k\right] \\
&= \text{tr}\left(R_k^T R_k\right) - \frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}\left(V_{k+1}^T R_k\right) \\
&= \|R_k\|^2 - \frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}\left(V_{k+1}^T \left(U_{k+1} - \frac{\|R_k\|^2}{\|R_{k-1}\|^2} U_k\right)\right) \\
&= \|R_k\|^2 - \frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}\left(V_{k+1}^T U_{k+1}\right) + \frac{\|R_k\|^2}{\alpha_{k+1}} \frac{\|R_k\|^2}{\|R_{k-1}\|^2} \text{tr}\left(V_{k+1}^T U_k\right) \\
&= \|R_k\|^2 - \frac{\|R_k\|^2}{\alpha_{k+1}} \alpha_{k+1} \\
&= 0,
\end{aligned}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

and

$$\begin{aligned}
\text{tr}\left(U_{k+2}^T V_{k+1}\right) &= \text{tr}\left[\left(R_{k+1} + \frac{\|R_{k+1}\|^2}{\|R_k\|^2} U_{k+1}\right)^T V_{k+1}\right] \\
&= \text{tr}\left(R_{k+1}^T V_{k+1}\right) + \frac{\|R_{k+1}\|^2}{\|R_k\|^2} \text{tr}\left(U_{k+1}^T V_{k+1}\right) \\
&= \text{tr}\left(R_{k+1}^T \left(\frac{-\alpha_{k+1}}{\|R_k\|^2}(R_{k+1} - R_k)\right)\right) + \frac{\|R_{k+1}\|^2}{\|R_k\|^2}\alpha_{k+1} \\
&= \frac{\alpha_{k+1}}{\|R_k\|^2} \text{tr}\left(R_{k+1}^T R_k\right) - \frac{\alpha_{k+1}}{\|R_k\|^2} \text{tr}\left(R_{k+1}^T R_{k+1}\right) + \frac{\|R_{k+1}\|^2}{\|R_k\|^2}\alpha_{k+1} \\
&= 0.
\end{aligned}$$

Hence, the conclusion Eq. (3.5) holds for any m. □

Lemma 3.4. Suppose that the iterative sequences $\{R_i\}$, $\{U_i\}$ and $\{V_i\}$ are generated by Algorithm 3. Then

$$\text{tr}\left(R_m^T R_0\right) = 0, \quad \text{tr}\left(U_{m+1}^T V_1\right) = 0 \qquad \textit{for any} \quad m. \tag{3.6}$$

Proof. We use induction principle to prove this lemma. From Lemma 3.3, In the initial step $m = 1$, we have $\text{tr}(R_1^T R_0) = 0$ and $\text{tr}(P_2^T Q_1) = 0$. Now, In the procedure of inductive step, assume that for all $m = 1, \ldots, k$ Eq. (3.6) is true. From Lemmas 3.1 and 3.2, for $m = k+1$ we get

$$\begin{aligned}
\text{tr}\left(R_{k+1}^T R_0\right) &= \text{tr}\left[\left(R_k - \frac{\|R_k\|^2}{\alpha_{k+1}} V_{k+1}\right)^T R_0\right] \\
&= \text{tr}\left(R_k^T R_0\right) - \frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}(V_{k+1}^T R_0) \\
&= -\frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}(V_{k+1}^T U_1) \\
&= -\frac{\|R_k\|^2}{\alpha_{k+1}} \text{tr}(U_{k+1}^T V_1) \\
&= 0,
\end{aligned}$$

and

$$\begin{aligned}
\text{tr}\left(U_{k+2}^T V_1\right) &= \text{tr}\left(V_{k+2}^T U_1\right) \\
&= \text{tr}\left[\left(\frac{-\alpha_{k+2}}{\|R_{k+1}\|^2}\left(R_{k+2} - R_{k+1}\right)\right)^T U_1\right] \\
&= \frac{-\alpha_{k+2}}{\|R_{k+1}\|^2} \text{tr}\left(R_{k+2}^T U_1\right) + \frac{\alpha_{k+2}}{\|R_{k+1}\|^2} \text{tr}\left(R_{k+1}^T U_1\right) \\
&= \frac{-\alpha_{k+2}}{\|R_{k+1}\|^2} \text{tr}\left(R_{k+2}^T R_0\right) + \frac{\alpha_{k+2}}{\|R_{k+1}\|^2} \text{tr}\left(R_{k+1}^T R_0\right) \\
&= 0.
\end{aligned}$$

Hence, the conclusion Eq. (3.6) holds for any m. □

Theorem 3.5. Suppose that the iterative sequences $\{R_i\}$, $\{U_i\}$ and $\{V_i\}$ are generated by Algorithm 3. Then for any m, n, $m \neq n$ is considered, under an assumption that the

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

matrix K in (3.2) is symmetric, we obtain

$$\text{tr}\left(R_{m-1}^T R_{n-1}\right) = 0 \quad \text{and} \quad \text{tr}\left(U_m^T V_n\right) = 0. \tag{3.7}$$

Proof. For any integers m, n, by Lemma 3.2 and the fact of trace of matrix that $\text{tr}(R_{m-1}^T R_{n-1}) = \text{tr}(R_{n-1}^T R_{m-1})$, that is enough to prove (3.7) for any m, n such that $m > n$. We use induction principle to prove this theorem. In the initial step, according to Lemma 3.3, Eq (3.7) holds for $m = n+1$. For $m = n+2$, we get

$$\begin{aligned}
\text{tr}\left(R_{n+2}^T R_n\right) &= \text{tr}\left[\left(R_{n+1} - \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} V_{n+2}\right)^T R_n\right] \\
&= \text{tr}(R_{n+1}^T R_n) - \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \text{tr}(V_{n+2}^T R_n) \\
&= -\frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \text{tr}\left[V_{n+2}^T \left(U_{n+1} - \frac{\|R_n\|^2}{\|R_{n-1}\|^2} U_n\right)\right] \\
&= -\frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \left[\text{tr}\left(V_{n+2}^T U_{n+1}\right) - \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \text{tr}\left(V_{n+2}^T U_n\right)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \text{tr}\left(U_{n+2}^T V_n\right) \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \text{tr}\left[\left(R_{n+1} + \frac{\|R_{n+1}\|^2}{\|R_n\|^2} U_{n+1}\right)^T V_n\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \left[\text{tr}\left(R_{n+1}^T V_n\right) + \frac{\|R_{n+1}\|^2}{\|R_n\|^2} \text{tr}\left(U_{n+1}^T V_n\right)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \text{tr}\left[R_{n+1}^T \left(\frac{\alpha_n}{\|R_{n-1}\|^2} (R_n - R_{n-1})\right)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \frac{\alpha_n}{\|R_{n-1}\|^2} \left[\text{tr}(R_{n+1}^T R_{n-1}) - \text{tr}(R_{n+1}^T R_n)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \frac{\alpha_n}{\|R_{n-1}\|^2} \text{tr}(R_{n+1}^T R_{n-1}),
\end{aligned}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

$$
\begin{aligned}
\mathrm{tr}\left(R_{n+1}^T R_{n-1}\right) &= \mathrm{tr}\left[\left(R_n - \frac{\|R_n\|^2}{\alpha_{n+1}} V_{n+1}\right)^T R_{n-1}\right] \\
&= \mathrm{tr}(R_n^T R_{n-1}) - \frac{\|R_n\|^2}{\alpha_{n+1}} \mathrm{tr}(V_{n+1}^T R_{n-1}) \\
&= -\frac{\|R_n\|^2}{\alpha_{n+1}} \mathrm{tr}\left[V_{n+1}^T \left(U_n - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} U_{n-1}\right)\right] \\
&= -\frac{\|R_n\|^2}{\alpha_{n+1}} \left[\mathrm{tr}\left(V_{n+1}^T U_n\right) - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \mathrm{tr}\left(V_{n+1}^T U_{n-1}\right)\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \mathrm{tr}\left(U_{n+1}^T V_{n-1}\right) \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \mathrm{tr}\left[\left(R_n + \frac{\|R_n\|^2}{\|R_{n-1}\|^2} U_n\right)^T V_{n-1}\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \left[\mathrm{tr}\left(R_n^T V_{n-1}\right) + \frac{\|R_n\|^2}{\|R_{n-1}\|^2} \mathrm{tr}\left(U_n^T V_{n-1}\right)\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \mathrm{tr}\left[R_n^T \left(\frac{\alpha_{n-1}}{\|R_{n-2}\|^2} \left(R_{n-1} - R_{n-2}\right)\right)\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \frac{\alpha_{n-1}}{\|R_{n-2}\|^2} \left[\mathrm{tr}(R_n^T R_{n-2}) - \mathrm{tr}(R_n^T R_{n-1})\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \frac{\alpha_{n-1}}{\|R_{n-2}\|^2} \mathrm{tr}(R_n^T R_{n-2}),
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{tr}\left(U_{n+2}^T V_n\right) &= \mathrm{tr}\left[\left(R_{n+1} - \frac{\|R_{n+1}\|^2}{\|R_n\|^2} U_{n+1}\right)^T V_n\right] \\
&= \mathrm{tr}(R_{n+1}^T V_n) + \frac{\|R_{n+1}\|^2}{\|R_n\|^2} \mathrm{tr}(U_{n+1}^T V_n) \\
&= \mathrm{tr}\left[R_{n+1}^T \left(\frac{-\alpha_n}{\|R_{n-1}\|^2} \left(R_n - R_{n-1}\right)\right)\right] \\
&= \frac{-\alpha_n}{\|R_{n-1}\|^2} \mathrm{tr}\left(R_{n+1}^T R_n - R_{n+1}^T R_{n-1}\right) \\
&= \frac{\alpha_n}{\|R_{n-1}\|^2} \mathrm{tr}\left[\left(R_n - \frac{\|R_n\|^2}{\alpha_{n+1}} V_{n+1}\right)^T R_{n-1}\right] \\
&= \frac{\alpha_n}{\|R_{n-1}\|^2} \left[\mathrm{tr}\left(R_n^T R_{n-1}\right) - \frac{\|R_n\|^2}{\alpha_{n+1}} \mathrm{tr}\left(V_{n+1}^T R_{n-1}\right)\right] \\
&= -\frac{\alpha_n}{\|R_{n-1}\|^2} \frac{\|R_n\|^2}{\alpha_{n+1}} \mathrm{tr}\left(V_n^T R_{n-1}\right) \\
&= -\frac{\alpha_n}{\|R_{n-1}\|^2} \frac{\|R_n\|^2}{\alpha_{n+1}} \mathrm{tr}\left[V_{n+1}^T \left(U_n - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} U_{n-1}\right)\right] \\
&= -\frac{\alpha_n}{\|R_{n-1}\|^2} \frac{\|R_n\|^2}{\alpha_{n+1}} \left[\mathrm{tr}\left(V_{n+1}^T U_n\right) - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \mathrm{tr}\left(V_{n+1}^T U_{n-1}\right)\right] \\
&= \frac{\alpha_n}{\|R_{n-1}\|^2} \frac{\|R_n\|^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2} \mathrm{tr}\left(U_{n+1}^T V_{n-1}\right),
\end{aligned}
$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

and

$$\begin{aligned}
\text{tr}\left(U_{n+1}^T V_{n-1}\right) &= \text{tr}\left[\left(R_n - \frac{\|R_n\|^2}{\|R_{n-1}\|^2}U_n\right)^T V_{n-1}\right] \\
&= \text{tr}(R_n^T V_{n-1}) + \frac{\|R_n\|^2}{\|R_{n-1}\|^2}\,\text{tr}(U_n^T V_{n-1}) \\
&= \text{tr}\left[R_n^T\left(\frac{-\alpha_{n-1}}{\|R_{n-2}\|^2}\left(R_{n-1} - R_{n-2}\right)\right)\right] \\
&= \frac{-\alpha_{n-1}}{\|R_{n-2}\|^2}\,\text{tr}\left(R_n^T R_{n-1} - R_n^T R_{n-2}\right) \\
&= \frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\,\text{tr}\left[\left(R_{n-1} - \frac{\|R_{n-1}\|^2}{\alpha_n}V_n\right)^T R_{n-2}\right] \\
&= \frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\left[\text{tr}\left(R_{n-1}^T R_{n-2}\right) - \frac{\|R_{n-1}\|^2}{\alpha_n}\,\text{tr}\left(V_n^T R_{n-2}\right)\right] \\
&= -\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\frac{\|R_{n-1}\|^2}{\alpha_n}\,\text{tr}\left(V_n^T R_{n-2}\right) \\
&= -\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\frac{\|R_{n-1}\|^2}{\alpha_n}\,\text{tr}\left[V_n^T\left(U_{n-1} - \frac{\|R_{n-2}\|^2}{\|R_{n-3}\|^2}U_{n-2}\right)\right] \\
&= -\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\frac{\|R_{n-1}\|^2}{\alpha_n}\left[\text{tr}\left(V_n^T U_{n-1}\right) - \frac{\|R_{n-2}\|^2}{\|R_{n-3}\|^2}\,\text{tr}\left(V_n^T U_{n-2}\right)\right] \\
&= \frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\frac{\|R_{n-1}\|^2}{\alpha_n}\frac{\|R_{n-2}\|^2}{\|R_{n-3}\|^2}\,\text{tr}\left(U_n^T V_{n-2}\right),
\end{aligned}$$

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. We keep repeating this until terms $\text{tr}(R_2^T R_0)$ and $\text{tr}(U_3^T V_1)$ show up. By Lemma 3.4, we get $\text{tr}(R_{n+2}^T R_n) = 0$ and $\text{tr}(U_{n+2}^T V_n) = 0$.

In the procedure of inductive step, Suppose that for all $m = n+1, \ldots, k$ Eq. (3.5) is true. Then for $m = k+1$, we obtain

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

and

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

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Hence, the conclusion $\text{tr}(R_{m-1}^T R_{n-1}) = 0$ and $\text{tr}(U_m^T V_n) = 0$ holds for any integers m, n such that $m \neq n$. □

Theorem 3.6. Suppose that the iterative sequence $\{X_i\}$ is generated by Algorithm 3 and assume that the matrix K in Eq. (3.2) is symmetric and full-column rank. For given initial matrix $X_0 \in \mathbb{R}^{n\times p}$ within finite steps, Algorithm 3 can solve Problem 1 and give an exact solution X in at most np iteration steps.

Proof. Assume that $R_i \neq 0$ for $i = 0, 1, \ldots, np$. According the Theorem 3.5, the set of residual matrices $\{R_0, R_1, ..., R_{np}\}$ is orthogonal in $\mathbb{R}^{n\times p}$. By Theorem 3.5, the set $\{R_0, R_1, ..., R_{np}\}$ is orthogonal in $\mathbb{R}^{n\times p}$. So,the set of residual matrices $\{R_0, R_1, ..., R_{np}\}$ of $np + 1$ elements is linearly independent. That contradicts with the dimension of $\mathbb{R}^{n\times p}$, which is np. Because the linearly independent subset of $\mathbb{R}^{n\times p}$ must not contain more than np elements. Thus, $R_{np} = 0$, hence X_{np} is a solution of the matrix equation (1.2). □

Next, we investigate the generalized Sylvester-transpose matrix equation when Eq. (1.1) is inconsistent. We will seek for the matrix X^* which is the least squares solution of Eq. (1.1) that solves the following minimization problem:

$$\min_{X\in\mathbb{R}^{n\times p}} \|E - AXB - CX^T D\|_F. \tag{3.8}$$

3.3 Least-squares solution via a direct Kronecker linearization

A traditional direct method to solve a generalized Sylvester-transpose matrix equation Eq. (1.1) when the equation has no solution. Indeed, taking Kronecker product and vector operator to (1.1) and utilizing Lemmas 2.5 and 2.8, the matrix equation (1.1) can be rewritten:

$$\begin{aligned}
\text{Vec}\, E &= \text{Vec}(AXB + CX^T D) \\
&= (B^T \otimes A)\,\text{Vec}\, X + (D^T \otimes C)\,\text{Vec}\, X^T \\
&= (B^T \otimes A)\,\text{Vec}\, X + (D^T \otimes C)P(n,p)\,\text{Vec}\, X \\
&= (B^T \otimes A) + (D^T \otimes C)P(n,p)\,\text{Vec}\, X.
\end{aligned} \tag{3.9}$$

Let us denote $x = \text{Vec}\, X$, $e = \text{Vec}\, E$, and

$$M = (B^T \otimes A) + (D^T \otimes C)P(n,p). \tag{3.10}$$

In this case, Eq. (1.1) is inconsistent. We will solve this problem through the associated normal equation

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

$$M^T M x \;=\; M^T e. \tag{3.11}$$

The normal linear system (3.11) is always consistent, then the matrix equation (1.1) always has a least-squares solution.

Moreover, the matrix equation (1.1) has a unique least-squares solution if and only if the matrix M is of full-column rank, i.e., $M^T M$ is invertible. The unique solution by direct method given by $\text{Vec}\, X^* = x^* = (M^T M)^{-1} M^T e$. If M is not of full-column rank, then the matrix equation (1.1) has many solutions. The least-squares solutions appear in the form $\text{Vec}\, X^* = x^* = M^\dagger e + u$ where $M^\dagger$ is the Moore–Penrose inverse of M, and u is an arbitrary vector in the kernel of M. Among all these solutions,

$$\text{Vec}\, X^* \;=\; x^* \;=\; M^\dagger e \tag{3.12}$$

is the unique one having minimal norm. Since $\text{Vec}(\cdot)$ is injective and linear, that imply least-squares solutions X^* of matrix equation (1.1).

3.4 Least-squares solution via a conjugate gradient algorithm

From now on, assume that M in (3.11) is of full-column rank. Then there is a unique solution X in minimization problem Eq. (3.8). Consider Eq. (3.11), applying Lemmas 2.2 – 2.8. Indeed, we can deduce:

Lemma 3.7 ([37])**.** Minimization problem Eq. (3.8) is equivalent to the following consistent matrix equation

$$A^T(AXB + CX^T D)B^T + D(B^T X^T A^T + D^T X C^T)C \;=\; A^T E B^T + D E^T C. \tag{3.13}$$

The residual matrix R_X of Eq. (3.13) corresponding to a matrix $X \in \mathbb{R}^{n\times p}$, is defined by:

$$R_X \;:=\; A^T E B^T + D E^T C - A^T(AXB + CX^T D)B^T - D(B^T X^T A^T + D^T X C^T)C. \tag{3.14}$$

We adapt the technique of conjugate gradient to solve a unique solution X in minimization problem Eq. (3.8). The form of conjugate gradient iterative algorithm for solving Least-squares solutions of Equation (1.1) can be presented as

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Algorithm 4: A conjugate gradient iterative algorithm for Equation. (1.1)

$A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, $C \in \mathbb{R}^{m\times p}$, $D \in \mathbb{R}^{n\times q}$ and $E \in \mathbb{R}^{m\times q}$;

Given an ϵ such that $\epsilon > 0$, set $r = 0$, $U_0 = 0$. Choose $X_0 \in \mathbb{R}^{n\times p}$;

$R_0 = A^T EB^T + DE^T C - A^T(AX_0B + CX_0^T D)B^T - D(B^T X_0^T A^T + D^T X_0 C^T)C$;

for $r = 0$ **to** np **do**

  if $\|R_r\|_F \leqslant \epsilon$ **then**

    X_r is a least-squares solution; break;

  else

    if $r = 0$ **then**

      set $U_{r+1} = R_r$;

      else

        set $U_{r+1} = R_r + \frac{\|R_r\|_F^2}{\|R_{r-1}\|_F^2} U_r$;

      end

    end

    $H_{r+1} = A^T(AU_{r+1}B + CU_{r+1}^T D)B^T + D(B^T U_{r+1}^T A^T + D^T U_{r+1} C^T)C$;

    $\alpha_{r+1} = \mathrm{tr}(U_{r+1}^T H_{r+1})$;

    $X_{r+1} = X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}} U_{r+1}$;

    $R_{r+1} = A^T EB^T + DE^T C - A^T(AX_{r+1}B + CX_{r+1}^T D)B^T$
    $\quad -D(B^T X_{r+1}^T A^T + D^T X_{r+1} C^T)C$;

    update r;

  end

end

We will show the set of residual matrices R_r is an orthogonal basis, for any given initial matrix X_0. The sequence of approximated solutions derived by Algorithm 4 converges to the solution X of Equation (3.13) within finite number of iterations. We will continue to prove several lemmas as follows:

Lemma 3.8. Assume that the iterative sequences $\{R_r\}$ and $\{H_r\}$ are constructed from Algorithm 4. We obtain

$$R_{r+1} = R_r - \frac{\|R_r\|_F^2}{\alpha_{r+1}} H_{r+1}, \qquad \textit{for} \quad r = 0, 1, 2, \ldots. \tag{3.15}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Proof. By Algorithm 4, for any r, we get

$$\begin{aligned}
R_{r+1} &= A^TEB^T + DE^TC - A^T(AX_{r+1}B + CX_{r+1}^TD)B^T - D(B^TX_{r+1}^TA^T + D^TX_{r+1}C^T)C \\
&= A^TEB^T + DE^TC - A^T\left(A(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})B + C(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})^TD\right)B^T \\
&\quad - D\left(B^T(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})^TA^T + D^T(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})C^T\right)C \\
&= A^TEB^T + DE^TC - A^T\left(AX_rB + \frac{\|R_r\|_F^2}{\alpha_{r+1}}AU_{r+1}B + CX_r^TD + \frac{\|R_r\|_F^2}{\alpha_{r+1}}CU_{r+1}^TD\right)B^T \\
&\quad - D\left(B^TX_r^TA^T + \frac{\|R_r\|_F^2}{\alpha_{r+1}}B^TU_{r+1}^TA^T + D^TX_rC^T + \frac{\|R_r\|_F^2}{\alpha_{r+1}}D^TU_{r+1}C^T\right)C \\
&= A^TEB^T + DE^TC - A^T(AX_rB + CX_r^TD)B^T - D(B^TX_r^TA^T + D^TX_rC^T)C \\
&\quad - \frac{\|R_r\|_F^2}{\alpha_{r+1}}[A^T(AU_{r+1}B + CU_{r+1}^TD)B^T + D(B^TU_{r+1}^TA^T + D^TU_{r+1}C^T)C] \\
&= R_r - \frac{\|R_r\|_F^2}{\alpha_{r+1}}H_{r+1}.\square
\end{aligned}$$

Lemma 3.9. Assume that the iterative sequences $\{U_r\}$ and $\{H_r\}$, constructed by Algorithm 4, satisfy

$$\text{tr}(U_m^TH_n) = \text{tr}(H_m^TU_n), \qquad \textit{for any} \quad m, n. \tag{3.16}$$

Proof. Applying the properties of Kronecker product and vectorization operator in Lem-

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

mas 2.2–2.8, we can obtain

$$\begin{aligned}
\mathrm{tr}(H_m^T U_n) &= (\mathrm{Vec}\, H_m)^T \mathrm{Vec}\, U_n \\
&= [\mathrm{Vec}(A^T(AU_mB + C_jU_m^TD)B^T + D(B^TU_m^TA^T + D^TU_mC^T)C)]^T \mathrm{Vec}\, U_n \\
&= [\mathrm{Vec}(A^TAU_mBB^T)]^T \mathrm{Vec}\, U_n + [\mathrm{Vec}(A^TCU_m^TDB^T)]^T \mathrm{Vec}\, U_n \\
&\quad + [\mathrm{Vec}(DB^TU_m^TA^TC)]^T \mathrm{Vec}\, U_n + [\mathrm{Vec}(DD^TU_mC^TC)]^T \mathrm{Vec}\, U_n \\
&= [(BB^T \otimes A^TA)\,\mathrm{Vec}\, U_m]^T \mathrm{Vec}\, U_n + [(BD^T \otimes A^TC)\,\mathrm{Vec}\, U_m^T]^T \mathrm{Vec}\, U_n \\
&\quad + [(C^TA \otimes DB^T)\,\mathrm{Vec}\, U_m^T]^T \mathrm{Vec}\, U_n + [(C^TC \otimes DD^T)\,\mathrm{Vec}\, U_m]^T \mathrm{Vec}\, U_n \\
&= (\mathrm{Vec}\, U_m)^T(BB^T \otimes A^TA)\,\mathrm{Vec}\, U_n + (\mathrm{Vec}\, U_m^T)^T(DB_k^T \otimes C^TA)\,\mathrm{Vec}\, U_n \\
&\quad + (\mathrm{Vec}\, U_m^T)^T(A^TC \otimes BD^T)\,\mathrm{Vec}\, U_n + (\mathrm{Vec}\, U_m)^T(C^TC \otimes DD^T)\,\mathrm{Vec}\, U_n \\
&= \mathrm{tr}(BB^TU_m^TA^TAU_n) + \mathrm{tr}(A^TCU_m^TDB^TU_n^T) + \mathrm{tr}(DB^TU_m^TA^TCU_n^T) \\
&\quad + \mathrm{tr}(C^TCU_m^TDD^TU_n) \\
&= (\mathrm{Vec}(BB_k^TU_n^TA^TA))^T \mathrm{Vec}\, U_m^T + (\mathrm{Vec}(C^TAU_nBD^T))^T \mathrm{Vec}\, U_n^T \\
&\quad + (\mathrm{Vec}(BD^TU_nC^TA))^T \mathrm{Vec}\, U_m^T + (\mathrm{Vec}(C^TCU_n^TDD^T))^T \mathrm{Vec}\, U_m^T \\
&= [(BB^T \otimes A^TA)\,\mathrm{Vec}\, U_n]^T \mathrm{Vec}\, U_m + [(C^TA \otimes DB^T)\,\mathrm{Vec}\, U_n^T]^T \mathrm{Vec}\, U_m \\
&\quad + [(BD^T \otimes A^TC)\,\mathrm{Vec}\, U_n^T]^T \mathrm{Vec}\, U_m + [(C^TC \otimes DD^T)\,\mathrm{Vec}\, U_n]^T \mathrm{Vec}\, U_m \\
&= [\mathrm{Vec}(A^TAU_nBB^T)]^T \mathrm{Vec}\, U_m + [\mathrm{Vec}(DB^TU_n^TA^TC)]^T \mathrm{Vec}\, U_m \\
&\quad + [\mathrm{Vec}(A^TCU_n^TDB^T)]^T \mathrm{Vec}\, U_m + [\mathrm{Vec}(DD^TU_nC^TC)]^T \mathrm{Vec}\, U_m \\
&= [\mathrm{Vec}(A^T(AU_nB + CU_n^TD)B^T + D(B^TU_n^TA^T + D^TU_nC^T)C)]^T \mathrm{Vec}\, U_m \\
&= (\mathrm{Vec}\, H_n)^T \mathrm{Vec}\, U_m \\
&= \mathrm{tr}(H_n^TU_m) \\
&= \mathrm{tr}(U_m^TH_n). \square
\end{aligned}$$

Lemma 3.10. Assume that the iterative sequences $\{R_r\}$, $\{U_r\}$ and $\{H_r\}$, constructed by Algorithm 4, satisfy

$$\mathrm{tr}(R_r^TR_{r-1}) = 0, \quad \textit{and} \quad \mathrm{tr}(U_{r+1}^TH_r) = 0, \qquad \textit{for any} \quad r. \tag{3.17}$$

Proof. The proof is similar to that of Lemma 3.3 by apply Lemmas 3.8 and 3.9. □

Lemma 3.11. Assume that the iterative sequences $\{R_r\}$, $\{U_r\}$ and $\{H_r\}$, constructed by Algorithm 4. Then

$$\mathrm{tr}(R_r^TR_0) = 0, \quad \mathrm{tr}(U_{r+1}^TH_1) = 0, \qquad \textit{for any} \quad r. \tag{3.18}$$

Proof. The proof is similar to that of Lemma 3.4, by using Lemmas 3.8 and 3.9. □

Theorem 3.12. Assume that the iterative sequences $\{R_r\}$, $\{U_r\}$ and $\{H_r\}$ are constructed by Algorithm 4. Then for any integers m, n, $(m \neq n)$, we have

$$\mathrm{tr}(R_{m-1}^TR_{n-1}) = 0, \quad \text{and} \quad \mathrm{tr}(U_m^TH_n) = 0. \tag{3.19}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Proof. The proof is similar to that of Theorem 3.5, by using Lemmas 3.10 and 3.11. □

Theorem 3.13. For any given initial matrix $X_0 \in \mathbb{R}^{n\times p}$, the solution X of minimization problem Eq. (3.8) can be obtained by Algorithm 4 within finite steps, that is, in at most np iterations.

Proof. According to Algorithm 4, assume that the set of residual matrices $R_r \neq 0$ for $r = 0, 1, \ldots, np-1$. We will get R_{np} by iterative Algorithm in the next step. Suppose that $R_{np} \neq 0$, then we have the set of residual matrices $\{R_0, R_1, \ldots, R_{np}\}$ having $np+1$ elements By Theorem 3.12, the set of residual matrices $\{R_0, R_1, \ldots, R_{np}\}$ is orthogonal in $\mathbb{R}^{n\times p}$ with respect to the Frobenius inner product (2.9). So, the set of residual matrix is linearly independent. That contradict with the dimension of $\mathbb{R}^{n\times p}$ is np. Because the linearly independent subset of $\mathbb{R}^{n\times p}$ must not contain more than np elements. Thus, $R_{np} = 0$, hence the matrix X_{np} is a solution of minimization problem Eq. (3.8), that mean, X_{np} is unique least-squares solution of Eq. (1.1). □

We propose a conjugate gradient algorithm to solve Problem 2. By adapting the same idea as shown above to derive an algorithm for Equation (1.2) as follows:

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Algorithm 5: A conjugate gradient iterative algorithm for Equation (1.2)

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

Given an ϵ such that $\epsilon > 0$, set $r = 0$, $U_0 = 0$. Choose $X_0 \in \mathbb{R}^{n\times p}$;

$$R_0 = \sum_{k=1}^{s} A_k^T E B_k^T + \sum_{l=1}^{t} D_l E^T C_l - \sum_{k=1}^{s} A_k^T \Big(\sum_{i=1}^{s} A_i X_0 B_i + \sum_{j=1}^{t} C_j X_0^T D_j \Big) B_k^T - \sum_{l=1}^{t} D_l \Big(\sum_{i=1}^{s} B_i^T X_0^T A_i^T + \sum_{j=1}^{t} D_j^T X_0 C_j^T \Big) C_l;$$

for $r = 0$ **to** np **do**

  if $\|R_r\|_F \leqslant \epsilon$ **then**

    X_r is a least-squares solution; break;

  else

    if $r = 0$ **then**

      set $U_{r+1} = R_r$;

    else

      set $U_{r+1} = R_r + \frac{\|R_r\|_F^2}{\|R_{r-1}\|_F^2} U_r$;

    end

  end

$$H_{r+1} = \sum_{k=1}^{s} A_k^T \Big(\sum_{i=1}^{s} A_i U_{r+1} B_i + \sum_{j=1}^{t} C_j U_{r+1}^T D_j \Big) B_k^T + \sum_{l=1}^{t} D_l \Big(\sum_{i=1}^{s} B_i^T U_{r+1}^T A_i^T + \sum_{j=1}^{t} D_j^T U_{r+1} C_j^T \Big) C_l;$$

$$\alpha_{r+1} = \mathrm{tr}\Big(U_{r+1}^T H_{r+1} \Big);$$

$$X_{r+1} = X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}} U_{r+1};$$

$$R_{r+1} = \sum_{k=1}^{s} A_k^T E B_k^T + \sum_{l=1}^{t} D_l E^T C_l - \sum_{k=1}^{s} A_k^T \Big(\sum_{i=1}^{s} A_i X_{r+1} B_i + \sum_{j=1}^{t} C_j X_{r+1}^T D_j \Big) B_k^T - \sum_{l=1}^{t} D_l \Big(\sum_{i=1}^{s} B_i^T X_{r+1}^T A_i^T + \sum_{j=1}^{t} D_j^T X_{r+1} C_j^T \Big) C_l;$$

  update r;

end

end

Theorem 3.14. Consider Eq. (1.2), for given matrices $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$, $D \in \mathbb{R}^{m\times q}$, $E \in \mathbb{R}^{m\times q}$ and unknown matrix $X \in \mathbb{R}^{n\times p}$. Assume that the matrix

$$M := \sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j) P(n, p) \tag{3.20}$$

is of full-column rank. Algorithm 5 solves Problem 2 for any given initial matrix $X_0 \in \mathbb{R}^{n\times p}$. The sequence $\{X_r\}$ constructed by Algorithm 5 can obtained a unique least-squares solution.

This document is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Proof. The proof of is similar to that of Theorem 3.13. □

3.5 Minimal-norm least-squares solution via a conjugate gradient algorithm

We consider the case when the matrix M in (3.11) is not full-column rank. The associated normal equation (3.11) has many solutions, that is, Equation (1.1) may have many least-squares solutions.

Recall that the set of Least-squares solutions of Equation (1.1) is denoted by $\mathcal{L}$, we will search for the elements of $\mathcal{L}$ with the minimal Frobenius norm.

Lemma 3.15. Assume that $\hat{X} \in \mathcal{L}$. Then, any arbitrary element $\check{X} \in \mathcal{L}$ can be expressed as $\hat{X} + Z$ for some matrix $Z \in \mathbb{R}^{n\times p}$ satisfying

$$A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C \;=\; 0. \tag{3.21}$$

Proof. Let us denote the residual of the least-squares solutions $\hat{X}$ and $\check{X}$, according to Equation (3.14), by $R_{\hat{X}}$ and $R_{\check{X}}$, respectively. We consider the different $Z := \check{X} - \hat{X}$. Now, we compute

$$\begin{aligned} R_{\check{X}} \;&=\; A^T(A(\hat{X}+Z)B + C(\hat{X}+Z)^TD)B^T + D(B^T(\hat{X}+Z)^TA^T + D^T(\hat{X}+Z)C^T)C \\ &\quad - A^TEB^T - DE^TC \\ &=\; A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C - R_{\hat{X}}. \end{aligned}$$

Since $\hat{X}, \check{X} \in \mathcal{L}$, by Lemma 3.7 we have $R_{\hat{X}} = R_{\check{X}} = 0$. It follows that Equation (3.21) holds as desired. □

Theorem 3.16. Consider the sequence $\{X_r\}$ generated by Algorithm 4 starting with the initial matrix

$$X_0 \;=\; A^T(AV_0B + CV_0^TD)B^T + D(B^TV_0^TA^T + D^TV_0C^T)C, \tag{3.22}$$

where $V_0 \in \mathbb{R}^{n\times p}$ is an arbitrary matrix, or especially $X_0 = 0$. Then the sequence $\{X_r\}$ generated by Algorithm 4 converges to the minimal-norm least-squares solution of Equatiom (1.1) in at most np iterations.

Proof. If we run Algorithm 4 starting with (3.22), then we can write the solution X^* of Eq. (3.8) so that

$$X^* \;=\; A^T(AV^*B + CV^{*T}D)B^T + D(B^TV^{*T}A^T + D^TV^*C^T)C,$$

for some matrix $V^* \in \mathbb{R}^{n\times p}$. Now, assume that $\check{X}$ is an arbitrary element in $\mathcal{L}$. By Lemma 3.15, there is a matrix $Z \in \mathbb{R}^{n\times p}$ such that $\check{X} = X^* + Z$ and

This material is reserved for educational use only, not allowed for commercial use.

$$A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C \;=\; 0.$$

Forbidden to modify the content, and cite the document when use.

Using the property 2.10, we get

$$\begin{aligned}\langle X^*, Z\rangle &= \left\langle A^T(AV^*B + CV^{*T}D)B^T + D(B^TV^{*T}A^T + D^TV^*C^T)C, Z\right\rangle \\ &= \left\langle V^*, A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C\right\rangle \\ &= 0.\end{aligned}$$

Since X^* is orthogonal to Z, it follows from the Pythagorean theorem that

$$\|\check{X}\|_F^2 = \|X^* + Z\|_F^2 = \|X^*\|_F^2 + 2\langle X^*, Z\rangle + \|Z\|_F^2 = \|X^*\|_F^2 + \|Z\|_F^2 \geq \|X^*\|_F^2.$$

Therefore, we have $\|\check{X}\|_F^2 \geq \|X^*\|_F^2$ for any arbitrary element $\check{X} \in \mathcal{L}$, hence we can conclude that the solution X^* is the minimal-norm solution. □

Theorem 3.17. Algorithm 5 solves Problem 3 in at most np iterations by starting with the initial matrix

$$X_0 = \sum_{k=1}^{s} A_k^T \left(\sum_{i=1}^{s} A_i V_0 B_i + \sum_{j=1}^{t} C_j V_0^T D_j\right) B_k^T + \sum_{l=1}^{t} D_l \left(\sum_{i=1}^{s} B_i^T V_0^T A_i^T + \sum_{j=1}^{t} D_j^T V_0 C_j^T\right) C_l,$$

where $V_0 \in \mathbb{R}^{n\times p}$ is an arbitrary matrix, or especially $X_0 = 0 \in \mathbb{R}^{n\times p}$. Then the sequence $\{X_r\}$ generated by Algorithm 5 converges to the minimal-norm least-squares solution of Equatiom (1.2) in at most np iterations.

Proof. The proof is similar to that of Theorem 3.16. □

3.6 Least-squares solution closest to a given matrix

In this case, consider the solution X of (1.1) generated by Algorithm 4 may have many least-squares solutions. We shall seek for one that closest to a given matrix Y with respect to the Frobenius norm.

Theorem 3.18. From Algorithm 4 by substituting E with $E_1 = E - (AYB + CY^TD)$, and choosing the initial matrix to be

$$W_0 = A^T(AVB + CV^TD)B^T + D(B^TV^TA^T + D^TVC^T)C, \tag{3.23}$$

where $V \in \mathbb{R}^{n\times p}$ is arbitrary, or specially $W_0 = 0 \in \mathbb{R}^{n\times p}$. Then the sequence X_r obtained by Algorithm 4 converges to the least-squares solution of (1.2) closest to Y within np iterations.

Proof. Let $Y \in \mathbb{R}^{n\times p}$ be given.

$$\begin{aligned}&\min_{X\in\mathbb{R}^{n\times p}} \|AXB + CX^TD - E\|_F \\ &\quad= \min_{X\in\mathbb{R}^{n\times p}} \|AXB + CX^TD - E - AYB - CY^TD + AYB + CY^TD\|_F \\ &\quad= \min_{X\in\mathbb{R}^{n\times p}} \|A(X-Y)B + C(X-Y)^TD - E + AYB + CY^TD\|_F.\end{aligned}$$

This material is reserved for educational purposes only. Allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Now, substituting $E_1 = E-(AYB+CY^TD)$ and $W = X-Y$, we see that the solution $\check{X}$ is equal to $W^* + Y$ where W^* is the minimal-norm least-squares solution of the equation

$$AWB + CW^TD \;=\; E_1,$$

in unknown W. By Theorem 3.16, the matrix W^* can be solved by Algorithm 4 with the initial matrix (3.23) where $V \in \mathbb{R}^{n\times p}$ is arbitrary matrix, or especially $W_0 = 0$. □

Theorem 3.19. Suppose that the matrix Equation (1.2) is inconsistent. Let $Y \in \mathbb{R}^{n\times p}$ be given. Consider Algorithm 5 solves Problem 4 when we replace the matrix E by

$$E_1 \;=\; E - \sum_{i=1}^{s} A_iYB_i - \sum_{j=1}^{t} C_jY^TD_j,$$

and choose the initial matrix

$$W_0 \;=\; \sum_{k=1}^{s} A_k^T \left(\sum_{i=1}^{s} A_iFB_i + \sum_{j=1}^{t} C_jF^TD_j \right) B_k^T + \sum_{l=1}^{t} D_l \left(\sum_{i=1}^{s} B_i^TF^TA_i^T + \sum_{j=1}^{t} D_j^TFC_j^T \right) C_l,$$

where $F \in \mathbb{R}^{n\times p}$ is arbitrary, or $W_0 = 0 \in \mathbb{R}^{n\times p}$. Then, the sequence $\{X_r\}$ obtained by Algorithm 5 converges to the least-squares solution of (1.2) closest to Y within np iterations.

Proof. The proof of the theorem is similar to that of Theorem 3.18. □

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 4

Numerical experiments

In this chapter, we provide numerical experiments to show the applicability and the performance of our algorithm. These algorithms have been carried out by MATLAB R2021a, on Mac operating system (M1 chip 8C CPU/8C GPU/8GB/512GB). We denote that ones(m,n) the m-by-n matrix whose all entries are 1. Each random matrix rand(m,n) has all entries belonging to the interval $(0,1)$. Here, we considered the coefficient matrices of generalized Sylvester-transpose matrix equation are moderate/large sizes. We shall present a few numerical results to compared our algorithm with the well-known algorithms and traditional direct method. The performance of our algorithms is investigated through the aspects of the number of iteration steps (denoted by 'ITs'), the norm of residual matrices (denoted by 'ERR'), and elapsed CPU time (denoted by 'CPU') measured in seconds by using the functions tic and toc in MATLAB.

4.1 Consistent matrix equations

In the following examples, some numerical results are presented to illustrate the stability and performance of Algorithm 3. Also the proposed algorithm is compared with the well-known algorithms and traditional direct method.

Example 1. Consider a generalized Sylvester-transpose equation

$$AXB + CX^TD = E$$

with 4×4 coefficient matrices $A = \text{tridiag}(-2,-3,-2)$, $B = \text{tridiag}(-1,1,-1)$, $C = \text{tridiag}(0,-1,0)$, $D = \text{tridiag}(0,2,0)$,

$$E = \begin{bmatrix} -7 & 6 & 0 & -2 \\ -5 & 9 & -2 & 0 \\ -4 & 5 & -1 & 1 \\ -2 & 2 & 4 & -3 \end{bmatrix}.$$

According to the discussion in Section 3.1, we can verify whether this matrix equation is consistent by inspecting the associated matrix $K \in \mathbb{R}^{16\times 16}$ defined by Eq. (3.2). It turns out that the matrix K is symmetric and $\text{rank}\, K = 16$. Thus, the matrix equation has a unique solution. 3.6 then implies that, starting with any initial matrix X_0, Algorithm 3 will produce an exact solution within $np = 16$ steps. Indeed, we run this algorithm using $X_0 = 0$, and we get the following information about the associated residual matrices in Table 4.1. We obtain the desire solution X_{15} in 0.028298 seconds.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

Table 4.1: The norm of residual matrix at k iteration 1.

k	The norm of residual matrix R_k
0	4.4922
1	14.0418
2	3.3348
3	2.4517
4	14.5817
5	1.1071
6	1.1241
7	2.5486
8	0.4739
9	10.8790
10	0.3062
11	0.3708
12	0.0193
13	0.0117
14	0.0013
15	0.0000

Example 2. Consider a generalized Sylvester-transpose equation

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

with 50×50 tridiagonal coefficient matrices given by

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

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

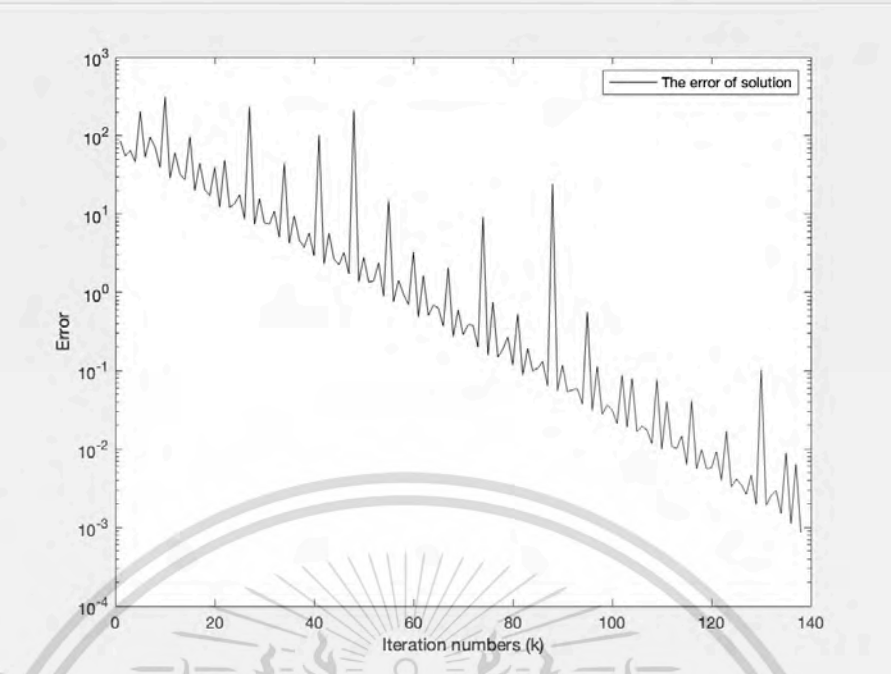

Figure 4.1: Error for Example 2.

We can check that $\text{rank}[K\ b] = \text{rank}\, K = 2500$, and $K^T = K$. Thus, this matrix equation has a unique solution. Results from running the Algorithm 3 by using an initial matrix $X_0 = 0.25 \times \text{ones} \in \mathbb{R}^{50\times 50}$ show in Figure 4.1. We get an approximate solution within 138 iterations (so that $\|R_k\| \leq 10^{-3}$), which is significantly less than the theoretical one (10^4 iterations) and only spends totally 0.131079 seconds while the direct Kronecker linearization takes 1.581769 seconds to obtain the exact soluton.

Example 3. Consider a large-scaled generalized Sylvester-transpose equation

$$A_1 X B_1 + C_1 X^T D_1 + C_2 X^T D_2 = E$$

where all matrices are 100×100 tridiagonal matrices given by

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

In this example, we can check $\text{rank}\, K = \text{rank}[K\ b] = 10000$ and the matrix K is symmetric. So, this matrix equation has a unique solution.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

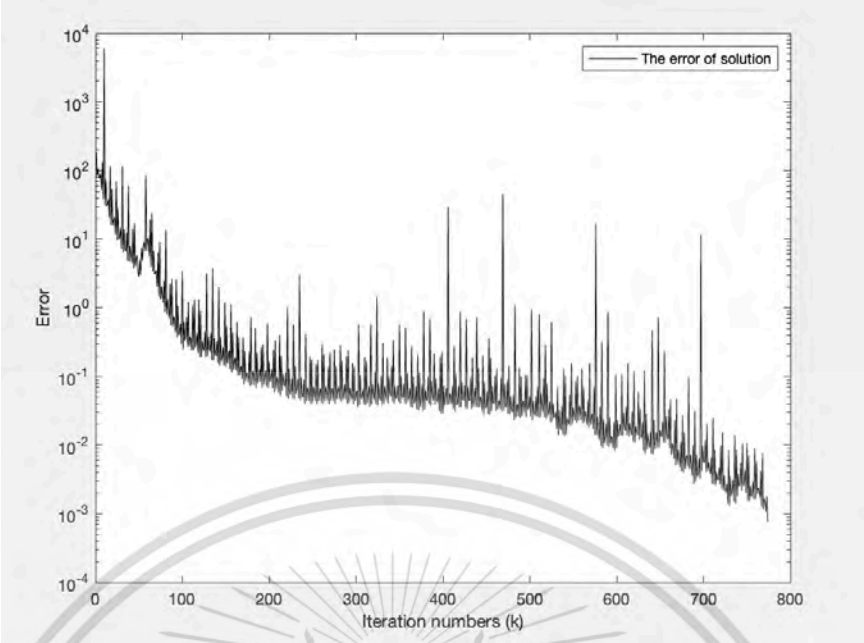

Figure 4.2: Error for Example 3.

Figure 4.2 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 $X_0 = 5 \times$ ones, $X_0 = 0$, and $X_0 = -5 \times$ ones. From Table 4.2 we can see that, no matter the initial point we will always get the approximate solution in a faster time than the direct method.

Table 4.2: Error and computational time for Example 3

Initial matrix	ITs	CPU	ERR
Direct	-	69.500953	0
$X_0 = 5 \times$ ones	830	2.247507	8.9145 $\times 10^{-4}$
$X_0 = 0.5 \times$ ones	774	1.986782	7.5755 $\times 10^{-4}$
$X_0 = 0$	16	0.106482	5.3862 $\times 10^{-4}$
$X_0 = -5 \times$ ones	830	2.190269	9.1232 $\times 10^{-4}$

Example 4. Consider a generalized Sylvester-transpose matrix equation

$$AXB + CX^T D = E,$$

with large-scaled matrices A, B, C, D, E size 100×100 as follows:

$$A = \text{tridiag}(-1, 3, -1), \quad B = \text{tridiag}(1, 7, 1), \quad C = 6 \times \text{ones}(m, p),$$
$$D = -3 \times \text{ones}(n, q), \quad E = 0.7 \times I_{100}.$$

This example is consistent and has a unique solution, because rank $K =$ rank$[K\ b] = 10000$, and $K \in \mathbb{R}^{10000 \times 10000}$ is of full-column rank and the matrix K is symmetric. We

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

show the efficiency of Algorithm 3 by compare with GI [19] and AGBI [20] algorithms. For GI algorithm, we use 3 different convergent factors namely, $m1 = 6.1728 \times 10^{-12}$, $m2 = 8.8183 \times 10^{-12}$ and $m3 = 3.0864 \times 10^{-11}$. We report the result in Figure 4.3 and Table 4.3. In conclusion, our algorithm takes a small time (0.073613 seconds) and a small-error solution, while GI, AGBI and the direct method consume a big amount of time to get the exact solution.

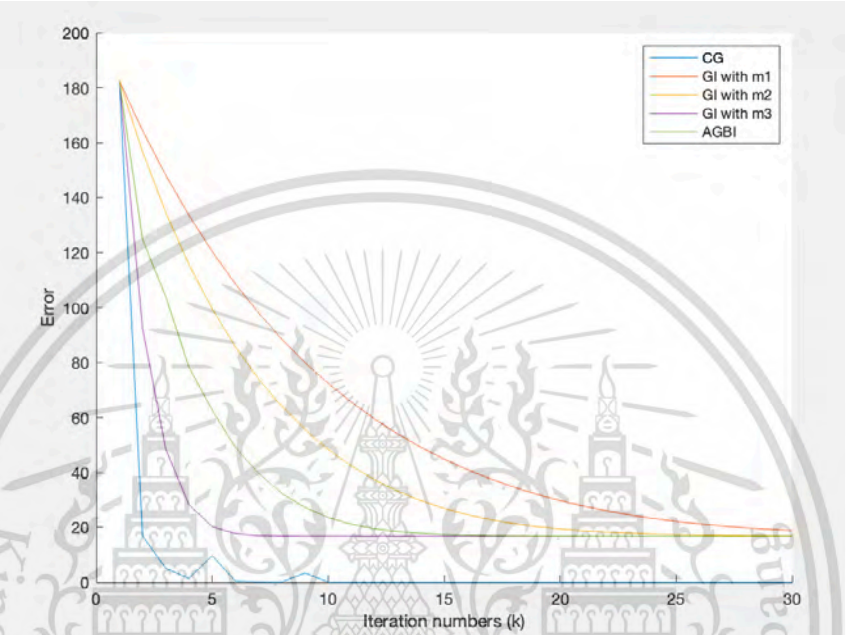

Figure 4.3: Error for Example 4.

Table 4.3: Error and computational time for Example 4

Method	ITs	CPU	ERR
CG	30	0.073613	0.000001
GI with m1	30	0.109370	18.788266
GI with m2	30	0.115890	16.853503
GI with m3	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:

$$A = \text{tridiag}(-1, 2, -1), \quad B = \text{tridiag}(3, -6, 3), \quad C = -3 \times \text{ones}(n, q),$$

$$D = \frac{1}{3} \times \text{ones}(p, q), \quad E = -1.2 \times \text{ones}(m, q).$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

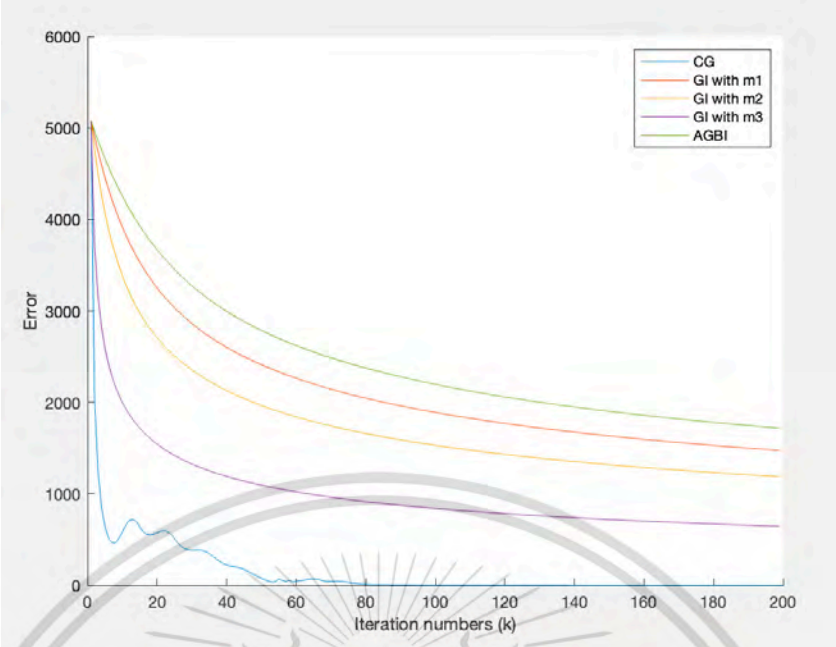

Figure 4.4: Error for Example 5.

Table 4.4: Error and computational time for Example 5

Method	ITs	CPU	ERR
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

Since associated matrix K in this example is symmetric and full-column rank, and rank K = rank$[K\ \ b] = 10000$. thus this matrix equation is consistent and has a unique solution. We run the Algorithm 3 compare with GI [19] and AGBI [20] algorithms by choosing the same initial matrix $X_0 = -0.4 \times$ ones $\in \mathbb{R}^{100\times 100}$. Figure 4.3 and Table 4.3 show that the comparison between Algorithm 3 and well-known algorithms (GI and AGBI). In 200 iterations, the computational time and error of conjugate gradient algorithm 3 is slightly better than those of GI with parameters $m1$, $m2$, $m3$ and AGBI algorithms.

4.2 Inconsistent matrix equations

In the following numerical examples, we use several examples to validate the stability and performance of Algorithm 5. We compared our algorithm with the well-known algorithms and traditional direct method.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Example 6. Consider a generalized Sylvester-transpose matrix equation

$$A_1XB_1 + A_2XB_2 + A_3XB_3 + C_1X^TD_1 + C_2X^TD_2 \;=\; E, \tag{4.1}$$

where the moderate-scaled coefficient matrices are given by

$A_1 = 0.5 \times \text{ones}(m,n) - \text{rand}(m,n) \in \mathbb{R}^{50 \times 50}$, $A_2 = 0.5 \times \text{ones}(m,n) - \text{rand}(m,n) \in \mathbb{R}^{50 \times 50}$,
$A_3 = 0.5 \times \text{ones}(m,n) - \text{rand}(m,n) \in \mathbb{R}^{50 \times 50}$, $B_1 = 0.5 \times \text{ones}(p,q) - \text{rand}(p,q) \in \mathbb{R}^{40 \times 50}$,
$B_2 = 0.5 \times \text{ones}(p,q) - \text{rand}(p,q) \in \mathbb{R}^{40 \times 50}$, $B_3 = 0.5 \times \text{ones}(p,q) - \text{rand}(p,q) \in \mathbb{R}^{40 \times 50}$,
$C_1 = 0.5 \times \text{ones}(m,p) - \text{rand}(m,p) \in \mathbb{R}^{50 \times 40}$, $C_2 = 0.5 \times \text{ones}(m,p) - \text{rand}(m,p) \in \mathbb{R}^{50 \times 40}$,
$D_1 = 0.5 \times \text{ones}(n,q) - \text{rand}(n,q) \in \mathbb{R}^{50 \times 50}$, $D_2 = 0.5 \times \text{ones}(n,q) - \text{rand}(n,q) \in \mathbb{R}^{50 \times 50}$,
$E = 0.5 \times \text{ones}(m,q) - \text{rand}(m,q) \in \mathbb{R}^{50 \times 50}$.

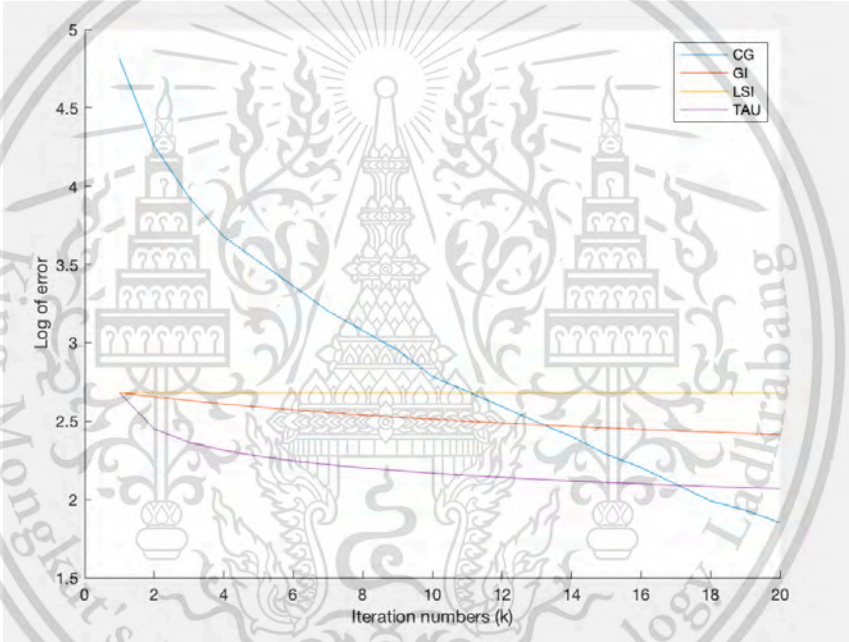

Figure 4.5: The logarithm of error for Example 6.

Table 4.5: Error and computational time for Example 6.

Method	ITs	CPU	ERR
CG	20	0.199308	6.407766
GI	20	0.129715	10.907665
LSI	20	0.179449	14.390460
TAUOpt	20	0.073866	7.806273
Direct	–	7.048632	0

In this example, we have $\text{rank}\, M = 2000 \neq 2001 = \text{rank}\,[M \;\; e]$, so the matrix equation Eq. (4.1) does not have an exact solution. However, we check M is of full-column rank, so this equation has a unique least-squares solution. The Figure 4.5 and

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Table 4.5 shows that, for an initial matrix $X_0 = 0 \in \mathbb{R}^{50\times 40}$ the Algorithm 5 takes 20 iterations to find a least-square solution of Example 6 with a tolerance error $\epsilon = \|Mx^* - e\| = 6.4812$, where $x^* = (M^TM)^{-1}M^Te$. Movrover, we also compare the performance of Algorithm 5 with GI method [24], LSI method [24], and TAUOpt method [38]. Those algorithms are all well-known algorithms. After running 20 iterations, our conjugate geadient Algorithm 5, It takes a little more time to compute a least-square solution than other methods but the relative error $\|R_r\|_F$ less than well-known algorithms.

Example 7. Consider a generalized Sylvester-transpose matrix equation

$$A_1XB_1 + C_1X^TD_1 + C_2X^TD_2 = E, \tag{4.2}$$

with the moderate-scaled coefficient matrices:

$A_1 = -0.08 \times \text{ones}(m,n) \in \mathbb{R}^{30\times 25}$, $B_1 = \text{tridiag}(0.11, -0.61, -0.29) \in \mathbb{R}^{30\times 30}$,

$C_1 = \text{tridiag}(-0.03, -0.22, -0.1) \in \mathbb{R}^{30\times 30}$, $C_2 = \text{tridiag}(0.38, 0.29, -0.41) \in \mathbb{R}^{30\times 30}$,

$D_1 = -0.13 \times \text{ones}(n,q) \in \mathbb{R}^{25\times 30}$, $D_2 = 0.04 \times \text{ones}(n,q) \in \mathbb{R}^{25\times 30}$,

$E = -0.01 \times I_{30} \in \mathbb{R}^{30\times 30}$.

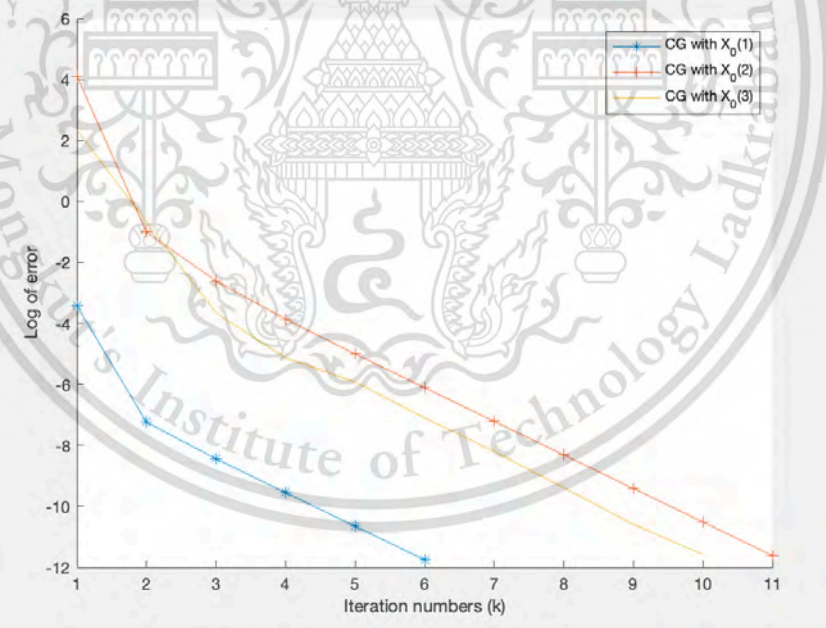

Figure 4.6: The logarithm of error for Example 7.

Table 4.6: Error and computational time for Example 7.

V_0	ITs	CPU	ERR
0	6	0.036523	0.000008
$0.02 \times \text{ones}(25, 30)$	11	0.036540	0.000009
$-0.01 \times I$	10	0.038425	0.000009

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

In this example, the matrix equation (4.2) is inconsistent and we check the associated matrix M is not full-column rank. So, Example 7 has many least-squares solutions. Figure 4.6 shows for three different initial matrices the relative errors $\|R_r\|_F$ compute by Algorithm 5 it is still decreasing rapidly. According to Theorem 3.17, we choose three matrices V_0 are different to generate the initial matrix X_0. The computational time and relative error are shown in Table 4.6. The direct method is another way that we try to compute the minimal-norm least-squares solutions in this example, Moore–Penrose inverse (3.12) takes 0.627019 seconds. While, MNLS method [39] cannot be used for this example because some coefficient matrices are triangular matrices with multiple zeros.

Example 8. Consider a generalized Sylvester-transpose matrix equation

$$A_1XB_1 + C_1X^TD_1 + C_2X^TD_2 = E, \tag{4.3}$$

with the moderate-scaled coefficient matrices:

$$A_1 = 0.2 \times \text{ones}(m,n) \in \mathbb{R}^{50\times 40}, \quad B_1 = \text{tridiag}(-0.2, 0.3, 0.3) \in \mathbb{R}^{50\times 50},$$
$$C_1 = \text{tridiag}(0.4, -0.2, -0.1) \in \mathbb{R}^{50\times 50}, \quad C_2 = \text{tridiag}(0.7, -0.2, 0.3) \in \mathbb{R}^{50\times 50},$$
$$D_1 = -0.2 \times \text{ones}(n,q) \in \mathbb{R}^{40\times 50}, \quad D_2 = 0.1 \times \text{ones}(n,q) \in \mathbb{R}^{40\times 50},$$
$$E = I_{50} \in \mathbb{R}^{50\times 50}.$$

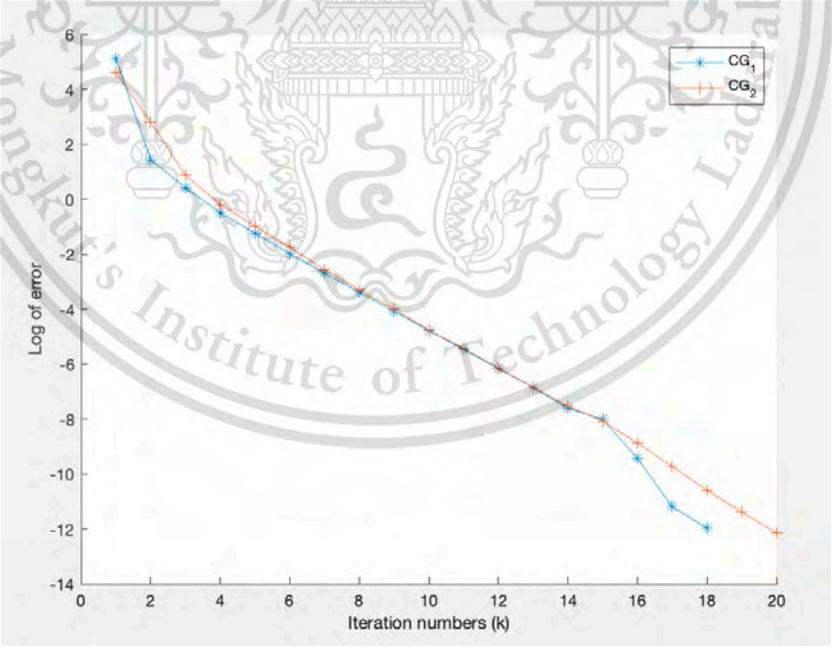

Figure 4.7: The logarithm of error for Example 8 with Y = 0.1 $\times$ ones(n,p).

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

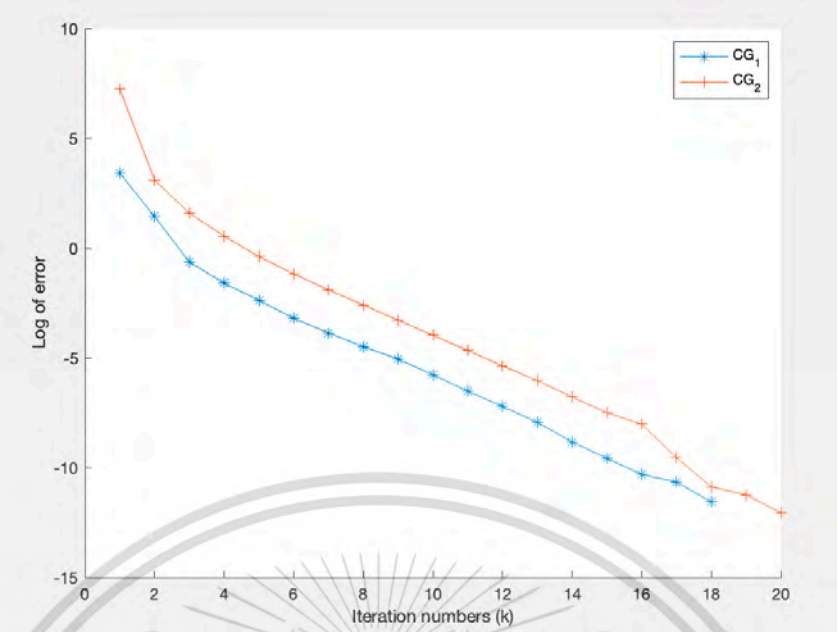

Figure 4.8: The logarithm of error for Example 8 with Y = I.

Table 4.7: Error and computational time for Example 8.

Y	Initial V	ITs	CPU	ERR	$\|X^* - Y\|_F$
$0.1 \times \text{ones}(n,p)$	0	18	0.104135	0.000006	4.3116
	$-0.19 \times I$	20	0.108153	0.000005	4.3116
I	0	18	0.113960	0.000009	0.8580
	$0.02 \times \text{ones}$	20	0.108499	0.000006	0.8580

We find that Equation (4.3) is inconsistent and has many least-squares solutions. In this example, we find the least-squares solution of Equation (4.3) closest to matrix Y. Figure 4.7 shows that Algorithm 5 with two different matrices $V = 0$ (CG_1) and $V = -0.19 \times I$ (CG_2) gets the least-squares solution closest to $Y = 0.1 \times \text{ones}(n,p) \in \mathbb{R}^{40\times 50}$. And the Figure 4.8 we see that, for $Y = I$ the Algorithm 5 can obtain a least-squares solution closest to matrix Y. with two different matrices $V = 0$ (CG_1) and $V = 0.02 \times \text{ones}$ (CG_2). For the number of iterations, computational time, relative error and $\|X^* - Y\|_F$ shown Table 4.7. The algorithm 5 performs well in terms of number of iterations and computation time. Moreover, changing the initial and desired matrices Y has no significant effect on the performance of our algorithm.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Chapter 5

Conclusion and suggestion

5.1 Conclusion

We propose conjugate gradient iterative algorithms, namely, Algorithms 3 and 5, to generate approximate solutions for the generalized Sylvester-transpose matrix equation (1.2). In the case that Eq. (1.2) is consistent, Algorithms 3 produces well-approximate solutions converging to the solution of Eq. (1.2) under an assumption that the matrix K in (3.2) is symmetric. The analysis shows that the residual matrices R_k, generated by Algorithms 3, is an orthogonal set. Thus, we get the desired solution within a finite step, says, np steps. For the case that Eq. (1.2) is inconsistent, we make an assumption that the associated matrix M is of full-column rank. The algorithm 5 will produce a unique least-squares solution of Eq. (1.2) within np iterations with the absence of round-off errors. If the matrix M is not full-column rank so, the matrix equation (1.2) has many least-squares solutions. In this caes, Algorithm 5 can search for the one with minimal Frobenius norm within np steps. Moreover, given a matrix Y, we can apply Algorithm 5 to find the least-squares solution closest to Y within np steps. The performance of algorithms are significantly better than the direct Kronecker linearization and well-known iterative methods for medium/large sizes of squares/non-squares matrices. We also show ability of the algorithm 5 is always applicable for any given initial matrix and the given matrix Y.

5.2 Suggestion

In this thesis, we propose CG algorithms to generate approximate solutions for the generalized Sylvester-transpose matrix equation (1.2) when coefficient matrices and an unknown matrix are real matrices. For future research, we may propose CG algorithm to solve for the generalized Sylvester-transpose matrix equation (1.2) over generalized quaternions.

Recall that the generalized quaternions $\mathbb{H}_G(a,b)$ over the real number field $\mathbb{R}$ can be expressed as:

$$x = x_1 + x_2 i + x_3 j + x_4 k, \qquad x_l \in \mathbb{R} \text{ for all } l = 1,2,3,4, \tag{5.1}$$

where $a, b \in \mathbb{R}$ are given, $i^2 = a$, $j^2 = b$, $k^2 = ijk = -ab$, $ij = -ji = k$, $jk = -kj = -bi$, $ik = -ki = aj$. When a and b take some special values, $\mathbb{H}_G(a,b)$ are corresponding some well-known structures. For example, $\mathbb{H}_G(-1,-1)$ is called the Hamilton quaternion $\mathbb{H}$, $\mathbb{H}_G(-1,1)$ is called the split quaternion ring, $\mathbb{H}_G(1,-1)$ is called the nectarine ring,

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

$\mathbb{H}_G(1,1)$ is called the conectarine ring, $\mathbb{H}_G(-1,0)$ is called the degenerate quaternions, $\mathbb{H}_G(1,0)$ is called the pseudo-degenerate quaternions, and $\mathbb{H}_G(0,0)$ is called the doubly degenerate quaternions (see e.g., [40, 41, 42]).

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

References

[1] B. Zhou, G. R. Duan, On the generalized Sylvester mapping and matrix equations, *Syst. Control Lett.*, **57** (2008), 200–208.

[2] L. Dai, *Singular control systems*, Berlin: Springer, 1989.

[3] G. R. Duan, Eigenstructure assignment in descriptor systems via output feedback: A new complete parametric approach, *Int. J. Control.*, **72** (1999), 345–364.

[4] F. Lewis, A survey of linear singular systems, *Circ. Syst. Signal Process.*, **5** (1986), 3–36.

[5] G. R. Duan, Parametric approaches for eigenstructure assignment in high-order linear systems, *Int. J. Control Autom. Syst.*, **3** (2005), 419–429.

[6] 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).

[7] 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.

[8] D. Hyland, D. Bernstein, The optimal projection equations for fixed order dynamic compensation, *IEEE Trans. Control.*, **29** (1984), 1034–1037.

[9] D. Calvetti, L. Reichel, Application of ADI iterative methods to the restoration of noisy images, *SIAM J. Matrix Anal. Appl.*, **17** (1996), 165–186.

[10] S. Y. Li, H. L. Shen, X. H. Shao, PHSS Iterative method for solving generalized Lyapunov equations, *Mathematics.*, **7** (2019), 38.

[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.

[12] H. L. Shen, Y. R. Li, X. H. Shao, The four-parameter PSS method for solving the Sylvester equation, *Mathematics.*, **7** (2019), 105.

[13] M. Dehghan, A. Shirilord, Solving complex Sylvester matrix equation by accelerated double-step scale splitting (ADSS) method, *Engineering with Computers*, **37** (2021), 489–508.

[14] F. Ding, T. Chen, Gradient based iterative algorithms for solving a class of matrix equations, *IEEE Trans. Automat. Comtr.*, **50** (2005), 1216–1221.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

[15] Q. Niu, X. Wang, L. Z. Lu, A relaxed gradient based algorithm for solving Sylvester equation, *Asian J. Control*, **13** (2011), 461–464.

[16] X. Wang, L. Dai, D. Liao, A modified gradient based algorithm for solving Sylvester equation, *Appl. Math. Comput.*, **218** (2012), 5620–5628.

[17] 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.

[18] N. Sasaki, P. Chansangiam, Modified Jacobi-gradient iterative method for generalized Sylvester matrix equation, *Symmetry.*, **12** (2020).

[19] L. Xie, J. Ding, F. Ding, Gradient based iterative solutions for general linear matrix equations, *Comput. Math. Appl.*, **58** (2009), 1441–1448.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] Xie, L.; Ding, J.; Ding, F. Gradient based iterative solutions for general linear matrix equations, *Comput. Math. Appl.* **2009**, *58*, 1441–1448.

[25] M. Hajarian, Developing BiCG and BiCR methods to solve generalized Sylvester-transpose matrix equations, *Int. J. Autom. Comput.*, **11** (2014), 25–29.

[26] M. Hajarian, Matrix form of the CGS method for solving general coupled matrix equations, *Appl. Math. Lett.*, **34** (2014), 37–42.

[27] 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.

[28] M. Hajarian, Generalized conjugate direction algorithm for solving the general coupled matrix equations over symmetric matrices, *Numer. Algor.*, **73** (2016), 591–609.

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

[29] M. Hajarian, Extending the CGLS algorithm for least squares solutions of the generalized Sylvester-transpose matrix equations, *J. Franklin Inst.*, **353** (2016), 1168–1185.

[30] 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.

[31] R. Horn, C. Johnson, *Topics in matrix analysis*, Cambridge: Cambridge University Press, 1991.

[32] J.R. Magnus, H. Neudecker, *Matrix Differential Calculus with Applications in Statistics and Econometrics*, Garsington Road, Oxford, UK, 2007.

[33] M.R. Hestenes, E. Stiefel, Methods of Conjugate Gradients for Solving LinearnSystems, *The National Bureau of Standards.*, **69(6)** (1952), 409–436.

[34] M. Rambo, *The Conjugate Gradient Method for Solving Linear Systems of Equations*, Saint Mary's College of California, 2016.

[35] C.C. Paige, M.A. Saunders, LSQR: An Algorithm for Sparse Linear Equations amd Least Squares., *ACM transactions.*, **8(1)** (1982), 43–71.

[36] A. Kittisopaporn, P. Chansangiam, Approximated least-squares solutions of a generalized Sylvester-transpose matrix equation via gradient-descent iterative algorithm, *Adv. Differ. Equ.*, **2021** (2021), 266.

[37] Wang, M.; Cheng, X. Iterative algorithms for solving the matrix equation $AXB + CX^TD = E$, *Appl. Math. Comput.* **2007**, *187*, 622–629.

[38] Kittisopaporn, A.; Chansangiam, P. Approximated least-squares solutions of a generalized Sylvester-transpose matrix equation via gradient-descent iterative algorithm, *Adv. Differ. Equations* **2021**, *2021(1)*. https://doi.org/10.1186/s13662-021-03427-4.

[39] Chen, X.; Ji, J. The minimum-norm least-squares solution of a linear system and symmetric rank-one updates, *Electron. J. Linear Algebra.* **2011**, *22*, 480–489.

[40] M. Akyiğit, H. H. Kösal, M. Tosun, *Fibonacci generalized quaternions*, *Adv. Appl. Clifford Algebras* **2014**, *24*, 631–641.

[41] Xin Liu and Yang Zhang, *Matrices over quaternion algebras*, Matrix and Operator Equations and Applications, Springer, pages 120–185, 2023.

[42] A.B. Mamagani and M. Jafari, *On properties of generalized quaternion algebra*, *Novel Applied Sciences* **2013**, *2*, 683–689.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Appendices

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Appendix A

The research Paper

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

AIMS Mathematics

http://www.aimspress.com/journal/Math

AIMS Mathematics, 7(4): 5386–5407.
DOI:10.3934/math.2022299
Received: 26 October 2021
Revised: 13 December 2021
Accepted: 21 December 2021
Published: 06 January 2022

Research article

Conjugate gradient algorithm for consistent generalized Sylvester-transpose matrix equations

Kanjanaporn Tansri, Sarawanee Choomklang and Pattrawut Chansangiam*

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

* **Correspondence:** Email: pattrawut.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 agorithm; 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:

$$\sum_{i=0}^{s_1} A_i x^{(i)} + \sum_{j=0}^{s_2} B_j u^{(j)} = 0 \tag{1.1}$$

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 $B_j \in \mathbb{R}^{n\times n}$ are the system coefficient matrices; see e.g., [1, 2]. The dynamical linear system (1.1) includes

$$A_1\dot{x} + A_0 x + B_0 u = 0, \qquad \text{the descriptor linear system,} \tag{1.2}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

$$A_2\ddot{x} + A_1\dot{x} + A_0x + B_0u = 0, \quad \text{the second-order linear system,} \tag{1.3}$$

$$A_kx^k + A_{k-1}x^{k-1} + \cdots + A_0x = Bu, \quad \text{the high-order dynamical linear system,} \tag{1.4}$$

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 realted 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:

$$\sum_{i=1}^{s} A_iXB_i + \sum_{j=1}^{t} C_jX^TD_j = E. \tag{1.5}$$

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 required memories 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 variantions 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 fora 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 haif-step-update of GI method, called MGI method, introducing by Wang et al. [17]. The idea of GI algorithm can be used in conjuction with matrix diagonal-extraction to get AJGI [18] and MJGI [19] algorithms. See more GI algorithms for a generalized Sylvester matrix equations in [20–22]. For a generalized Sylvester-transpose matrix equation $AXB + CX^TD = E$, 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

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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-by-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 $\mathrm{Vec}(\cdot)$ assigns to each matrix $A = [a_{ij}] \in \mathbb{R}^{m\times n}$ the column vector

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

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*

$$\mathrm{Vec}\, AXB \;=\; (B^T \otimes A)\,\mathrm{Vec}\, X.$$

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

$$P(m,n) \;=\; \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij} \otimes E_{ij}^T \;\in\; \mathbb{R}^{mn\times mn} \tag{2.1}$$

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*

$$B \otimes A \;=\; P(n,p)^T\,(A \otimes B)\,P(n,q). \tag{2.2}$$

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

$$\mathrm{Vec}(X^T) \;=\; P(m,n)\,\mathrm{Vec}(X). \tag{2.3}$$

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

$$(\mathrm{Vec}\,(Y))^T\,(A \otimes B)\,\mathrm{Vec}\,(X) \;=\; \mathrm{tr}\left(A^T Y^T B X\right). \tag{2.4}$$

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

$$\|A\| \;=\; \left(\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2\right)^{\frac{1}{2}} \;=\; \left(\mathrm{tr}\left(A^T A\right)\right)^{\frac{1}{2}}.$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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 $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$, $D \in \mathbb{R}^{m\times q}$, $E \in \mathbb{R}^{m\times q}$ 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

$$\begin{aligned}
\operatorname{Vec} E &= \operatorname{Vec}\left(\sum_{i=1}^{s} A_iXB_i + \sum_{j=1}^{t} C_jX^TD_j\right)\\
&= \sum_{i=1}^{s} (B_i^T \otimes A_i)\operatorname{Vec} X + \sum_{j=1}^{t} (D_j^T \otimes C_j)\operatorname{Vec} X^T\\
&= \sum_{i=1}^{s} (B_i^T \otimes A_i)\operatorname{Vec} X + \sum_{j=1}^{t} (D_j^T \otimes C_j)P\operatorname{Vec} X\\
&= \left(\sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j)P\right)\operatorname{Vec} X. \qquad (3.1)
\end{aligned}$$

Let us denote $x = \operatorname{Vec} X$, $b = \operatorname{Vec} E$, and

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

Thus, Eq (3.1) is equivalent to a linear algebraic system $Kx = b$. Hence, Eq (3.1) is consistent if and only if the associated linear system is consistent (i.e., $\operatorname{rank}[K\ b] = \operatorname{rank} K$). 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):

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 $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$, $D \in \mathbb{R}^{m\times q}$, $E \in \mathbb{R}^{m\times q}$ 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

$$\left\|E - \sum_{i=1}^{s} A_i\tilde{X}B_i - \sum_{j=1}^{t} C_j\tilde{X}^TD_j\right\| < \epsilon.$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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 $X_0 \in \mathbb{R}^{n\times p}$
$R_0 = E - \sum_{i=1}^{s} A_i X_0 B_i - \sum_{j=1}^{t} C_j X_0 D_j$
for $k = 0$ **to** np **do**
  if $\|R_k\| \leqslant \epsilon$ **then**
    X_k is the disire 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
    end
    $V_{k+1} = \sum_{i=1}^{s} A_i U_{k+1} B_i + \sum_{j=1}^{t} C_j U_{k+1}^T D_j$
    $\alpha_{k+1} = \mathrm{tr}\left(U_{k+1}^T V_{k+1}\right)$
    $X_{k+1} = X_k + \frac{\|R_k\|^2}{\alpha_{k+1}} U_{k+1}$
    $R_{k+1} = E - \sum_{i=1}^{s} A_i X_{k+1} B_i - \sum_{j=1}^{t} C_j X_{k+1}^T D_j$
    update k
  end
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 $\|R_k\| \leqslant \epsilon$, e.g., the norm of the difference $X_{k+1} - X_k$ between sucessive 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.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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 disire solution in a finite step. We divides the proof into several lemmas.

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

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

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

$$\begin{aligned} R_{k+1} &= E - \sum_{i=1}^{s} A_i X_{k+1} B_i - \sum_{j=1}^{t} C_j X_{k+1}^T D_j \\ &= E - \sum_{i=1}^{s} A_i \left(X_k + \frac{||R_k||^2}{\alpha_{k+1}} U_{k+1} \right) B_i - \sum_{j=1}^{t} C_j \left(X_k^T + \frac{||R_k||^2}{\alpha_{k+1}} U_{k+1}^T \right) D_j \\ &= E - \sum_{i=1}^{s} A_i X_k B_i - \sum_{j=1}^{t} C_j X_k^T D_j - \frac{||R_k||^2}{\alpha_{k+1}} \left(\sum_{i=1}^{s} A_i U_k B_i + \sum_{j=1}^{t} C_j U_k^T D_j \right) \\ &= R_k - \frac{||R_k||^2}{\alpha_{k+1}} V_{k+1}. \end{aligned}$$

□

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

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

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

$$\begin{aligned} \mathrm{tr}\left(V_m^T U_n\right) &= (\mathrm{Vec}\, V_m)^T \mathrm{Vec}\, U_n \\ &= \left[\sum_{i=1}^{s} (B_i^T \otimes A_i) \mathrm{Vec}\, U_m + \sum_{j=1}^{t} (D_j^T \otimes C_j) \mathrm{Vec}\, U_m^T \right]^T \mathrm{Vec}\, U_n \\ &= [K \mathrm{Vec}\, U_m]^T \mathrm{Vec}\, U_n \\ &= (\mathrm{Vec}\, U_m)^T \ K \mathrm{Vec}\, U_n \\ &= (\mathrm{Vec}\, U_m)^T \left[\mathrm{Vec} \left(\sum_{i=1}^{s} A_i U_n B_i + \sum_{j=1}^{t} C_j U_n^T D_j \right) \right] \\ &= (\mathrm{Vec}\, U_m)^T \mathrm{Vec}\, V_n \\ &= \mathrm{tr}\left(U_m^T V_n\right) \end{aligned}$$

for any m, n. □

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Lemma 8. *Assume that the matrix K is symmetric. The sequences $\{R_i\}$, $\{U_i\}$ and $\{V_i\}$ are generated by Algorithm 1 satisfy*

$$\mathrm{tr}\left(R_m^T R_{m-1}\right) = 0 \quad \textit{and} \quad \mathrm{tr}\left(U_{m+1}^T V_m\right) = 0 \qquad \textit{for any} \quad m. \tag{4.3}$$

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

$$\begin{aligned}
\mathrm{tr}\left(R_1^T R_0\right) &= \mathrm{tr}\left[\left(R_0 - \frac{\|R_0\|^2}{\alpha_1} V_1\right)^T R_0\right] \\
&= \mathrm{tr}\left(R_0^T R_0\right) - \frac{\|R_0\|^2}{\alpha_1} \mathrm{tr}\left(V_1^T R_0\right) \\
&= \|R_0\|^2 - \|R_0\|^2 = 0,
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{tr}\left(U_2^T V_1\right) &= \mathrm{tr}\left[\left(R_1 + \frac{\|R_1\|^2}{\|R_0\|^2} U_1\right)^T V_1\right] \\
&= \mathrm{tr}\left(R_1^T V_1\right) + \frac{\|R_1\|^2}{\|R_0\|^2} \mathrm{tr}\left(U_1^T V_1\right) \\
&= -\frac{\alpha_1}{\|R_0\|^2} \mathrm{tr}\left(R_1^T R_1\right) + \alpha_1 \frac{\|R_1\|^2}{\|R_0\|^2} = 0.
\end{aligned}$$

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

In the inductive step, for $m = k$ we assume that $\mathrm{tr}(R_k^T R_{k-1}) = 0$ and $\mathrm{tr}(U_{k+1}^T V_k) = 0$. Then

$$\begin{aligned}
\mathrm{tr}\left(R_{k+1}^T R_k\right) &= \mathrm{tr}\left[\left(R_k - \frac{\|R_k\|^2}{\alpha_{k+1}} V_{k+1}\right)^T R_k\right] \\
&= \mathrm{tr}\left(R_k^T R_k\right) - \frac{\|R_k\|^2}{\alpha_{k+1}} \mathrm{tr}\left(V_{k+1}^T R_k\right) \\
&= \mathrm{tr}\left(R_k^T R_k\right) - \frac{\|R_k\|^2}{\alpha_{k+1}} \mathrm{tr}\left(V_{k+1}^T \left(U_{k+1} - \frac{\|R_k\|^2}{\|R_{k-1}\|^2} U_k\right)\right) \\
&= \|R_k\|^2 - \frac{\|R_k\|^2}{\alpha_{k+1}} \mathrm{tr}\left(V_{k+1}^T U_{k+1}\right) \\
&= 0,
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{tr}\left(U_{k+2}^T V_{k+1}\right) &= \mathrm{tr}\left[\left(R_{k+1} + \frac{\|R_{k+1}\|^2}{\|R_k\|^2} U_{k+1}\right)^T V_{k+1}\right] \\
&= \mathrm{tr}\left(R_{k+1}^T V_{k+1}\right) + \frac{\|R_{k+1}\|^2}{\|R_k\|^2} \mathrm{tr}\left(U_{k+1}^T V_{k+1}\right) \\
&= \mathrm{tr}\left(R_{k+1}^T \left(\frac{-\alpha_{k+1}}{\|R_k\|^2}(R_{k+1} - R_k)\right)\right) + \frac{\|R_{k+1}\|^2}{\|R_k\|^2} \alpha_{k+1}
\end{aligned}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

$$= \frac{\alpha_{k+1}}{\|R_k\|^2}\operatorname{tr}\left[\left(R_{k+1}^T R_k\right) - \left(R_{k+1}^T R_{k+1}\right)\right] + \frac{\|R_{k+1}\|^2}{\|R_k\|^2}\alpha_{k+1}$$
$$= 0.$$

Hence, Eq (4.3) holds for any m. □

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*

$$\operatorname{tr}\left(R_m^T R_0\right) = 0, \quad \operatorname{tr}\left(U_{m+1}^T V_1\right) = 0 \qquad \text{for any} \quad m. \tag{4.4}$$

Proof. From Lemma 8 for $m = 1$, we get $\operatorname{tr}(R_1^T R_0) = 0$ and $\operatorname{tr}(P_2^T Q_1) = 0$. 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

$$\begin{aligned}
\operatorname{tr}\left(R_{k+1}^T R_0\right) &= \operatorname{tr}\left[\left(R_k - \frac{\|R_k\|^2}{\alpha_{k+1}} V_{k+1}\right)^T R_0\right] \\
&= \operatorname{tr}\left(R_k^T R_0\right) - \frac{\|R_k\|^2}{\alpha_{k+1}}\operatorname{tr}(V_{k+1}^T R_0) \\
&= -\frac{\|R_k\|^2}{\alpha_{k+1}}\operatorname{tr}(V_{k+1}^T U_1) \\
&= -\frac{\|R_k\|^2}{\alpha_{k+1}}\operatorname{tr}(U_{k+1}^T V_1) = 0,
\end{aligned}$$

and

$$\begin{aligned}
\operatorname{tr}\left(U_{k+2}^T V_1\right) &= \operatorname{tr}\left(V_{k+2}^T U_1\right) \\
&= \operatorname{tr}\left[\left(\frac{-\alpha_{k+2}}{\|R_{k+1}\|^2}\left(R_{k+2} - R_{k+1}\right)\right)^T U_1\right] \\
&= \frac{-\alpha_{k+2}}{\|R_{k+1}\|^2}\left[\operatorname{tr}\left(R_{k+2}^T U_1\right) - \operatorname{tr}\left(R_{k+1}^T U_1\right)\right] \\
&= \frac{-\alpha_{k+2}}{\|R_{k+1}\|^2}\left[\operatorname{tr}\left(R_{k+2}^T R_0\right) - \operatorname{tr}\left(R_{k+1}^T R_0\right)\right] = 0.
\end{aligned}$$

Hence, Eq (4.4) holds for any m. □

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*

$$\operatorname{tr}\left(R_{m-1}^T R_{n-1}\right) = 0 \quad \text{and} \quad \operatorname{tr}\left(U_m^T V_n\right) = 0. \tag{4.5}$$

Proof. By Lemma 7 and the fact that $\operatorname{tr}(R_{m-1}^T R_{n-1}) = \operatorname{tr}(R_{n-1}^T R_{m-1})$ 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

$$\operatorname{tr}\left(R_{n+2}^T R_n\right) = \operatorname{tr}\left[\left(R_{n+1} - \frac{\|R_{n+1}\|^2}{\alpha_{n+2}} V_{n+2}\right)^T R_n\right]$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

$$
\begin{aligned}
&= -\frac{\|R_{n+1}\|^2}{\alpha_{n+2}} \operatorname{tr}\left[V_{n+2}^T\left(U_{n+1} - \frac{\|R_n\|^2}{\|R_{n-1}\|^2}U_n\right)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\frac{\|R_n\|^2}{\|R_{n-1}\|^2}\left[\operatorname{tr}\left(R_{n+1} + \frac{\|R_{n+1}\|^2}{\|R_n\|^2}U_{n+1}\right)^T V_n\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\frac{\|R_n\|^2}{\|R_{n-1}\|^2}\left[\operatorname{tr}\left(\frac{\alpha_n}{\|R_{n-1}\|^2}R_{n+1}^T\left(R_n - R_{n-1}\right)\right)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\frac{\|R_n\|^2}{\|R_{n-1}\|^2}\frac{\alpha_n}{\|R_{n-1}\|^2}\operatorname{tr}(R_{n+1}^T R_{n-1}),
\end{aligned}
$$

$$
\begin{aligned}
\operatorname{tr}\left(R_{n+1}^T R_{n-1}\right) &= \operatorname{tr}\left[\left(R_n - \frac{\|R_n\|^2}{\alpha_{n+1}}V_{n+1}\right)^T R_{n-1}\right] \\
&= -\frac{\|R_n\|^2}{\alpha_{n+1}}\operatorname{tr}\left[V_{n+1}^T\left(U_n - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}U_{n-1}\right)\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\left[\operatorname{tr}\left(R_n + \frac{\|R_n\|^2}{\|R_{n-1}\|^2}U_n\right)^T V_{n-1}\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\left[\operatorname{tr}\left(\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}R_n^T\left(R_{n-1} - R_{n-2}\right)\right)\right] \\
&= \frac{\|R_n\|^2}{\alpha_{n+1}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\operatorname{tr}(R_n^T R_{n-2}),
\end{aligned}
$$

$$
\begin{aligned}
\operatorname{tr}\left(U_{n+2}^T V_n\right) &= \operatorname{tr}\left[\left(R_{n+1} - \frac{\|R_{n+1}\|^2}{\|R_n\|^2}U_{n+1}\right)^T V_n\right] \\
&= \operatorname{tr}\left[R_{n+1}^T\left(\frac{-\alpha_n}{\|R_{n-1}\|^2}\left(R_n - R_{n-1}\right)\right)\right] \\
&= \frac{\alpha_n}{\|R_{n-1}\|^2}\operatorname{tr}\left[\left(R_n - \frac{\|R_n\|^2}{\alpha_{n+1}}V_{n+1}\right)^T R_{n-1}\right] \\
&= -\frac{\alpha_n}{\|R_{n-1}\|^2}\frac{\|R_n\|^2}{\alpha_{n+1}}\left[\operatorname{tr}\left(V_{n+1}^T U_n\right) - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\operatorname{tr}\left(V_{n+1}^T U_{n-1}\right)\right] \\
&= \frac{\alpha_n}{\|R_{n-1}\|^2}\frac{\|R_n\|^2}{\alpha_{n+1}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\operatorname{tr}\left(U_{n+1}^T V_{n-1}\right),
\end{aligned}
$$

and

$$
\begin{aligned}
\operatorname{tr}\left(U_{n+1}^T V_{n-1}\right) &= \operatorname{tr}\left[\left(R_n - \frac{\|R_n\|^2}{\|R_{n-1}\|^2}U_n\right)^T V_{n-1}\right] \\
&= \operatorname{tr}\left[R_n^T\left(\frac{-\alpha_{n-1}}{\|R_{n-2}\|^2}\left(R_{n-1} - R_{n-2}\right)\right)\right] \\
&= \frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\operatorname{tr}\left[\left(R_{n-1} - \frac{\|R_{n-1}\|^2}{\alpha_n}V_n\right)^T R_{n-2}\right]
\end{aligned}
$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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

Similarly, we can write $\operatorname{tr}(R_{n+2}^T R_n)$ and $\operatorname{tr}(U_{n+2}^T V_n)$ in terms of $\operatorname{tr}(R_{n+1}^T R_{n-1})$ and $\operatorname{tr}(U_{n+1}^T V_{n-1})$, respectively. Repeating this process until the terms $\operatorname{tr}(R_2^T R_0)$ and $\operatorname{tr}(U_3^T V_1)$ show up. By Lemma 9, we get $\operatorname{tr}(R_{n+2}^T R_n) = 0$ and $\operatorname{tr}(U_{n+2}^T V_n) = 0$.

Next, for $m = n + 3$, we have

$$\begin{aligned}
\operatorname{tr}\left(R_{n+3}^T R_n\right) &= \operatorname{tr}\left[\left(R_{n+2} - \frac{\|R_{n+2}\|^2}{\alpha_{n+3}}V_{n+3}\right)^T R_n\right] \\
&= -\frac{\|R_{n+2}\|^2}{\alpha_{n+3}}\operatorname{tr}\left[V_{n+3}^T\left(U_{n+1} - \frac{\|R_n\|^2}{\|R_{n-1}\|^2}U_n\right)\right] \\
&= \frac{\|R_{n+2}\|^2}{\alpha_{n+3}}\frac{\|R_n\|^2}{\|R_{n-1}\|^2}\left[\operatorname{tr}\left(R_{n+2} + \frac{\|R_{n+2}\|^2}{\|R_{n+1}\|^2}U_{n+2}\right)^T V_n\right] \\
&= \frac{\|R_{n+2}\|^2}{\alpha_{n+3}}\frac{\|R_n\|^2}{\|R_{n-1}\|^2}\left[\operatorname{tr}\left(\frac{\alpha_n}{\|R_{n-1}\|^2}R_{n+2}^T\left(R_n - R_{n-1}\right)\right)\right] \\
&= \frac{\|R_{n+2}\|^2}{\alpha_{n+3}}\frac{\|R_n\|^2}{\|R_{n-1}\|^2}\frac{\alpha_n}{\|R_{n-1}\|^2}\operatorname{tr}(R_{n+2}^T R_{n-1}),
\end{aligned}$$

$$\begin{aligned}
\operatorname{tr}\left(R_{n+2}^T R_{n-1}\right) &= \operatorname{tr}\left[\left(R_{n+1} - \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}V_{n+2}\right)^T R_{n-1}\right] \\
&= -\frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\operatorname{tr}\left[V_{n+2}^T\left(U_n - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}U_{n-1}\right)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\left[\operatorname{tr}\left(R_{n+1} + \frac{\|R_{n+1}\|^2}{\|R_n\|^2}U_{n+1}\right)^T V_{n-1}\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\left[\operatorname{tr}\left(\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}R_{n+1}^T\left(R_{n-1} - R_{n-2}\right)\right)\right] \\
&= \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\operatorname{tr}(R_{n+1}^T R_{n-2}),
\end{aligned}$$

$$\begin{aligned}
\operatorname{tr}\left(U_{n+3}^T V_n\right) &= \operatorname{tr}\left[\left(R_{n+2} - \frac{\|R_{n+2}\|^2}{\|R_{n+1}\|^2}U_{n+2}\right)^T V_n\right] \\
&= \operatorname{tr}\left[R_{n+2}^T\left(\frac{-\alpha_n}{\|R_{n-1}\|^2}\left(R_n - R_{n-1}\right)\right)\right] \\
&= \frac{\alpha_n}{\|R_{n-1}\|^2}\operatorname{tr}\left[\left(R_{n+1} - \frac{\|R_{n+1}\|^2}{\alpha_{n+2}}V_{n+2}\right)^T R_{n-1}\right] \\
&= -\frac{\alpha_n}{\|R_{n-1}\|^2}\frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\left[\operatorname{tr}\left(V_{n+2}^T U_n\right) - \frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\operatorname{tr}\left(V_{n+2}^T U_{n-1}\right)\right]
\end{aligned}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

$$= \frac{\alpha_n}{\|R_{n-1}\|^2}\frac{\|R_{n+1}\|^2}{\alpha_{n+2}}\frac{\|R_{n-1}\|^2}{\|R_{n-2}\|^2}\operatorname{tr}\left(U_{n+2}^T V_{n-1}\right),$$

and

$$\begin{aligned}
\operatorname{tr}\left(U_{n+2}^T V_{n-1}\right) &= \operatorname{tr}\left[\left(R_{n+1} - \frac{\|R_{n+1}\|^2}{\|R_n\|^2}U_{n+1}\right)^T V_{n-1}\right] \\
&= \operatorname{tr}\left[R_{n+1}^T\left(\frac{-\alpha_{n-1}}{\|R_{n-2}\|^2}(R_{n-1} - R_{n-2})\right)\right] \\
&= \frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\operatorname{tr}\left[\left(R_n - \frac{\|R_n\|^2}{\alpha_{n+1}}V_{n+1}\right)^T R_{n-2}\right] \\
&= -\frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\frac{\|R_n\|^2}{\alpha_{n+1}}\left[\operatorname{tr}\left(V_{n+1}^T U_{n-1}\right) - \frac{\|R_{n-2}\|^2}{\|R_{n-3}\|^2}\operatorname{tr}\left(V_{n+1}^T U_{n-2}\right)\right] \\
&= \frac{\alpha_{n-1}}{\|R_{n-2}\|^2}\frac{\|R_n\|^2}{\alpha_{n+1}}\frac{\|R_{n-2}\|^2}{\|R_{n-3}\|^2}\operatorname{tr}\left(U_{n+1}^T V_{n-2}\right).
\end{aligned}$$

Hence, we can write $\operatorname{tr}(R_{n+3}^T R_n)$ and $\operatorname{tr}(U_{n+3}^T V_n)$ in terms of $\operatorname{tr}(R_{n+2}^T R_{n-1})$ and $\operatorname{tr}(U_{n+2}^T V_{n-1})$, respectively. Repeating this process until the terms $\operatorname{tr}(R_3^T R_0)$ and $\operatorname{tr}(U_4 V_1)$ by Lemma 9, we get $\operatorname{tr}(R_{n+3}^T R_n) = 0$ and $\operatorname{tr}(U_{n+3}^T V_n) = 0$.

Suppose that $\operatorname{tr}(R_{m-1}^T R_{n-1}^T) = \operatorname{tr}(U_m^T V_n^T) = 0$ for $m = n+1, \ldots, k$. Then for $m = k+1$, we have

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

and

$$\begin{aligned}
\operatorname{tr}\left(U_{k+1}^T V_{n-1}\right) &= \operatorname{tr}\left[\left(R_k + \frac{\|R_k\|^2}{\|R_{k-1}\|^2}U_k\right)^T V_{n-1}\right] \\
&= \operatorname{tr}(R_k^T V_{n-1}) + \frac{\|R_k\|^2}{\|R_{k-1}\|^2}\operatorname{tr}(U_k^T V_{n-1}) \\
&= \operatorname{tr}\left[R_k^T\left(\frac{-\alpha_{n-1}}{\|R_{n-2}\|^2}(R_{n-1} - R_{n-2})\right)\right] \\
&= \frac{-\alpha_{n-1}}{\|R_{n-2}\|^2}\operatorname{tr}\left(R_k^T R_{n-1} - R_k^T R_{n-2}\right) = 0.
\end{aligned}$$

Hence, $\operatorname{tr}(R_{m-1}^T R_{n-1}) = 0$ and $\operatorname{tr}(U_m^T V_n) = 0$ 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 $X_0 \in \mathbb{R}^{n\times p}$, an exact solution X can be obtained in at most np iteration steps.*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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 $\{R_0, R_1, ..., R_{np}\}$ is orthogonal in $\mathbb{R}^{n\times p}$. So, $\{R_0, R_1, ..., R_{np}\}$ is linearly independent. Since the dimension of $\mathbb{R}^{n\times p}$ is np, any linearly independent subset of $\mathbb{R}^{n\times p}$ must have at most np elements. So this is false because the set $\{R_0, R_1, ..., R_{np}\}$ 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 $\|R_k\| \leqslant \epsilon$ 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 50×50 *tridiagonal matrices given by*

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

We run Algorithm 1 using an initial matrix $X_0 = 0.25 \times \text{ones} \in \mathbb{R}^{50\times 50}$. *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.*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

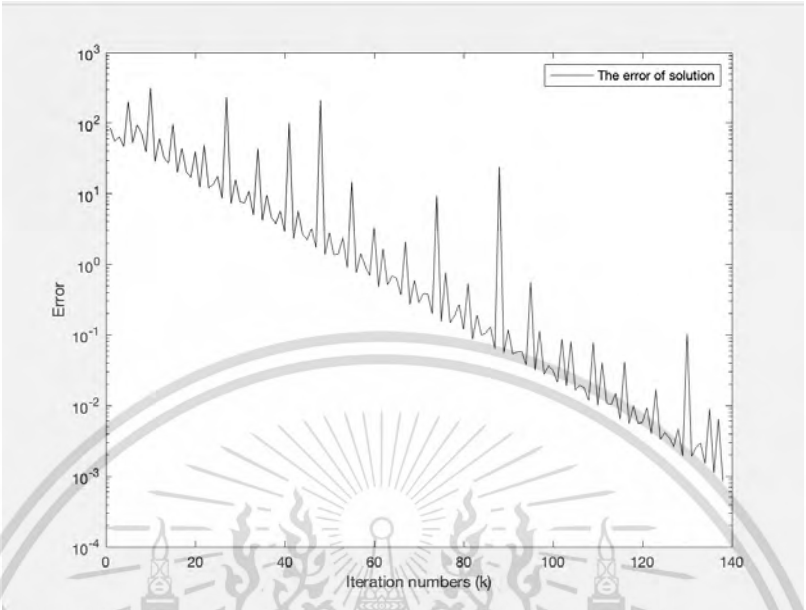

Figure 1. Relative error for Example 1.

Althouh 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* $||R_k|| \leqslant 10^{-3}$*), 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 soluton. Thus, the performace of Algorithm 1 is significantly better than the direct method. Moreover, for sparse coefficient matrices, Agorithm 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:

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

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* $||R_k|| \leqslant 10^{-3}$ *in* 164 *steps, using* 0.196250 *seconds. We see in Figure 2 that during iterations, althouh 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, Agorithm 1 is applicable and effective.*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

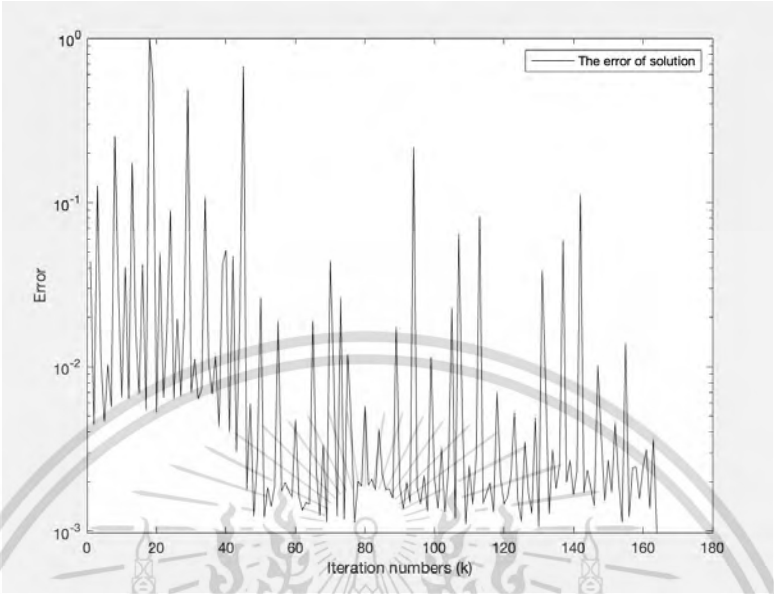

Figure 2. Relative error for Example 2.

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 100×100 *tridiagonal matrices given by*

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

The resulting simulation of Agorithm 1 using an initial matrix $X_0 = 0.5 \times \text{ones} \in \mathbb{R}^{100\times100}$ *is shown in the next figure.*

Figure 3 shows the error gradually decreasing into $\epsilon = 10^{-3}$ *in* 774 *steps, consuming around* 2 *seconds.*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

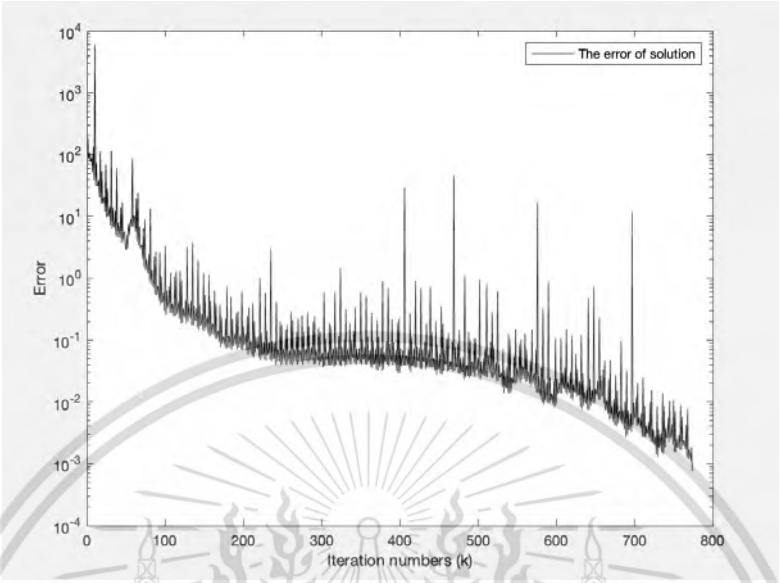

Figure 3. Relative error for Example 3.

Next, we investigate the effect of changing initial points. So we make experiments for the initial matrices $X_0 = 5 \times$ ones, $X_0 = 0$, *and* $X_0 = -5 \times$ 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 exacy solution. Thus, Agorithm 1 significantly outperforms the direct mathod.*

Table 1. Relative error and CTs for Example 3.

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

$$AXB + CX^T D = E,$$

where A, B, C, D, E *are* 100×100 *matrices as follows:*

$$A = \text{tridiag}(-1, 3, -1), \quad B = \text{tridiag}(1, 7, 1), \quad C = 6 \times \text{ones}, \quad D = -3 \times \text{ones}, \quad E = 0.7 \times I_{100}.$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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 $X_0 = -0.001 \times I_{100} \in \mathbb{R}^{100\times 100}$.

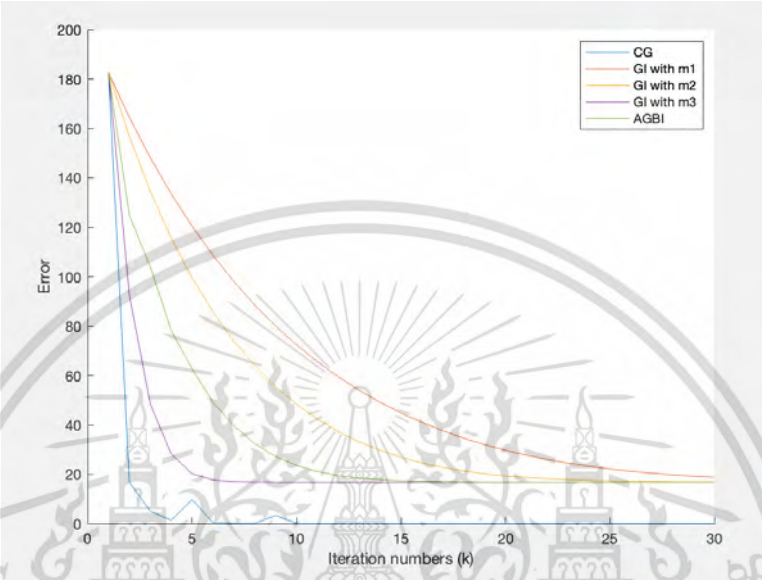

Figure 4. Relative error for Example 4.

According to [23], the GI algorithm is applicable as long as a convergent factor μ *satisfies*

$$0 < \mu < \frac{2}{\lambda_{max}(AA^T)\lambda_{max}(B^TB) + \lambda_{max}(CC^T)\lambda_{max}(D^TD)},$$

where $\lambda_{max}(AA^T)$ *is the largest eigenvalue of* AA^T. *We run GI algorithm under* 3 *different convergent factors, namely,* $m1 = 6.1728 \times 10^{-12}$, $m2 = 8.8183 \times 10^{-12}$ *and* $m3 = 3.0864 \times 10^{-11}$. *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* $m1$, *GI with* $m2$, *GI with* $m3$, *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 m1	30	0.109370	18.788266
GI with m2	30	0.115890	16.853503
GI with m3	30	0.109668	16.724640
AGBI	30	0.118294	16.724926
Direct	-	66.928143	0

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Example 5. *Consider a consistent generalized Sylvester-transpose matrix equation*

$$AXB + CX^T D = E,$$

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 $X_0 = -0.4 \times \text{ones} \in \mathbb{R}^{100\times100}$.

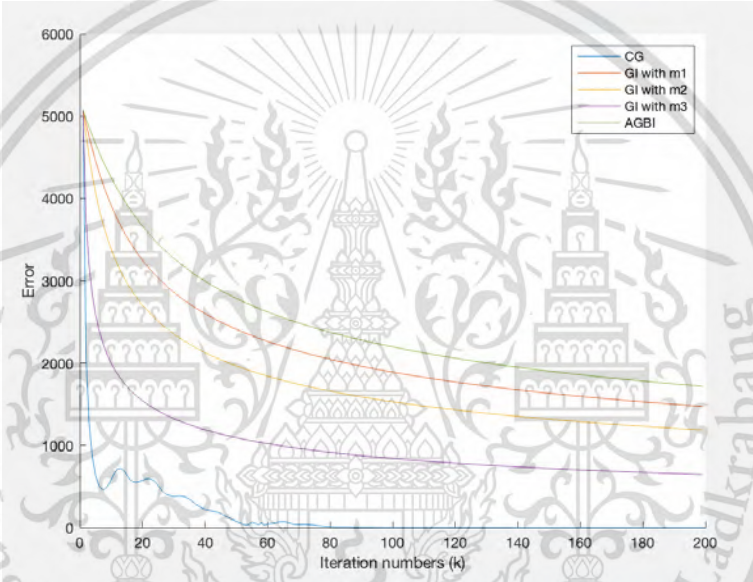

Figure 5. Relative error for Example 5.

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, $m1 = 1.7132 \times 10^{-8}$, $m2 = 3.0837 \times 10^{-8}$ *and* $m3 = 1.5418 \times 10^{-7}$. *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*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

is slightly less than those of GI (with parameters m1, m2, m3) and AGBI algorithms. However, the outcoming 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 100×100 *tridiagonal matrices given by*

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

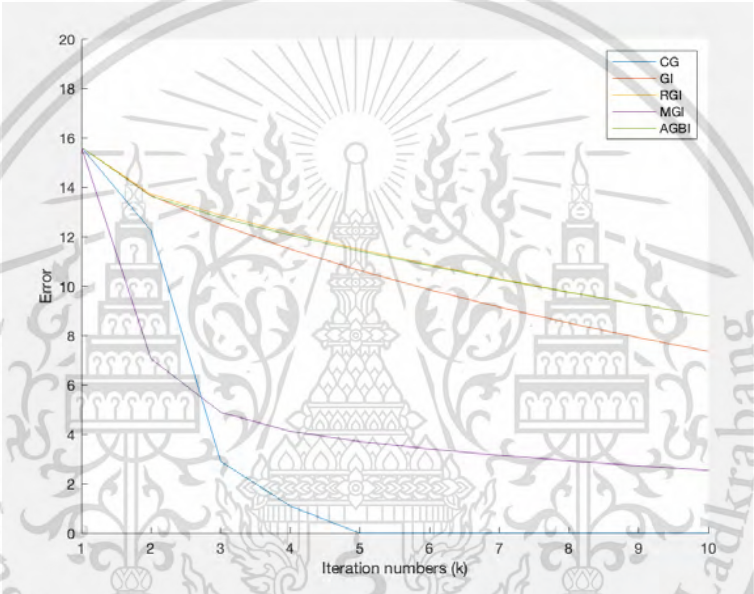

Figure 6. Relative error for Example 6.

Table 4. Relative error and CTs for Example 6.

Method	Parameters		Iterations	CT	Relative error
	Convergent factor	weighted factor			
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 $X_0 = -5 \times \text{ones} \in \mathbb{R}^{100\times100}$. *Table 4 shows that the computational times for*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

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. Comtr.*, **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

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

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

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

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

AIMS Press

© 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)

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Appendix B

The research Paper

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

symmetry

MDPI

Article

Conjugate Gradient Algorithm for Least-Squares Solutions of a Generalized Sylvester-Transpose Matrix Equation

Kanjanaporn Tansri [1,†] and Pattrawut Chansangiam [1,*,†]

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

* Correspondence: pattrawut.ch@kmitl.ac.th; Tel.: +66-9352-666-00

† These authors contributed equally to this work.

Abstract: We derive a conjugate-gradient type algorithm to produce approximate least-squares (LS) solutions for an inconsistent generalized Sylvester-transpose matrix equation. The algorithm is always applicable for any given initial matrix and will arrive at an LS solution within finite steps. When the matrix equation has many LS solutions, the algorithm can search for the one with minimal Frobenius-norm. Moreover, given a matrix Y, the algorithm can find a unique LS solution closest to Y. Numerical experiments show the relevance of the algorithm for square/non-square dense/sparse matrices of medium/large sizes. The algorithm works well in both the number of iterations and the computation time, compared to the direct Kronecker linearization and well-known iterative methods.

Keywords: generalized Sylvester-transpose matrix equation; conjugate gradient algorithm; least-squares solution; orthogonality; Kronecker product

MSC: 65F45; 65F10; 15A60; 15A69

check for updates

Citation: Tansri, K.; Chansangiam, P. Conjugate Gradient Algorithm for Least-Squares Solutions of a Generalized Sylvester-Transpose Matrix Equation. *Symmetry* **2022**, *14*, 1868. https://doi.org/10.3390/sym14091868

Academic Editor: Dongfang Li

Received: 2 August 2022
Accepted: 2 September 2022
Published: 7 September 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Sylvester-type matrix equations are closely related to ordinary differential equations (ODEs), which can be adapted to several problems in control engineering and information sciences; see e.g., monographs [1–3]. The Sylvester matrix equation $AX + XB = E$ and the famous special case Lyapunov equation $AX + XA^T = E$ have several applications in numerical methods for ODEs, control and system theory, signal processing, and image restoration; see e.g., [4–9]. The Sylvester-transpose equation $AX + X^TB = C$ is utilized in eigenstructure assignment in descriptor systems [10], pole assignment [3], and fault identification in dynamic systems [11]. In addition, if we require that the solution X to be symmetric, then the Sylvester-transpose equation coincides with the Sylvester one. A generalized Sylvester equation $AXB + CXD = E$ can be applied to implicit ODEs, and a general dynamical linear model for vibration and structural analysis, robot and spaceship controls; see e.g., [12,13].

The mentioned matrix equations are special cases of a generalized Sylvester-transpose matrix equation:

$$AXB + CX^TD = E, \tag{1}$$

or more generally

$$\sum_{i=1}^{s} A_iXB_i + \sum_{j=1}^{t} C_jX^TD_j = E. \tag{2}$$

A direct algebraic method to find a solution of Equation (2) is to use the Kronecker linearization transforming the matrix equation into an equivalent linear system; see e.g., [14]

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

(Ch. 4). The same technique together with the notion of weighted Moore–Penrose inverse were adapted to solve a coupled inconsistent Sylvester-type matrix equations [15] for least-squares (LS) solutions. Another algebraic method is to apply a generalized Sylvester mapping [13], so that the solution is expressed in terms of polynomial matrices. However, when the sizes of coefficient matrices are moderate or large, it is inconvenient to use matrix factorizations or another traditional methods since they require a large memory to calculate an exact solution. Thus, the Kronecker linearization and other algebraic methods are only suitable for small matrices. That is why it is important to find solutions that are easy to compute, leading many researchers to come up with algorithms that can reduce the time and memory usage of solving large matrix equations.

In the literature, there are two notable techniques to derive iterative procedures for solving linear matrix equations; see more information in a monograph [16]. The conjugate gradient (CG) technique aims to create an approximate sequence of solutions so that the respective residual matrix creates a perpendicular base. The desired solution will come out in the final step of iterations. In the last decade, many authors developed CG-type algorithms for Equation (2) and its special cases, e.g., BiCG [17], BiCR [17], CGS [18], GCD [19], GPBiCG [20], and CMRES [21]. The second technique, known as gradient-based iterative (GI) technique, intends to construct a sequence of approximate solutions from the gradient of the associated norm-error function. If we carefully set parameters of GI algorithm, then the generated sequence would converge to the desired solution. In the last five years, many GI algorithms have been introduced; see e.g., GI [22,23], relaxed GI [24], accelerated GI [25], accelerated Jacobi GI [26], modified Jacobi GI [27], gradient-descent algorithm [28], and global generalized Hessenberg algorithm [29]. For LS solutions of Sylvester-type matrix equations, there are iterative solvers, e.g., [30,31].

Recently, the work [32] developed an effective gradient-descent iterative algorithm to produce approximated LS solutions of Equation (2). When Equation (2) is consistent, a CG-type algorithm was derived to obtain a solution within finite steps; see [33]. This work is a continuation of [33], i.e., we consider Equation (1) with rectangular coefficient matrices and a rectangular unknown matrix X. Suppose that Equation (1) is inconsistent. We propose a CG-type algorithm to approximate LS solutions, which will solve the following problems:

Problem 1. *Find a matrix* $\hat{X} \in \mathbb{R}^{n\times p}$ *that minimizes* $\|E - AXB - CX^TD\|_F$.

Let $\mathcal{L}$ be the set of least-squares solutions of Equation (1). The second problem is to find an LS solution with the minimal norm:

Problem 2. *Find the matrix* X^* *such that*

$$\|X^*\|_F = \min_{\hat{X}\in\mathcal{L}} \|\hat{X}\|_F. \tag{3}$$

The last one is to find an LS solution closest to a given matrix:

Problem 3. *Let* $Y \in \mathbb{R}^{n\times p}$. *Find the matrix* $\check{X}$ *such that*

$$\|\check{X} - Y\|_F = \min_{\hat{X}\in\mathcal{L}} \|\hat{X} - Y\|_F. \tag{4}$$

Moreover, we extend our studies to the matrix Equation (2). We verify the results from theoretical and numerical points of view.

The organization of this article is as follows. In Section 2, we recall preliminary results from matrix theory that will be used in later discussions. In Section 3, we explain how the Kronecker linearization can transform Equation (1) into an equivalent linear system to obtain LS solutions. In Section 4, we propose a CG-type algorithm to solve Problem 1 and verify the theoretical capability of the algorithm. After that, Problems 2 and 3 are investigated in Sections 5 and 6, respectively. To verify the theory, we provide numerical

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

experiments in Section 7 to show the applicability and efficiency of the algorithm, compared to the Kronecker linearization and recent iterative algorithms. We summarize the whole work in the last section.

2. Auxiliary Results from Matrix Theory

Throughout, let us denote by $\mathbb{R}^{m\times n}$ the set of all m-by-n real matrices. Recall that the standard (Frobenius) inner product of $A, B \in \mathbb{R}^{m\times n}$ is defined by

$$\langle A, B\rangle \;:=\; \mathrm{tr}(B^T A) \;=\; \mathrm{tr}(AB^T). \tag{5}$$

If $\langle A, B\rangle = 0$, we say that A is orthogonal to B. A well-known property of the inner product is that

$$\langle A, BCD\rangle \;=\; \langle B^T AD^T, C\rangle, \tag{6}$$

for any matrices A, B, C, D with appropriate dimensions. The Frobenius norm of matrix $A \in \mathbb{R}^{m\times n}$ is defined by

$$\|A\|_F \;:=\; \sqrt{\mathrm{tr}(A^T A)}.$$

The Kronecker product $A \otimes B$ of $A = [a_{ij}] \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ is defined to be the mp-by-nq matrix whose each (i, j)-th block is given by $a_{ij}B$.

Lemma 1 ([14]). *For any real matrices A and B, we have $(A \otimes B)^T \;=\; A^T \otimes B^T$.*

The vector operator $\mathrm{Vec}(\cdot)$ transforms a matrix $A = [a_{ij}] \in \mathbb{R}^{m\times n}$ to the vector

$$\mathrm{Vec}\, A \;:=\; [a_{11} \cdots a_{m1}\, a_{12} \cdots a_{m2} \cdots a_{1n} \cdots a_{mn}]^T \;\in\; \mathbb{R}^{mn}.$$

The vector operator is bijective, linear, and related to the usual matrix multiplication as follows.

Lemma 2 ([14]). *For any $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$ and $C \in \mathbb{R}^{p\times q}$, we have*

$$\mathrm{Vec}\, ABC \;=\; (C^T \otimes A)\,\mathrm{Vec}\, B.$$

For each $m, n \in \mathbb{N}$, we define a commutation matrix

$$P(m, n) \;:=\; \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij} \otimes E_{ij}^T \;\in\; \mathbb{R}^{mn\times mn}, \tag{7}$$

where the (i, j)-th position of $E_{ij} \in \mathbb{R}^{m\times n}$ is 1 and all other entries are 0. Indeed, $P(m, n)$ acts on a vector by permuting its entries as follows.

Lemma 3 ([14]). *For any matrix $A \in \mathbb{R}^{m\times n}$, we have*

$$\mathrm{Vec}(A^T) \;=\; P(m, n)\,\mathrm{Vec}(A). \tag{8}$$

Moreover, commutation matrices permute the entries of $A \otimes B$ as follows.

Lemma 4 ([14]). *For any $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$, we have*

$$B \otimes A \;=\; P(m, p)^T (A \otimes B) P(n, q). \tag{9}$$

The next result will be used in the later discussions.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Lemma 5 ([14]). *For any matrices A, B, C, D with appropriate dimensions, we get*

$$\operatorname{tr}(A^T D^T B C) = (\operatorname{Vec} D)^T (A \otimes B) \operatorname{Vec} C. \tag{10}$$

3. Least-Squares Solutions via the Kronecker Linearization

From now on, we investigate the generalized Sylvester-transpose matrix Equation (1), with corresponding coefficient matrices $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, $C \in \mathbb{R}^{m\times p}$, $D \in \mathbb{R}^{n\times q}$, $E \in \mathbb{R}^{m\times q}$, and a rectangular unknown matrix $X \in \mathbb{R}^{n\times p}$. We focus our attention when Equation (1) is inconsistent. In this case, we will seek for its LS solution, that is, a matrix X^* that solves the following minization problem:

$$\min_{X\in\mathbb{R}^{n\times p}} \|E - AXB - CX^T D\|_F. \tag{11}$$

A traditional algebraic way to solve a linear matrix equation is known as the Kronecker linearization–to transform the matrix equation into an equivalent linear system using the notions of vector operator and Kronecker products. Indeed, taking the vector operator to Equation (1) and applying Lemmas 2 and 3 yield

$$\begin{aligned} \operatorname{Vec} E &= \operatorname{Vec}(AXB + CX^T D) = (B^T \otimes A) \operatorname{Vec} X + (D^T \otimes C) \operatorname{Vec} X^T \\ &= (B^T \otimes A) \operatorname{Vec} X + (D^T \otimes C) P(n,p) \operatorname{Vec} X. \end{aligned} \tag{12}$$

Let us denote $x = \operatorname{Vec} X$, $e = \operatorname{Vec} E$, and

$$M = (B^T \otimes A) + (D^T \otimes C) P(n,p). \tag{13}$$

Thus, a matrix X is an LS solution of Equation (1) if and only if x is an LS solution of the linear system $Mx = e$, or equivalently, a solution of the associated normal equation

$$M^T M x = M^T e. \tag{14}$$

The linear system (14) is always consistent, i.e., Equation (1) always has an LS solution. From the normal Equation (14) and Lemmas 2 and 3, we can deduce:

Lemma 6 ([34]). *Problem 1 is equivalent to the following consistent matrix equation*

$$A^T(AXB + CX^T D)B^T + D(B^T X^T A^T + D^T X C^T)C = A^T E B^T + D E^T C. \tag{15}$$

Moreover, the normal Equation (14) has a unique solution if and only if the matrix M is of full-column rank, i.e., $M^T M$ is invertible. In this case, the unique solution is given by $x^* = (M^T M)^{-1} M^T e$, and the LS error can be computed as follows:

$$\|Mx^* - e\|^2 = \|e\|^2 - e^T M x^*. \tag{16}$$

If M is of not full-column rank (i.e., the kernel of M is nontrivial), then the system $Mx = e$ has many solutions. In this case, the LS solutions appear in the form $x^* = M^\dagger e + u$ where $M^\dagger$ is the Moore–Penrose inverse of M, and u is an arbitrary vector in the kernel of M. Among all these solutions,

$$x^* = M^\dagger e \tag{17}$$

is the unique one having minimal norm.

4. Least-Squares Solution via a Conjugate Gradient Algorithm

In this section, we propose a CG-type algorithm to solve Problem 1. We do not impose any assumption on the matrix M, so that LS solutions of Equation (1) may not be unique.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

We shall adapt the conjugate-gradient technique to solve the equivalent matrix Equation (15). Recall that the set of LS solutions of Equation (1) is denoted by $\mathcal{L}$. From Lemma 6, observe that the residual of a matrix $X \in \mathbb{R}^{n\times p}$ according to Equation (1) is given by

$$R_X := A^TEB^T + DE^TC - A^T(AXB + CX^TD)B^T - D(B^TX^TA^T + D^TXC^T)C. \quad (18)$$

Lemma 6 states that $X \in \mathcal{L}$ if and only if $R_X = 0$. From this, we propose the following algorithm. Indeed, the next approximate solution X_{r+1} is equal to the current approximation X_r along with a search direction U_{r+1} of suitable step size.

Algorithm 1: A conjugate gradient iterative algorithm for Equation (1)

$A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, $C \in \mathbb{R}^{m\times p}$, $D \in \mathbb{R}^{n\times q}$ and $E \in \mathbb{R}^{m\times q}$;
Set $r = 0$, $U_0 = 0$. Choose $X_0 \in \mathbb{R}^{n\times p}$;
$R_0 = A^TEB^T + DE^TC - A^T(AX_0B + CX_0^TD)B^T - D(B^TX_0^TA^T + D^TX_0C^T)C$;
for $r = 0$ **to** np **do**
  if $\|R_r\|_F \leqslant \epsilon$ **then**
    X_r is an LS solution; break;
  else
    if $r = 0$ **then**
      set $U_{r+1} = R_r$;
    else
      set $U_{r+1} = R_r + \frac{\|R_r\|_F^2}{\|R_{r-1}\|_F^2}U_r$;
    end
  end
  $H_{r+1} = A^T(AU_{r+1}B + CU_{r+1}^TD)B^T + D(B^TU_{r+1}^TA^T + D^TU_{r+1}C^T)C$;
  $\alpha_{r+1} = \mathrm{tr}(U_{r+1}^TH_{r+1})$;
  $X_{r+1} = X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1}$;
  $R_{r+1} = A^TEB^T + DE^TC - A^T(AX_{r+1}B + CX_{r+1}^TD)B^T$
    $-D(B^TX_{r+1}^TA^T + D^TX_{r+1}C^T)C$;
  update r;
  end
end

Remark 1. *To terminate the algorithm, one can alternatively set the stopping rule to be* $\|R_r\|_F - \delta \leqslant \epsilon'$ *where* $\delta := \|Mx^* - e\|$ *is the positive square root of the LS error described in Equation* (16) *and* $\epsilon' > 0$ *is a small tolerance.*

For any given initial matrix X_0, we will show that Algorithm 1 generates a sequence of approximate solutions X_r of Equation (1), so that the set of residual matrices R_r is orthogonal. It follows that a unique LS solution will be obtained within finite steps.

Lemma 7. *Assume that the sequences* $\{R_r\}$ *and* $\{H_r\}$ *are generated by Algorithm 1. We get*

$$R_{r+1} = R_r - \frac{\|R_r\|_F^2}{\alpha_{r+1}}H_{r+1}, \quad \text{for} \quad r = 1, 2, \ldots. \quad (19)$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Proof. From Algorithm 1, we have that for any r,

$$\begin{aligned}
R_{r+1} &= A^TEB^T + DE^TC - A^T(AX_{r+1}B + CX_{r+1}^TD)B^T - D(B^TX_{r+1}^TA^T + D^TX_{r+1}C^T)C \\
&= A^TEB^T + DE^TC - A^T\left(A(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})B + C(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})^TD\right)B^T \\
&\quad - D\left(B^T(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})^TA^T + D^T(X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}}U_{r+1})C^T\right)C \\
&= A^TEB^T + DE^TC - A^T\left(AX_rB + \frac{\|R_r\|_F^2}{\alpha_{r+1}}AU_{r+1}B + CX_r^TD + \frac{\|R_r\|_F^2}{\alpha_{r+1}}CU_{r+1}^TD\right)B^T \\
&\quad - D\left(B^TX_r^TA^T + \frac{\|R_r\|_F^2}{\alpha_{r+1}}B^TU_{r+1}^TA^T + D^TX_rC^T + \frac{\|R_r\|_F^2}{\alpha_{r+1}}D^TU_{r+1}C^T\right)C \\
&= A^TEB^T + DE^TC - A^T(AX_rB + CX_r^TD)B^T - D(B^TX_r^TA^T + D^TX_rC^T)C \\
&\quad - \frac{\|R_r\|_F^2}{\alpha_{r+1}}[A^T(AU_{r+1}B + CU_{r+1}^TD)B^T + D(B^TU_{r+1}^TA^T + D^TU_{r+1}C^T)C] \\
&= R_r - \frac{\|R_r\|_F^2}{\alpha_{r+1}}H_{r+1}. \quad \square
\end{aligned}$$

Lemma 8. *The sequences* $\{U_r\}$ *and* $\{H_r\}$ *generated by Algorithm 1 satisfy*

$$\operatorname{tr}(U_m^TH_n) = \operatorname{tr}(H_m^TU_n), \qquad \text{for any} \quad m, n. \tag{20}$$

Proof. Using the properties of the Kronecker product and the vector operator in Lemmas 1–5, we have

$$\begin{aligned}
\operatorname{tr}(H_m^TU_n) &= (\operatorname{Vec}H_m)^T\operatorname{Vec}U_n \\
&= [\operatorname{Vec}(A^T(AU_mB + C_jU_m^TD)B^T + D(B^TU_m^TA^T + D^TU_mC^T)C)]^T\operatorname{Vec}U_n \\
&= [\operatorname{Vec}(A^TAU_mBB^T)]^T\operatorname{Vec}U_n + [\operatorname{Vec}(A^TCU_m^TDB^T)]^T\operatorname{Vec}U_n \\
&\quad + [\operatorname{Vec}(DB^TU_m^TA^TC)]^T\operatorname{Vec}U_n + [\operatorname{Vec}(DD^TU_mC^TC)]^T\operatorname{Vec}U_n \\
&= (\operatorname{Vec}U_m)^T(BB^T \otimes A^TA)\operatorname{Vec}U_n + (\operatorname{Vec}U_m^T)^T(DB_k^T \otimes C^TA)\operatorname{Vec}U_n \\
&\quad + (\operatorname{Vec}U_m^T)^T(A^TC \otimes BD^T)\operatorname{Vec}U_n + (\operatorname{Vec}U_m)^T(C^TC \otimes DD^T)\operatorname{Vec}U_n \\
&= \operatorname{tr}(BB^TU_m^TA^TAU_n) + \operatorname{tr}(A^TCU_m^TDB^TU_n^T) + \operatorname{tr}(DB^TU_m^TA^TCU_n^T) \\
&\quad + \operatorname{tr}(C^TCU_m^TDD^TU_n) \\
&= (\operatorname{Vec}(BB_k^TU_n^TA^TA))^T\operatorname{Vec}U_m^T + (\operatorname{Vec}(C^TAU_nBD^T))^T\operatorname{Vec}U_n^T \\
&\quad + (\operatorname{Vec}(BD^TU_nC^TA))^T\operatorname{Vec}U_m^T + (\operatorname{Vec}(C^TCU_n^TDD^T))^T\operatorname{Vec}U_m^T \\
&= [\operatorname{Vec}(A^TAU_nBB^T)]^T\operatorname{Vec}U_m + [\operatorname{Vec}(DB^TU_n^TA^TC)]^T\operatorname{Vec}U_m \\
&\quad + [\operatorname{Vec}(A^TCU_n^TDB^T)]^T\operatorname{Vec}U_m + [\operatorname{Vec}(DD^TU_nC^TC)]^T\operatorname{Vec}U_m \\
&= [\operatorname{Vec}(A^T(AU_nB + CU_n^TD)B^T + D(B^TU_n^TA^T + D^TU_nC^T)C)]^T\operatorname{Vec}U_m \\
&= (\operatorname{Vec}H_n)^T\operatorname{Vec}U_m \\
&= \operatorname{tr}(H_n^TU_m) \\
&= \operatorname{tr}(U_m^TH_n). \quad \square
\end{aligned}$$

Lemma 9. *The sequences* $\{R_r\}$, $\{U_r\}$ *and* $\{H_r\}$ *generated by Algorithm 1 satisfy*

$$\operatorname{tr}(R_r^TR_{r-1}) = 0, \quad \text{and} \quad \operatorname{tr}(U_{r+1}^TH_r) = 0, \qquad \text{for any} \quad r. \tag{21}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Proof. We use the induction principle to prove (21). In order to calculate related terms, we utilize Lemmas 7 and 8. For $r = 1$, we get

$$\begin{aligned}\operatorname{tr}(R_1^T R_0) &= \operatorname{tr}(R_0^T R_0) - \frac{\|R_0\|_F^2}{\alpha_1}\operatorname{tr}(H_1^T R_0) = 0,\\ \operatorname{tr}(U_2^T H_1) &= \operatorname{tr}(R_1^T H_1) + \frac{\|R_1\|_F^2}{\|R_0\|_F^2}\operatorname{tr}(U_1^T H_1)\\ &= -\frac{\alpha_1}{\|R_0\|_F^2}\operatorname{tr}(R_1^T R_1) + \alpha_1\frac{\|R_1\|_F^2}{\|R_0\|_F^2} = 0.\end{aligned}$$

These imply that (21) holds for $r = 1$. Now, we proceed the inductive step by assuming that $\operatorname{tr}(R_r^T R_{r-1}) = 0$ and $\operatorname{tr}(U_{r+1}^T H_r) = 0$. Then

$$\begin{aligned}\operatorname{tr}(R_{r+1}^T R_r) &= \operatorname{tr}(R_r^T R_r) - \frac{\|R_r\|_F^2}{\alpha_{r+1}}\operatorname{tr}\left(H_{r+1}^T\left(U_{r+1} - \frac{\|R_r\|_F^2}{\|R_{r-1}\|_F^2}U_r\right)\right)\\ &= \|R_r\|_F^2 - \frac{\|R_r\|_F^2}{\alpha_{r+1}}\operatorname{tr}(H_{r+1}^T U_{r+1}) = 0,\\ \operatorname{tr}(U_{r+2}^T H_{r+1}) &= \operatorname{tr}\left(R_{r+1}^T\left(\frac{-\alpha_{r+1}}{\|R_r\|_F^2}(R_{r+1} - R_r)\right)\right) + \frac{\|R_{r+1}\|_F^2}{\|R_r\|_F^2}\alpha_{r+1}\\ &= \frac{\alpha_{r+1}}{\|R_r\|_F^2}\operatorname{tr}[(R_{r+1}^T R_r) - (R_{r+1}^T R_{r+1})] + \frac{\|R_{r+1}\|_F^2}{\|R_r\|_F^2}\alpha_{r+1} = 0.\end{aligned}$$

Hence, Equationd (21) holds for any r. □

Lemma 10. *Suppose the sequences $\{R_r\}$, $\{U_r\}$ and $\{H_r\}$ are constructed from Algorithm 1. Then*

$$\operatorname{tr}(R_r^T R_0) = 0, \quad \operatorname{tr}(U_{r+1}^T H_1) = 0, \quad \text{for any} \quad r. \tag{22}$$

Proof. The initial step $r = 1$ holds due to Lemma 9. Now, assume that Equation (22) is valid for all $r = 1, \ldots, k$. From Lemmas 7 and 8, we get

$$\begin{aligned}\operatorname{tr}(R_{k+1}^T R_0) &= \operatorname{tr}\left(\left(R_k - \frac{\|R_k\|_F^2}{\alpha_{k+1}}V_{k+1}\right)^T R_0\right) = \operatorname{tr}(R_k^T R_0) - \frac{\|R_k\|_F^2}{\alpha_{k+1}}\operatorname{tr}(H_{k+1}^T R_0)\\ &= -\frac{\|R_k\|_F^2}{\alpha_{k+1}}\operatorname{tr}(H_{k+1}^T U_1) = -\frac{\|R_k\|_F^2}{\alpha_{k+1}}\operatorname{tr}(U_{k+1}^T H_1) = 0,\end{aligned}$$

and

$$\begin{aligned}\operatorname{tr}(U_{k+2}^T H_1) &= \operatorname{tr}(H_{k+2}^T U_1) = \operatorname{tr}\left(\left(\frac{-\alpha_{k+2}}{\|R_{k+1}\|_F^2}(R_{k+2} - R_{k+1})\right)^T U_1\right)\\ &= \frac{-\alpha_{k+2}}{\|R_{k+1}\|_F^2}[\operatorname{tr}(R_{k+2}^T R_0) - \operatorname{tr}(R_{k+1}^T R_0)] = 0.\end{aligned}$$

Hence, Equation (22) holds for any r. □

Lemma 11. *Suppose the sequences $\{R_r\}$, $\{U_r\}$ and $\{H_r\}$ are constructed from Algorithm 1. Then for any integers m and n such that $m \neq n$, we have*

$$\operatorname{tr}(R_{m-1}^T R_{n-1}) = 0, \quad \text{and} \quad \operatorname{tr}(U_m^T H_n) = 0. \tag{23}$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Proof. According to Lemma 8 and the fact that $\mathrm{tr}(R_{m-1}^T R_{n-1}) = \mathrm{tr}(R_{n-1}^T R_{m-1})$ for any integers m and n, it remains to prove two equalities in (23) for only m, n such that $m > n$. For $m = n+1$, the two equalities hold by Lemma 9. For $m = n+2$, we have

$$\begin{aligned}
\mathrm{tr}(R_{n+2}^T R_n) &= \mathrm{tr}\left(\left(R_{n+1} - \frac{\|R_{n+1}\|_F^2}{\alpha_{n+2}} H_{n+2}\right)^T R_n\right) \\
&= -\frac{\|R_{n+1}\|_F^2}{\alpha_{n+2}} \mathrm{tr}\left(H_{n+2}^T\left(U_{n+1} - \frac{\|R_n\|_F^2}{\|R_{n-1}\|_F^2} U_n\right)\right) \\
&= \frac{\|R_{n+1}\|_F^2}{\alpha_{n+2}} \frac{\|R_n\|_F^2}{\|R_{n-1}\|_F^2} \mathrm{tr}\left(\left(R_{n+1} + \frac{\|R_{n+1}\|_F^2}{\|R_n\|_F^2} U_{n+1}\right)^T H_n\right) \\
&= \frac{\|R_{n+1}\|_F^2}{\alpha_{n+2}} \frac{\|R_n\|_F^2}{\|R_{n-1}\|_F^2} \frac{\alpha_n}{\|R_{n-1}\|_F^2} \mathrm{tr}(R_{n+1}^T R_{n-1}),
\end{aligned}$$

and, similarly, we have

$$\begin{aligned}
\mathrm{tr}(R_{n+1}^T R_{n-1}) &= \frac{\|R_n\|_F^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|_F^2}{\|R_{n-2}\|_F^2} \frac{\alpha_{n-1}}{\|R_{n-2}\|_F^2} \mathrm{tr}(R_n^T R_{n-2}), \\
\mathrm{tr}(R_n^T R_{n-2}) &= \frac{\|R_{n-1}\|_F^2}{\alpha_n} \frac{\|R_{n-2}\|_F^2}{\|R_{n-3}\|_F^2} \frac{\alpha_{n-2}}{\|R_{n-3}\|_F^2} \mathrm{tr}(R_{n-1}^T R_{n-3}).
\end{aligned}$$

Moreover,

$$\begin{aligned}
\mathrm{tr}(U_{n+2}^T H_n) &= \mathrm{tr}\left(\left(R_{n+1} - \frac{\|R_{n+1}\|_F^2}{\|R_n\|_F^2} U_{n+1}\right)^T H_n\right) \\
&= \mathrm{tr}\left(R_{n+1}^T\left(\frac{-\alpha_n}{\|R_{n-1}\|_F^2}(R_n - R_{n-1})\right)\right) \\
&= \frac{\alpha_n}{\|R_{n-1}\|_F^2} \mathrm{tr}\left(\left(R_n - \frac{\|R_n\|_F^2}{\alpha_{n+1}} H_{n+1}\right)^T R_{n-1}\right) \\
&= -\frac{\alpha_n}{\|R_{n-1}\|_F^2} \frac{\|R_n\|_F^2}{\alpha_{n+1}}\left[\mathrm{tr}(H_{n+1}^T U_n) - \frac{\|R_{n-1}\|_F^2}{\|R_{n-2}\|_F^2} \mathrm{tr}(H_{n+1}^T U_{n-1})\right] \\
&= \frac{\alpha_n}{\|R_{n-1}\|_F^2} \frac{\|R_n\|_F^2}{\alpha_{n+1}} \frac{\|R_{n-1}\|_F^2}{\|R_{n-2}\|_F^2} \mathrm{tr}(U_{n+1}^T H_{n-1}),
\end{aligned}$$

and, similarly,

$$\mathrm{tr}(U_{n+1}^T H_{n-1}) = \frac{\alpha_{n-1}}{\|R_{n-2}\|_F^2} \frac{\|R_{n-1}\|_F^2}{\alpha_n} \frac{\|R_{n-2}\|_F^2}{\|R_{n-3}\|_F^2} \mathrm{tr}(U_n^T H_{n-2}).$$

In a similar way, we can write $\mathrm{tr}(R_{n+1}^T R_{n-1})$ and $\mathrm{tr}(U_{n+2}^T H_n)$ in terms of $\mathrm{tr}(R_n^T R_{n-2})$ and $\mathrm{tr}(U_{n+1}^T H_{n-1})$, respectively. Continuing this process until the terms $\mathrm{tr}(R_2^T R_0)$ and $\mathrm{tr}(U_3^T H_1)$ appear. By Lemma 10, we get $\mathrm{tr}(R_{n+1}^T R_{n-1}) = 0$ and $\mathrm{tr}(U_{n+2}^T H_n) = 0$. Similarly, we have $\mathrm{tr}(R_m^T R_{n-1}) = 0$ and $\mathrm{tr}(U_m^T H_n) = 0$ for $m = n+3, \ldots, k$.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Suppose that $\mathrm{tr}(R_{m-1}^T R_{n-1}) = \mathrm{tr}(U_m^T H_n) = 0$ for $m \in \{n+1, \ldots, k\}$. Then for $m = k+1$, we have

$$\begin{aligned}\mathrm{tr}(R_k^T R_{n-1}) &= \mathrm{tr}(R_{k-1}^T R_{n-1}) - \frac{\|R_{k-1}\|_F^2}{\alpha_k}\mathrm{tr}(H_k^T R_{n-1}) \\ &= -\frac{\|R_{k-1}\|_F^2}{\alpha_k}\mathrm{tr}\left(H_k^T\left(U_n - \frac{\|R_{n-1}\|_F^2}{\|R_{n-2}\|_F^2}U_{n-1}\right)\right) \\ &= -\frac{\|R_{k-1}\|_F^2}{\alpha_k}\left[\mathrm{tr}(H_k^T U_n) - \frac{\|R_{n-1}\|_F^2}{\|R_{n-2}\|_F^2}\mathrm{tr}(H_k^T U_{n-1})\right] = 0.\end{aligned}$$

and

$$\begin{aligned}\mathrm{tr}(U_{k+1}^T H_n) &= \mathrm{tr}(R_k^T H_n) + \frac{\|R_k\|_F^2}{\|R_{k-1}\|_F^2}\mathrm{tr}(U_k^T H_n) = \mathrm{tr}\left(R_k^T\left(\frac{-\alpha_n}{\|R_{n-1}\|_F^2}(R_n - R_{n-1})\right)\right) \\ &= \frac{-\alpha_n}{\|R_{n-1}\|_F^2}\mathrm{tr}(R_k^T R_n - R_k^T R_{n-1}) = 0.\end{aligned}$$

Hence, $\mathrm{tr}(R_{m-1}^T R_{n-1}) = 0$ and $\mathrm{tr}(U_m^T H_n) = 0$ for any m, n such that $m \neq n$. □

Theorem 1. *Algorithm 1 solves Problem 1 within finite steps. More precisely, for any given initial matrix $X_0 \in \mathbb{R}^{n\times p}$, the sequence $\{X_r\}$ constructed from Algorithm 1 converges to an LS solution of Equation (1) in at most np iterations.*

Proof. Assume that $R_r \neq 0$ for $r = 0, 1, \ldots, np-1$. Assume that $R_{np} \neq 0$. By Lemma 11, the set $\{R_0, R_1, ..., R_{np}\}$ of residual matrices is orthogonal in $\mathbb{R}^{n\times p}$ with respect to the Frobenius inner product (5). Therefore, the set $\{R_0, R_1, ..., R_{np}\}$ of $np+1$ elements is linearly independent. This contradicts the fact that the dimension of $\mathbb{R}^{n\times p}$ is np. Thus, $R_{np} = 0$, and X_{np} satisfies Equation (15) in Lemma 6. Hence X_{np} is an LS solution of Equation (1). □

We adapt the same idea as that for Algorithm 1 to derive an algorithm for Equation (2) as follows:

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Algorithm 2: A conjugate gradient iterative algorithm for Equation (2)

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

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

$$R_0 = \sum_{k=1}^{s} A_k^T E B_k^T + \sum_{l=1}^{t} D_l E^T C_l - \sum_{k=1}^{s} A_k^T \Big(\sum_{i=1}^{s} A_i X_0 B_i + \sum_{j=1}^{t} C_j X_0^T D_j \Big) B_k^T - \sum_{l=1}^{t} D_l \Big(\sum_{i=1}^{s} B_i^T X_0^T A_i^T + \sum_{j=1}^{t} D_j^T X_0 C_j^T \Big) C_l;$$

for $r = 0$ **to** np **do**

  if $\|R_r\|_F \leqslant \epsilon$ **then**

    X_r is an LS solution; break;

  else

    if $r = 0$ **then**

      set $U_{r+1} = R_r$;

    else

      set $U_{r+1} = R_r + \frac{\|R_r\|_F^2}{\|R_{r-1}\|_F^2} U_r$;

    end

  end

$$H_{r+1} = \sum_{k=1}^{s} A_k^T \Big(\sum_{i=1}^{s} A_i U_{r+1} B_i + \sum_{j=1}^{t} C_j U_{r+1}^T D_j \Big) B_k^T + \sum_{l=1}^{t} D_l \Big(\sum_{i=1}^{s} B_i^T U_{r+1}^T A_i^T + \sum_{j=1}^{t} D_j^T U_{r+1} C_j^T \Big) C_l;$$

$$\alpha_{r+1} = \mathrm{tr}\left(U_{r+1}^T H_{r+1} \right);$$

$$X_{r+1} = X_r + \frac{\|R_r\|_F^2}{\alpha_{r+1}} U_{r+1};$$

$$R_{r+1} = \sum_{k=1}^{s} A_k^T E B_k^T + \sum_{l=1}^{t} D_l E^T C_l - \sum_{k=1}^{s} A_k^T \Big(\sum_{i=1}^{s} A_i X_{r+1} B_i + \sum_{j=1}^{t} C_j X_{r+1}^T D_j \Big) B_k^T - \sum_{l=1}^{t} D_l \Big(\sum_{i=1}^{s} B_i^T X_{r+1}^T A_i^T + \sum_{j=1}^{t} D_j^T X_{r+1} C_j^T \Big) C_l;$$

  update r;

end

The stopping rule of Algorithm 2 may be described as $\|R_r\|_F - \delta \leqslant \epsilon'$ where δ is the positive square root of the associated LS error and $\epsilon' > 0$ is a small tolerance.

Theorem 2. *Consider Equation (2) where $A_i \in \mathbb{R}^{m\times n}$, $B_i \in \mathbb{R}^{p\times q}$, $C_j \in \mathbb{R}^{m\times p}$, $D_j \in \mathbb{R}^{n\times q}$, $D \in \mathbb{R}^{m\times q}$, $E \in \mathbb{R}^{m\times q}$ are given constant matrices and $X \in \mathbb{R}^{n\times p}$ is an unknown matrix. Assume that the matrix*

$$M := \sum_{i=1}^{s} (B_i^T \otimes A_i) + \sum_{j=1}^{t} (D_j^T \otimes C_j) P(n, p) \tag{24}$$

is of full-column rank. Then, for any given initial matrix $X_0 \in \mathbb{R}^{n\times p}$, the sequence $\{X_r\}$ constructed from Algorithm 2 converges to a unique LS solution.

Proof. The proof of is similar to that of Theorem 1. □

This material is reserved for educational use only, not allowed for commercial use.

Forbidden to modify the content, and cite the document when use.

5. Minimal-Norm Least-Squares Solution via Algorithm 1

In this section, we investigate Problem 2. That is, we consider the case when the matrix M may not have full-column rank, so that Equation (1) may have many LS solutions. We shall seek for an element of $\mathcal{L}$ with the minimal Frobenius norm.

Lemma 12. *Assume $\hat{X} \in \mathcal{L}$. Then, any arbitrary element $\tilde{X} \in \mathcal{L}$ can be expressed as $\hat{X} + Z$ for some matrix $Z \in \mathbb{R}^{n\times p}$ satisfying*

$$A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C \;=\; 0. \tag{25}$$

Proof. Let us denote the residual of the LS solutions $\hat{X}$ and $\tilde{X}$, according to Equation (18), by $R_{\hat{X}}$ and $R_{\tilde{X}}$, respectively. We consider the different $Z := \tilde{X} - \hat{X}$. Now, we compute

$$\begin{aligned} R_{\tilde{X}} \;&=\; A^T(A(\hat{X}+Z)B + C(\hat{X}+Z)^TD)B^T + D(B^T(\hat{X}+Z)^TA^T + D^T(\hat{X}+Z)C^T)C \\ &\quad - A^TEB^T - DE^TC \\ &=\; A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C - R_{\hat{X}}. \end{aligned}$$

Since $\hat{X}, \tilde{X} \in \mathcal{L}$, by Lemma 6 we have $R_{\hat{X}} = R_{\tilde{X}} = 0$. It follows that Equation (25) holds as desired. □

Theorem 3. *Algorithm 1 solves Problem 2 in at most np iterations by starting with the initial matrix*

$$X_0 \;=\; A^T(AV_0B + CV_0^TD)B^T + D(B^TV_0^TA^T + D^TV_0C^T)C, \tag{26}$$

where $V_0 \in \mathbb{R}^{n\times p}$ is an arbitrary matrix, or especially $X_0 = 0$.

Proof. If we run Algorithm 1 starting with (26), then we can write the solution X^* of Problem 2 so that

$$X^* \;=\; A^T(AV^*B + CV^{*T}D)B^T + D(B^TV^{*T}A^T + D^TV^*C^T)C,$$

for some matrix $V^* \in \mathbb{R}^{n\times p}$. Now, assume that $\tilde{X}$ is an arbitrary element in $\mathcal{L}$. By Lemma 12, there is a matrix $Z \in \mathbb{R}^{n\times p}$ such that $\tilde{X} = X^* + Z$ and

$$A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C \;=\; 0.$$

Using the property (6), we get

$$\begin{aligned} \langle X^*, Z\rangle \;&=\; \left\langle A^T(AV^*B + CV^{*T}D)B^T + D(B^TV^{*T}A^T + D^TV^*C^T)C, Z \right\rangle \\ &=\; \left\langle V^*, A^T(AZB + CZ^TD)B^T + D(B^TZ^TA^T + D^TZC^T)C \right\rangle \\ &=\; 0. \end{aligned}$$

Since X^* is orthogonal to Z, it follows from the Pythagorean theorem that

$$\|\tilde{X}\|_F^2 \;=\; \|X^* + Z\|_F^2 \;=\; \|X^*\|_F^2 + \|Z\|_F^2 \;\geq\; \|X^*\|_F^2.$$

This implies that X^* is the minimal-norm solution. □

Theorem 4. *Consider the sequence $\{X_r\}$ generated by Algorithm 2 starting with the initial matrix*

$$X_0 = \sum_{k=1}^{s} A_k^T\left(\sum_{i=1}^{s} A_iV_0B_i + \sum_{j=1}^{t} C_jV_0^TD_j\right)B_k^T + \sum_{l=1}^{t} D_l\left(\sum_{i=1}^{s} B_i^TV_0^TA_i^T + \sum_{j=1}^{t} D_j^TV_0C_j^T\right)C_l,$$

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

where $V_0 \in \mathbb{R}^{n\times p}$ is an arbitrary matrix, or especially $X_0 = 0 \in \mathbb{R}^{n\times p}$. Then the sequence $\{X_r\}$ converges to the minimal-norm LS solution of Equatiom (2) in at most np iterations.

Proof. The proof is similar to that of Theorem 3. □

6. Least-Squares Solution Closest to a Given Matrix

In this section, we investigate Problem 3. In this case, Equation (1) may have many LS solutions. We shall seek for one that closest to a given matrix with respect to the Frobenius norm.

Theorem 5. *Algorithm 1 solves Problem 3 by substituting E with $E_1 = E - (AYB + CY^TD)$, and choosing the initial matrix to be*

$$W_0 = A^T(AVB + CV^TD)B^T + D(B^TV^TA^T + D^TVC^T)C, \tag{27}$$

where $V \in \mathbb{R}^{n\times p}$ is arbitrary, or specially $W_0 = 0 \in \mathbb{R}^{n\times p}$.

Proof. Let $Y \in \mathbb{R}^{n\times p}$ be given. We can translate Problem 3 into Problem 2 as follows:

$$\begin{aligned}\min_{X\in\mathbb{R}^{n\times p}} &\|AXB + CX^TD - E\|_F \\ &= \min_{X\in\mathbb{R}^{n\times p}} \|AXB + CX^TD - E - AYB - CY^TD + AYB + CY^TD\|_F \\ &= \min_{X\in\mathbb{R}^{n\times p}} \|A(X-Y)B + C(X-Y)^TD - E + AYB + CY^TD\|_F.\end{aligned}$$

Now, substituting $E_1 = E - (AYB + CY^TD)$ and $W = X - Y$, we see that the solution $\check{X}$ of Problem 3 is equal to $W^* + Y$ where W^* is the minimal-norm LS solution of the equation

$$AWB + CW^TD = E_1,$$

in unknown W. By Theorem 3, the matrix W^* can be solved by Algorithm 1 with the initial matrix (27) where $V \in \mathbb{R}^{n\times p}$ is arbitrary matrix, or especially $W_0 = 0$. □

Theorem 6. *Suppose that the matrix Equation (2) is inconsistent. Let $Y \in \mathbb{R}^{n\times p}$ be given. Consider Algorithm 2 when we replace the matrix E by*

$$E_1 = E - \sum_{i=1}^{s} A_iYB_i - \sum_{j=1}^{t} C_jY^TD_j,$$

and choose the initial matrix

$$W_0 = \sum_{k=1}^{s} A_k^T\left(\sum_{i=1}^{s} A_iFB_i + \sum_{j=1}^{t} C_jF^TD_j\right)B_k^T + \sum_{l=1}^{t} D_l\left(\sum_{i=1}^{s} B_i^TF^TA_i^T + \sum_{j=1}^{t} D_j^TFC_j^T\right)C_l,$$

where $F \in \mathbb{R}^{n\times p}$ is arbitrary, or $W_0 = 0 \in \mathbb{R}^{n\times p}$. Then, the sequence $\{X_r\}$ obtained by Algorithm 2 converges to the LS solution of (2) closest to Y within np iterations.

Proof. The proof of the theorem is similar to that of Theorem 5. □

7. Numerical Experiments

In this section, we provide numerical results to show the efficiency and effectiveness of Algorithm 2 (denoted by CG), which is an extension of Algorithm 1. We perform experiments when the coefficients in a given matrix equation are dense/sparse rectangular matrices of moderate/large sizes. We denote by ones(m, n) the m-by-n matrix whose all entries are 1. Each random matrix rand(m, n) has all entries belonging to the interval $(0, 1)$. Each experiment contains some comparisons of Algorithm 2 with the direct Kronecker linearization as well as well-known iterative algorithms. All iterations were performed by

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

MATLAB R2021a, on Mac operating system (M1 chip 8C CPU/8C GPU/8GB/512GB). The performance of algorithms is investigated through the number of iterations, the norm of residual matrices, and the CPU time. The latter is measured in seconds using the functions *tic* and *toc* on MATLAB.

The next is an example of Problem 1.

Example 1. *Consider a generalized Sylvester-transpose matrix equation*

$$A_1XB_1 + A_2XB_2 + A_3XB_3 + C_1X^TD_1 + C_2X^TD_2 = E, \tag{28}$$

where the coefficient matrices are given by

$$\begin{aligned}
&A_1 = 0.5\,\mathrm{ones}(m,n) - \mathrm{rand}(m,n) \in \mathbb{R}^{50\times 50}, && A_2 = 0.5\,\mathrm{ones}(m,n) - \mathrm{rand}(m,n) \in \mathbb{R}^{50\times 50},\\
&A_3 = 0.5\,\mathrm{ones}(m,n) - \mathrm{rand}(m,n) \in \mathbb{R}^{50\times 50}, && B_1 = 0.5\,\mathrm{ones}(p,q) - \mathrm{rand}(p,q) \in \mathbb{R}^{40\times 50},\\
&B_2 = 0.5\,\mathrm{ones}(p,q) - \mathrm{rand}(p,q) \in \mathbb{R}^{40\times 50}, && B_3 = 0.5\,\mathrm{ones}(p,q) - \mathrm{rand}(p,q) \in \mathbb{R}^{40\times 50},\\
&C_1 = 0.5\,\mathrm{ones}(m,p) - \mathrm{rand}(m,p) \in \mathbb{R}^{50\times 40}, && C_2 = 0.5\,\mathrm{ones}(m,p) - \mathrm{rand}(m,p) \in \mathbb{R}^{50\times 40},\\
&D_1 = 0.5\,\mathrm{ones}(n,q) - \mathrm{rand}(n,q) \in \mathbb{R}^{50\times 50}, && D_2 = 0.5\,\mathrm{ones}(n,q) - \mathrm{rand}(n,q) \in \mathbb{R}^{50\times 50},\\
&E = 0.5\,\mathrm{ones}(m,q) - \mathrm{rand}(m,q) \in \mathbb{R}^{50\times 50}.
\end{aligned}$$

In fact, we have $\mathrm{rank}\, M = 2000 \neq 2001 = \mathrm{rank}\,[M\ \ e]$*, i.e., the matrix equation does not have an exact solution. However, M is of full-column rank, so this equation has a unique LS solution. We run Algorithm 2 using an initial matrix* $X_0 = 0 \in \mathbb{R}^{50\times 40}$ *and a tolerance error* $\epsilon = \|Mx^* - e\| = 6.4812$*, where* $x^* = (M^TM)^{-1}M^Te$*. It turns out that Algorithm 2 takes 20 iterations to get a least-square solution, consuming around 0.2 s, while the direct method consumes around 7 s. Thus, Algorithm 2 takes 35 times less computational time than the direct method. We compare the performance of Algorithm 2 with other well-known iterative algorithms: GI method [31], LSI method [31], and TAUOpt method [32]. The numerical results are shown in Table 1 and Figure 1. We see that after running 20 iterations, Algorithm 2 consumes CTs slightly more than other methods, but the relative error* $\|R_r\|_F$ *is less than those of the others. Hence, Algorithm 2 is applicable and has a good performance.*

Table 1. Relative error and computational time for Example 1.

Method	Iterations	CPU	$\|R_r\|_F$
CG	20	0.199308	6.407766
GI	20	0.129715	10.907665
LSI	20	0.179449	14.390460
TAUOpt	20	0.073866	7.806273
Direct	–	7.048632	0

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

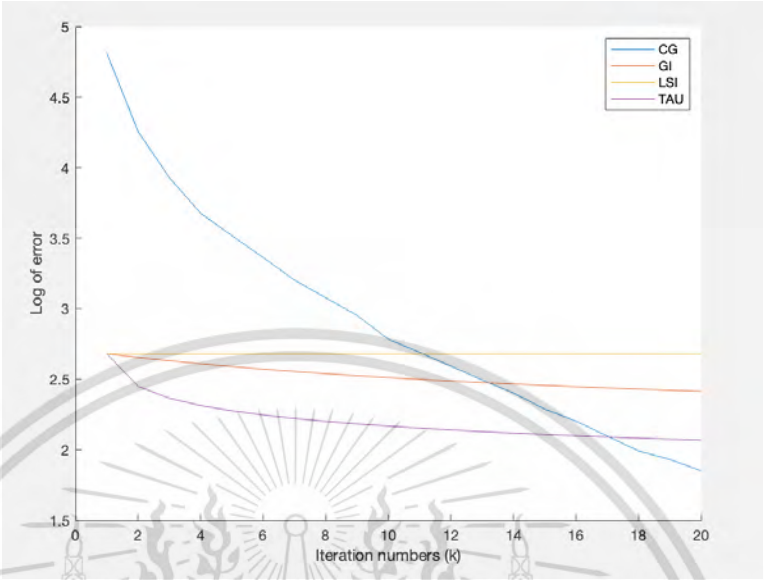

Figure 1. The logarithm of the relative error $\|R_r\|_F$ for Example 1.

The next is an example of Problem 2.

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

$$A_1XB_1 + C_1X^TD_1 + C_2X^TD_2 = E, \tag{29}$$

where

$$A_1 = -0.08 \times \text{ones}(m,n) \in \mathbb{R}^{30\times 25}, \quad B_1 = \text{tridiag}(0.11,-0.61,-0.29) \in \mathbb{R}^{30\times 30},$$
$$C_1 = \text{tridiag}(-0.03,-0.22,-0.1) \in \mathbb{R}^{30\times 30}, \quad C_2 = \text{tridiag}(0.38,0.29,-0.41) \in \mathbb{R}^{30\times 30},$$
$$D_1 = -0.13 \times \text{ones}(n,q) \in \mathbb{R}^{25\times 30}, \quad D_2 = 0.04 \times \text{ones}(n,q) \in \mathbb{R}^{25\times 30},$$
$$E = -0.01 \times I_{30} \in \mathbb{R}^{30\times 30}.$$

In this case, Equation (29) is inconsistent and the associated matrix M is not of full-column rank. Thus, Equation (29) has many LS solutions. The direct method concerning Moore–Penrose inverse (17) takes 0.627019 s to get the minimal-norm LS solutions. Alternatively, MNLS method [35] can be also used to this kind of problem. However, some coefficient matrices are triangular matrices with multiple zeros, causing the MNLS algorithm diverges and cannot provide answer. Therefore, let us apply Algorithm 2 using a tolerance error $\epsilon = 10^{-5}$. *According to Theorem 4, we choose three different matrices* V_0 *to generate the initial matrix* X_0. *The numerical results are shown in Table 2 and Figure 2.*

Table 2. Relative error and computational time for Example 2.

V_0	Iterations	CPU	$\|R_r\|_F$
0	6	0.036523	0.000008
$0.02 \times$ ones	11	0.036540	0.000009
$-0.01 \times I$	10	0.038425	0.000009

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

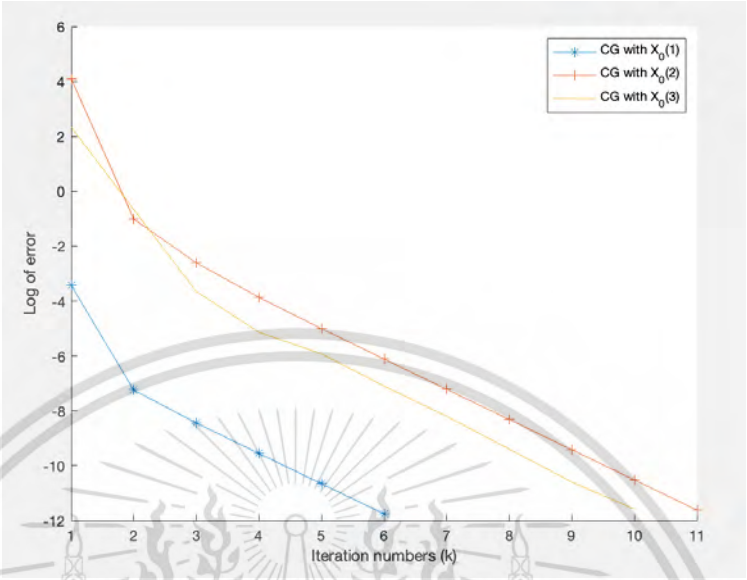

Figure 2. The logarithm of the relative error for Example 2.

Figure 2 shows that the logarithm of the relative errors $\|R_r\|_F$ for CG algorithms, using three initial matrices, are rapidly decreasing to zero. All of them consume around 0.037 s to arrive at the desired solution, which is 16 times less than the direct method.

The following is an example of Problem 3.

Example 3. *Consider a generalized Sylvester-transpose matrix equation*

$$A_1XB_1 + C_1X^TD_1 + C_2X^TD_2 = E, \tag{30}$$

where

$$A_1 = 0.2 \times \text{ones}(m,n) \in \mathbb{R}^{50\times 40}, \quad B_1 = \text{tridiag}(-0.2, 0.3, 0.3) \in \mathbb{R}^{50\times 50},$$
$$C_1 = \text{tridiag}(0.4, -0.2, -0.1) \in \mathbb{R}^{50\times 50}, \quad C_2 = \text{tridiag}(0.7, -0.2, 0.3) \in \mathbb{R}^{50\times 50},$$
$$D_1 = -0.2 \times \text{ones}(n,q) \in \mathbb{R}^{40\times 50}, \quad D_2 = 0.1 \times \text{ones}(n,q) \in \mathbb{R}^{40\times 50},$$
$$E = I_{50} \in \mathbb{R}^{50\times 50}.$$

In fact, Equation (30) is inconsistent and has many LS solutions. The first task is to find the LS solution of Equation (30) closest to $Y = 0.1 \times \text{ones}(n,p) \in \mathbb{R}^{40\times 50}$. *According to Theorem 6, we apply Algorithm 2 with two different matrices V to construct the initial matrix* W_0. *Algorithm 2 with* $V = 0$ *and* $V = -0.19 \times I$ *are denoted in Figure 3 by* CG_1 *and* CG_2, *respectively.*

The second task is to solve Problem 3 when we are given $Y = 0.1 \times \text{ones}(n,p) \in \mathbb{R}^{40\times 50}$. *Similarly, we use two different matrices* $V = 0$ *and* $V = 0.02 \times \text{ones}(n,p) \in \mathbb{R}^{40\times 50}$ *to construct the initial matrix.*

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

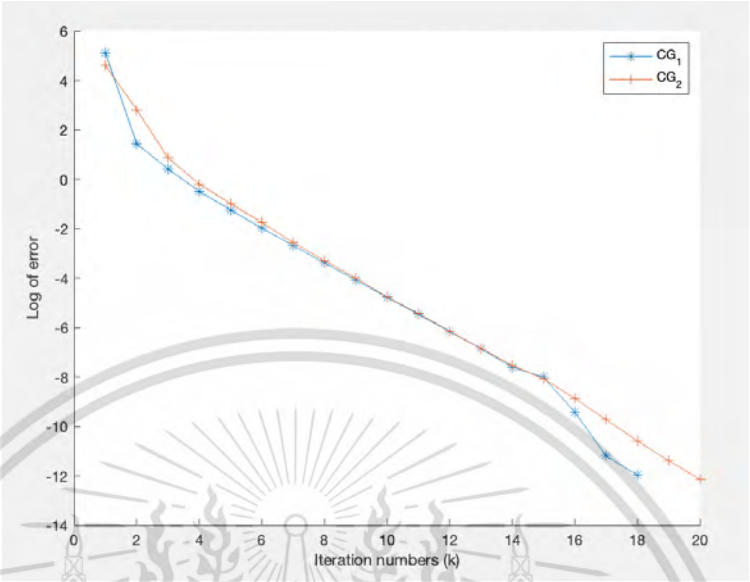

Figure 3. The logarithm of the relative error for Example 3 with Y = 0.1 × ones(n, p).

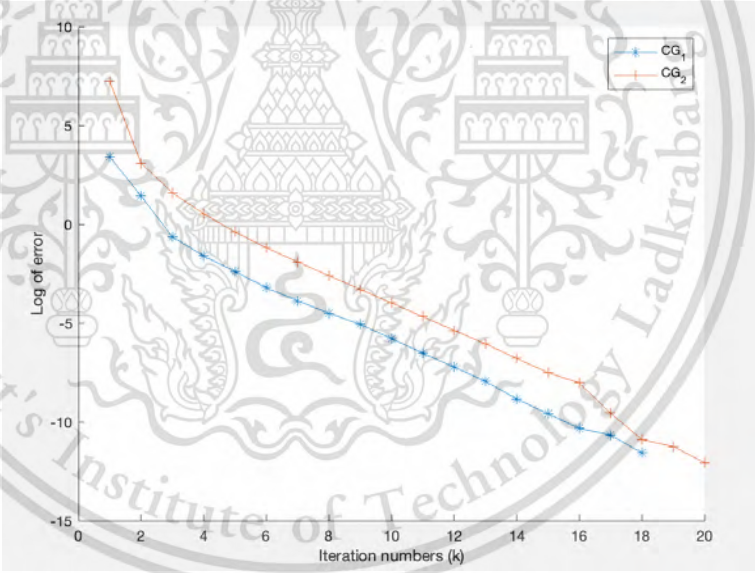

Figure 4. The logarithm of the relative error for Example 3 with Y = I.

Table 3. Relative error and computational time for Example 3.

Y	Initial V	Iterations	CPU	$\|R_r\|_F$	$\|X^* - Y\|_F$
0.1 × ones(n, p)	0	18	0.104135	0.000006	4.3116
	$-0.19 \times I$	20	0.108153	0.000005	4.3116
I	0	18	0.113960	0.000009	0.8580
	0.02 × ones	20	0.108499	0.000006	0.8580

We apply Algorithm 2 with a tolerance error $\epsilon = 10^{-5}$. The numerical results in Figures 3 and 4, and Table 3 illustrate that, in each case, the relative error converges rapidly to zero within 20 iterations, consuming around 0.1 s. Thus, Algorithm 2 performs well in

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

both the number of iterations and computational time. Moreover, changing initial matrix and the desired matrix Y does not siginificantly affect the performance of algorithm.

8. Conclusions

We propose CG-type iterative algorithms, namely, Algorithms 1 and 2, to generate approximate solutions for the generalized Sylvester-transpose matrix Equations (1) and (2), respectively. When the matrix equation is inconsistent, the algorithm will arrive at an LS solution within np iterations with the absence of round-off errors. When the matrix equation has many LS solutions, the algorithm can search for the one with minimal Frobenius norm within np steps. Moreover, given a matrix Y, the algorithm can find the LS solution closest to Y within np steps. The numerical simulations validate the relevance of the algorithm for medium/large sizes of squares/non-squares matrices. The algorithm is always applicable for any given initial matrix and the given matrix Y. The algorithm performs well in both the number of iterations and computational times, compared to the direct Kronecker linearization and well-known iterative methods.

Author Contributions: Writing—original draft preparation, K.T.; writing—review and editing, P.C.; data curation, K.T.; supervision, P.C. All authors have read and agreed to the published version of the manuscript.

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

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare there is no conflict of interest.

References

1. Geir, E.D.; Fernando, P. *A Course in Robust Control Theory: A Convex Approach*; Springer: New York, NY, USA, 1999.
2. Lewis, F. A survey of linear singular systems. *Circ. Syst. Signal Process.* **1986**, *5*, 3–36. [CrossRef]
3. Dai, L. *Singular Control Systems*; Springer: Berlin, Germany, 1989.
4. Enright, W.H. Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations. *ACM Trans. Math. Softw.* **1978**, *4*, 127–136. [CrossRef]
5. Aliev, F.A.; Larin, V.B. *Optimization of Linear Control Systems: Analytical Methods and Computational Algorithms*; Stability Control Theory, Methods Applications; CRC Press: Boca Raton, FL, USA, 1998
6. Calvetti, D.; Reichel, L. Application of ADI iterative methods to the restoration of noisy images. *SIAM J. Matrix Anal. Appl.* **1996**, *17*, 165–186. [CrossRef]
7. Duan, G.R. Eigenstructure assignment in descriptor systems via output feedback: A new complete parametric approach. *Int. J. Control* **1999**, *72*, 345–364. [CrossRef]
8. Duan, G.R. Parametric approaches for eigenstructure assignment in high-order linear systems. *Int. J. Control Autom. Syst.* **2005**, *3*, 419–429.
9. Kim, Y.; Kim, H.S. Eigenstructure assignment algorithm for second order systems. *J. Guid. Control Dyn.* **1999**, *22*, 729–731. [CrossRef]
10. Fletcher, L.R.; Kuatsky, J.; Nichols, N.K. Eigenstructure assignment in descriptor systems. *IEEE Trans. Autom. Control* **1986**, *31*, 1138–1141. [CrossRef]
11. Frank, P.M. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy a survey and some new results. *Automatica* **1990**, *26*, 459–474. [CrossRef]
12. Epton, M. Methods for the solution of $AXD - BXC = E$ and its applications in the numerical solution of implicit ordinary differential equations. *BIT Numer. Math.* **1980**, *20*, 341–345. [CrossRef]
13. Zhou, B.; Duan, G.R. On the generalized Sylvester mapping and matrix equations. *Syst. Control Lett.* **2008**, *57*, 200–208. [CrossRef]
14. Horn, R.; Johnson, C. *Topics in Matrix Analysis*; Cambridge University Press: New York, NY, USA, 1991.
15. Kilicman, A.; Al Zhour, Z.A. Vector least-squares solutions for coupled singular matrix equations. *Comput. Appl. Math.* **2007**, *206*, 1051–1069. [CrossRef]
16. Simoncini, V. Computational methods for linear matrix equations. *SIAM Rev.* **2016**, *58*, 377–441. [CrossRef]
17. Hajarian, M. Developing BiCG and BiCR methods to solve generalized Sylvester-transpose matrix equations. *Int. J. Autom. Comput.* **2014**, *11*, 25–29. [CrossRef]
18. Hajarian, M. Matrix form of the CGS method for solving general coupled matrix equations. *Appl. Math. Lett.* **2014**, *34*, 37–42. [CrossRef]

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

19. Hajarian, M. Generalized conjugate direction algorithm for solving the general coupled matrix equations over symmetric matrices. *Numer. Algorithms* **2016**, *73*, 591–609. [CrossRef]
20. Dehghan, M.; Mohammadi-Arani, R. Generalized product-type methods based on Bi-conjugate gradient (GPBiCG) for solving shifted linear systems. *Comput. Appl. Math.* **2017**, *36*, 1591–1606. [CrossRef]
21. Zadeh, N.A.; Tajaddini, A.; Wu, G. Weighted and deflated global GMRES algorithms for solving large Sylvester matrix equations. *Numer. Algorithms* **2019**, *82*, 155–181. [CrossRef]
22. Kittisopaporn, A.; Chansangiam, P.; Lewkeeratiyukul, W. 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. [CrossRef]
23. Boonruangkan, N.; Chansangiam, P. Convergence analysis of a gradient iterative algorithm with optimal convergence factor for a generalized Sylvester-transpose matrix equation. *AIMS Math.* **2021**, *6*, 8477–8496. [CrossRef]
24. Zhang, X.; Sheng, X. The relaxed gradient based iterative algorithm for the symmetric (skew symmetric) solution of the Sylvester equation $AX + XB = C$. *Math. Probl. Eng.* **2017**, *2017*. [CrossRef]
25. Xie, Y.J.; Ma, C.F. The accelerated gradient based iterative algorithm for solving a class of generalized Sylvester transpose matrix equation. *Appl. Math. Comp.* **2016**, *273*, 1257–1269. [CrossRef]
26. Tian, Z.; Tian, M.; Gu, C.; Hao, X. An accelerated Jacobi-gradient based iterative algorithm for solving Sylvester matrix equations. *Filomat* **2017**, *31*, 2381–2390. [CrossRef]
27. Sasaki, N.; Chansangiam, P. Modified Jacobi-gradient iterative method for generalized Sylvester matrix equation. *Symmetry* **2020**, *12*, 1831. [CrossRef]
28. Kittisopaporn, A.; Chansangiam, P. 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. [CrossRef]
29. Heyouni, M.; Saberi-Movahed, F.; Tajaddini, A. On global Hessenberg based methods for solving Sylvester matrix equations. *Comp. Math. Appl.* **2018**, *2019*, 77–92. [CrossRef]
30. Hajarian, M. Extending the CGLS algorithm for least squares solutions of the generalized Sylvester-transpose matrix equations. *J. Franklin Inst.* **2016**, *353*, 1168–1185. [CrossRef]
31. Xie, L.; Ding, J.; Ding, F. Gradient based iterative solutions for general linear matrix equations. *Comput. Math. Appl.* **2009**, *58*, 1441–1448. [CrossRef]
32. Kittisopaporn, A.; Chansangiam, P. Approximated least-squares solutions of a generalized Sylvester-transpose matrix equation via gradient-descent iterative algorithm. *Adv. Differ. Equ.* **2021**, *2021*, 266. [CrossRef]
33. Tansri, K.; Choomklang, S.; Chansangiam, P. Conjugate gradient algorithm for consistent generalized Sylvester-transpose matrix equations. *AIMS Math.* **2022**, *7*, 5386–5407. [CrossRef]
34. Wang, M.; Cheng, X. Iterative algorithms for solving the matrix equation $AXB + CX^TD = E$. *Appl. Math. Comput.* **2007**, *187*, 622–629. [CrossRef]
35. Chen, X.; Ji, J. The minimum-norm least-squares solution of a linear system and symmetric rank-one updates. *Electron. J. Linear Algebra* **2011**, *22*, 480–489. [CrossRef]

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.

Author Biography

Name	Miss Kanjanaporn Tansri
Date of Birth	4 June 1995
Address	203 Moo 1, Nonmakkheng, Watthana nakhon, Sakaeo, 27160
Education	2014 - 2017 Bachelor of Science in Applied Mathematics GPA 3.26, King Mongkut's Institute of Technology Ladkrabang 2018 - 2019 Master of Science in Applied Mathematics GPA 3.90, King Mongkut's Institute of Technology Ladkrabang 2020 - 2023 Doctor of Philosophy in Applied Mathematics GPA 3.92, King Mongkut's Institute of Technology Ladkrabang
Scholarship	2020 - 2021 The RA-TA graduate scholarship from the Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Grant No. RA/TA-2563-D-004 2021 - 2024 The Royal Golden Jubilee (RGJ) Ph.D. Scholarship from the National Research Council of Thailand (NRCT), Grant No. N41A640234
Academic Publications	1. K. Tansri, S. Choomklang, P. Chansangiam, "Conjugate gradient algorithm for consistent generalized Sylvester-transpose matrix equations", AIMS Math., 2022, 7, 5386–5407. Web of Science, Q2. 2. K. Tansri, P. Chansangiam, "Conjugate Gradient Algorithm for Least-Squares Solutions of a Generalized Sylvester-Transpose Matrix Equation", Symmetry, 2022, 14(9). Web of Science, Q2.

This material is reserved for educational use only, not allowed for commercial use.
Forbidden to modify the content, and cite the document when use.