GRADIENT-DESCENT ITERATIVE ALGORITHM FOR SOLVING A CLASS OF LINEAR MATRIX EQUATIONS WITH APPLICATIONS TO HEAT AND POISSON EQUATIONS

ADISORN KITTISOPAPORN

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

2022

KMITL-2022-SC-D-001-070

Thesis Title	Gradient-Descent Iterative Algorithm for Solving a Class of Linear Matrix Equations with Applications to Heat and Poisson Equations
Student Name	Mr. Adisorn Kittisopaporn
Student ID	62605006
Degree	Doctor of Philosophy (Applied Mathematics)
Department	Mathematics
Year	2022
Thesis Advisor	Assoc.Prof.Dr. Pattrawut Chansangiam

Abstract

In this research, we introduce a new iterative algorithm for solving a generalized Sylvester-transpose matrix equation $\sum_{t=1}^{p} A_t X B_t + \sum_{s=1}^{q} C_s X^T D_s = E$ and its special cases. The objective of the algorithm is to minimize an error at each iteration by the idea of gradient-descent. The algorithm is widely suitable for any problems as long as the associated matrix has full column-rank. The convergence analysis includes the convergence rate and error estimates which point out that our algorithm converges fast for the small condition number of the associated matrix. Numerical simulations illustrate the capability and effectiveness of the proposed algorithm compared to well-known and recent methods. Furthermore, we apply our algorithm to the discretizations of the heat, and Poisson equations.

Keywords : Gradient descent, Heat equation, iterative method, Linear matrix equation, Matrix norms and conditioning, Poisson equation

Acknowledgements

Foremost, I would like to express my extreme sincere gratitude to my advisor Assoc.Prof.Dr. Pattrawut Chansangiam, for imparting his knowledge and expertise in this study as well as his patience, motivation, and endless support. His guidance helped me in all the time of research and writing of this thesis. He is my best advisor and mentor for my whole time at KMITL.

Besides my advisor, I would like to express my special thanks to the rest of my thesis committees: Assoc.Prof.Dr. Wicharn Lewkeeratiyutkul, Asst.Prof.Dr. Nopparat Pochai, Asst.Prof.Dr Kanchana Kumnungkit, and also Dr.Wannaporn Sanprasert for their encouragement, insightful comments, and valuable guidance.

My sincere thanks also goes to Assoc.Prof.Dr Puntani Pongsumpun for offering me the opportunity to join her classes and teaching me the diverse interesting subjects.

I would also like to acknowledge my scholarship. My completion of this thesis could not have been accomplished without the financial support from KMITL Doctoral Scholarships, grant no. KDS 2019/022.

Last but not the least, I would like to thank my parents for giving birth to me at the first place and supporting me spiritually throughout my life.

Mr. Adisorn Kittisopaporn

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

Chapter 1

Introduction

In applied mathematics and control engineering, there have been attention in the following linear matrix equations:

- $AXB = C$,
- $AX + XA^T = B$: the Lyapunov equation,
- $AX + XB = C$: the Sylvester equation,
- $AXB + CXD = E$: the generalized Sylvester equation,
- $AXB + CX^TD = E$: the generalized Sylvester-transpose equation,
- $X + AXB = C$: the Kalman-Yakubovich equation,
- $X + AX^TB = C$: the T-Stein equation.

All of the above equations are special cases of the generalized Sylvester-transpose matrix equation:

$$\sum_{t=1}^{p} A_t X B_t + \sum_{s=1}^{q} C_s X^T D_s = E.$$

These equations play important roles in control and system theory, robust simulation, neural network, and statistics; see, e.g. [1, 2, 3, 4].

A traditional method of finding their exact solutions is to use the Kronecker product of a matrix and the vectorization to reduce the matrix equation to a linear system. However, the dimension of the linear system can be very large due to the Kronecker multiplication, so that the step of finding the inversion of the associated matrix will result in excessive computer storage memory. For that reason, iterative approaches have received much attention and are very famous in matrix analysis and numerical linear algebra. Sometimes, solving the equations by using direct methods can be waste of our time. Although, the direct method is a way of finding the exact solution, it is suitable only for small systems because solving large systems will end up with excess time-consuming. Another weakness is that in real applications, the coefficient matrix could be sparse, meaning that most of its elements are zero. In this case, to keep track of the whole matrix is wasteful.

Iterative methods are numerical methods. They create a sequence of approximated solution which converges to the exact solution. To develop the iterative methods, we utilize the knowledge from linear algebra together with matrix analysis such as norms of vector and matrix, convergence of sequence of vectors and matrices, etc.

The iterative methods have few steps of computation but the result gets closer and closer to the exact solution. The iterative methods are suit for large systems. The iteration can be stopped at any time whenever desired accuracy is obtained. All of these processes take significant shorter computation time than the direct methods would require. There are several ideas to formulate an iterative procedure, namely, one can use matrix sign function [5], block recursion [6, 7], Krylov subspace [8, 9], Hermitian and skew-Hermitian splitting [10, 11]. In addition, the conjugate gradient (CG) is an interesting idea to formulate finite-step iterative procedures to obtain the exact solution at the final step. There are variants of CG method for solving linear matrix equations, namely, the generalized conjugate direction method (GCD) [12], the conjugated gradient least-squares method (CGLs) [13], generalized product-type methods based on a bi-conjugate gradient (GPBiCG) [14]. For more related research works; see, e.g., [15, 16, 17, 18].

In the recent decade, the ideas of gradients, hierarchical identification and minimization of associated norm-error functions have encouraged and brought about many researches. In this way, we decompose an equation into some subsystems, and then the unknown parameters of each subsystem are identified successively. Here, we regard the unknown matrix X as the parameters (parameter matrix) of a system to be identified, and propose ways of decomposing the system equations, leading to gradient based iterative methods for solutions of the matrix equations involved. For example, we recall the Sylvester matrix equation

$$AX + XB = C, \tag{1.1}$$

where $A \in M_m$, $B \in M_n$ and $C \in M_{m,n}$ are given constant matrices, $X \in M_{m,n}$ is the unknown matrix to be solved. According to the hierarchical identification principle, the equation (1.1) is decomposed into two subsystems as follows. Define two matrices

$$b_1 := C - XB \quad \text{and} \quad b_2 := C - AX.$$

Then, we obtain two fictitious subsystems

$$AX = b_1 \quad \text{and} \quad XB = b_2.$$

Based on least squares optimization, we form two criterion functions and the parameters of each subsystems are identified, respectively.

$$J_1(X) := \|AX - b_1\|_F^2 \quad \text{and} \quad J_2(X) := \|XB - b_2\|_F^2$$

where $\|\cdot\|_F$ is the Frobenius norm. Minimizing $J_1(X)$ and $J_2(X)$ leads to the following recursive equations, for which we called the gradient based iterative method:

$$X_{1,k+1} = X_{1,k} + \mu A^T(b_1 - AX_{1,k})$$

$$X_{2,k+1} = X_{2,k} + \mu(b_2 - X_{2,k}B)B^T.$$

Here, μ is called the iterative step-size or convergence factor. For the criterion of the convergence factor and more details about the gradient based iterative method see, e.g. [19, 20, 21]. The gradient based iterative method has been developed and played a role in many pieces of research in a few decades, for instances,

- the least-squares based iterative method [20, 21, 22],
- the relaxed gradient based iterative method [23],
- the modified gradient based iterative method [24],
- the Jacobi-gradient based iterative method [25],
- the accelerated gradient based iterative method [26].

See more related methods in [27, 28, 29, 30, 31, 32, 33, 34]. The developed iterative methods can be applied to state-space models [35], controlled autoregressive systems [36], and parameter estimation in signal processing [37]. Every method has something in common, that is the criterion to choose the convergence factor. It comes in the form of an interval. It is proved that if we choose the convergence factor in that interval, then the algorithm will converge to the exact solution or the least-squares solution. Unfortunately, choosing the good convergence factor becomes one of the problems because some values of the convergence factor effects the performance of algorithm to converge slowly. Recently, there have been some researches that work on finding the optimal convergence factor. Consequently, the algorithm will result in the fastest performance of convergence compared with all of the convergence factors in the given interval, see e.g., [33, 34]. In this work, we generate the optimal convergence factor for each iteration. The result will cause the algorithm to take slightly more computation time but the performance of convergence is far faster than those of other algorithms.

Furthermore, the activity in this field involves exploiting the underlying mathematical or physical problem that gives rise to the linear system in order to design better iterative methods. The underlying problems are often finite difference or finite element models of physical systems, usually involving a differential equation. There are many kinds of physical systems, differential equations, and finite difference and finite element models, and so many methods. We cannot hope to cover all or even most interesting situations, so we will limit ourselves to a model problem, the standard finite difference approximation to heat and Poisson equations on a rectangular domain [38]. The heat equation is of fundamental importance in diverse scientific fields. In mathematics, it is the prototypical parabolic partial differential equation. In statistics, the heat equation is connected with the study of Brownian motion via the Fokker-Planck equation. The diffusion equation, a more general version of the heat equation, arises in connection with the study of chemical diffusion and other related

processes [39]. Moreover, Engineering, fluid mechanics, atmospheric science, climate physics, weather forecasting, option pricing, geophysics, solar physics, all use the heat equation to solve their problems. Poisson equation and its close relation, Laplace equation, arise in many applications, including electromagnetics, fluid mechanics, heat flow, diffusion, and quantum mechanics, see, e.g. [40, 41].

The organization of this thesis is as follows. In Chapter 2, famous and recent iterative methods for solving the linear matrix equations are presented. In addition, the auxiliary tools to solve the linear matrix equations and to make a convergence analysis of an iterative method are supplied. In Chapter 3, a new iterative algorithm based on the techniques of gradient-descent is proposed. The direct method to find the exact or least-squares solution of the generalized Sylvester-transpose matrix equation is shown in details. Also, we make the convergence analysis of the proposed algorithm. In Chapter 4, we present numerical simulations to illustrate the effectiveness and capability of the proposed algorithm. In Chapter 5, the proposed algorithm is applied to the discretization of the certain differential equations, i.e., the one-dimensional heat equation and the two-dimensional Poisson's equation. Finally, we make a conclusion and suggestion for further study in Chapter 6.

Chapter 2

Preliminaries on iterative methods for solving linear matrix equations

In this chapter, we present famous and recent iterative methods for solving the linear matrix equations. All of the mentioned iterative methods will be used to compare with our proposed algorithm afterwards. In addition, we supply auxiliary tools to solve linear matrix equations and to make a convergence analysis of an iterative method for solving such equations. Throughout this thesis, all matrices considered here are real. Let us denote the set of $m \times n$ matrices by $M_{m,n}$. When $m = n$ we write M_n instead of $M_{n,n}$. The (i, j)-th entry of a matrix A is denoted by $A(i, j)$ or a_{ij}.

2.1 Iterative methods for the matrix equation $AXB = C$

Consider the matrix equation

$$AXB = C \tag{2.1}$$

where $A \in M_{l,m}$ is full column-rank, $B \in M_{n,r}$ is full row-rank, $C \in M_{l,r}$ is a known constant matrix, and $X \in M_{m,n}$ is an unknown matrix.

2.1.1 Gradient based iterative (GI) method

In 2008, Ding, Liu, and Ding [20] proposed the following iterative method for Eq. (2.1).

Proposition 2.1. [20] If Eq. (2.1) has a unique solution X^*, then the gradient based iterative (GI) method,

$$X_{k+1} = X_k + \mu A^T(C - AX_kB)B^T,$$
$$0 < \mu < \frac{2}{\lambda_{\max}(AA^T)\lambda_{\max}(B^TB)} \text{ or } \mu \leqslant \frac{2}{\|A\|^2\|B\|^2}$$

yields $X_k \to X^*$.

2.1.2 Least-squares based iterative (LS) method

In 2008, Ding, Liu, and Ding [20] proposed the following iterative method for Eq. (2.1).

Proposition 2.2. [20] If Eq. (2.1) has a unique solution X^*, then the least squares (LS) iterative method,

$$X_{k+1} = X_k + \mu(A^TA)^{-1}A^T(C - AX_kB)B^T(BB^T)^{-1}, \qquad 0 < \mu < 2$$

yields $X_k \to X^*$.

2.2 Iterative methods for the Sylvester matrix equation

Consider the Sylvester matrix equation

$$AX + XB = C \tag{2.2}$$

where $A \in M_{l,m}$ is full column-rank, $B \in M_{n,r}$ is full row-rank, $C \in M_{l,r}$ is a known constant matrix, and $X \in M_{m,n}$ is an unknown matrix.

2.2.1 Gradient based iterative (GI) method

In 2005, Ding and Chen [19] proposed the following method for solving (2.2).

Method 2.3. [19] **The GI method**

Step 1 Input matrices $A, B, C \in M_n$, give any small positive number ϵ. Choose the initial matrices $X_{1,0}$ and $X_{2,0}$. Compute $X_0 = (X_{1,0} + X_{2,0})/2$. Set $k := 1$.

Step 2 If $\delta_k := \|AX_k + X_kB - C\|/\|C\| < \epsilon$, stop; otherwise, go to Step 3.

Step 3 Update the sequences

$$\begin{aligned} X_{1,k+1} &= X_k + \tau A^T(C - AX_k - X_kB), \\ X_{2,k+1} &= X_k + \tau(C - AX_k - X_kB)B^T, \\ X_{k+1} &= \frac{X_{1,k+1} + X_{2,k+1}}{2}. \end{aligned}$$

Step 4 Set $k := k + 1$, return to Step 2.

If we choose the convergence factor τ in $(0, \frac{2}{\lambda_{\max}(AA^T)+\lambda_{\max}(B^TB)})$, the iterative solutions X_k will converge to the exact solution of Eq. (2.2).

2.2.2 Relaxed gradient based iterative (RGI) method

In 2011, Niu, Wang, and Lu [23] proposed a relaxed gradient based iterative method for solving Eq. (2.2). The RGI method has been showed to be convergent when τ is chosen in $(0, \frac{2}{\omega(1-\omega)(\lambda_1+\lambda_2+\lambda_3)})$ with $0 < \omega < 1$, $\lambda_1 = \lambda_{\max}(AA^T)$, $\lambda_2 = \lambda_{\max}(B^TB)$, $\lambda_3 = \sigma_{\max}(BA^T)$.

Method 2.4. [23] **The RGI method**

Step 1 Input matrices $A, B, C \in M_n$, given any small positive numbere ϵ and appropriative positive number $\omega < 1$. Choose the initial matrices $X_{1,0}$ and $X_{2,0}$. Compute $X_0 = \omega X_{1,0} + (1-\omega)X_{2,0}$. Set $k := 1$.

Step 2 If $\delta_k := \|AX_k + X_kB - C\|/\|C\| < \epsilon$, stop; otherwise, go to Step 3.

Step 3 Update the sequences

$$\begin{aligned} X_{1,k+1} &= X_k + (1-\omega)\tau A^T(C - AX_k - X_kB), \\ X_{2,k+1} &= X_k + \omega\tau(C - AX_k + X_kB)B^T, \\ X_{k+1} &= \omega X_{1,k+1} + (1-\omega)X_{2,k+1}. \end{aligned}$$

Step 4 Set $k := k + 1$, return to Step 2.

2.2.3 Modified gradient based iterative (MGI) method

In 2012, Wang, Dai, and Liao [24] presented a modified gradient based iterative (MGI) method for solving Eq. (2.2). It is a half-step-update modification of the GI method.

Method 2.5. [24] **The MGI method**

Step 1 Input matrices $A, B, C \in M_n$, given any small positive number ϵ. Choose the initial matrices $X_{1,0}$ and $X_{2,0}$. Compute $X_0 = (X_{1,0} + X_{2,0})/2$. Set $k := 1$.

Step 2 If $\delta_k := \|AX_k + X_kB - C\|/\|C\| < \epsilon$, stop; otherwise, go to Step 3.

Step 3 Update the sequence

$$X_{1,k+1} = X_k + \tau A^T(C - AX_k - X_kB).$$

Step 4 Compute

$$X_{k+1} = \frac{X_{1,k+1} + X_{2,k}}{2}.$$

Step 5 Update the sequence

$$X_{2,k+1} = X_{k+1} + \tau(C - AX_{k+1} - X_{k+1}B)B^T.$$

Step 6 Compute

$$X_{k+1} = \frac{X_{1,k+1} + X_{2,k+1}}{2}.$$

Step 7 Set $k := k + 1$, return to Step 2.

Here, τ is chosen in $0 < \tau < \min\{\frac{1}{\lambda_{\max}(AA^T)}, \frac{1}{\lambda_{\max}(B^TB)}\}$.

2.2.4 Jacobi-gradient based iterative (JGI) method

In 2017, Tian et al. [25] presented three algorithms for solving the Sylvester matrix equation (2.2). The first method is developed from the GI method and called the Jacobi-gradient based iterative (JGI) method. Furthermore, they introduced relaxation factors for accelerate the speed of convergence and called the accelerated Jacobi-gradient based iterative (AJGI) method.

Method 2.6. [25] **The JGI method**

Step 1: Given any two initial approximate solution block vectors $X_{1,0}$, $X_{2,0}$, and then $X_0 = (X_{1,0} + X_{2,0})/2$.

Step 2: For $k = 1, 2, \ldots$, until converges, do:

Step 3: $X_{1,k+1} = X_k + \mu D_1 (C - AX_k - X_kB)$.

Step 4: $X_{2,k+1} = X_k + \mu (C - AX_k - X_kB) D_2$.

Step 5: $X_{k+1} = (X_{1,k+1} + X_{2,k+1})/2$.

Step 6: End.

Here, D_1 and D_2 are the diagonal parts of A and B respectively.

Method 2.7. [25] **The AJGI1 method**

Step 1: Given two initial approximate solution block vectors $X_{1,0}$ and $X_{2,0}$.

Step 2: For $k = 1, 2, \ldots$, until converges, do:

Step 3: $X_k = (X_{1,k} + X_{2,k})/2$.

Step 4: $X_{1,k+1} = X_k + (1-\omega_1)\mu D_1 (C - AX_k - X_kB)$.

Step 5: $\tilde{X}_k = (1-\omega_2)X_k + \omega_2 X_{1,k+1}$.

Step 6: $X_{2,k+1} = \tilde{X}_k + \omega_1\mu \left(C - A\tilde{X}_k - \tilde{X}_kB\right) D_2$.

Step 7: End.

Based on the technique of half-step-update, the authors also provided the second version of the AJGI method as follows:

Method 2.8. [25] **The AJGI2 method**

Step 1: Given two initial approximate solution block vectors $X_{1,0}$ and $X_{2,0}$.

Step 2: For $k = 1, 2, \ldots$, until converges, do:

Step 3: $X_k = \omega_1 X_{1,k} + (1-\omega_1)X_{2,k}$.

Step 4: $X_{1,k+1} = X_k + (1-\omega_1)\mu A^T (C - AX_k - X_kB)$.

Step 5: $\tilde{X}_k = (1-\omega_2)X_k + \omega_2 X_{1,k+1}$.

Step 6: $X_{2,k+1} = \tilde{X}_k + \omega_1\mu(C - A\tilde{X}_k - \tilde{X}_kB)B^T$.

Step 7: End.

The convergence criteria to select the convergence factor, μ, for the JGI and the AJGI methods are given as follows:

Proposition 2.9. If Eq. (2.2) is consistent and has a unique solution X^*, then the iterative sequence X_k generated by the JGI method converge to X^*, i.e., $\lim_{k\to\infty} X_k = X$; or the error $X_k - X$ converge to zero for any initial value X_0 when μ satisfies

$$\|I - \mu D_1 A\|_2 + \|I - \mu B D_2\|_2 + \mu\tau_1\sigma_2 + \mu\tau_2\sigma_1 < 2,$$

where $\tau_1 = \|D_1\|_2$, $\tau_2 = \|D_2\|_2$, $\sigma_1 = \|A\|_2$, and $\sigma_2 = \|B\|_2$.

Proposition 2.10. If Eq. (2.2) is consistent and has a unique solution X^*, then the iterative sequence X_k generated by the AJGI method converge to X^*, i.e., $\lim_{k\to\infty} X_k = X$; or the error $X_k - X$ converge to zero for any initial value X_0 when μ satisfies

$$\|I - (1-\omega_1)\mu D_1 A\|_2 + (1-\omega_1)\mu\tau_1\sigma_2 + \tau\|I - \omega_1\mu B D_2\|_2 + \omega_1\mu\tau\tau_2\sigma_1 < 2,$$

where $\tau = \|I - \mu\omega_2(1-\omega_1)D_1 A\|_2 + \mu\omega_2(1-\omega_1)\tau_1\sigma_2$ and $0 < \omega_1 < 1$, $\omega_2 > 0$.

2.2.5 Least-squares iterative (LSIA) method for Lyapunov matrix equation

Recently, Sun, Wang, and Liu [22] proposed two modified least-squares iterative methods called LSIA1 and LSIA2 for the Lyapunov matrix equation:

$$AX + XA^T = B.$$

Method 2.11. [22] **The LSIA1 method**

$$X_{1,k+1} = X_k + \mu(A^TA)^{-1}A^T\left(B - AX_k - X_kA^T\right),$$
$$X_{2,k+1} = X_k + \mu\left(B - AX_k - X_kA^T\right)A(A^TA)^{-1},$$
$$X_{k+1} = (X_{1,k+1} + X_{2,k+1})/2, \qquad 0 < \mu < 4.$$

Method 2.12. [22] **The LSIA2 method**

$$X_{k+1} = X_k - \mu(X_k - (A^TA)^{-1}A^T(B - X_kA^T)).$$

Here the convergence factor μ satisfies

$$0 < \mu < \frac{2 + 2\lambda_{\min}(A \otimes A^{-1})}{1 + \lambda^2_{\max}(A \otimes A^{-1}) + 2\lambda_{\min}(A \otimes A^{-1})}.$$

2.3 Iterative methods for the generalized Sylvester matrix equation

In this section, we present the iterative method for solving a more general linear matrix equation

$$\sum_{t=1}^{p} A_tXB_t = C, \tag{2.3}$$

where $A_t \in M_m(\mathbb{R})$, $B_t \in M_n(\mathbb{R})$, and $C \in M_{m,n}(\mathbb{R})$ are given constant matrices, and $X \in M_{m,n}(\mathbb{R})$ is the unknown matrix to be solved.

2.3.1 Gradient based iterative (GI) method

In 2005, Ding and Chen [19] proposed the following gradient based iterative (GI) method for solving Eq. (2.3).

Proposition 2.13. [19] If the matrix equation in (2.3) has a unique solution X^*, then the iterative solution X_k given by the GI method:

$$X_{i,k+1} = X_k + \mu A_i^T\left(C - \sum_{t=1}^{p} A_tX_kB_t\right)B_i^T$$
$$X_{k+1} = (X_{1,k+1} + X_{2,k+1} + \cdots + X_{p,k+1})/p$$

where

$$\frac{1}{\mu} = \sum_{t=1}^{p} \lambda_{\max}(A_tA_t^T)\lambda_{\max}(B_t^TB_t) \text{ or } \frac{1}{\mu} = \sum_{t=1}^{p} \|A_t\|_F^2\|B_t\|_F^2$$

converges to the exact solution X^*.

2.3.2 Least-squares based iterative (LSI) method

In 2008, Ding, Liu, and Ding [20] proposed the following Least-squares based iterative (LSI) method for solving Eq. (2.3).

Method 2.14. [20] **The LSI method**

$$X_{k+1} = X_k + \mu\sum_{i=1}^{p}(A_i^TA_i)^{-1}A_i^T\left(C - \sum_{t=1}^{p} A_tX_kB_t\right)B_i^T(B_iB_i^T)^{-1}, \qquad 0 < \mu < 2p.$$

2.3.3 Gradient iterative method with optimal convergence factor (GIO)

In 2020, Boonruangkan and Chansangiam [33] proposed the following gradient iterative method with optimal convergence factor (GIO) for solving Eq. (2.3).

Method 2.15. [33] **The GIO method**

$$E_k = C - \sum_{j=1}^{p} A_j X_k B_j,$$
$$X_{k+1} = \frac{1}{p}\sum_{i=1}^{p}(X_k + \tau A_i^T E_k B_i^T).$$

The optimal value of $\tau > 0$ for which Method 2.15 has the fastest asymptotic convergence rate is determined by

$$\tau_{opt} = \frac{2}{\lambda_{\max}(P^T P) + \lambda_{\min}(P^T P)},$$

where $P = \sum_{i=1}^{p}(B_i^T \otimes A_i)$.

2.3.4 Gradient based iterative (GI (θ_{opt})) method

In 2021, A.K., Chansangiam, and Lewkeeratiyutkul [34] proposed the gradient based iterative method with the optimal convergence factor for solving Eq. (2.3) in case that $p = 2$.

Method 2.16. [34] **GI θ_{opt} method**

$$F_k = C - A_1 X_k B_1 - A_2 X_k B_2,$$
$$X_{k+1} = X_k + \theta(A_1^T F_k B_1^T + A_2^T F_k B_2^T).$$

Let $P = B_1^T \otimes A_1 + B_2^T \otimes A_2$. The optimal convergence factor is defined by

$$\theta_{opt} = \frac{2}{\lambda_{\min}(P^T P) + \lambda_{\max}(P^T P)}.$$

2.4 Iterative methods for the generalized Sylvester-transpose matrix equation

In this section, we consider the generalized Sylvester-transpose matrix equation

$$\sum_{t=1}^{p} A_t X B_t + \sum_{s=1}^{q} C_s X^T D_s = E, \tag{2.4}$$

where for each $t = 1, \ldots, p$, $A_t \in M_{l,m}$, $B_t \in M_{n,r}$, for each $s = 1, \ldots, q$, $C_s \in M_{l,n}$, $D_s \in M_{m,r}$, $E \in M_{l,r}$ are known matrices and $X \in M_{m,n}$ is the matrix to be determined.

2.4.1 Gradient based iterative (GI) method

In 2009, Xie, Ding and Ding [21] proposed the GI method for solving Eq. (2.4).

Method 2.17. [21] **The GI method**

$$X_{k+1} = \frac{1}{p+q}\left(\sum_{j=1}^{p} X_{j,k} + \sum_{l=1}^{q} X_{p+l,k}\right),$$

$$X_{j,k+1} = X_k + \mu A_j^T\left(E - \sum_{i=1}^{p} A_i X_k B_i - \sum_{i=1}^{q} C_i X_k^T D_i\right) B_j^T,$$

$$X_{p+l,k+1} = X_k + \mu D_l\left(E - \sum_{i=1}^{p} A_i X_k B_i - \sum_{i=1}^{q} C_i X_k^T D_i\right)^T C_l.$$

A conservative choice of the convergence factor μ is

$$0 < \mu < 2\left(\sum_{j=1}^{p} \lambda_{\max}(A_j A_j^T)\lambda_{\max}(B_j^T B_j) + \sum_{l=1}^{q} \lambda_{\max}(C_l C_l^T)\lambda_{\max}(D_l^T D_l)\right).$$

2.4.2 Least-squares based iterative (LSI) method

In 2009, Xie, Ding and Ding [21] proposed the LSI method for solving Eq. (2.4).

Method 2.18. [21] **The LSI method**

$$R_{k+1} = E - \sum_{i=1}^{p} A_i X_k B_i - \sum_{i=1}^{q} C_i X_k D_i,$$

$$X_{j,k+1} = X_k + \mu(A_j^T A_j)^{-1} A_j^T R_{k+1} B_j^T (B_j B_j^T)^{-1},$$

$$X_{p+l,k+1} = X_k + \mu(D_l D_l^T)^{-1} D_l R_{k+1} C_l (C_l^T C_l)^{-1},$$

$$X_{k+1} = \frac{1}{p+q}\left(\sum_{j=1}^{p} X_{j,k+1} + \sum_{l=1}^{q} X_{p+l,k+1}\right), \quad 0 < \mu < 2(p+q).$$

2.4.3 The accelerated gradient based iterative (AGBI) method

In 2016, Xie and Ma [26] constructed an accelerated gradient based iterative (AGBI) method for solving the generalized Sylvester-transpose matrix equation (2.4) where $p, q = 1$. The method is based on the ideas of the GI (Method 2.3), the RGI (Method 2.4) and the MGI (Method 2.5).

Method 2.19. [26] **The AGBI method**

Step 1: Input matrices $A, B, C, D, E \in M_n(\mathbb{R})$, given any small positive number ϵ and appropriate positive number $\omega < 1$. Choose the initial matrices $X_{1,0}$ and $X_{2,0}$. Compute $X_0 = (1-\omega)X_{1,0} + \omega X_{2,0}$. Set $k := 1$.

Step 2: If $\delta_k := \|AX_kB + CX_k^T D - E\|_F < \epsilon$, stop; otherwise, go to Step 3.

Step 3: Update the sequence

$$X_{1,k+1} = X_k + \omega\tau A^T(E - AX_kB - CX_k^T D)B^T.$$

Step 4: Compute

$$X_k = (1-\omega)X_{1,k+1} + \omega X_{2,k}.$$

Step 5: Update the sequence

$$X_{2,k+1} \;=\; X_k + (1-\omega)\tau D(E - AX_kB - CX_k^TD)^TC.$$

Step 6: Compute

$$X_{k+1} \;=\; (1-\omega)X_{1,k+1} + \omega X_{2,k+1}.$$

Step 7: Set $k := k+1$, return to Step 2.

Proposition 2.20. [26] Assume that $AXB + CXD = E$ is consistent and has a unique solution X^*, if the parameter τ is chosen in

$$0 < \tau < \min\left\{\frac{2}{\omega\|A\|_F^2\|B\|_F^2}, \frac{2}{(1-\omega)\|C\|_F^2\|D\|_F^2}\right\},$$

then the iterative sequences $\{X_k\}$ generated by Method 2.19 converge to X^*, i.e., $\lim_{k\to\infty} X_k = X^*$ or the matrix $X_k - X^*$ converges to zero for any initial value X_0.

2.5 Auxiliary tools for analysis

Definition 2.21. Let $A \in M_{m,n}$. A is said to be full-row rank if the rank of A is equal to the number of rows of A, i.e., $\text{rank}(A) = m$. And A is said to be full-column rank if the rank of A is equal to the number of columns of A, i.e., $\text{rank}(A) = n$.

Lemma 2.22. Let $A \in M_{m,n}$. Then the followings are equivalent:

1) A is of full-row rank,

2) AA^T is invertible.

Lemma 2.23. Let $A \in M_{m,n}$. Then the followings are equivalent:

1) A is of full-column rank.

2) A^TA is invertible.

Definition 2.24. Let $A \in M_n$. The *trace* of A is denoted by $\text{tr}(A)$ and defined by

$$\text{tr}(A) \;=\; \sum_{i=1}^{n} a_{ii}.$$

Lemma 2.25. [42] Let A, B be compatible constant matrices, X be a variable matrix and $c \in \mathbb{R}$. The properties and derivatives of trace of matrix are given as follows:

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

2) $\text{tr}(cA) \;=\; c\,\text{tr}(A)$,

3) $\text{tr}(A^T) \;=\; \text{tr}(A)$,

4) $\text{tr}(AB) = \text{tr}(BA)$,

5) $\frac{d}{dX}\,\mathrm{tr}(AX) = \frac{d}{dX}\,\mathrm{tr}(X^TA^T) = A^T$,

6) $\frac{d}{dX}\,\mathrm{tr}(XAX^TB) = BXA + B^TXA^T$.

Definition 2.26. Let $A = [a_{ij}] \in M_{m,n}$ and $B = [b_{ij}] \in M_{p,q}$. The *Kronecker product* of A and B is denoted by $A \otimes B$ and defined to be the block matrix

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix} \in M_{mp,nq}.$$

One of many uses of the Kronecker product is to solve or determine properties of solutions to linear matrix equations. These matrix equations can be converted into an equivalent system of equations where the coefficient matrix involves the Kronecker product. In this transition, matrices are also converted into vectors. The function that implements this conversion is the subject of the next definition.

Definition 2.27. For each matrix $A = [a_{ij}] \in M_{m,n}$, the associated *vector operator* is defined by

$$\mathrm{Vec}(A) = [a_{11} \dots a_{m1}\ a_{12} \dots a_{m2}\ \dots\ a_{1n} \dots a_{mn}]^T.$$

The vector operator creates a column vector from a matrix A by stacking column vectors of A below one another for example:

Example 2.28. Let $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$. Then

$$\mathrm{Vec}(A) = \begin{pmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{pmatrix}$$

It is clear that the mapping $\mathrm{Vec} : M_{m,n} \to \mathbb{R}^{mn}$ is a linear transformation and an isomorphism. The next lemma shows the properties of the Kronecker product and the vector operator.

Lemma 2.29. For compatible matrices A, B, and C, we have the following properties of the Kronecker product and the vector operator.

1) $(A \otimes B)^T = A^T \otimes B^T$,

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

Consider the generalized Sylvester-transpose matrix equation, the equation contains the transpose of the unknown matrix i.e., X^T. So, we need some tools to transform Vec(X^T) into Vec(X). For that reason, we are introducing the permutation matrix. A permutation matrix $P \in M_n$ is so named because the matrix is created by permuting the rows of an $n \times n$ identity matrix according to some permutation of the numbers $1, \ldots, n$. Thus, each column and row contains exactly one 1 and zeros elsewhere. Using the usual matrix multiplication, the product of a square matrix A of the same dimensions as the permutation matrix, PA, creates a matrix $B = PA$ such that B is a permutation of rows of A matching the same rearrangement as the permutation matrix.

Lemma 2.30. Let $P \in M_n$ be a matrix with exactly one 1 in each row and column and zeros elsewhere. Then, P is a permutation matrix.

Lemma 2.31. Let m, n be given positive integers. There exists a unique matrix $P(m,n) \in M_{mn}$ such that for all $X \in M_{m,n}$

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

The matrix $P(m,n)$ depends only on the dimensions m and n and is given by

$$P(m,n) = \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij} \otimes E_{ij}^T$$

where each $E_{ij} \in M_{m,n}$ contains 1 in the i,j^{th} entry and every other entry is zero. Moreover, $P(m,n)$ is a permutation matrix and

$$P(m,n) = P(n,m)^T.$$

Next, we will introduce the tools we use to perform convergence analysis namely, the condition number, the Frobenius norm and the Spectral norm.

Definition 2.32. The *condition number* of an $m \times n$ matrix A is the ratio between its largest and smallest singular values:

$$\kappa(A) = \left(\frac{\lambda_{\max}(A^TA)}{\lambda_{\min}(A^TA)}\right)^{1/2}.$$

Definition 2.33. The *Frobenius norm* is a norm of an $m \times n$ matrix A defined as the square root of the matrix trace of AA^T:

$$\|A\|_F = \sqrt{\text{tr}(AA^T)}.$$

Definition 2.34. The *Spectral norm* is a norm of an $m \times n$ matrix A defined as the square root of the largest singular value of A:

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^TA)}.$$

Lemma 2.35. [42] For any compatible matrices A and B, we have

1) $\|A^TA\|_2 = \|A\|_2^2$,

2) $\|A^T\|_2 = \|A\|_2$,

3) $\|AB\|_F \leqslant \|A\|_2\|B\|_F$.

Definition 2.36. Let $A \in M_n$ be symmetric. Then A is called

- **Positive definite** if $x^TAx > 0$ for all $x \in \mathbb{R}^n - \{0\}$,
- **Positive semidefinite** if $x^TAx \geqslant 0$ for all $x \in \mathbb{R}^n$.

To show that our proposed algorithm converges, the analysis will hold for strongly convex functions. So, we will first recall its definition and then provide some important properties.

Definition 2.37. A twice-differentiable convex funtion $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *strongly convex* if there exist constant $0 \leqslant m \leqslant M$ such that for all $x \in \mathbb{R}^n$,

$$mI \leqslant \nabla^2 f(x) \leqslant MI.$$

Indeed, m (M, resp.) is a lower (upper, resp.) bound on the smallest (largest, resp.) eigenvalue of $\nabla^2 f(x)$ for all $x \in \mathbb{R}^n$.

Lemma 2.38. [43] If f is strongly convex on $\mathbb{R}^n$, then for $x, y \in \mathbb{R}^n$ we have

$$f(y) \geqslant f(x) + \nabla f(x)^T(y-x) + \frac{m}{2}\|y-x\|_F^2, \tag{2.5}$$

$$f(y) \leqslant f(x) + \nabla f(x)^T(y-x) + \frac{M}{2}\|y-x\|_F^2. \tag{2.6}$$

Chapter 3

Gradient-descent iterative algorithm for solving the generalized Sylvester-transpose matrix equation

In this chapter, we apply the idea of gradient-descent to derive an iterative method for solving the generalized Sylvester-transpose matrix equation. We have divided this chapter into three sections: first, we describe about the direct method that is used to find the exact or least-squares solution of the matrix equation. Second, according to the techniques of gradient-descent, we find the search direction vector and the optimal convergence factor. Then we sum up things altogether to obtain the iterative algorithm. Finally, we make convergence analysis to guarantee that our algorithm converge to a unique solution for any given initial matrix.

3.1 Exact and least-squares solutions of the generalized Sylvester-transpose matrix equation by the Kronecker linearization

In this section, we explain how to solve the generalized Sylvester-transpose matrix equation directly using the Kronecker linearization. Recall the generalized Sylvester-transpose matrix equation

$$\sum_{t=1}^{p} A_t X B_t + \sum_{s=1}^{q} C_s X^T D_s = E, \tag{3.1}$$

where for each $t = 1, \ldots, p$, $A_t \in M_{l,m}$, $B_t \in M_{n,r}$, for each $s = 1, \ldots, q$, $C_s \in M_{l,n}$, $D_s \in M_{m,r}$, $E \in M_{l,r}$ are known matrices whereas $X \in M_{m,n}$ is the matrix to be determined. We can transform Eq. (3.1) to an equivalent linear system by applying the vector operator and utilizing Lemmas 2.29 and 2.31. It follows from Eq. (3.1) that

$$\begin{aligned}
\mathrm{Vec}\left(\sum_{t=1}^{p} A_t X B_t + \sum_{s=1}^{q} C_s X^T D_s\right) &= \mathrm{Vec}(E) \\
\sum_{t=1}^{p} \mathrm{Vec}(A_t X B_t) + \sum_{s=1}^{q} \mathrm{Vec}(C_s X^T D_s) &= \mathrm{Vec}(E) \\
\sum_{t=1}^{p} (B_t^T \otimes A_t)\,\mathrm{Vec}(X) + \sum_{s=1}^{q} (D_s^T \otimes C_s)\,\mathrm{Vec}(X^T) &= \mathrm{Vec}(E) \\
\sum_{t=1}^{p} (B_t^T \otimes A_t)\,\mathrm{Vec}(X) + \sum_{s=1}^{q} (D_s^T \otimes C_s) P(m,n)\,\mathrm{Vec}(X) &= \mathrm{Vec}(E) \\
\left(\sum_{t=1}^{p} (B_t^T \otimes A_t) + \sum_{s=1}^{q} (D_s^T \otimes C_s) P(m,n)\right)\mathrm{Vec}(X) &= \mathrm{Vec}(E).
\end{aligned}$$

Thus,

$$Q\,\mathrm{Vec}(X) = \mathrm{Vec}(E) \tag{3.2}$$

where

$$Q \;=\; \sum_{t=1}^{p}(B_t^T \otimes A_t) + \sum_{s=1}^{q}(D_s^T \otimes C_s)P(m,n) \in M_{rl,mn}. \tag{3.3}$$

Hence, Eq. (3.1) has a (unique) solution if and only if Eq. (3.2) does. To make the algorithm widely useful, we impose an assumption that Q is of full column-rank, or equivalently, Q^TQ is invertible, or equivalently, $\ker Q = \{0\}$.

If Eq. (3.1) has no solution, then we can seek for a least-squares solution according to the following definition

Definition 3.1. [44] A *least-squares solution* to a linear system of equations

$$Ax \;=\; b$$

is a vector $x^* \in \mathbb{R}^n$ that minimizes the squared Euclidean norm $\|Ax - b\|^2$.

Note that the Euclidean norm is sometimes called the Frobenius norm.

Theorem 3.2. [44] Assume that $\ker A = \{0\}$. Then the least-squares solution to the linear system $Ax = b$ under the Euclidean norm is the unique solution x^* to the *normal equations*

$$(A^TA)x = A^Tb, \quad \text{namely} \quad x^* = (A^TA)^{-1}A^Tb.$$

The least-squares error is

$$\|Ax^* - b\|^2 = \|b\|^2 - b^TA(A^TA)^{-1}A^Tb.$$

If Eq. (3.1) has a solution, then it is automatically the least-squares solution. Thus, we obtain the exact (vector) solution to be

$$\mathrm{Vec}(X^*) \;=\; (Q^TQ)^{-1}Q^T\,\mathrm{Vec}(E). \tag{3.4}$$

Otherwise, we will find a matrix X^* that minimizes the squared Frobenius norm

$$\|Q\,\mathrm{Vec}(X) - \mathrm{Vec}(E)\|_F^2.$$

The assumption on Q implies that the least-squares solution for Eq. (3.1) is uniquely determined by the solution of the associated normal equation, and it is also given by Eq. (3.4). In this case, the least-squares error is given by

$$\begin{aligned}\|Q\,\mathrm{Vec}(X^*) - \mathrm{Vec}(E)\|_F^2 \;&=\; \|\,\mathrm{Vec}(E)\|_F^2 - \mathrm{Vec}^T(E)Q\,\mathrm{Vec}(X^*) \\ &=\; \|E\|_F^2 - \mathrm{Vec}^T(E)Q(Q^TQ)^{-1}Q^T\,\mathrm{Vec}(E). \end{aligned}\tag{3.5}$$

We denote both the exact and the least-squares solutions of Eq. (3.1) by X^*.

3.2 Proposing the gradient-descent iterative algorithm for the generalized Sylvester-transpose matrix equation

In this section, we intent to propose the gradient-descent iterative algorithm for creating a sequence of approximated solutions $\{X_k\}$ of Eq. (3.1). The sequence converges to the exact solution X^* and the proof will be shown in the next section. The proposed algorithm is applicable to any problems as long as the matrix Q is of full column-rank, no matter Eq. (3.1) has a solution or not.

Let us consider the Frobenius-norm error

$$\|\sum_{t=1}^{p} A_t X B_t + \sum_{s=1}^{q} C_s X^T D_s - E\|_F.$$

By using Lemmas 2.29 and 2.31, the above norm can be transformed into

$$\|Qx - \mathrm{Vec}(E)\|_F,$$

where $x = \mathrm{Vec}(X)$. We define the quadratic norm-error function $f : \mathbb{R}^{mn} \to \mathbb{R}$ by

$$f(x) := \frac{1}{2}\|Qx - \mathrm{Vec}(E)\|_F^2. \tag{3.6}$$

Since we assume that Q is of full-column rank, the system (3.2) has a unique (exact or least-squares) solution, i.e., an optimal vector x^* of f exists. Hence, the gradient-descent iterative method can be shown as the following recursive equation:

$$x_{k+1} = x_k - \tau_{k+1}\nabla f(x_k),$$

where $-\nabla f(x_k) \in \mathbb{R}^{mn}$ is the search direction vector and $\tau_{k+1} \in [0, \infty)$ is the step size.

3.2.1 The search direction

We shall find the derivative of f by applying Lemma 2.25. For convenience, we let $\tilde{e} = \mathrm{Vec}(E)$.

$$\begin{aligned}
\nabla f(x) = \frac{d}{dx} f(x) &= \frac{1}{2}\frac{d}{dx}\,\mathrm{tr}((Qx - \tilde{e})^T(Qx - \tilde{e})) \\
&= \frac{1}{2}\frac{d}{dx}\,\mathrm{tr}(Qxx^TQ^T - \tilde{e}x^TQ^T - Qx\tilde{e}^T + \tilde{e}\tilde{e}^T) \\
&= \frac{1}{2}(Q^TQx + Q^TQx - Q^T\tilde{e} - Q^T\tilde{e}) \\
&= Q^TQx - Q^T\tilde{e} \\
&= Q^T(Qx - \tilde{e})
\end{aligned} \tag{3.7}$$

Thus, our new iterative equation is in the form

$$x_{k+1} = x_k + \tau_{k+1}Q^T(\tilde{e} - Qx_k).$$

We would like to write our iterative (vector) equation in the form of iterative (matrix) equation. First we will show that $\tilde{e} - Qx_k = \mathsf{Vec}(R_k)$ where

$$R_k \;=\; E - \sum_{t=1}^{p} A_t X_k B_t - \sum_{s=1}^{q} C_s X_k^T D_s.$$

By Lemma 2.29, we have

$$\begin{aligned}
\tilde{e} - Qx_k &= \mathsf{Vec}(E) - \left(\sum_{t=1}^{p} (B_t^T \otimes A_t)\,\mathsf{Vec}(X_k) + \sum_{s=1}^{q} (D_s^T \otimes C_s) P(m,n)\,\mathsf{Vec}(X_k) \right) \\
&= \mathsf{Vec}(E) - \left(\sum_{t=1}^{p} \mathsf{Vec}(A_t X_k B_t) + \sum_{s=1}^{q} \mathsf{Vec}(C_s X_k^T D_s) \right) \\
&= \mathsf{Vec}(E - \sum_{t=1}^{p} A_t X_k B_t - \sum_{s=1}^{q} C_s X_k^T D_s) \\
&= \mathsf{Vec}(R_k).
\end{aligned}$$

Second, we will show that $Q^T(\tilde{e} - Qx_k) = \mathsf{Vec}(W_k)$ where

$$W_k \;=\; \sum_{t=1}^{p} A_t^T R_k B_t^T + \sum_{s=1}^{q} (C_s^T R_k D_s^T)^T.$$

By Lemma 2.29, we have

$$\begin{aligned}
Q^T(\tilde{e} - Qx_k) &= Q^T\,\mathsf{Vec}(R_k) \\
&= \sum_{t=1}^{p} (B_t^T \otimes A_t)^T\,\mathsf{Vec}(R_k) + \sum_{s=1}^{q} P^T(m,n)(D_s^T \otimes C_s)^T\,\mathsf{Vec}(R_k) \\
&= \sum_{t=1}^{p} (B_t \otimes A_t^T)\,\mathsf{Vec}(R_k) + \sum_{s=1}^{q} P(n,m)(D_s \otimes C_s^T)\,\mathsf{Vec}(R_k) \\
&= \sum_{t=1}^{p} \mathsf{Vec}(A_t^T R_k B_t^T) + \sum_{s=1}^{q} P(n,m)\,\mathsf{Vec}(C_s^T R_k D_s^T) \\
&= \sum_{t=1}^{p} \mathsf{Vec}(A_t^T R_k B_t^T) + \sum_{s=1}^{q} (C_s^T R_k D_s^T)^T \\
&= \mathsf{Vec}(\sum_{t=1}^{p} A_t^T R_k B_t^T + \sum_{s=1}^{q} (C_s^T R_k D_s^T)^T) \\
&= \mathsf{Vec}(W_k).
\end{aligned}$$

Then, our (vector form) iterative equation is transformed into a matrix form via Lemma 2.29 and the linearity of the vector operator as follows:

$$\begin{aligned}
\mathsf{Vec}(X_{k+1}) &= \mathsf{Vec}(X_k) + \tau_{k+1}\,\mathsf{Vec}(W_k) \\
&= \mathsf{Vec}(X_k + \tau_{k+1} W_k).
\end{aligned}$$

Since the vector operator is injective, we have our iterative equation in a matrix form as follows

$$X_{k+1} \;=\; X_k + \tau_{k+1} W_k.$$

3.2.2 The step size

To choose a step size, we minimize an error which occurs at the step $k+1$. We define $\phi_{k+1} : [0,\infty) \to \mathbb{R}$ by for each $k \in \mathbb{N} \cup \{0\}$,

$$\begin{aligned}\phi_{k+1}(\tau) = f(x_{k+1}) &= \frac{1}{2}\|Qx_{k+1} - \mathsf{Vec}(E)\|_F^2 \\ &= \frac{1}{2}\|\tau QQ^t(\tilde{e} - Qx_k) + Qx_k - \tilde{e}\|_F^2.\end{aligned}$$

For convenience, we let $\tilde{b} = QQ^t(\tilde{e} - Qx_k)$ and $\tilde{c} = -(Qx_k - \tilde{e})$. Then

$$\phi_{k+1}(\tau) = \frac{1}{2}\|\tau\tilde{b} - \tilde{c}\|_F^2.$$

We differentiate ϕ_{k+1} by using Lemma 2.25 and obtain

$$\begin{aligned}\frac{d}{d\tau}\phi_{k+1}(\tau) &= \frac{1}{2}\frac{d}{d\tau}\mathsf{tr}\left((\tau\tilde{b} - \tilde{c})(\tau\tilde{b} - \tilde{c})^T\right) \\ &= \frac{1}{2}\frac{d}{d\tau}\mathsf{tr}\left(\tau\tilde{b}\tau\tilde{b}^T - \tau\tilde{b}\tilde{c}^T - \tilde{c}\tau\tilde{b}^T + \tilde{c}\tilde{c}^T\right) \\ &= \frac{1}{2}\left(\frac{d}{d\tau}\tau^2\mathsf{tr}(\tilde{b}\tilde{b}^T) - \frac{d}{d\tau}\tau\mathsf{tr}(\tilde{b}\tilde{c}^T) - \frac{d}{d\tau}\tau\mathsf{tr}(\tilde{c}\tilde{b}^T) + \frac{d}{d\tau}\mathsf{tr}(\tilde{c}\tilde{c}^T)\right) \\ &= \frac{1}{2}\left(2\tau\mathsf{tr}(\tilde{b}\tilde{b}^T) - 2\mathsf{tr}(\tilde{b}\tilde{c}^T)\right) \\ &= \tau\mathsf{tr}(\tilde{b}\tilde{b}^T) - \mathsf{tr}(\tilde{b}\tilde{c}^T).\end{aligned}$$

Hence,

$$\frac{d}{d\tau}\phi_{k+1}(\tau) = \tau\|QQ^T(\tilde{e} - Qx_k)\|_F^2 - \|Q^T(\tilde{e} - Qx_k)\|_F^2.$$

It is obvious that the second-order derivative of ϕ_{k+1} is

$$\frac{d^2}{d\tau^2}\phi_{k+1}(\tau) = \|QQ^T(\tilde{e} - Qx_k)\|_F^2 > 0.$$

So when $\frac{d}{d\tau}\phi_{k+1}(\tau) = 0$, we get the minimizer of ϕ_{k+1} as follows

$$\tau_{k+1} = \frac{\|Q^T(\tilde{e} - Qx_k)\|_F^2}{\|QQ^T(\tilde{e} - Qx_k)\|_F^2}.$$

To avoid duplicate manipulations, the denominator of τ_{k+1} is described more precisely via Lemma 2.29 as follows.

$$\begin{aligned}QQ^T(\tilde{e} - Qx_k) &= Q\,\mathsf{Vec}(W_k) \\ &= \sum_{t=1}^{q}(B_t^T \otimes A_t)\,\mathsf{Vec}(W_k) + \sum_{s=1}^{q}(D_s^T \otimes C_s)P(m,n)\,\mathsf{Vec}(W_k) \\ &= \sum_{t=1}^{p}\mathsf{Vec}(A_tW_kB_t) + \sum_{s=1}^{q}\mathsf{Vec}(C_sW_k^TD_s) \\ &= \mathsf{Vec}(\sum_{t=1}^{p}A_tW_kB_t + \sum_{s=1}^{q}C_sW_k^TD_s).\end{aligned}$$

Hence, we have another form of the optimal convergence factor as follows.

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|\sum_{t=1}^{p}A_tW_kB_t + \sum_{s=1}^{q}C_sW_k^TD_s\|_F^2}.$$

3.2.3 The algorithm

An implementation of the gradient-descent iterative algorithm for solving Eq. (3.1) is given by the following algorithm where the search direction and the step size are taken into account. To terminate the algorithm, one can alternatively set the stopping rule to be $\|R_k\|_F - \delta < \epsilon$ where $\epsilon > 0$ is a small number and δ is the least-squares error described in Eq. (3.5).

Algorithm 1: The gradient-descent iterative algorithm for Eq. (3.1)

A_t, B_t, C_s, D_s, E, X_0;
for $k = 0, \ldots, n$ **do**
  $R_k = E - \sum_{t=1}^{p} A_t X_k B_t - \sum_{s=1}^{q} C_s X_k^T D_s$;
  $W_k = \sum_{t=1}^{p} A_t^T R_k B_t^T + \sum_{s=1}^{q} (C_s^T R_k D_s^T)^T$;
  $\tau_{k+1} = \|W_k\|_F^2 / \| \sum_{t=1}^{p} A_t W_k B_t + \sum_{s=1}^{q} C_s W_k^T D_s \|_F^2$;
  $X_{k+1} = X_k + \tau_{k+1} W_k$
end

3.2.4 Iterative algorithms for special cases of the generalized Sylvester-transpose matrix equation

In this section, we describe how to use Algorithm 1 to solve the interesting special cases of the generalized Sylvester-transpose equation.

- **The matrix equation $AXB = E$**
  We recall the following matrix equation

$$AXB = E \tag{3.8}$$

  where $A \in M_{l,m}$, $B \in M_{n,r}$, $E \in M_{l,r}$ and $X \in M_{m,n}$. For this equation, the optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|AW_kB\|_F^2} \tag{3.9}$$

  where $W_k = A^T R_k B^T$ and $R_k = E - AX_kB$.

- **The Lyapunov equation**
  We recall the Lyapunov equation

$$AX + XA^T = E \tag{3.10}$$

  where A, E and X are square matrices of equal dimension. For this equation, the optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|AW_k + W_kA^T\|_F^2} \tag{3.11}$$

  where $W_k = A^T R_k + R_k A$ and $R_k = E - AX_k - X_k A^T$.

- **The Sylvester equation**
  We recall the Sylvester equation

$$AX + XB = E \tag{3.12}$$

  where A, B, E, and X are square matrices of equal dimension. For this equation, the optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|AW_k + W_kB\|_F^2} \tag{3.13}$$

  where $W_k = A^T R_k + R_k B^T$ and $R_k = E - AX_k - X_k B$.

- **The Kulman-Yakubovich equation**
  We recall the Kulman-Yakubovich equation

$$X - AXB = E \tag{3.14}$$

  where $A \in M_m$, $B \in M_n$, $E \in M_{m,n}$, and $X \in M_{m,n}$. For this equation, the optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|W_k - AW_kB\|_F^2}$$

  where $W_k = R_k - A^T R_k B^T$ and $R_k = E - X_k + AX_kB$.

- **The T-Stein equation**
  We recall the T-Stein equation

$$X + AX^T B = E \tag{3.15}$$

  where A, B, E, and X are square matrices of equal dimension. For this equation, the optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|W_k + AW_k^T B\|_F^2}$$

  where $W_k = R_k + (A^T R_k B^T)^T$ and $R_k = E - X_k - AX_k^T B$.

3.3 Convergence analysis

In this section, we will prove that Algorithm 1 does converge to the exact solution. In addition, we give convergence rates and error estimates of the algorithm. First of all, we define an extension of the Frobenius norm. This following norm will be used in the convergence analysis.

Definition 3.3. Given a full-column-rank matrix $P \in M_{k,n}$, we define the *P-weighted Frobenius norm* of $A \in M_{m,n}$ by

$$\|A\|_P := \|PA\|_F = \sqrt{\text{tr}(A^T P^T PA)}. \tag{3.16}$$

Theorem 3.4. Consider Eq. (3.1). Assume that Q is of full column-rank.

(i) Suppose Eq. (3.1) has a solution (and thus, the solution is unique). Then for any initial matrix X_0, the sequence X_k of approximated solutions generated by Algorithm 1 converges to the exact solution X^*.

(ii) Suppose Eq. (3.1) has no solution (and thus, it has the unique least-squares solution X^*). Then, $\|X_k\|_Q \to \|X^*\|_Q$ for any initial matrix X_0. Here, $\|\cdot\|_Q$ is the Q-weighted Frobenius norm defined by Eq. (3.16).

Proof. Since $x^* = \text{Vec}(X^*)$ is the optimal solution of $\min_{x\in\mathbb{R}^{mn}} f(x)$, we denote the minimum value, $\inf_{x\in\mathbb{R}^{mn}} f(x) = f(x^*)$ as δ. Note that δ is equal to the least-squares error determined by Eq. (3.5) and is zero if X^* is the unique exact solution. If there exists $k \in \mathbb{N}$ such that $\nabla f(x_k) = 0$, then $X_k = X^*$ and the result holds. To investigate the convergence of the algorithm, we assume that $\nabla f(x_k) \neq 0$ for all k. Consider the strong convexity of f, we have from Eq. (3.7) that $\nabla^2 f(x_k) = Q^T Q$. Let $\lambda_{\min}$ ($\lambda_{\max}$) be a minimum (maximum) eigenvalue of $Q^T Q$, respectively. Since $Q^T Q$ is symmetric, we have

$$\lambda_{\min} I \leqslant \nabla^2 f(x_k) \leqslant \lambda_{\max} I,$$

Thus, f is strongly convex. From (2.5), substituting $y = x_{k+1}$ and $x = x_k$ yields

$$f(y) \geqslant f(x_k) - \tau\|\nabla f(x_k)\|_F^2 + \frac{\lambda_{\min}\tau^2}{2}\|\nabla f(x_k)\|_F^2.$$

We minimize the RHS by taking $\tau = 1/\lambda_{\min}$, so that

$$f(y) \geqslant f(x_k) - \frac{1}{2\lambda_{\min}}\|\nabla f(x_k)\|_F^2.$$

Since the above equation is true for all $y \in \mathbb{R}^{mn}$, we have

$$\delta \geqslant f(x_k) - \frac{1}{2\lambda_{\min}}\|\nabla f(x_k)\|_F^2. \tag{3.17}$$

Similarly, from (2.6), we have

$$f(x_{k+1}) \leqslant f(x_k) - \tau\|\nabla f(x_k)\|_F^2 + \frac{\lambda_{\max}\tau^2}{2}\|\nabla f(x_k)\|_F^2.$$

Minimizing the RHS by taking $\tau = 1/\lambda_{\max}$ yields

$$f(x_{k+1}) \leqslant f(x_k) - \frac{1}{2\lambda_{\max}}\|\nabla f(x_k)\|_F^2. \tag{3.18}$$

Subtracting each side of (3.18) by δ and combining $\|\nabla f(x_k)\|_F^2 \geqslant 2\lambda_{\min}(f(x_k) - \delta)$ (from (3.17)), we get

$$\begin{aligned} f(x_{k+1}) - \delta &\leqslant f(x_k) - \delta - \frac{1}{2\lambda_{\max}}\|\nabla f(x_k)\|_F^2 \\ &\leqslant (f(x_k) - \delta) - \frac{2\lambda_{\min}}{2\lambda_{\max}}(f(x_k) - \delta) \\ &\leqslant \left(1 - \frac{\lambda_{\min}}{\lambda_{\max}}\right)(f(x_k) - \delta) \end{aligned}$$

Putting $\alpha := 1 - \lambda_{\text{min}}/\lambda_{\text{max}}$, we have

$$f(x_{k+1}) - \delta \leqslant \alpha(f(x_k) - \delta). \tag{3.19}$$

By induction, we obtain

$$f(x_k) - \delta \leqslant \alpha^k(f(x_0) - \delta). \tag{3.20}$$

Since Q^TQ is assumed to be invertible, $Q^TQ > 0$, it follows that $\lambda_{\text{min}} > 0$ and hence $0 < \alpha < 1$. Thus, $f(x_k) - \delta \to 0$, or equivalently, $f(x_k) \to \delta$ as $k \to \infty$.

Consider the case of X^* is the unique exact solution, i.e., $\delta = 0$. We have $f(x_k) \to 0$, or equivalently $Qx_k - \text{Vec}(E) \to 0$ as $k \to \infty$. Now, the assumption that Q is of full column-rank implies that

$$x_k \to (Q^TQ)^{-1}Q^T \text{Vec}(E).$$

Therefore, $X_k = \text{Vec}^{-1}(x_k) \to X^*$ as $k \to \infty$.

The other case is X^* is the unique least-squares solution, i.e., $\delta > 0$. We have from $f(x_k) \to \delta$, then

$$\begin{aligned}
&\frac{1}{2}\|Qx_k - \tilde{e}\|_F^2 \to \|\tilde{e}\|_F^2 - \tilde{e}^TQx^* \\
\Leftrightarrow\ &\frac{1}{2}\,\text{tr}[(Qx_k - \tilde{e})^T(Qx_k - \tilde{e})] \to \text{tr}(\tilde{e}^T\tilde{e}) - \tilde{e}^TQx^* \\
\Leftrightarrow\ &\frac{1}{2}[\text{tr}(Qx_kx_k^TQ^T) - \text{tr}(\tilde{e}x_k^TQ^T) - \text{tr}(Qx_k\tilde{e}^T) + \text{tr}(\tilde{e}\tilde{e}^T)] \to \text{tr}(\tilde{e}^T\tilde{e}) - \tilde{e}^TQx^* \\
\Leftrightarrow\ &\frac{1}{2}\,\text{tr}(Qx_kx_k^TQ^T) - \text{tr}(\tilde{e}^TQx^*) + \frac{1}{2}\,\text{tr}(\tilde{e}\tilde{e}^T) \to \text{tr}(\tilde{e}^T\tilde{e}) - \tilde{e}^tQx^* \\
\Leftrightarrow\ &\frac{1}{2}\,\text{tr}(Qx_kx_k^TQ^T) \to \frac{1}{2}\,\text{tr}(\tilde{e}^T\tilde{e}) \\
\Leftrightarrow\ &\|Qx_k\|_F^2 \to \|\tilde{e}\|_F^2 \\
\Leftrightarrow\ &\|Qx_k\|_F^2 \to \|Qx^*\|_F^2 \\
\Leftrightarrow\ &\|x_k\|_Q^2 \to \|x^*\|_Q^2.
\end{aligned}$$

Therefore, $\|X_k\|_Q \to \|X^*\|_Q$ as $k \to \infty$. □

We denote the condition number of Q by $\kappa = \kappa(Q)$. Observe that $\alpha = 1 - \kappa^{-2}$. Making use of Lemma 2.29 (2), a relation between the quadratic norm-error $f(x_k)$ and the norm of residual error $\|R_k\|$ is given by

$$f(x_k) = \frac{1}{2}\|R_k\|_F^2. \tag{3.21}$$

From the relation (3.21) and the inequality (3.19), we obtain

$$\begin{aligned}
\frac{1}{2}\|R_k\|_F^2 - \delta &\leqslant \alpha(\frac{1}{2}\|R_{k-1}\|_F^2 - \delta) \\
\|R_k\|_F^2 &\leqslant 2\alpha(\frac{1}{2}\|R_{k-1}\|_F^2 - \delta) + 2\delta \\
&= \alpha\|R_{k-1}\|_F^2 - 2\alpha\delta + 2\delta \\
&= \alpha\|R_{k-1}\|_F^2 - 2\delta(\alpha - 1) \\
&= \alpha\|R_{k-1}\|_F^2 + 2\delta\kappa^{-2}.
\end{aligned}$$

Similarly, from the inequality (3.20), we obtain

$$\begin{aligned}\frac{1}{2}\|R_k\|_F^2 - \delta &\leqslant \alpha^k(\frac{1}{2}\|R_0\|_F^2 - \delta)\\ \|R_k\|_F^2 &\leqslant 2(\alpha^k(\frac{1}{2}\|R_0\|_F^2 - \delta) + \delta)\\ &= \alpha^k\|R_0\|_F^2 + 2\delta(1-\alpha^k).\end{aligned}$$

Hence, the inequalities (3.19) and (3.20) become the following estimation respectively:

$$\|R_k\|_F^2 \leqslant \alpha\|R_{k-1}\|_F^2 + 2\delta\kappa^{-2} \tag{3.22}$$

$$\|R_k\|_F^2 \leqslant \alpha^k\|R_0\|_F^2 + 2\delta(1-\alpha^k). \tag{3.23}$$

In case of Eq. (3.1) has a unique exact solution ($\delta = 0$), then the error estimation (3.22) and (3.23) reduce to (3.24) and (3.25), respectively.

$$\|R_k\|_F \leqslant \alpha^{\frac{1}{2}}\|R_{k-1}\|_F, \tag{3.24}$$

$$\|R_k\|_F \leqslant \alpha^{\frac{k}{2}}\|R_0\|_F. \tag{3.25}$$

Since $0 < \alpha < 1$, it follows that if $\|R_{k-1}\|_F$ are nonzero, then

$$\|R_k\|_F < \|R_{k-1}\|_F. \tag{3.26}$$

The above discussion is summarized to the following theorem.

Theorem 3.5. Assume that Q is of full column-rank.

(i) Suppose Eq. (3.1) has a unique solution. The error estimation $\|R_k\|_F$ compared with $\|R_{k-1}\|_F$ (the preceding iteration) and $\|R_0\|_F$ (the initial iteration) are given by (3.24) and (3.25), respectively. Particularly, the relative error $\|R_k\|_F$ gets smaller than the preceding (nonzero) error, as in (3.26).

(ii) When Eq. (3.1) has a unique least-squares solution, the error estimation (3.22) and (3.23) hold.

In both cases, the convergence rate of Algorithm 1 (regarding the error $\|R_k\|_F$) is governed by $\sqrt{1-\kappa^{-2}}$.

Remark 3.6. The relative errors (3.22) and (3.23) do not seem to decrease every step of iteration since the terms $2\delta\kappa^{-2}$ and $2\delta(1-\alpha^k)$ are positive. However, the inequality (3.18) implies that $\{\|R_k\|_F\}_{k=1}^{\infty}$ is a strictly decreasing sequence converging to δ.

Theorem 3.7. Suppose that Q is of full column-rank and Eq. (3.1) has a unique exact solution. We have the error estimation $\|X_k - X^*\|_F$ compared with the preceding iteration and the initial iteration of Algorithm 1 are provided by:

$$\|X_k - X^*\|_F \leqslant \kappa\sqrt{\kappa^2-1}\|X_{k-1} - X^*\|_F, \tag{3.27}$$

$$\|X_k - X^*\|_F \leqslant \kappa^2(1-\kappa^{-2})^{\frac{k}{2}}\|X_0 - X^*\|_F. \tag{3.28}$$

Particularly, the convergence rate of the algorithm is governed by $\sqrt{1-\kappa^{-2}}$.

Proof. Utilizing (3.25) and Lemma 2.35, we have

$$\begin{aligned}
\|X_k - X^*\|_F &= \|x_k - x^*\|_F \\
&= \|(Q^TQ)^{-1}(Q^TQ)x_k - (Q^TQ)^{-1}(Q^TQ)x^*\|_F \\
&\leqslant \|(Q^TQ)^{-1}\|_2\|Q^T\|_2\|Qx_k - Qx^*\|_F \\
&\leqslant (1-\kappa^{-2})^{\frac{k}{2}}\|(Q^TQ)^{-1}\|_2\|Q^T\|_2\|Qx_0 - \tilde{e}\|_F \\
&\leqslant (1-\kappa^{-2})^{\frac{k}{2}}\|(Q^TQ)^{-1}\|_2\|Q^T\|_2\|Q\|_2\|X_0 - X^*\|_F \\
&= (1-\kappa^{-2})^{\frac{k}{2}}\frac{\lambda_{\max}(Q^TQ)}{\lambda_{\min}(Q^TQ)}\|X_0 - X^*\|_F \\
&= \kappa^2(1-\kappa^{-2})^{\frac{k}{2}}\|X_0 - X^*\|_F.
\end{aligned}$$

As the limiting behaviour of $\|X_k - X^*\|_F$ depends on $(1-\kappa^{-2})^{\frac{k}{2}}$, the convergence rate for Algorithm 1 is governed by $\sqrt{1-\kappa^{-2}}$. Similarly, using (3.24), it follows that

$$\begin{aligned}
\|X_k - X^*\|_F &\leqslant (1-\kappa^{-2})^{\frac{1}{2}}\|(Q^TQ)^{-1}\|_2\|Q^T\|_2\|Qx_{k-1} - \tilde{e}\|_F \\
&\leqslant (1-\kappa^{-2})^{\frac{1}{2}}\|(Q^TQ)^{-1}\|_2\|Q^T\|_2\|Q\|_2\|X_{k-1} - X^*\|_F \\
&= \kappa^2(1-\kappa^{-2})^{\frac{1}{2}}\|X_{k-1} - X^*\|_F,
\end{aligned}$$

and hence (3.28) is reached. □

Theorem 3.8. Suppose Q is of full column-rank and Eq. (3.1) has a unique least-squares solution. We have the error estimation $\|X_k - X^*\|_F^2$ compared to the preceding iteration and the initial iteration of Algorithm 1 are provided by:

$$\|X_k - X^*\|_F^2 \leqslant \alpha\kappa^4\|X_{k-1} - X^*\|_F^2 + 2\delta\lambda_{\min}^{-1}, \tag{3.29}$$

$$\|X_k - X^*\|_F^2 \leqslant \alpha^k\kappa^4\|X_0 - X^*\|_F^2 + 2\delta\kappa^2(1-\alpha^k)\lambda_{\min}^{-1}. \tag{3.30}$$

Proof. The proof is similar to that of Theorem 3.7 and carried out by (3.22) and (3.23). We, therefore, omit the proof. □

Chapter 4

Numerical simulations

In this chapter, we illustrate the effectiveness and capability of our proposed algorithm. We make the comparison reports as tables and figures of our algorithm (denoted by *TauOpt*) with the famous algorithms we have presented in Chapter 2. The tables in each example provide the (relative) errors at the terminal iteration and the computational time, CT (in seconds). The CT is measured by *tic* and *toc* in MATLAB. In addition, the figure in each example displays the error plot graphic which combine all algorithms from ours and others. All iterations have been carried out by MATLAB r2020b, Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz 2.60 GHz, RAM 8.00 GB. PC environment. At step kth of the iteration we consider the following error:

$$\|X(k) - X^*\|_F$$

where $X(k)$ is the kth approximated solution of the corresponding system. In case of unawareness of the exact solution, we will consider the relative error $\|R_k\|_F$ instead.

Example 4.1. Suppose that the generalized Sylvester-transpose matrix equations are $\sum_{t=1}^{3} A_t X B_t + \sum_{s=1}^{2} C_s X^T D_s = E$ with

$$A_1 = \begin{bmatrix} 0.491 & 0.064 \\ 0.071 & 0.436 \\ 0.887 & 0.826 \end{bmatrix}, A_2 = \begin{bmatrix} 0.394 & 0.886 \\ 0.613 & 0.931 \\ 0.818 & 0.190 \end{bmatrix}, A_3 = \begin{bmatrix} 0.258 & 0.503 \\ 0.897 & 0.612 \\ 0.593 & 0.819 \end{bmatrix},$$

$$B_1 = \begin{bmatrix} 0.531 & 0.453 & 0.966 \\ 0.202 & 0.427 & 0.620 \end{bmatrix}, B_2 = \begin{bmatrix} 0.695 & 0.346 & 0.556 \\ 0.720 & 0.517 & 0.156 \end{bmatrix}, B_3 = \begin{bmatrix} 0.562 & 0.426 & 0.731 \\ 0.694 & 0.836 & 0.360 \end{bmatrix},$$

$$C_1 = \begin{bmatrix} 0.454 & 0.734 \\ 0.386 & 0.430 \\ 0.775 & 0.693 \end{bmatrix}, C_2 = \begin{bmatrix} 0.945 & 0.109 \\ 0.784 & 0.389 \\ 0.705 & 0.590 \end{bmatrix}, D_1 = \begin{bmatrix} 0.459 & 0.228 & 0.015 \\ 0.050 & 0.834 & 0.863 \end{bmatrix},$$

$$D_2 = \begin{bmatrix} 0.078 & 0.500 & 0.571 \\ 0.669 & 0.218 & 0.122 \end{bmatrix} \text{ and } E = \begin{bmatrix} 0.671 & 0.056 & 0.435 \\ 0.599 & 0.152 & 0.832 \\ 0.056 & 0.019 & 0.617 \end{bmatrix}.$$

We find that $4 = \text{rank}\, Q \neq \text{rank}[Q\ \text{Vec}(E)] = 5$, i.e., the matrix equation does not have an exact solution. However, the size of Q is 9×4, i.e., Q is of full-column rank. Hence, according to Theorem 3.4, Algorithm 1 will converge to the least-squares solution in which the least-squares error (3.5) is equal to 0.0231. We choose an initial matrix $X_0 = \text{zero}(2)$, where $\text{zero}(n)$ is an $n \times n$ zero matrix. Algorithm 1 is compared with the list of algorithms as follows:

- the gradient based iterative (GI, Method 2.17),

- the least-squares based iterative (LSI, Method 2.18),
- the direct method Eq. (3.4).

In this case, we consider the error $\|X^* - X_k\|_F$ where X^* is the least-squares solution. Figure 4.1 displays the error plot. Table 4.1 shows the errors and CTs for TauOpt, GI, LSI and the direct method which can be implied that our algorithm gives the fastest convergence to the least-squares solution as well as the fastest time.

Table 4.1*:* Errors and CTs at 100th iteration for Example 4.1

Method	Error	CT
TauOpt	7.3178e-04	0.0015
GI	0.6164	0.0025
LSI	0.8453	0.0076
direct		0.0020

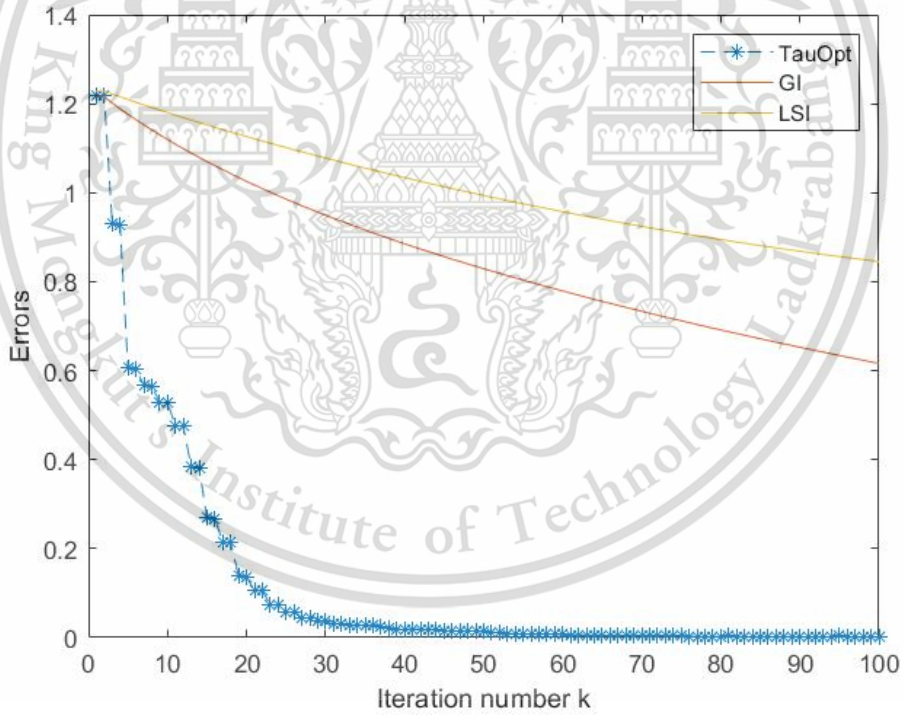

Figure 4.1*:* Errors for Example 4.1

Example 4.2. We consider the equation $AXB = E$ with

$$A = \begin{bmatrix} 1 & -1 & 2 & 3 & 1 & -3 & 3 & 2 \\ 2 & 3 & -2 & 2 & 2 & 1 & 3 & 3 \\ 3 & 1 & 1 & -1 & -3 & -2 & -1 & 3 \end{bmatrix}^T,$$

$$B = \begin{bmatrix} 1 & 2 & -5 & 9 & 7 & 5 & 1 & 0 & -6 & 3 \\ 2 & -7 & 8 & 3 & 0 & 1 & 2 & 3 & 5 & -6 \\ 6 & -5 & 2 & 1 & 0 & 3 & -9 & 8 & 7 & 6 \end{bmatrix}.$$

And

$$E = \begin{bmatrix} 106 & -155 & 104 & 215 & 112 & 125 & -14 & 127 & 65 & -30 \\ 186 & -168 & 54 & 222 & 126 & 174 & -174 & 228 & 120 & 126 \\ -143 & 83 & 15 & -98 & -63 & -112 & 202 & -178 & -97 & -177 \\ -72 & -23 & 70 & 173 & 98 & 27 & 258 & -107 & -123 & -228 \\ -32 & 7 & 2 & 107 & 70 & 29 & 118 & -53 & -85 & -84 \\ 123 & -25 & -67 & -8 & 7 & 66 & -252 & 128 & 99 & 237 \\ -32 & -66 & 88 & 246 & 140 & 74 & 238 & -60 & -106 & -216 \\ 103 & -186 & 139 & 303 & 161 & 155 & 58 & 117 & 38 & -99 \end{bmatrix}$$

We choose the initial matrix $X_0 = 10^{-6}\,\text{ones}(3,3)$ where $\text{ones}(m,n)$ denotes the m -by n matrix with contains 1 at every position. After running Algorithm 1 (with the optimal step size τ described by Eq. (3.9)), the numerical solutions converge to the exact solution

$$X^* = \begin{bmatrix} 1 & 5 & -9 \\ 6 & 5 & 4 \\ 1 & 2 & 3 \end{bmatrix}.$$

In this example, we compare the Algorithm 1 with the list of algorithms as follows:

- the gradient based iterative (GI, Proposition 2.1),
- the least-squares based iterative (LS, Proposition 2.2),
- the direct method Eq. (3.4).

All reports are presented after running 100 iterations. Table 4.2 shows the errors at the final iteration as well as the computational time. Figure 4.2 displays the error plot. Table 4.2 implies that our algorithm takes significantly less computational time than the direct method. For comparison to other two algorithms, it seems that our algorithm takes a little more time but both Table 4.2 and Figure 4.2 indicate that our algorithm obtains a highly satisfactory approximated solution.

Table 4.2: Error and CT for Example 4.2

Method	Error	CT
TauOpt	7.2231e-14	0.0057
GI	12.1879	0.0004
LS	13.8387	0.0027
direct		0.3051

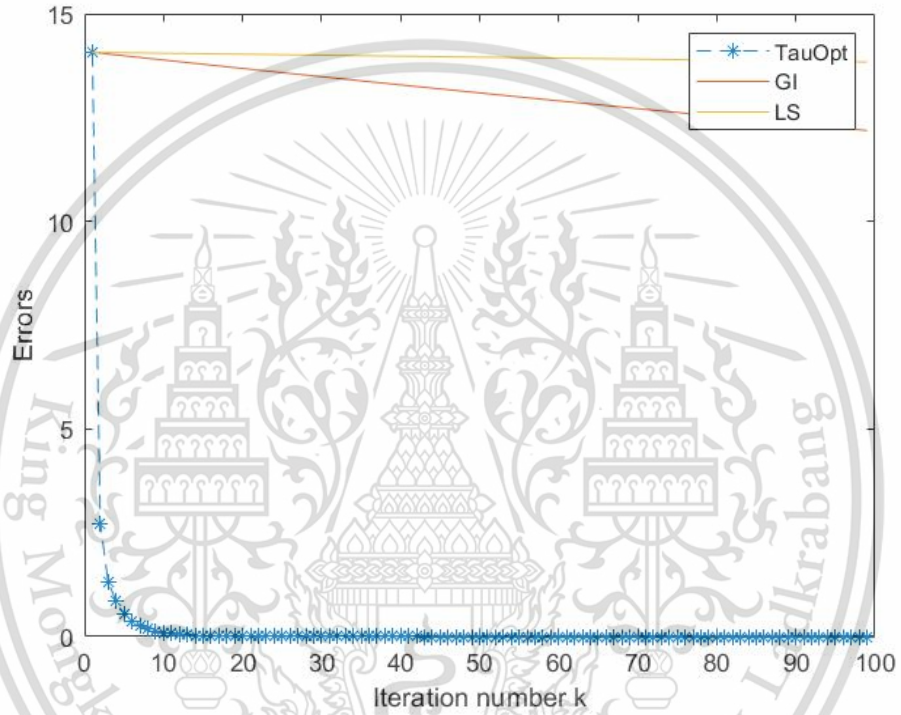

Figure 4.2: Errors for Example 4.2

Example 4.3. We consider the Lyapunov equation Eq. (3.10) with medium-scale coefficient matrices

$$A = -\,\mathrm{triu}(\mathrm{rand}(n), 1) + \mathrm{diag}(8 - \mathrm{diag}(\mathrm{rand}(n))), \qquad C = \mathrm{rand}(n).$$

Choose $n = 20$ and set $X_0 = \mathrm{zero}(20)$. In this example, we run Algorithm 1 (with the step size τ described by Eq. (3.11)) for 50 iterations. We compare our algorithm with the list of algorithms as follows:

- the gradient based iterative (GI, Method 2.3),
- the relaxed gradient based iterative (RGI, Method 2.4),
- the modified gradient based iterative (MGI, Method 2.5),
- the Jacobi-gradient based iterative (JGI, Method 2.6),

- the accelerated Jacobi-gradient based iterative (AJGI, Method 2.7),
- the 1st version modified least-squares iterative (LSIA1, Method 2.11),
- the 2nd version modified least-squares iterative (LSIA2, Method 2.12),
- the accelerated gradient based iterative (AGBI, Method 2.19),
- the direct method Eq. (3.4).

We report the result in Figure 4.3 and Table 4.3. In conclusion, our algorithm takes a slightly more CT than some other algorithms but still outperforms most of the algorithms in performance of convergence.

Table 4.3*:* Relative errors and CTs at 50th iteration for Example 4.3

Method	Relative error	CT
TauOpt	1.0525e-05	8.3420e-04
GI	9.2192	0.0016
RGI	7.2664	0.0010
MGI	2.9296	9.8290e-04
AGBI	10.1582	0.0014
JGI	7.4067	5.2160e-04
AJGI	0.7471	6.9470e-04
LSIA 1	5.2781	0.0024
LSIA 2	5.2853	0.0014
direct		0.4636

Example 4.4. Suppose that the Sylvester equation (3.12) has large-scales tridiagonal coefficient matrices, i.e.,

$$A = \text{tridiag}(10, -2, 9), B = \text{tridiag}(-1, 2, -5), \text{ and } C = \text{tridiag}(-45, 13, -20)$$

where $A, B, C \in M_{100}$. We choose an initial matrix $X_0 = \text{zero}(100)$. Here, the symmetric exact solution is given by $X^* = \text{tridiag}(1, -5, 1)$ so that AGBI method can be applicable. In this example, Algorithm 1 (with the step size τ described by Eq. (3.13)) is compared with the list of algorithms as follows:

- the gradient based iterative (GI, Method 2.3),
- the relaxed gradient based iterative (RGI, Method 2.4),
- the modified gradient based iterative (MGI, Method 2.5),
- the Jacobi-gradient based iterative (JGI, Method 2.6),

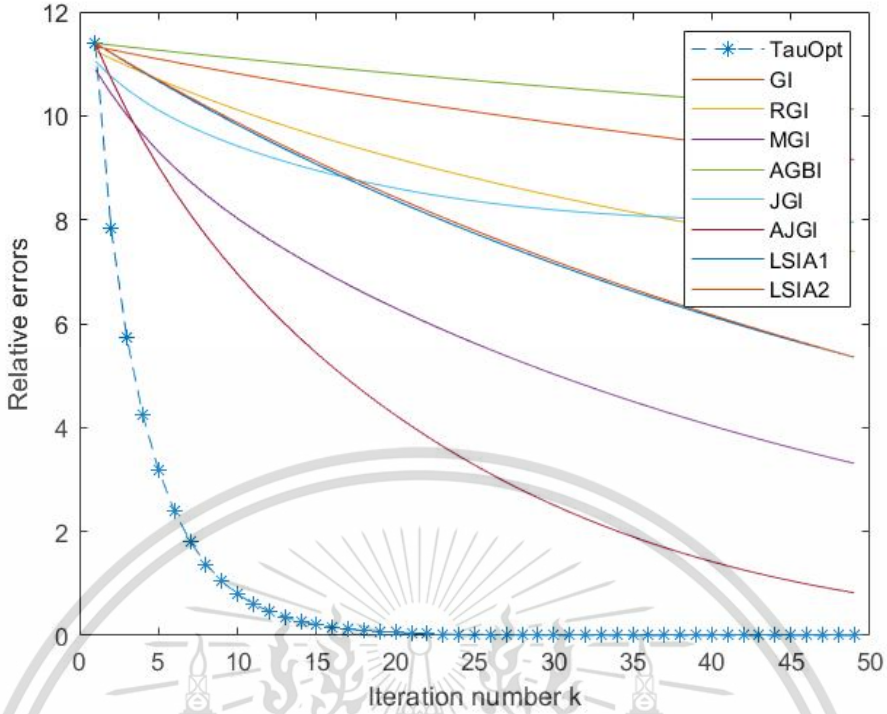

Figure 4.3: Relative errors for Example 4.3

- the accelerated Jacobi-gradient based iterative (AJGI, Method 2.7),
- the gradient iterative method with optimal convergence factor (GIO, Method 2.15),
- the gradient based iterative (GI θ_{opt}, Method 2.16),
- the accelerated gradient based iterative (AGBI, Method 2.19),
- the direct method Eq. (3.4).

The results of running every algorithm for 100 iterations are shown in Table 4.4 and Figure 4.4. Although Table 4.4 tells us that our algorithm takes a slightly more time than some other algorithms, Figure 4.4 illustrates that our algorithm reaches the fastest convergence. Note that the computational time of GIO and GI θ_{opt} algorithms is extremely too much compared to the others. It is because according to Method 2.15 and Method 2.16, both methods need to evaluate the maximum and minimum eigenvalues of the matrix $P^TP \in M_{10000}$ which causes a large amount of time.

Table 4.4: Relative errors and CTs at 100th iteration for Example 4.4

Method	Relative error	CT
TauOpt	0.1457	0.0681
GI	183.4122	0.0731
RGI	33.0116	0.0661
MGI	115.7206	0.0640
AGBI	57.8981	0.0839
JGI	515.9767	0.0385
AJGI	469.0704	0.0547
GIO	1.5718	148.1816
GI θ_{opt}	0.1646	120.2540
direct		5.4606e+03

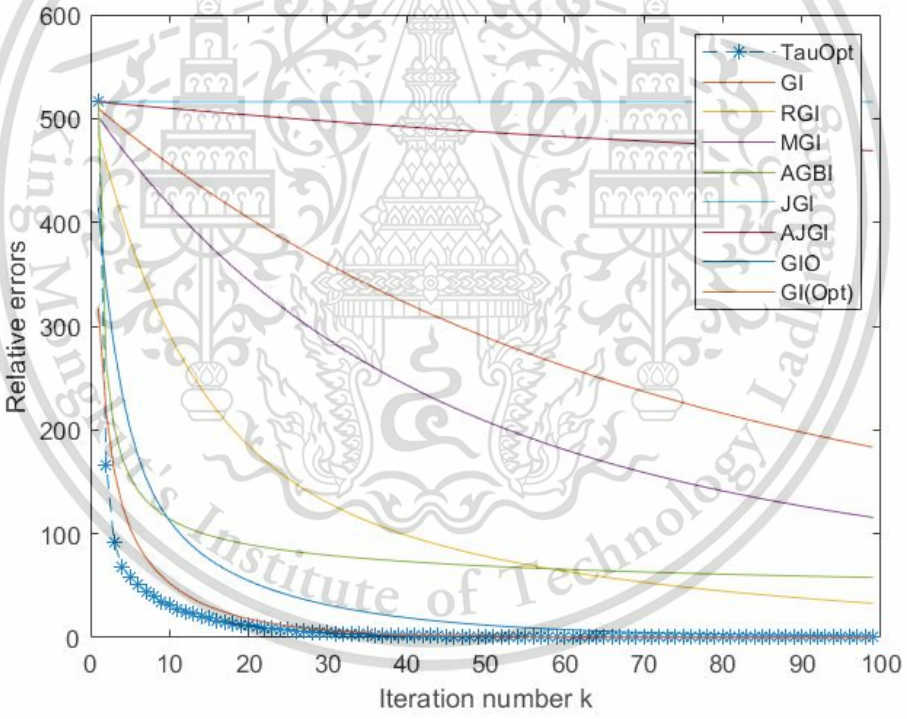

Figure 4.4: Relative errors for Example 4.4

Chapter 5

Applications to heat and Poisson's equations

In this chapter, we apply our algorithm to the discretization of a differential equation. Indeed, there are many kinds of the differential equations. Since we cannot cover all of them, we limit our study to two interesting model problems, i.e., the one-dimensional heat equation and the two-dimensional Poisson's equation. First, we apply finite difference methods to the differential equations and transform them into the linear system. Then, we make two new gradient-descent algorithms for each of the equations specifically. Furthermore, the numerical examples for each equation are also provided in their own sections.

5.1 An application to a discretization of one-dimensional heat equation

In this section, we apply our proposed algorithm to a discretization of one-dimensional heat equation in the following form

$$\frac{\partial u}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2} \quad \text{on } [0, \beta_t] \times [\alpha_x, \beta_x] \tag{5.1}$$

subject to the boundary conditions

$$u(\alpha_x, t) = g_l, u(\beta_x, t) = g_r, u(x, 0) = g_d$$

where g_l, g_r, g_d are given functions.

5.1.1 Discretization of the heat equation

We make discretization at the grid points in the rectangle which are at (x_i, t_j) with $x_i = \alpha_x + ih_x$ and $t_j = jh_t$ where

$$h_x = \frac{\beta_x - \alpha_x}{N_x + 1} \text{ and } h_t = \frac{\beta_t}{N_t}. \tag{5.2}$$

We denote $u_{ij} = u(x_i, t_j)$. By Forward Time Central Space (FTCS) method, we obtain

$$\frac{\partial u}{\partial t} = \frac{u_{i,j+1} - u_{ij}}{h_t} = c^2 \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h_x^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

or equivalently

$$u_{i,j+1} = F\left(u_{i-1,j} + u_{i+1,j}\right) + (1 - 2F)u_{ij},$$

where $F = h_t c^2 / h_x^2$ for $1 \leqslant i \leqslant N_x, 1 \leqslant j \leqslant N_t$. We transform Eq. (5.1) into a linear system of $N_x N_t$ equations in $N_x N_t$ unknowns $u_{11}, \ldots, u_{N_x N_t}$:

$$T_H \, \text{Vec}(U) = V, \tag{5.3}$$

where $U = [u_{ij}]$, $T_H \in M_{N_x N_t}$ has N_t-by-N_t blocks of the form I_{N_x} on its diagonal and tridiag$(-F, -(1-2F), -F)$ under its diagonal. Here is the example of T_H where $N_t = 3$ and $N_x = 2$

$$T_H = \left[\begin{array}{cc|cc|cc} 1 & & & & & \\ & 1 & & & & \\ \hline -(1-2F) & -F & 1 & & & \\ -F & -(1-2F) & & 1 & & \\ \hline & & -(1-2F) & -F & 1 & \\ & & -F & -(1-2F) & & 1 \end{array}\right].$$

The vector V is partitioned in N_x periods as $\begin{bmatrix} V_1^T & V_2^T & \dots & V_{N_t}^T \end{bmatrix}^T$ where

$$V_1 = \begin{bmatrix} Fg_d(\alpha_x, 0) + (1-2F)g_d(x_1, 0) + Fg_d(x_2, 0) \\ Fg_d(x_1, 0) + (1-2F)g_d(x_2, 0) + Fg_d(x_3, 0) \\ \vdots \\ Fg_d(x_{N_t-2}, 0) + (1-2F)g_d(x_{N_t-1}, 0) + Fg_d(N_t, 0) \end{bmatrix} \text{ and } V_i = \begin{bmatrix} Fg_l(\alpha_x, i-1) \\ 0 \\ \vdots \\ 0 \\ Fg_r(\beta_x, i-1) \end{bmatrix}$$

for $i = 2, \dots, N_t$.

Consequently, Eq. (5.3) is formed $AXB = E$ where $A = T_H$, $X = \text{Vec}(U)$, $B = I$ and $E = V$. According to Algorithm 1 (with the optimal step size τ described in Subsect. 3.2.4), we obtain an algorithm for Eq. (5.3) as follows:

Algorithm 2: The gradient-descent iterative algorithm for Eq. (5.1)

T_H, V, Vec(U_0);
for $k = 0, \dots, n$ **do**
  $R_k = V - T_H \text{Vec}(U_k)$;
  $W_k = T_H^T R_k$;
  $\tau_{k+1} = \|W_k\|_F^2 / \|T_H W_k\|_F^2$;
  $\text{Vec}(U_{k+1}) = \text{Vec}(U_k) + \tau_{k+1} W_k$
end

5.1.2 Numerical simulation for the heat equation

To obtain the approximate solutions, we need to partition the rectangular domain. The accuracy of the solutions depends on the size of the grid partitioned. The better accuracy must be from the finer grid system and it causes the sizes of the associated matrix T_H to be larger.

Example 5.1. Consider the heat equation (5.1) on $\{(x,t) : 0 < x < 1,\ t > 0\}$ with the boundary conditions and the initial condition:

$$u(0,t) \ = \ u(1,t) \ = \ 0 \text{ and } u(x,0) = \sin \pi x.$$

Given $c = 1$, $N_x = 4$, $h_t = 0.01$. We have $h_x = 0.2$ and $F = 0.25$. In this case, we consider $N_t = 10$ so the size of the matrix T_H is of 40×40. We run the Algorithm 2 with the initial vector $\text{Vec}(U_0) = 10^{-6}[1 \ \dots \ 1]^T$ and the approximate solutions converge to the exact solution

$$u^*(x,t) \ = \ e^{-\pi^2 t} \sin(\pi x).$$

This example we compare our algorithm to the following algorithms:

- the gradient based iterative algorithm (GI, Proposition 2.1),
- the least-squares based iterative algorithm (LSI, Proposition 2.2),
- the relaxed gradient based iterative algorithm (RGI, Method 2.4),
- the modified gradient based iterative algorithm (MGI, Method 2.5),
- the Jacobi-gradient based iterative algorithm (JGI, Method 2.6),
- the accelerated Jacobi-gradient based iterative algorithm (AJGI, Method 2.7).

The results after running 500 iterations are shown in Figure 5.1, Figure 5.2, Table 5.1, and Table 5.2.

Table 5.1*:* Comparison of approximate and analytical results for Example 5.1

	$u(x,t)$							
	$x = 0.2$		$x = 0.4$		$x = 0.6$		$x = 0.8$	
t	approx.	exact	approx.	exact	approx.	exact	approx.	exact
0.01	0.5317	0.5325	0.8602	0.8617	0.8602	0.8617	0.5317	0.5325
0.02	0.4809	0.4825	0.7781	0.7807	0.7781	0.7807	0.4809	0.4825
0.03	0.4350	0.4371	0.7038	0.7073	0.7038	0.7073	0.4350	0.4371
0.04	0.3934	0.3961	0.6366	0.6408	0.6366	0.6408	0.3934	0.3961
0.05	0.3559	0.3588	0.5758	0.5806	0.5758	0.5806	0.3559	0.3588
0.06	0.3219	0.3251	0.5208	0.5261	0.5208	0.5261	0.3219	0.3251
0.07	0.2911	0.2946	0.4711	0.4766	0.4711	0.4766	0.2911	0.2946
0.08	0.2633	0.2669	0.4261	0.4318	0.4261	0.4318	0.2633	0.2669
0.09	0.2382	0.2418	0.3854	0.3912	0.3854	0.3912	0.2382	0.2418
0.10	0.2154	0.2191	0.3486	0.3545	0.3486	0.3545	0.2151	0.2191

Table 5.2: Error and CT for Example 5.1

Method	Error	CT
TauOpt	0.0445	0.0368
GI	2.0528	0.0072
RGI	2.5198	0.0076
MGI	1.6413	0.0078
LSI	3.0821	0.1155
JGI	2.1186	0.0072
AJGI	2.6405	0.0090

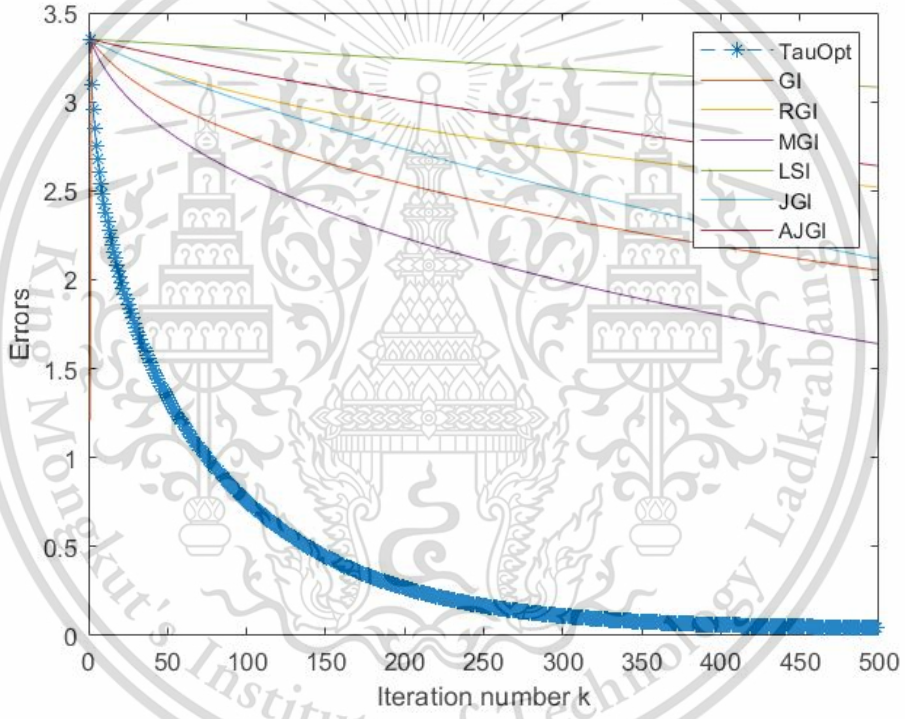

Figure 5.1: Errors for Example 5.1

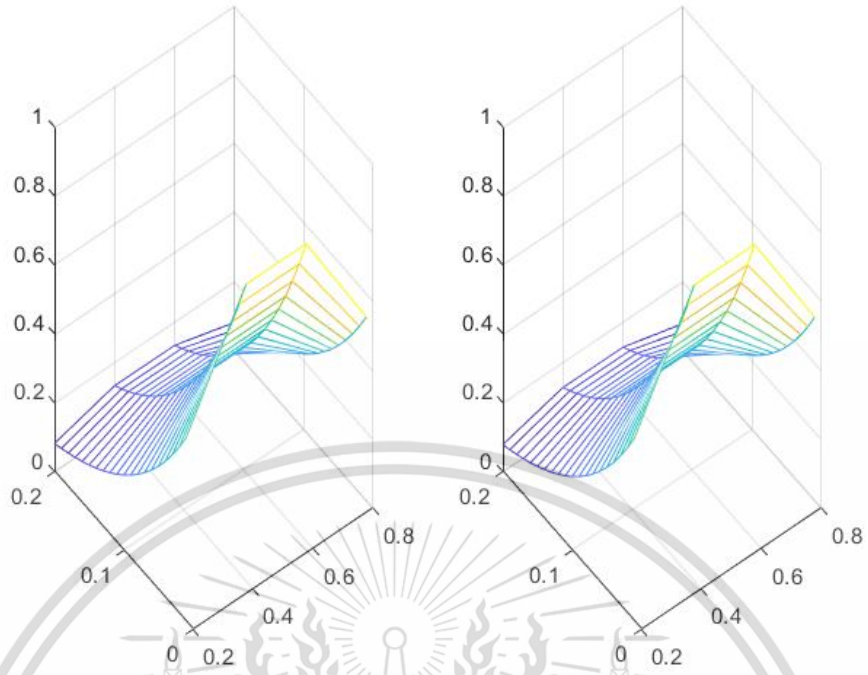

Figure 5.2*:* The 3D-plot of the analytical solution (left) and the approximate solution (right) for Example 5.1

According to Table 5.1, it is seen that the approximate solutions are close to the exact solutions with two decimals accuracy. However, these results can be more accurate since the error occurs from two reasons. Firstly, the error comes from the discretization. To reduce this error, we can adjust the size of the step size h_x and h_t to be smaller. Secondly, the error comes from the algorithm. To reduce this error, we can set the algorithm to run with more iteration numbers. All of these will improve the accuracy of the approximate solutions.

5.2 An application to a discretization of two-dimensional Poisson's equation

In this section, we give an application of the proposed algorithm to a discretization of two-dimensional Poisson's equation in the following form:

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y) \qquad \text{on } [\alpha_x, \beta_x] \times [\alpha_y, \beta_y] \tag{5.4}$$

with the boundary conditions

$$u(x, \beta_y) = g_u, u(x, \alpha_y) = g_d,$$
$$u(\alpha_x, y) = g_l, u(\beta_x, y) = g_r$$

where g_u, g_d, g_l, g_r are given functions. Notice that the two-dimensional Laplace's equation is a homogeneous case of the Poisson's equation when the RHS function is zero, i.e., $f(x, y) = 0$.

5.2.1 Discretization with rectangular grid

We make discretization at the grid points in the rectangle which are at (x_i, y_j) with $x_i = \alpha_x + ih_x$ and $y_j = \alpha_y + jh_y$ where

$$h_x = \frac{\beta_x - \alpha_x}{N_x + 1} \text{ and } h_y = \frac{\beta_y - \alpha_y}{N_y + 1}. \tag{5.5}$$

We denote $u_{ij} = u(x_i, y_j)$, $f_{ij} = f(x_i, y_j)$ as well as g_u, g_d, g_l, g_r . By the standard finite difference approximation, we obtain

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{ij} + u_{i,j+1}}{h_y^2} \tag{5.6}$$

or equivalently

$$h_y^2 (2u_{ij} - u_{i-1,j} - u_{i+1,j}) + h_x^2 (2u_{ij} - u_{i,j+1} - u_{i,j+1}) = -h_x^2 h_y^2 f_{ij},$$

for $1 \leqslant i \leqslant N_x, 1 \leqslant j \leqslant N_y$. Now, we can convert the differential equation (5.4) to a linear system of $N_x N_y$ equations in $N_x N_y$ unknowns $u_{11}, \ldots, u_{N_x N_y}$:

$$T_N \,\mathrm{Vec}(U) = E, \tag{5.7}$$

where

$$U = [u_{ij}],\ T_N = (h_y^2 T_1 + h_x^2 T_2),\ E = -h_x^2 h_y^2\, \mathrm{Vec}[f_{ij}] + h_x^2 (\overline{g_u} + \overline{g_d}) + h_y^2 (\overline{g_l} + \overline{g_r}).$$

Here, T_1 has N_y-by-N_y blocks of the form $\mathrm{tridiag}(-1, 2, -1)$ of N_x-by-N_x on its diagonal and T_2 also has N_y-by-N_y blocks of the form $2I_{N_x}$ on its diagonal and $-I_{N_x}$ blocks on its off-diagonals. The boundary conditions produce constant vectors $\overline{g_u}, \overline{g_d}, \overline{g_l}, \overline{g_r}$ at the RHS of Eq. (5.7) as follows:

$$\begin{aligned}
\overline{g_u} &= \left[g_{u_{x_1,\beta_y}}\ g_{u_{x_2,\beta_y}}\ \cdots\ g_{u_{x_{N_x},\beta_y}}\ 0\ \cdots\ 0\right]^T, \\
\overline{g_d} &= \left[0\ \cdots\ 0\ g_{d_{x_1,\alpha_y}}\ g_{d_{x_2,\alpha_y}}\ \cdots\ g_{d_{x_{N_x},\alpha_y}}\right]^T, \\
\overline{g_l} &= \left[g_{l_{\alpha_x,y_{N_y}}}\ 0\ \cdots\ 0\ g_{l_{\alpha_x,y_{N_y-1}}}\ 0\ \cdots\ g_{l_{\alpha_x,y_1}}\ 0\ \cdots\ 0\right]^T, \\
\overline{g_r} &= \left[0\ \cdots\ 0\ g_{r_{\beta_x,y_{N_y}}}\ 0\ \cdots\ 0\ g_{r_{\beta_x,y_{N_y-1}}}\ 0\ \cdots\ g_{r_{\beta_x,y_1}}\right]^T.
\end{aligned}$$

In case of the Laplace's equation, the matrix E in Eq. (5.7) will be reduced to

$$E = h_x^2 (\overline{g_u} + \overline{g_d}) + h_y^2 (\overline{g_l} + \overline{g_r}).$$

Consequently, Eq. (5.7) is formed $AXB = E$ where $A = T_N$, $X = \mathrm{Vec}(U)$, and $B = I$. According to Algorithm 1 (with the optimal step size τ described in Subsect. 3.2.4), we obtain an algorithm for the rectangular-grid case as follows:

Algorithm 3: The gradient-descent iterative algorithm for Eq. (5.4)

T_N, E, $\mathrm{Vec}(U_0)$;
for $k = 0, \ldots, n$ **do**
  $R_k = E - T_N \mathrm{Vec}(U_k)$;
  $W_k = T_N^T R_k$;
  $\tau_{k+1} = \|W_k\|_F^2 / \|T_N W_k\|_F^2$;
  $\mathrm{Vec}(U_{k+1}) = \mathrm{Vec}(U_k) + \tau_{k+1} W_k$
end

Since, the coefficient matrix T_N is a sparse matrix, the error norm can be described more precisely:

$$\begin{aligned}\|E - T_N \mathrm{Vec}(U_k)\|_F^2 &= \|E\|_F^2 - 2\,\mathrm{tr}(E^T T_N \mathrm{Vec}(U_k)) + \|T_N \mathrm{Vec}(U_k)\|_F^2 \\ &= \|E\|_F^2 - 2h_x^2 h_y^2 \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} h_Y^2 f_{ij}(-u_{i-1,j} + 2u_{ij} - u_{i+1,j}) \\ &\quad + h_x^2 f_{ij}(-u_{i,j+1} + 2u_{ij} - u_{i,j-1}) + \|T_N \mathrm{Vec}(U_k)\|_F^2.\end{aligned}$$

5.2.2 Discretization with square grid

Now, we consider the Poisson's equation (5.4) on the square $[\alpha, \beta] \times [\alpha, \beta]$ with the boundary condition $u = 0$ on the boundary of the square. In this case, $h := h_x = h_y$ and $N := N_x = N_y$ and hence

$$T_N = I_N \otimes T_r + T_r \otimes I_N$$

where $T_r = \mathrm{tridiag}(-1, 2, -1) \in M_N$. Thereby, (5.7) can be transformed into

$$T_N \mathrm{Vec}(U) = -h^2 \mathrm{Vec}([f_{ij}]) + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}, \tag{5.8}$$

or equivalently $T_r U + U T_r = G$ where $G = -h^2 \mathrm{Vec}([f_{ij}]) + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}$. Thus (5.8) can be solved by Algorithm 1 (with the optimal step size τ described in Subsect. 3.13) where $A, B = T_r$, $X = U$ and $E = G$.

To have the condition number of T_N, we consider the smallest and the largest eigenvalues of T_r which are given respectively by (see e.g. [38])

$$\lambda_1 = 2\left(1 - \cos\frac{\pi}{N+1}\right) \approx \left(\frac{\pi}{N+1}\right)^2, \qquad \lambda_N = 2\left(1 - \cos\frac{N\pi}{N+1}\right) \approx 4.$$

Since $T_N = I_N \otimes T_r + T_r \otimes I_N$, the eigenvalue of T_N is $\lambda_i + \lambda_j$ where $\lambda_i, \lambda_j \in \sigma(T_r)$. Thus, the condition number of T_N for large N is

$$\kappa_{T_N} = \frac{\lambda_N + \lambda_N}{\lambda_1 + \lambda_1} \approx \frac{4}{\pi^2}(N+1)^2. \tag{5.9}$$

Corollary 5.2. The discretization (5.8) of the Poisson's equation (5.4) can be solved by using Algorithm 3 in which $E = -h^2 \mathrm{Vec}[f_{ij}] + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}$ so that the approximated

solution U_k converges to the exact solution U^* for any initial vector U_0. The convergent rate of the algorithm is governed by $\sqrt{1-\kappa_{T_N}^{-2}}$, where κ_{T_N} is given by (5.9). Moreover, error estimates are given as follows:

$$\|U_k - U^*\|_F \leqslant \kappa_{T_N}(1-\kappa_{T_N}^{-2})^{\frac{1}{2}}\|U_{k-1} - U^*\|_F,$$
$$\|U_k - U^*\|_F \leqslant \kappa_{T_N}(1-\kappa_{T_N}^{-2})^{\frac{k}{2}}\|U_0 - U^*\|_F.$$

5.2.3 Numerical Simulations for the Poisson's equation

Example 5.3. We consider an application of our algorithm to the two-dimensional Poisson's equation (5.4) with

$$f(x,y) = -2\pi^2 \sin(\pi x)\sin(\pi y), \qquad 0 < x < 1,\ 0 < y < 1,$$

and the boundary condition $u = 0$ on the boundary of the rectangle. It is called *Dirichlet problem*. We choose an initial vector $\text{Vec}(U_0) = 10^{-6}[1\ \ldots\ 1]^T$. We run Algorithm 3 with the rectangular grid of 10×20 which causes the sizes of the matrix T_N to be of 200×200. The analytical solution is

$$u^*(x,y) = \sin(\pi x)\sin(\pi y).$$

In this example, we provide only the comparison of approximate solution and analytical solution in Table 5.3, and the 3D-plot of both solutions in Figure 5.3.

Table 5.3: Comparison of approximate and analytical results for Example 5.3

	$u(x,y)$							
	$x = 0.3636$		$x = 0.5454$		$x = 0.7272$		$x = 0.9090$	
y	approx.	exact	approx.	exact	approx.	exact	approx.	exact
0.1904	0.5136	0.5124	0.5588	0.5576	0.4267	0.4257	0.1587	0.1591
0.3808	0.8488	0.8468	0.9236	0.9214	0.7052	0.7035	0.2629	0.2623
0.5712	0.8889	0.8868	0.9673	0.9650	0.7386	0.7368	0.2753	0.2747
0.7616	0.6202	0.6187	0.6749	0.6732	0.5153	0.5140	0.1921	0.1916
0.9520	0.1359	0.1356	0.1479	0.1475	0.1129	0.1126	0.0423	0.0420

Example 5.4. Consider the two-dimensional Laplace's equation on $[0,1] \times [0,\pi]$ with the boundary conditions:

$$u(0,y) = \sin y,\ u(1,y) = e\sin y,\ u(x,0) = 0,\ u(x,\pi) = 0.$$

We run the Algorithm 3 with the initial vector $\text{Vec}(U_0) = [1\ \ldots\ 1]^T$. We choose two types of grid partitions: one is $h_x = 0.25$, $h_y = \pi/4$ and the another one is $h_x = 0.0625$, $h_y = \pi/32$. So the sizes of the matrix T_N are 9×9 and 465×465, respectively. The

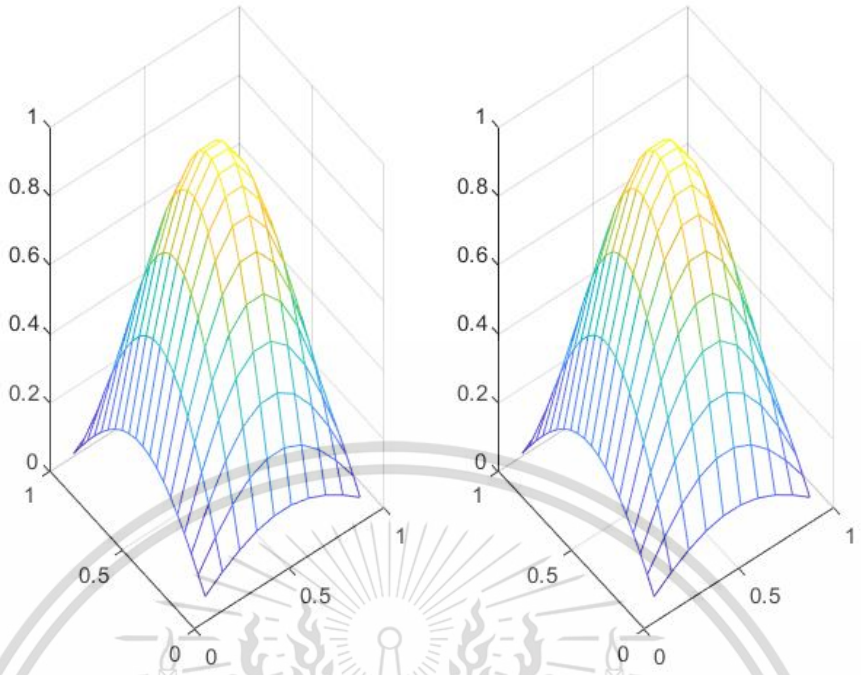

Figure 5.3: The 3D-plot of the analytical solution (left) and the approximate solution (right) for Example 5.3

comparison of approximate and analytical results is shown in the Table 5.4. The Figure 5.4 displays the 3D-plot of the approximate and the analytical results for the latter grid partition. Note that the analytical solution is

$$u^*(x, y) = e^x \sin y.$$

Table 5.4: Comparison of approximate and analytical results for Example 5.4

		$h_x = 0.25,\ h_y = \pi/4$		$h_x = 0.0625,\ h_y = \pi/32$	
	exact	approx.	error (%)	approx.	error (%)
$u(0.25, \pi/4)$	0.9079	0.9131	0.57	0.9080	0.01
$u(0.50, \pi/2)$	1.6487	1.6593	0.64	1.6489	0.01
$u(0.75, 3\pi/4)$	1.4969	1.5031	0.41	1.4971	0.01

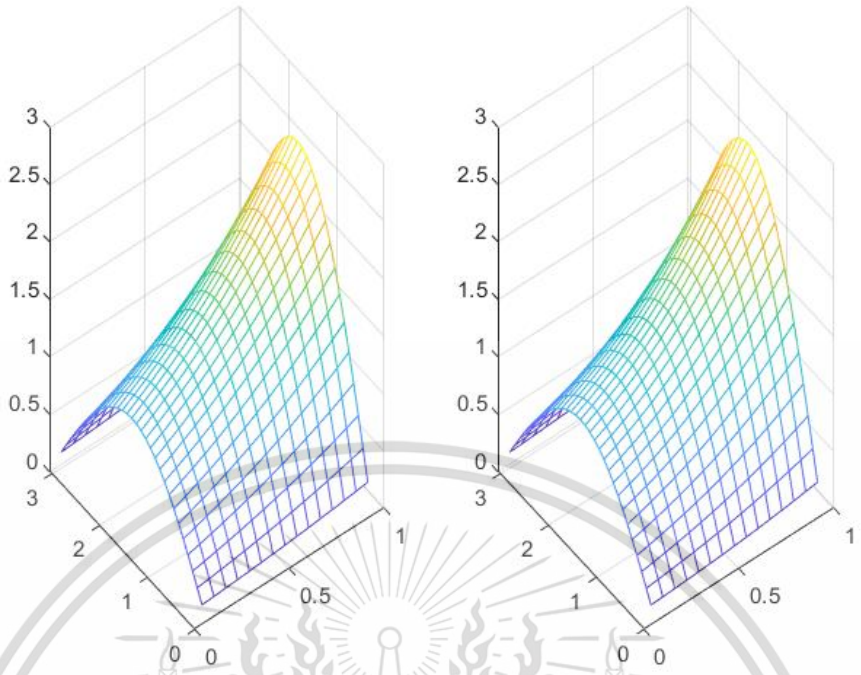

Figure 5.4*:* The 3D-plot of the analytical solution (left) and the approximate solution (right) for Example 5.4

Chapter 6

Concluding remarks and suggestion for further study

In this work, a new algorithm, i.e., the gradient-descent iterative algorithm, is well established for solving the generalized Sylvester-transpose matrix equation. This equation can be reduced to many of the linear matrix equations for examples:

- $AXB = C$,
- $AX + XA^T = B$: the Lyapunov equation,
- $AX + XB = C$: the Sylvester equation,
- $AXB + CXD = E$: the generalized Sylvester equation,
- $AXB + CX^TD = E$: the generalized Sylvester-transpose equation,
- $X + AXB = C$: the Kalman-Yakubovich equation,
- $X + AX^TB = C$: the T-Stein equation.

By utilizing knowledge in gradient-descent, and convex optimization, we hence obtain the effective algorithm with the optimal step size that can be applicable for any linear matrix equations as long as the certain matrix Q, defined by Eq. (3.3), is full column-rank. The convergence of our algorithm is divided into the following two cases, i.e.,

1) If the problem has the unique exact solution, then the approximate solutions converge to the exact solution.
2) If the problem has no solution, then $\|X\|_Q \to \|X^*\|_Q$ where X^* is the unique least-squares solution.

The asymptotic rate of convergence is governed by $\sqrt{1-\kappa^{-2}}$, where κ is the condition number of Q. As applications, our algorithm can be adapted to the discretization of the one-dimensional heat equation and the two-dimensional Poisson's equation. The numerical simulations are provided to illustrate the efficiency and capability. We also reveal the competency of our algorithm comparing to well-known and recent methods. All of the examples including the heat and the Poisson's equations verify our theoretical findings and indicate that our algorithm can be a good choice for solving a class of linear matrix equations.

As far as further study is concerned, the techniques of gradient-descent and convex optimization could be useful but require additional research to extend to nonlinear matrix equations, for examples:

- the continuous time algebraic Riccati equation

$$A^T X + XA - XBR^{-1}B^T X + Q = 0,$$

- the discrete time algebraic Riccati equation

$$X = A^T XA - (A^T XB)(R + B^T XB)^{-1}(B^T XA) + Q.$$

Furthermore, the proposed gradient-descent iterative algorithm can be adapted to the discretization of the other types of differential equations, for examples:

- the wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}),$$

- the convection-diffusion equation

$$\frac{\partial T}{\partial t} = a\frac{\partial^2 T}{\partial x^2} - \epsilon u\frac{\partial T}{\partial x} + \frac{Q}{c\rho},$$

- the Korteweg-De Vries (KdV) equation

$$\frac{\partial \phi}{\partial t} + \frac{\partial^3 \phi}{\partial x^3} - c\phi\frac{\partial \phi}{\partial x} = 0,$$

- the 2D space-time fractional diffusion equation

$${}^{C}D_t^{\beta} u(x,y,t) - {}^{RL}D_x^{\alpha_1} u(x,y,t) - {}^{RL}D_y^{\alpha_2} u(x,y,t) = f(x,y,t).$$

Here, ${}^{C}D^{\beta}$ and ${}^{RL}D^{\alpha}$ are the β-order Caputo's derivative and the α-order Riemann-Liouville derivative, respectively. The discretization of fractional derivatives is often done by finite difference schemes based on Grünwald-Letnikov type (see e.g. [45, 46]).

References

[1] Geir, E.D. and Fernando, P. 1999. **A Course in Robust Control Theory: A Convex Approach.** New York: Springer.

[2] Varga, A. 2000. "Robust pole assignment via Sylvester equation based state feedback parametrization." *Proceedings of the 2000 IEEE International Symposium on Computer-Aided Control System.* 13-18.

[3] Magnus, J.R. and Neudecker, H. 2007. **Matrix Differential Calculus with Applications in Statistics and Econometries 3rd edn.** Chichester: Wiley.

[4] Nouri, K. and Beik, S.P.A. and et al. 2020. "An iterative algorithm for robust simulation of the Sylvester matrix differential equations." *Advances in Difference Equations.* 2020(1): article no. 287.

[5] Benner, P. and Quintana-Ortí, E.S. 1999. "Solving stable generalized Lyapunov equations with the matrix sign function." *Numerical Algorithms* 20(1):75-100.

[6] Isak, J. and Bo, K. 2002. "Recursive blocked algorithms for solving triangular systems-part I: one-sided and couple Sylvester-type matrix equations." *ACM Transactions on Mathematical Software* 28(4): 392-415.

[7] Isak, J. and Bo, K. 2002. "Recursive blocked algorithms for solving triangular systems-part II: two-sided and generalized Sylvester and Lyapunov matrix equations." *ACM Transactions on Mathematical Software* 28(4): 416-435.

[8] Amer, K. and Asghar, K. and Faezeh, T. 2006. "A new version of successive approximations method for solving Sylvester matrix equations." *Applied Mathematics and Computation* 186(1): 638-645.

[9] Li, Y.-Q. 2005. "Implicity restarted global FOM and GMRES for nonsymmetric matrix equations and Sylvester equations." *Applied Mathematics and Computation* 167(2): 1004-1025.

[10] Bai, Z. 2011. "On Hermitian and skew-Hermitian splitting iteration methods for continuous Sylvester equation." *Journal of Computational Mathematics* 29(2): 185-198.

[11] Dehghan, M. and Shirilord, A. 2019. "A generalized modified Hermitian and skew-Hermitian splitting (GMHSS) method for solving complex Sylvester matrix equation." *Applied Mathematics and Computation* 348: 632-651.

[12] Hajarian, M. 2016. "Generalized conjugate direction algorithm for solving the general coupled matrix equations over symmetric matrices." *Numerical Algorithms* 73(3): 591-609.

[13] Hajarian, M. 2016. "Extending the CGLS algorithm for least squares solutions of the generalized Sylvester-transpose matrix equations." *Journal of the Franklin Institute* 353(5): 1168-1185.

[14] Dehghan, M. and Mohammadi-Arani, R. 2017. "Generalized product-type methods based on Bi-conjugate gradient (GPBiCG) for solving shifted linear systems." *Computational and Applied Mathematics* 36(4): 1591-1606.

[15] Dehghan, M. and Hajarian, M. 2008. "An iterative algorithm for the reflexive solutions of the generalized coupled Sylvester matrix equations and its optimal approximation." *Applied Mathematics and Computation* 202(2): 571-588.

[16] Dehghan, M. and Hajarian, M. 2010. "An iterative method for solving the generalized coupled Sylvester matrix equations over generalized bisymmetric matrices." *Applied Mathematical Modelling* 34(3): 639-654.

[17] Dehghan, M. and Shirilord, A. 2019. "The double-step scale splitting method for solving complex Sylvester matrix equation." *Computational and Applied Mathematics* 38: article no. 146.

[18] Dehghan, M. and Shirilord, A. 2019. "Solving complex Sylvester matrix equation by accelerated double-step scale splitting (ADSS) method." *Engineering with Computers* 37: 489-508.

[19] Ding, F. and Chen, T. 2005. "Gradient based iterative algorithms for solving a class of matrix equations." *IEEE Transactions on Automatic Control.* 50(8): 1216-1221.

[20] Ding, F. Liu, P.X. and Ding, J. 2008. "Iterative solutions of the generalized Sylvester matrix equations by using the hierarchical identification principle." *Applied Mathematics and Computation.* 197: 41-50.

[21] Xie, L. Ding, J. and Ding, F. 2009. "Gradient based iterative solutions for general linear matrix equations." *Computers and Mathematics with Applications.* 58(7): 1441-1448.

[22] Sun, M. Wang, Y. and Liu, J. 2019. "Two modified Least-squares iterative algorithms for the Lyapunov matrix equations." *Advances in Difference Equations.* article no. 305.

[23] Niu, Q. Wang, X. and Lu, L.-Z. 2011. "A relaxed gradient based algorithm for solving Sylvester equations." *Asian Journal of Control.* 13(3): 461-464.

[24] Wang, X. Dai, L. and Liao, D. 2012. "A modified gradient based algorithm for solving Sylvester equations." *Applied Mathematics and Computation.* 218: 5620-5628.

[25] Tian, Z. Tian, M. and et al. 2017. "An Accelerated Jacobi-gradient Based Iterative Algorithm for solving Sylvester Matrix Equations." *Filomat.* 31(8): 2381-2390.

[26] Xie, Y.-J. and Ma, C.-F. 2016. "The accelerated gradient based iterative algorithm for solving a class of generalized Sylvester-transpose matrix equation." *Applied Mathematics and Computation.* 273: 1257-1269.

[27] Ding, F. and Chen, T. 2005. "Hierarchical gradient-based identification of multivariable discrete-time systems." *Automatica* 41(2): 315-325.

[28] Ding, F. and Chen, T. 2005. "Hierarchical least squares identification methods for multivariable systems." *IEEE Transactions on Automatic Control* 50(3): 397-402.

[29] Wu, A. and Duan, G. and Zhou, B. 2008. "Solution to generalized Sylvester matrix equations." *IEEE Transactions on Automatic Control* 53(3): 811-815.

[30] Xie, L. and Liu, Y. and Yang, H. 2010. "Gradient based and least squares based iterative algorithms for matrix equations $AXB+CX^TD=F$." *Applied Mathematics and Computation.* 217(5): 2191-2199.

[31] Zhang, X. and Sheng, X. 2017. "The relaxed gradient based iterative algorithm for the symmetric (skew symmetric) solution of the Sylvester equation $AX+XB=C$." *Mathematical Problems in Engineering* 2017: 1-8.

[32] Kittisopaporn, A. and Chansangiam, P. 2020. "The steepest descent of gradient-based iterative method for solving rectangular linear systems with an application to Poisson's equation." *Advances in Difference Equations* 2020(1): article no. 259.

[33] Boonruangkan, N. and Chansangiam, P. 2020. "Gradient iterative method with optimal convergent factor for solving a generalized Sylvester matrix equation with applications to diffusion equations." *Symmetry* 12(10): article no. 1732.

[34] Kittisopaporn, A. and Chansangiam, P. and Lewkeeratiyutkul, W. 2021. "Convergence analysis of gradient-based iterative algorithms for a class of rectangular Sylvester matrix equations based on Banach contraction principle." *Advances in Difference Equations* 2021(1): article no. 17.

[35] Ding, F. and Zhang, X. and Xu, L. 2019. "The innovation algorithms for multivariable state-space models." *International Journal of Adaptive Control and Signal Processing* 33(10): 1601-1618.

[36] Ding, F. and Lv, L. and Pan, J. and et al. 2020. "Two-stage gradient-based iterative estimation methods for controlled autoregressive systems using the measurement data." *International Journal of Control, Automation and Systems* 18: 886-896.

[37] Ding, F. and Xu, L. and Meng, D. and et al. 2020. ”Gradient estimation algorithms for the parameter identification of bilinear systems using the auxiliary model.” *Journal of Computational and Applied Mathematics* 369: 112575.

[38] James, W.D. 1997. **Applied Numerical Linear Algebra.** Philadelphia: Society for Industrial and Applied Mathematics.

[39] Sammy, K.N. 2009. ”Heat Equations and Their Applications (One and Two Dimension Heat Equations)” M.Sc. project, University of Nairobi.

[40] Slavko, V. and Petar, S. 2006. ”2D BEM analysis of power cables thermal field.” *International Journal for Engineering Modelling* 19(1-4): 87-94.

[41] Yildirim, S. 2008. ”Exact and numerical solutions of Poisson equation for electrostatic potential problems.” *Mathematical Problem in Engineering* 2008: 578723.

[42] Horn, R.A. and Johnson, C.R. 1990. **Matrix Analysis 2nd edition.** New York: Cambridge University Press.

[43] Stephen, P.B. and Lieven, V. 2004. **Convex Optimization.** Cambridge: Cambridge University Press.

[44] Peter, J.O. and Chehrzad, S. 2018. **Applied Linear Algebra.** Switzerland: Springer.

[45] Meerschaert, M. and Tadjeran, C. 2004. ”Finite difference approximations for fractional advection-dispersion flow equations.” *Journal of Computational and Applied Mathematics* 172: 65-77.

[46] Scherer, R. and Kalla, L. and Tang, Y.F. and Huang, J.F. 2011. ”The Grunwald-Letnikov method for fractional differential equations.” *Computers and Mathematics with Applications* 62: 813-823.

Appendices

Appendix A

The research paper 1

Kittisopaporn and Chansangiam *Advances in Difference Equations* (2020) 2020:324
https://doi.org/10.1186/s13662-020-02785-9

Advances in Difference Equations
a SpringerOpen Journal

RESEARCH Open Access

Gradient-descent iterative algorithm for solving a class of linear matrix equations with applications to heat and Poisson equations

Adisorn Kittisopaporn[1] and Pattrawut Chansangiam[1]*

*Correspondence:
pattrawut.ch@kmitl.ac.th
[1]Department of Mathematics, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, 10520, Bangkok, Thailand

Abstract

In this paper, we introduce a new iterative algorithm for solving a generalized Sylvester matrix equation of the form $\sum_{t=1}^{p} A_t X B_t = C$ which includes a class of linear matrix equations. The objective of the algorithm is to minimize an error at each iteration by the idea of gradient-descent. We show that the proposed algorithm is widely applied to any problems with any initial matrices as long as such problem has a unique solution. The convergence rate and error estimates are given in terms of the condition number of the associated iteration matrix. Furthermore, we apply the proposed algorithm to sparse systems arising from discretizations of the one-dimensional heat equation and the two-dimensional Poisson's equation. Numerical simulations illustrate the capability and effectiveness of the proposed algorithm comparing to well-known methods and recent methods.

MSC: 15A60; 15A69; 26B25; 31A30; 65F45; 65F50

Keywords: Generalized Sylvester matrix equation; Gradient descent; Iterative method; Matrix norms and conditioning; Heat equation; Poisson's equation

1 Introduction

Linear matrix equations have played a crucial role in control theory and differential equations; see, e.g., [1–4]. There was much attention given to the following matrix equations: the equation $AXB = C$, the Sylvester equation $AX + XB = C$, the Kalman–Yakubovich equation $AXB + X = C$, and, more generally, the equation $AXB + CXD = F$. Using the notions of the matrix Kronecker product and the vector operator, we can obtain their exact solutions. However, matrices with high dimensions (e.g., A, B of size $10^2 \times 10^2$) cause their Kronecker product dimension to be very high ($10^4 \times 10^4$, in that case). The dimension problem leads to a computational difficulty due to exceeding computer memory when computing an inverse of the large matrix.

In practical applications, we solve the linear matrix equations of large dimensions by effective iterative methods. There are several ideas to formulate an iterative procedure, namely, one can use matrix sign function [5], block recursion [6, 7], Krylov subspace [8, 9], Hermitian and skew-Hermitian splitting [10, 11], and other related research works; see, e.g., [12–15]. In the recent decade, the ideas of gradients, hierarchical identification and

minimization of associated norm-error functions have encouraged and brought about many researches; see, e.g., [16–28]. Such iterative schemes turn out to have wide applications in many engineering problems, especially in systems identification for parameter estimation; see, e.g., [29–31].

In 2005, Ding and Chen applied the hierarchical identification principle to develop the gradient-based iterative (GI) algorithms for solving the equation $\sum_{j=1}^{p} A_j X B_j = C$, which includes the Sylvester equation, as follows.

Proposition 1.1 ([32]) *If the matrix equation $\sum_{j=1}^{p} A_j X B_j = C$ has a unique solution X, then the iterative solution $X(k)$ obtained from the gradient-based iterative (GI) algorithm given by*

$$X(k) = \big(X_1(k) + X_2(k) + \cdots + X_p(k)\big)/p,$$

$$X_i(k) = X(k-1) + \mu A_i^T \left(C - \sum_{j=1}^{p} A_j X(k-1) B_j \right) B_i^T,$$

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

converges to the solution X.

In 2008, Ding, Liu, and Ding derived the following three iterative methods for the equation $AXB = C$, and the equation $\sum_{j=1}^{p} A_j X B_j = F$.

Proposition 1.2 ([33]) *If the equation $AXB = C$ has a unique solution X^*, then the gradient-based iterative (GI) algorithm,*

$$X(k+1) = X(k) + \mu A^T \big(C - AX(k)B\big)B^T,$$

$$0 < \mu < \frac{2}{\lambda_{\max}(AA^T)\lambda_{\max}(B^T B)} \quad \text{or} \quad \mu \le \frac{2}{\|A\|^2 \|B\|^2},$$

is such that $X(k) \to X^*$.

Proposition 1.3 ([33]) *If the equation $AXB = C$ has a unique solution X^*, then the least squares (LS) iterative algorithm,*

$$X(k+1) = X(k) + \mu \big(A^T A\big)^{-1} A^T \big(C - AX(k)B\big)B^T \big(BB^T\big)^{-1}, \quad 0 < \mu < 2$$

is such that $X(k) \to X^*$.

Proposition 1.4 ([33]) *If the matrix equation $\sum_{j=1}^{p} A_j X B_j = F$ has a unique solution X, then the iterative solution $X(k)$ obtained from the least-squares-iterative (LSI) algorithm given by*

$$X(k) = X(k-1) + \mu \sum_{i=1}^{p} \big(A_i^T A_i\big)^{-1} A_i^T \left(F - \sum_{j=1}^{p} A_j X(k-1) B_j \right) B_i^T \big(B_i B_i^T\big)^{-1},$$

where $0 < \mu < 2p$, converges to the solution X.

There are many variations and modifications of the GI algorithm [32], namely the RGI algorithm [34], the MGI algorithm [35], the JGI algorithm [36], and the AJGI algorithm [36].

In this paper, we introduce a gradient-descent iterative algorithm for solving the generalized Sylvester equation that takes the form

$$\sum_{t=1}^{p} A_t X B_t = C. \tag{1}$$

Note that this equation includes all mentioned matrix equations as special cases. The obtained algorithm is based on the vector representation and the variants of the previous works in [32, 33]. The algorithm aims to minimize an error at each iteration by the idea of gradient-descent. We show that the proposed algorithm can be applied to any problems with any initial matrices as long as such problem has a unique solution. The convergence rate and error estimates are given in terms of the condition number of the associated iteration matrix. Numerical simulations reveal that our proposed algorithm performs well compared to the mentioned iterative methods. Moreover, our algorithm can be employed to a discretization of famous partial differential equations namely, the one-dimensional heat equation and the two-dimensional Poisson's equation. Both equations are widely used in many areas of theoretical physics, electrostatic and mechanical engineering; see, e.g., [37] and [38]. According to our numerical results, the algorithm is applicable to both heat and Poisson's equations comparing to their analytical solutions.

The outline of this paper is as follows. In Sect. 2, we supply auxiliary tools to solve linear matrix equations and to make a convergence analysis of an iterative method for solving such equations. We propose new algorithms for the equations $AXB = C$ and $\sum_{t=1}^{p} A_t X B_t$ in Sects. 3 and 4, respectively. In Sect. 5, we presented numerical simulations for various kinds of the linear matrix equations. In Sects. 6 and 7, we apply our algorithm to the one-dimensional heat equation and the two-dimensional Poisson's equation, respectively. The numerical simulations for heat and Poisson's equations are provided in their own sections. Finally, we present a conclusion in Sect. 8.

2 Preliminaries on matrix analysis

Throughout this paper, all considered matrices are real. Denote the set of $m \times n$ matrices by $M_{m,n}$. When $m = n$, we write M_n instead of $M_{n,n}$. Let I be an identity matrix of compatible dimension. The (i,j)th entry of a matrix A is denoted by $A(i,j)$ or a_{ij}.

Recall the Löwner partial order $\preceq$ for real symmetric matrices:

$$A \preceq B \quad \Leftrightarrow \quad B - A \text{ is positive definite} \quad \Leftrightarrow \quad x^T A x \leq x^T B x, \quad \text{for all } x \in \mathbb{R}^n.$$

The Kronecker (tensor) product of $A = [a_{ij}] \in M_{m,n}$ and $B \in M_{p,q}$ is defined by

$$A \otimes B = [a_{ij} B]_{ij} \in M_{mp,nq}.$$

The vector operator is defined for each $A = [a_{ij}] \in M_{m,n}$ by

$$\mathrm{Vec}(A) = \begin{bmatrix} a_{11} \cdots a_{m1} & a_{12} \cdots a_{m2} & \cdots & a_{1n} \cdots a_{mn} \end{bmatrix}^T.$$

It is clear that the vector operator is linear and injective.

Lemma 2.1 ([39]) *The Kronecker product and the vector operator posses the following properties provided that all matrices are compatible*:

(i) $(A \otimes B)^T = A^T \otimes B^T$,

(ii) $(A \otimes B)(C \otimes D) = AC \otimes BD$,

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

To perform convergence analysis, the spectral norm, the Frobenius norm, and the condition number of $A \in M_{m,n}$ are used and respectively defined by

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)}, \qquad \|A\|_F = \sqrt{\mathrm{tr}(A^T A)}, \qquad \kappa(A) = \left(\frac{\lambda_{\max}(A^T A)}{\lambda_{\min}(A^T A)}\right)^{1/2}.$$

We recall the following properties:

Lemma 2.2 ([40]) *For any compatible matrices A and B, we have*

(i) $\|A^T A\|_2 = \|A\|_2^2$,

(ii) $\|A^T\|_2 = \|A\|_2$,

(iii) $\|AB\|_F \le \|A\|_2 \|B\|_F$.

3 The equation $AXB = C$

Consider the matrix equation

$$AXB = C, \tag{2}$$

where $A \in M_{p,m}$ has full column-rank, $B \in M_{n,q}$ has full row-rank, $C \in M_{p,q}$ is a known constant matrix, and $X \in M_{m,n}$ is unknown. The hypotheses imply the invertibility of $A^T A$ and BB^T, and thus we obtain the unique solution to be

$$X^* = (A^T A)^{-1} A^T C B^T (BB^T)^{-1}. \tag{3}$$

However, to compute $(A^T A)^{-1}$ and $(BB^T)^{-1}$ requires a large amount of data storage if the sizes of matrices are large. Thus, in this section, we shall propose a new iterative method to solve (2) based on gradients and the steepest descend which provides an appropriate sequence of convergent factors for minimizing an error at each iteration. Moreover, the discussion in this section leads to a treatment for a general matrix equation in Sect. 4.

3.1 Proposed algorithm

We consider the Frobenius norm-error $\|AXB - C\|_F$ which can be equally transform into $\|(B^T \otimes A)\,\mathrm{Vec}(X) - \mathrm{Vec}(C)\|_F$ via Lemma 2.1(iii). So, we define the quadratic norm-error function $f : \mathbb{R}^{mn} \to \mathbb{R}$ by

$$f(x) := \frac{1}{2}\left\|(B^T \otimes A)x - \mathrm{Vec}(C)\right\|_F^2.$$

We know that a norm function is a convex function, so f is convex. We assume that the exact solution X^* of (2) is uniquely determined, hence an optimal matrix X^* of f exists. We start by having an arbitrary initial matrix $X(0)$ and then at every step $k > 0$ we iteratively move to the next matrix $X(k+1)$ along an appropriate direction, i.e., the negative

gradient of f, together with a suitable step size. In the kth step, the step size τ_{k+1} is changed appropriately in order to incur the minimum error. The gradient-descent iterative method thus can be described through the following recursive rule:

$$X(k+1) = X(k) - \tau_{k+1}\nabla f\big(\mathrm{Vec}(X(k))\big).$$

In order to do that, we recall the following gradient formula:

$$\frac{d}{dX}\mathrm{tr}(AX) = \frac{d}{dX}\mathrm{tr}\big(X^TA^T\big) = A^T.$$

Now, we find the gradient of function f and deduce its derivatives in detail. Letting $S = B^T \otimes A$, $x = \mathrm{Vec}(X)$, and $\widehat{c} = \mathrm{Vec}(C)$, we have

$$\begin{aligned}\nabla f(x) = \frac{df(x)}{dx} &= \frac{1}{2}\frac{d}{dx}\mathrm{tr}\big((Sx-\widehat{c})^T(Sx-\widehat{c})\big)\\ &= \frac{1}{2}\frac{d}{dx}\mathrm{tr}\big(Sxx^TS^T - \widehat{c}x^TS^T - Sx\widehat{c}^T + \widehat{c}\widehat{c}^T\big)\\ &= S^T(Sx-\widehat{c}).\end{aligned} \tag{4}$$

Thus, our new iterative equation is in the form

$$\mathrm{Vec}\big(X(k+1)\big) = \mathrm{Vec}\big(X(k)\big) + \tau_{k+1}\big(B^T\otimes A\big)^T\big(\mathrm{Vec}(C) - \big(B^T\otimes A\big)\,\mathrm{Vec}(X)\big).$$

Using Lemma 2.1, we have

$$X(k+1) = X(k) + \tau_{k+1}\big(A^T(C - AX(k)B)B^T\big).$$

Next, we choose a step size. To generate the best step size at each iteration, we minimize an error which occurs at the next iteration, $X(k+1)$. Then, for each $k \in \mathbb{N}\cup 0$, we define $\phi_{k+1} : [0,\infty) \to \mathbb{R}$ by

$$\begin{aligned}\phi_{k+1}(\tau) &:= f\big(\mathrm{Vec}\big(X(k+1)\big)\big)\\ &= \frac{1}{2}\big\|\big(B^T\otimes A\big)\,\mathrm{Vec}\big(X(k) + \tau_{k+1}\big(A^T(C - AX(k)B)B^T\big)\big) - \mathrm{Vec}(C)\big\|_F^2.\end{aligned}$$

Now, we shall minimize the function $\phi_{k+1}(\tau)$ by applying the properties of matrix trace. Before that, we may transform $\phi_{k+1}(\tau)$ into a convenient form by letting $\overline{c} = \widehat{c} - Sx$ and $\tilde{b} = SS^T\overline{c}$, so that

$$\begin{aligned}\phi_{k+1}(\tau) &= \frac{1}{2}\big\|S\big(x(k) + \tau_{k+1}S^T\big(\widehat{c} - Sx(k)\big)\big) - \widehat{c}\big\|_F^2\\ &= \frac{1}{2}\big\|\tau_{k+1}SS^T\big(\widehat{c} - Sx(k)\big) + Sx(k) - \widehat{c}\big\|_F^2\\ &= \frac{1}{2}\|\tau_{k+1}\tilde{b} - \overline{c}\|_F^2.\end{aligned}$$

Differentiating both sides, we have

$$\frac{d\phi_{k+1}(\tau)}{d\tau} = \frac{1}{2}\frac{d}{d\tau}\mathrm{tr}\big((\tau\tilde{b}-\overline{c})^T(\tau\tilde{b}-\overline{c})\big)$$

$$= \frac{1}{2}\frac{d}{d\tau}\operatorname{tr}\left(\tau\tilde{b}\tau\tilde{b}^T - \tau\tilde{b}\overline{c}^T - \overline{c}\tau\tilde{b}^T + \overline{c}\overline{c}^T\right)$$
$$= \tau\operatorname{tr}\left(\tilde{b}\tilde{b}^T\right) - \operatorname{tr}\left(\tilde{b}\overline{c}^T\right).$$

Note that the second derivative of $\phi_{k+1}(\tau)$ is the constant $\operatorname{tr}(\tilde{b}\tilde{b}^T)$, which is positive. Setting $d\phi_{k+1}(\tau)/d\tau = 0$ and using Lemma 2.1(iii), we obtain the minimizer of $\phi_{k+1}(\tau)$ as follows:

$$\tau_{k+1} = \frac{\|(B^T\otimes A)^T(\operatorname{Vec}(C) - (B^T\otimes A)\operatorname{Vec}(X(k)))\|_F^2}{\|(B^T\otimes A)(B^T\otimes A)^T(\operatorname{Vec}(C) - (B^T\otimes A)\operatorname{Vec}(X(k)))\|_F^2}$$
$$= \frac{\|A^T(C - AX(k)B)B^T\|_F^2}{\|AA^T(C - AX(k)B)B^TB\|_F^2}.$$

Summarizing the direction and the step size altogether, we get:

Algorithm 3.1 *The gradient-descent iterative algorithm for solving* (2).

Initialization step. *Given any small error* $\epsilon > 0$, *choose an initial matrix* $X(0)$. *Set* $k := 0$. *Compute* $\widehat{A} = AA^T$, *and* $\widehat{B} = B^TB$.

Stopping rule. *Compute* $E(k) = C - AX(k)B$. *If* $\|E(k)\|_F < \epsilon$, *stop. Otherwise, go to the next step.*

Updating step. *Compute*

$$\tau_{k+1} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(\sum_{\beta=1}^{q}(\sum_{\alpha=1}^{p}A^T(i,\alpha)E(\alpha,\beta))B^T(\beta,j))^2}{\sum_{i=1}^{p}\sum_{j=1}^{q}(\sum_{\beta=1}^{q}(\sum_{\alpha=1}^{p}\widehat{A}(i,\alpha)E(\alpha,\beta))\widehat{B}(\beta,j))^2},$$

$$X(k+1) = X(k) + \tau_{k+1}A^TE(k)B^T.$$

Set $k := k+1$ *and return to Stopping rule.*

Remark 3.2 In Algorithm 3.1, we introduce the matrices $\widehat{A}$, $\widehat{B}$, and $E(k)$ to avoid duplicate manipulations. The term $E(k)$ or $E(\alpha,\beta)$ in the denominator of the formula of τ_{k+1} does not cause a severe propagation of errors when $X(k)$ is close to the exact solution. This is because the Stopping rule prevents $E(\alpha,\beta)$ from being a very small number, and there is also the term $E(\alpha,\beta)$ in the numerator. A similar comment is applied to any developed algorithms in this paper.

3.2 Convergence of the algorithm

Here, we will prove that Algorithm 3.1 converges to the exact solution. The following analysis will hold for strongly convex functions. Recall that a twice-differentiable convex function $f:\mathbb{R}^n\to\mathbb{R}$ is said to be *strongly convex* if there exist $m,M\in[0,\infty)$ such that $mI \preceq \nabla^2 f(x) \preceq MI$ for all $x\in\mathbb{R}^n$.

Lemma 3.3 ([41]) *If* f *is strongly convex on* $\mathbb{R}^n$, *then for any* $x,y\in\mathbb{R}^n$

$$f(y) \geq f(x) + \nabla f(x)^T(y-x) + \frac{m}{2}\|y-x\|_F^2, \tag{5}$$

$$f(y) \leq f(x) + \nabla f(x)^T(y-x) + \frac{M}{2}\|y-x\|_F^2. \tag{6}$$

Theorem 3.4 *If (2) is consistent and has a unique solution X^*, then the iterative sequence $\{X(k)\}$ generated by Algorithm 3.1 converges to X^* for any initial matrix $X(0)$, i.e., $X(k) \to X^*$ as $k \to \infty$.*

Proof If $\nabla f(\mathrm{Vec}(X(k))) = 0$ for some k, then $X(k) = X^*$ and the result holds. So assume that $\nabla f(\mathrm{Vec}(X(k))) \neq 0$ for all k. To investigate its convexity, let us find the second derivative. Indeed, we have from (4) and Lemma 2.1 that

$$\nabla^2 f\big(\mathrm{Vec}(X)\big) = \big(B^T \otimes A\big)^T \big(B^T \otimes A\big) = BB^T \otimes A^T A.$$

For convenience, we write $\lambda_{\min}$ and $\lambda_{\max}$ instead of $\lambda_{\min}(BB^T \otimes A^T A)$ and $\lambda_{\max}(BB^T \otimes A^T A)$, respectively. Since $BB^T \otimes A^T A$ is symmetric, we have

$$\lambda_{\min} I \preceq \nabla^2 f\big(\mathrm{Vec}(X)\big) \preceq \lambda_{\max} I,$$

meaning that f is strongly convex. Considering $\phi_{k+1}(\tau) = f(\mathrm{Vec}(X(k+1)))$ and applying (6) in Lemma 3.3, we obtain

$$\phi_{k+1}(\tau) \leq f\big(\mathrm{Vec}\big(X(k)\big)\big) - \tau \left\| \nabla f\big(\mathrm{Vec}\big(X(k)\big)\big) \right\|_F^2 + \frac{\lambda_{\max}\tau^2}{2} \left\| \nabla f\big(\mathrm{Vec}\big(X(k)\big)\big) \right\|_F^2.$$

The right-hand side is minimized by $\tau_{k+1} = 1/\lambda_{\max}$, and

$$\begin{aligned} f\big(\mathrm{Vec}\big(X(k+1)\big)\big) &= \phi_{k+1}(\tau_{k+1}) \\ &\leq f\big(\mathrm{Vec}\big(X(k)\big)\big) - \frac{1}{2\lambda_{\max}} \left\| \nabla f\big(\mathrm{Vec}\big(X(k)\big)\big) \right\|_F^2. \end{aligned} \tag{7}$$

It follows from (5) that

$$\begin{aligned} f\big(\mathrm{Vec}\big(X(k+1)\big)\big) \geq{}& f\big(\mathrm{Vec}\big(X(k)\big)\big) - \tau \left\| \nabla f\big(\mathrm{Vec}\big(X(k)\big)\big) \right\|_F^2 \\ &+ \frac{\lambda_{\min}\tau^2}{2} \left\| \nabla f\big(\mathrm{Vec}\big(X(k)\big)\big) \right\|_F^2. \end{aligned} \tag{8}$$

We find that $\tau = 1/\lambda_{\min}$ minimizes the RHS of (8), i.e.,

$$0 \geq f\big(\mathrm{Vec}\big(X(k)\big)\big) - \frac{1}{2\lambda_{\min}} \left\| \nabla f\big(\mathrm{Vec}\big(X(k)\big)\big) \right\|_F^2.$$

Hence,

$$\left\| \nabla f\big(\mathrm{Vec}\big(X(k)\big)\big) \right\|_F^2 \geq 2\lambda_{\min} f\big(\mathrm{Vec}\big(X(k)\big)\big). \tag{9}$$

Substituting (9) into (7) and then putting $c := 1 - \lambda_{\min}/\lambda_{\max}$, we have

$$f\big(\mathrm{Vec}\big(X(k+1)\big)\big) \leq c f\big(\mathrm{Vec}\big(X(k)\big)\big).$$

We obtain inductively that

$$f\big(\mathrm{Vec}\big(X(k)\big)\big) \leq c^k f\big(\mathrm{Vec}\big(X(0)\big)\big).$$

Since A has full column-rank and B has full row-rank, $BB^T \otimes A^T A$ is invertible. It follows that $BB^T \otimes A^T A$ is positive definite, which implies $\lambda_{\min} > 0$ and thus $0 < c < 1$. Hence, $f(\mathrm{Vec}(X(k))) \to 0$ as $k \to \infty$. □

4 The equation $\sum_{t=1}^{p} A_t X B_t$

In this section, we consider the generalized Sylvester equation

$$\sum_{t=1}^{p} A_t X B_t = C, \tag{10}$$

where for each $t = 1, \ldots, p$ $A_t \in M_{q,m}$ is a full column-rank matrix, $B_t \in M_{n,r}$ is a full row-rank matrix, $C \in M_{q,r}$ is a known constant matrix, and $X \in M_{m,n}$ is an unknown matrix. An equivalent condition for (10) to have a unique solution is that $P = \sum_{t=1}^{p} B_t^T \otimes A_t$ is invertible. Its unique solution is given by

$$\mathrm{Vec}(X^*) = (P^T P)^{-1} P^T \mathrm{Vec}(C). \tag{11}$$

We shall introduce a new iterative method for solving (10) based on gradients and the steepest descend which provides an appropriate sequence of convergent factors for minimizing an error at each iteration.

4.1 Proposed algorithm

We define the quadratic norm-error function $\tilde{f} : \mathbb{R}^{mn} \to \mathbb{R}$ by

$$\tilde{f}(x) := \frac{1}{2} \|Px - \mathrm{Vec}(C)\|_F^2.$$

It is obvious that $\tilde{f}$ is convex. For convenience, we let $P = \sum_{t=1}^{p} B_t^T \otimes A$. We assume that P is invertible, then the exact solution exists. The gradient-descent iterative method therefore can be described through the following recursive rule:

$$X(k+1) = X(k) - \tilde{\tau}_{k+1} \nabla \tilde{f}(\mathrm{Vec}(X(k))).$$

To search for the direction, we use the same techniques as in the previous section and then obtain

$$\nabla \tilde{f}(\mathrm{Vec}(X)) = P^T (P \mathrm{Vec}(X) - \mathrm{Vec}(C)).$$

Thus, our new iterative equation is in the form

$$\mathrm{Vec}(X(k+1)) = \mathrm{Vec}(X(k)) + \tilde{\tau}_{k+1} P^T (P \mathrm{Vec}(X) - \mathrm{Vec}(C)).$$

Using Lemma 2.1, we obtain

$$X(k+1) = X(k) + \tilde{\tau} \sum_{t=1}^{p} \left(A_t^T \left(C - \sum_{l=1}^{p} A_l X(k) B_l \right) B_t^T \right).$$

Next, we choose a step size. With the same technique as in the previous section, we minimize $\tilde{\phi} : [0, \infty) \to \mathbb{R}$ by for each $k = 0, 1, \ldots$, $\tilde{\phi}_{k+1}(\tilde{\tau}) := \tilde{f}(X(k+1))$. Similarly, the minimizer of function $\tilde{\phi}_{k+1}(\tilde{\tau})$ is

$$\tilde{\tau}_{k+1} = \frac{\| \sum_{t=1}^{p} (A_t^T (C - \sum_{l=1}^{p} A_l X(k) B_l) B_t^T) \|_F^2}{\| \sum_{t=1}^{p} \sum_{h=1}^{p} (A_t A_h^T (C - \sum_{l=1}^{p} A_l X(k) B_l) B_h^T B_t) \|_F^2}.$$

Summarizing the direction and the step size altogether, we get:

Algorithm 4.1 *The gradient-descent iterative algorithm for solving* (10).

Initialization step. *Given any small error $\epsilon > 0$, choose an initial matrix $X(0)$. Set $k := 0$. Compute $A_{\alpha,\beta} = A_\alpha A_\beta^T$, and $B_{\alpha,\beta} = B_\alpha^T B_\beta$ for all $\alpha, \beta = 1, \ldots, p$.*

Stopping rule. *Compute $E(k) = C - \sum_{t=1}^{p} A_t X(k) B_t$. If $\|E(k)\|_F < \epsilon$, stop. Otherwise, go to the next step.*

Updating step. *Compute*

$$\tilde{\tau}_{k+1} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} (\sum_{t=1}^{p} \sum_{\beta=1}^{r} \sum_{\alpha=1}^{q} A_t^T(i,\alpha) E(\alpha,\beta) B_t^T(\beta,j))^2}{\sum_{i=1}^{q} \sum_{j=1}^{r} (\sum_{t=1}^{p} \sum_{h=1}^{p} \sum_{\beta=1}^{r} \sum_{\alpha=1}^{q} A_{t,h}(i,\alpha) E(\alpha,\beta) B_{t,h}(\beta,j))^2},$$

$$X(k+1) = X(k) + \tilde{\tau}_{k+1} \sum_{t=1}^{p} A_t^T E(k) B_t^T.$$

Set $k := k + 1$ and return to the Stopping rule.

4.2 Convergence analysis of the algorithm

In this subsection, we shall show that Algorithm 4.1 is applicable for any choice of the initial matrix $X(0)$ as long as equation (10) has a unique solution. After that, we shall discuss error estimates and the asymptotic convergence rate of the algorithm.

Theorem 4.2 *If (10) is consistent and has a unique solution X^*, or equivalently, P is invertible, then the iterative sequence $\{X(k)\}$ generated by Algorithm 4.1 converges to X^* for any initial matrix $X(0)$, i.e., $X(k) \to X^*$ as $k \to \infty$.*

Proof Convergence of Algorithm 4.1 can be proved similarly as in Theorem 3.4. In this case, we have

$$\lambda_{\min}(P^T P) I \preceq \nabla^2 \tilde{f}(\mathrm{Vec}(X(k))) = P^T P \preceq \lambda_{\max}(P^T P) I,$$

which implies the strong convexity of $\tilde{f}$. In a similar manner, we get

$$\tilde{f}(\mathrm{Vec}(X(k+1))) \leq \tilde{c} \tilde{f}(\mathrm{Vec}(X(k))), \tag{12}$$

where $\tilde{c} := 1 - \lambda_{\min}(P^T P)/\lambda_{\max}(P^T P)$. By induction, we obtain

$$\tilde{f}(\mathrm{Vec}(X(k))) \leq \tilde{c}^k \tilde{f}(\mathrm{Vec}(X(0))). \tag{13}$$

The uniqueness of the solution implies that P is positive definite, and thus $0 < \tilde{c} < 1$. Hence, $\tilde{f}(\mathrm{Vec}(X(k))) \to 0$ as $k \to \infty$. □

From now on, we denote $\kappa = \kappa(P)$, the condition number of P. Observe that $\tilde{c} = 1 - \kappa^{-2}$. According to Lemma 2.1(iii), the bounds (12) and (13) give rise to the following estimates:

$$\left\| \sum_{t=1}^{p} A_t X(k) B_t - C \right\|_F \leq \left(1 - \kappa^{-2}\right)^{\frac{1}{2}} \left\| \sum_{t=1}^{p} A_t X(k-1) B_t - C \right\|_F, \tag{14}$$

$$\left\| \sum_{t=1}^{p} A_t X(k) B_t - C \right\|_F \leq \left(1 - \kappa^{-2}\right)^{\frac{k}{2}} \left\| \sum_{t=1}^{p} A_t X(0) B_t - C \right\|_F. \tag{15}$$

Since $0 < \tilde{c} < 1$, it follows that if $\| \sum_{t=1}^{p} A_t X(k-1) B_t - C \|_F$ are nonzero, then

$$\left\| \sum_{t=1}^{p} A_t X(k) B_t - C \right\|_F < \left\| \sum_{t=1}^{p} A_t X(k-1) B_t - C \right\|_F. \tag{16}$$

We can summarize the above discussion as follows:

Theorem 4.3 *Suppose the hypothesis of Theorem 4.2 holds. The convergence rate of Algorithm 4.1 (with respect to the certain error $\| \sum_{t=1}^{p} A_t X(k) B_t - C \|_F$) is governed by $\sqrt{1 - \kappa^{-2}}$. Moreover, the error estimates $\| \sum_{t=1}^{p} A_t X(k) B_t - C \|_F$ compared to the previous iteration and the first iteration are provided by (14) and (15), respectively. In particular, the relative error at each iteration gets smaller than the previous (nonzero) error, as in (16).*

Theorem 4.4 *Suppose the hypothesis of Theorem 4.2 holds. Then the error estimates $\|X(k) - X^*\|_F$ compared to the previous iteration and the first iteration of Algorithm 4.1 are given as follows:*

$$\left\|X(k) - X^*\right\|_F \leq \kappa \sqrt{\kappa^2 - 1} \left\|X(k-1) - X^*\right\|_F, \tag{17}$$

$$\left\|X(k) - X^*\right\|_F \leq \kappa^2 \left(1 - \kappa^{-2}\right)^{\frac{k}{2}} \left\|X(0) - X^*\right\|_F. \tag{18}$$

In particular, the convergence rate of the algorithm is governed by $\sqrt{1 - \kappa^{-2}}$.

Proof Utilizing equation (15) and Lemma 2.2, we have

$$\begin{aligned}
\left\|X(k) - X^*\right\|_F &= \left\|\operatorname{Vec}\left(X(k)\right) - \operatorname{Vec}\left(X^*\right)\right\|_F \\
&= \left\|\left(P^T P\right)^{-1} \left(P^T P\right) \operatorname{Vec}\left(X(k)\right) - \left(P^T P\right)^{-1} \left(P^T P\right) \operatorname{Vec}\left(X^*\right)\right\|_F \\
&\leq \left\|\left(P^T P\right)^{-1}\right\|_2 \left\|P^T\right\|_2 \left\|P \operatorname{Vec}\left(X(k)\right) - P \operatorname{Vec}\left(X^*\right)\right\|_F \\
&\leq \left(1 - \kappa^{-2}\right)^{\frac{k}{2}} \left\|\left(P^T P\right)^{-1}\right\|_2 \left\|P^T\right\|_2 \left\|P \operatorname{Vec}\left(X(0)\right) - \operatorname{Vec}(C)\right\|_F \\
&\leq \left(1 - \kappa^{-2}\right)^{\frac{k}{2}} \left\|\left(P^T P\right)^{-1}\right\|_2 \left\|P^T\right\|_2 \|P\|_2 \left\|X(0) - X^*\right\|_F \\
&= \left(1 - \kappa^{-2}\right)^{\frac{k}{2}} \frac{\lambda_{\max}(P^T P)}{\lambda_{\min}(P^T P)} \left\|X(0) - X^*\right\|_F \\
&= \kappa^2 \left(1 - \kappa^{-2}\right)^{\frac{k}{2}} \left\|X(0) - X^*\right\|_F.
\end{aligned}$$

Since the asymptotic behavior of the above error depends on the term $(1-\kappa^{-2})^{\frac{k}{2}}$, the asymptotic convergence rate for the algorithm is governed by $\sqrt{1-\kappa^{-2}}$. In a similar manner but making use of (14) instead of (15), we obtain

$$\begin{aligned}\left\|X(k)-X^*\right\|_F &\le \left(1-\kappa^{-2}\right)^{\frac{1}{2}}\left\|\left(P^TP\right)^{-1}\right\|_2\left\|P^T\right\|_2\left\|P\,\mathrm{Vec}\left(X(k-1)\right)-\mathrm{Vec}(C)\right\|_F\\ &\le \left(1-\kappa^{-2}\right)^{\frac{1}{2}}\left\|\left(P^TP\right)^{-1}\right\|_2\left\|P^T\right\|_2\|P\|_2\left\|X(k-1)-X^*\right\|_F\\ &= \kappa^2\left(1-\kappa^{-2}\right)^{\frac{1}{2}}\left\|X(k-1)-X^*\right\|_F,\end{aligned}$$

and hence (17) is reached. □

Thus, the condition number κ determines the asymptotic convergence rate, as well as how far our initial matrix was from the exact solution. The closer κ gets to 1, the faster the algorithm converges to the required result.

5 Numerical simulations for a class of the generalized Sylvester matrix equations

In this section, we present applications of our proposed algorithms to the certain linear matrix equations. To show the effectiveness and capability of our algorithms, we compare our proposed algorithms to the mentioned existing algorithms as well as the direct methods (3) and (11). For convenience, we abbreviate TauOpt to represent our algorithms. To measure the computational time taken for each program, we apply the *tic* and *toc* functions in MATLAB and abbreviate CT for it. The readers are recommended to consider all reported results, such as errors, CTs, figures, while comparing the performance of any algorithms. To measure the error at the kth step of the iteration, we consider the following error:

$$\gamma_k := \left\|X(k)-X^*\right\|_F.$$

All iterations have been carried out by MATLAB R2018a, Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz, 8.00 GB RAM, PC environment.

Example 5.1 We consider the equation $AXB = C$ with

$$A=\begin{bmatrix}1&-1&2&3&1&-3&3&2\\2&3&-2&2&2&1&3&3\\3&1&1&-1&-3&-2&-1&3\end{bmatrix}^T \quad \text{and}$$

$$B=\begin{bmatrix}1&2&-5&9&7&5&1&0&-6&3\\2&-7&8&3&0&1&2&3&5&-6\\6&-5&2&1&0&3&-9&8&7&6\end{bmatrix}.$$

We choose the initial matrix $X(0) = 10^{-6}\,\mathrm{ones}(3,3)$ where $\mathrm{ones}(m,n)$ denotes the $m \times n$ matrix with contains 1 at every position. After running Algorithm 3.1, the numerical so-

Table 1 Error and CT for Example 5.1

Method	Error	CT
TauOpt	7.2231e−14	0.0057
GI	12.1879	0.0004
LS	13.8387	0.0027
direct		0.3051

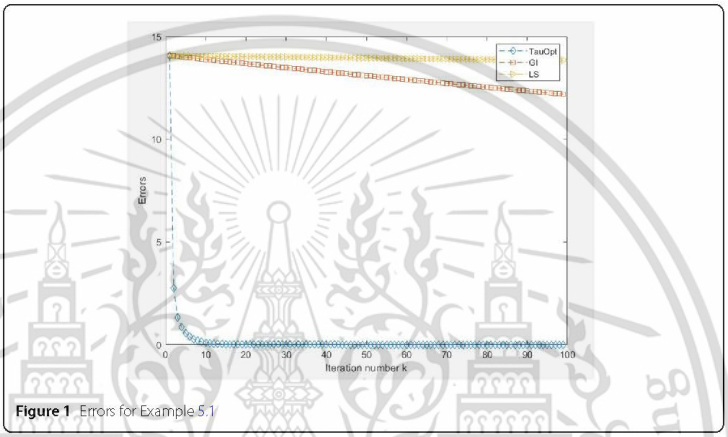

Figure 1 Errors for Example 5.1

lutions converge to the exact solution

$$X^* = \begin{bmatrix} 1 & 5 & -9 \\ 6 & 5 & 4 \\ 1 & 2 & 3 \end{bmatrix}.$$

In this example, we compare Algorithm 3.1 with GI (Proposition 1.2) and LS (Proposition 1.3). All reports are presented after running 100 iterations. Table 1 shows the errors at the final iteration as well as the computational time. Figure 1 displays the error plot. Table 1 implies that our algorithm takes significantly less computational time than the direct method. For comparison to other two algorithms, it seems that our algorithm takes a little more time but both Table 1 and Fig. 1 indicate that ours obtains a highly satisfactory approximated solution.

Example 5.2 In this example we consider the Sylvester equation with

$$A = \text{tridiag}(3, -9, 1) \in M_{100} \quad \text{and} \quad B = \text{tridiag}(-1, -2, 5) \in M_{100}.$$

After running Algorithm 4.1 with an initial matrix $X(0) = 10^{-6}\,\text{ones}(100, 100)$, the numerical solution converges to the exact solution $X^* = \text{tridiag}(1, 2, 3) \in M_{100}$.

We compare Algorithm 4.1 with the following algorithms: GI [32], RGI [34], MGI [35], JGI (Algorithm 4, [36]) and AJGI (Algorithm 5, [36]). The results after running for 100 iterations are shown in Fig. 2 and Table 2. According to the error and CT in Table 2 and

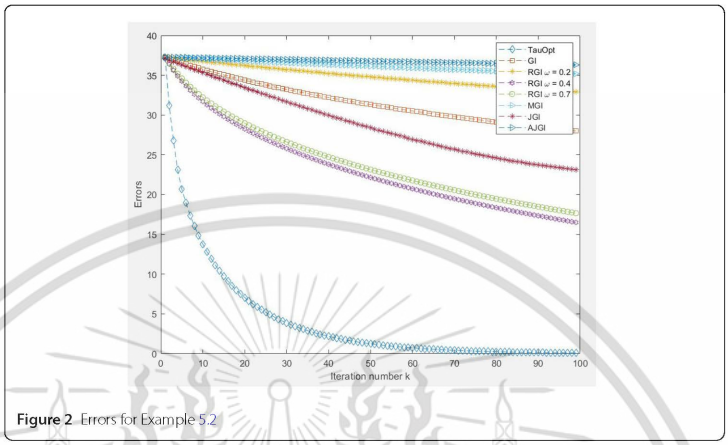

Figure 2 Errors for Example 5.2

Table 2 Error and CT for Example 5.2

Method	Error	CT
TauOpt	0.0891	0.0568
GI	27.9847	0.0341
RGI $\omega = 0.2$	32.9285	0.0300
RGI $\omega = 0.4$	16.5017	0.0288
RGI $\omega = 0.7$	17.6606	0.0261
MGI	35.1852	0.0415
JGI	23.1308	0.0292
AJGI	36.3178	0.0423
direct		71.2048

Table 3 Errors and CT for Example 5.3

Method	Error	CT
TauOpt	2.0180e−16	0.0496
GI	0.3227	0.0124
LSI	13.1666	0.0242
direct		0.3867

Fig. 2, we find that despite a little longer computational time, our final error outperforms the other algorithms.

Example 5.3 In this example we consider equation (10) when $p = 3$ with

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 3 & 1 \\ 2 & -2 & 1 \\ 3 & 2 & -1 \\ 1 & 2 & -3 \\ -3 & 1 & -2 \\ 3 & 3 & -1 \\ 2 & 3 & 3 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 3 & 6 & 5 \\ 6 & 9 & -4 \\ 3 & 2 & -1 \\ 1 & 2 & -3 \\ -3 & 1 & -2 \\ 3 & 3 & -1 \\ 6 & -1 & 0 \\ 2 & 3 & 3 \end{bmatrix}, \quad A_3 = \begin{bmatrix} -2 & 0 & 5 \\ 6 & 9 & -4 \\ 9 & 5 & -4 \\ 0 & 1 & 6 \\ 9 & -2 & 0 \\ 3 & 3 & -1 \\ -7 & 2 & 0 \\ -8 & 8 & 1 \end{bmatrix},$$

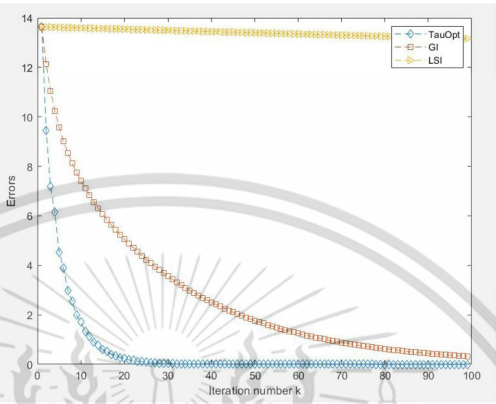

Figure 3 Errors for Example 5.3

$$B_1 = \begin{bmatrix} 1 & 2 & 6 \\ 2 & -7 & -5 \\ -5 & 8 & 2 \\ 9 & 3 & 1 \\ 7 & 0 & 0 \\ 5 & 1 & 3 \\ 1 & 2 & -9 \\ 0 & 3 & 8 \\ -6 & 5 & 7 \\ 3 & -6 & 6 \end{bmatrix}^T, \quad B_2 = \begin{bmatrix} 1 & 6 & 6 \\ 2 & -2 & -5 \\ -5 & 0 & 2 \\ 4 & 5 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 3 \\ 3 & 2 & 3 \\ -9 & 3 & -5 \\ -6 & 5 & 9 \\ 3 & -6 & 1 \end{bmatrix}^T, \quad B_3 = \begin{bmatrix} 3 & 6 & 6 \\ 2 & -2 & 6 \\ 1 & 0 & 3 \\ 1 & 5 & 0 \\ 1 & 0 & -7 \\ 0 & 1 & 3 \\ 3 & 0 & 3 \\ -9 & 9 & -5 \\ -6 & -4 & 9 \\ 3 & -6 & 1 \end{bmatrix}^T.$$

We choose an initial matrix $X(0) = 10^{-6}\,\mathrm{ones}(3,3)$. After running Algorithm 4.1, the numerical solutions converge to the exact solution

$$X^* = \begin{bmatrix} 6 & 2 & 0 \\ -9 & 4 & -2 \\ 3 & 6 & 0 \end{bmatrix}.$$

In this example, we compare Algorithm 4.1 with GI (Proposition 1.1) and LSI (Proposition 1.4). The results after 100 iterations are shown in Fig. 3 and Table 3. We find that Algorithm 4.1 gives the fastest convergence.

6 An application to a discretization of one-dimensional heat equation

In this section, we apply our proposed algorithm to a discretization of one-dimensional heat equation:

$$\frac{\partial u}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2} \quad \text{on } [0, \beta_t] \times [\alpha_x, \beta_x] \tag{19}$$

subject to the boundary conditions $u(\alpha_x, t) = g_l, u(\beta_x, t) = g_r, u(x, 0) = g_d$ where g_l, g_r, g_d are given functions.

6.1 Discretization of the heat equation

We make discretization at the grid points in the rectangle which are at (x_i, t_j) with $x_i = \alpha_x + ih_x$ and $t_j = jh_t$ where

$$h_x = \frac{\beta_x - \alpha_x}{N_x + 1} \quad \text{and} \quad h_t = \frac{\beta_t}{N_t}. \tag{20}$$

We denote $u_{ij} = u(x_i, t_j)$. By the Forward Time Central Space (FTCS) method, we obtain

$$\frac{\partial u}{\partial t} = \frac{u_{i,j+1} - u_{ij}}{h_t} = c^2 \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h_x^2} = c^2 \frac{\partial^2 u}{\partial x^2},$$

or equivalently,

$$u_{i,j+1} = F(u_{i-1,j} + u_{i+1,j}) + (1 - 2F)u_{ij},$$

where $F = h_t c^2 / h_x^2$ for $1 \le i \le N_x$, $1 \le j \le N_t$. We transform equation (19) into a linear system of $N_x N_t$ equations in $N_x N_t$ unknowns $u_{11}, \dots, u_{N_x N_t}$:

$$T_H \operatorname{Vec}(U) = V, \tag{21}$$

where $U = [u_{ij}]$, $T_H \in M_{N_x N_t}$ has $N_t \times N_t$ blocks of the form I_{N_x} on its diagonal and tridiag$(-F, -(1-2F), -F)$ under its diagonal. Here is an example of T_H where $N_t = 3$ and $N_x = 2$:

$$T_H = \left[\begin{array}{cc|cc|cc} 1 & & & & & \\ & 1 & & & & \\ \hline -(1-2F) & -F & 1 & & & \\ -F & -(1-2F) & & 1 & & \\ \hline & & -(1-2F) & -F & 1 & \\ & & -F & -(1-2F) & & 1 \end{array}\right].$$

The vector V is partitioned in N_x periods as $\left[V_1^T\ V_2^T \cdots V_{N_y}^T\right]^T$ where

$$V_1 = \begin{bmatrix} Fg_d(\alpha_x, 0) + (1-2F)g_d(x_1, 0) + Fg_d(x_2, 0) \\ Fg_d(x_1, 0) + (1-2F)g_d(x_2, 0) + Fg_d(x_3, 0) \\ \vdots \\ Fg_d(x_{N_t-2}, 0) + (1-2F)g_d(x_{N_t-1}, 0) + Fg_d(N_t, 0) \end{bmatrix} \quad \text{and} \quad V_i = \begin{bmatrix} Fg_l(\alpha_x, i-1) \\ 0 \\ \vdots \\ 0 \\ Fg_r(\beta_x, i-1) \end{bmatrix}$$

for $i = 2, \dots, N_y$.

Equation (21) is formed as $AXB = C$ where $A = T_H$, $X = \operatorname{Vec}(U)$, $B = I$ and $C = V$. According to Algorithm 3.1, we obtain an algorithm for (21) as follows:

Algorithm 6.1 *The gradient-descent iterative algorithm for solving one-dimensional heat equation.*

Input step. *Input* $N_x, N_t \in \mathbb{N}$ *as numbers of partition.*

Initialization step. *Let* h_x *and* h_t *be as in* (20) *and, for each* $i = 1,\dots,N_x$ *and* $j = 1,\dots,N_t$, $x_i = \alpha_x + ih_x$ *and* $t_j = jh_t$, *compute* $s = T_H V$, $S = T_H^2$, $\widehat{s} = T_H s$, *and* $\widehat{S} = T_H S$. *Choose* $u(0) \in \mathbb{R}^{N_x N_t}$ *and set* $k := 0$.

Updating step. *Compute*

$$\tau_{k+1} = \frac{\sum_{p=1}^{N_x N_t} (s_p - \sum_{q=1}^{N_x N_t} S_{pq} u_q(k))^2}{\sum_{p=1}^{N_x N_t} (\widehat{s} - \sum_{q=1}^{N_x N_t} \widehat{S}_{pq} u_q(k))^2},$$

$$u(k+1) = u(k) + \tau_{k+1}\big(s - Su(k)\big).$$

Set $k := k + 1$ *and repeat the Updating step.*

Here, we denote s_p the pth entry of a vector s and S_{pq} the (p,q)-entry of S. To stop the algorithm, an appropriate stopping rule is $\|V - T_H u(k)\|_F^2 < \epsilon$ where ϵ is a small positive number.

6.2 Numerical simulation for the heat equation

To obtain the numerical solutions, we need to partition the rectangular domain. The accuracy of the solution depends on the size of the partition grid. A better accuracy must be from a finer grid system and it causes the size of the associated matrix T_H to be larger.

Example 6.2 Consider the heat equation (19) on $\{(x,t) : 0 < x < 1, t > 0\}$ with the boundary and initial conditions given as:

$$u(0,t) = u(1,t) = 0 \quad \text{and} \quad u(x,0) = \sin \pi x.$$

Let $c = 1$, $N_x = 4$, $h_t = 0.01$. We have $h_x = 0.2$ and $F = 0.25$. In this case, we consider $N_t = 10$, so the size of the matrix T_H is 40×40. We run Algorithm 6.1 with the initial vector $u(0) = 10^{-6}[1 \ \cdots \ 1]^T$ and the numerical solutions converge to the exact solution

$$u^*(x,t) = e^{-\pi^2 t} \sin(\pi x).$$

In this example we compare our algorithm to the following algorithms: GI (Proposition 1.2), RGI [34], MGI [35], LS (Proposition 1.3), JGI (Algorithm 4, [36]) and AJGI (Algorithm 5, [36]). The results after running 500 iterations are shown in Figs. 4 and 5, as well as Tables 4 and 5.

7 An application to a discretization of two-dimensional Poisson's equation

In this section, we give an application of the proposed algorithm to a discretization of two-dimensional Poisson's equation:

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y) \quad \text{on } [\alpha_x, \beta_x] \times [\alpha_y, \beta_y] \tag{22}$$

with the boundary conditions $u(x,\beta_y) = g_u$, $u(x,\alpha_y) = g_d$, $u(\alpha_x,y) = g_l$, and $u(\beta_x,y) = g_r$ where g_u, g_d, g_l, g_r are given functions. Notice that the two-dimensional Laplace's equation is a homogeneous case of the Poisson's equation when the RHS function is zero, i.e., $f(x,y) = 0$.

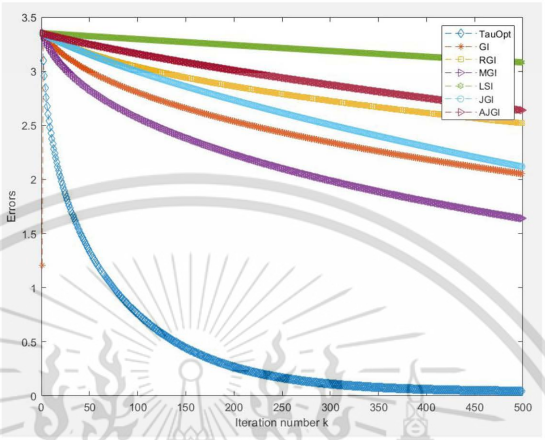

Figure 4 Errors for Example 6.2

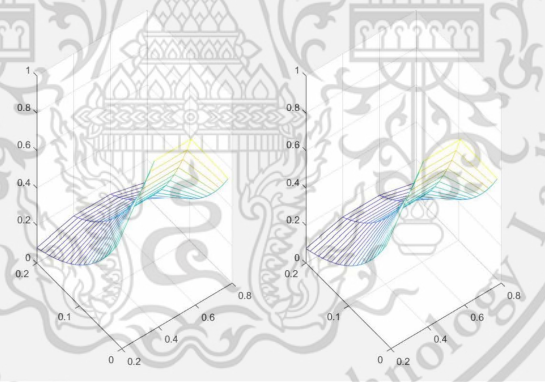

Figure 5 The 3D-plot of the analytical solution (left) and the numerical solution (right) for Example 6.2

7.1 Discretization with rectangular grid

We make discretization at the grid points in the rectangle which are at (x_i, y_j) with $x_i = \alpha_x + ih_x$ and $y_j = \alpha_y + jh_y$ where

$$h_x = \frac{\beta_x - \alpha_x}{N_x + 1} \quad \text{and} \quad h_y = \frac{\beta_y - \alpha_y}{N_y + 1}. \tag{23}$$

Table 4 Comparison of numerical and analytical results for Example 6.2

t	$u(x,t)$							
	$x=0.2$		$x=0.4$		$x=0.6$		$x=0.8$	
	Numer	Exact	Numer	Exact	Numer	Exact	Numer	Exact
0.01	0.5317	0.5325	0.8602	0.8617	0.8602	0.8617	0.5317	0.5325
0.02	0.4809	0.4825	0.7781	0.7807	0.7781	0.7807	0.4809	0.4825
0.03	0.4350	0.4371	0.7038	0.7073	0.7038	0.7073	0.4350	0.4371
0.04	0.3934	0.3961	0.6366	0.6408	0.6366	0.6408	0.3934	0.3961
0.05	0.3559	0.3588	0.5758	0.5806	0.5758	0.5806	0.3559	0.3588
0.06	0.3219	0.3251	0.5208	0.5261	0.5208	0.5261	0.3219	0.3251
0.07	0.2911	0.2946	0.4711	0.4766	0.4711	0.4766	0.2911	0.2946
0.08	0.2633	0.2669	0.4261	0.4318	0.4261	0.4318	0.2633	0.2669
0.09	0.2382	0.2418	0.3854	0.3912	0.3854	0.3912	0.2382	0.2418
0.10	0.2154	0.2191	0.3486	0.3545	0.3486	0.3545	0.2151	0.2191

Table 5 Errors and computational time for Example 6.2

Method	Error	CT
TauOpt	0.0445	0.0368
GI	2.0528	0.0072
RGI	2.5198	0.0076
MGI	1.6413	0.0078
LSI	3.0821	0.1155
JGI	2.1186	0.0072
AJGI	2.6405	0.0090

We denote $u_{ij} = u(x_i, y_j)$, $f_{ij} = f(x_i, y_j)$, as well as g_u, g_d, g_l, g_r. By the standard finite difference approximation, we obtain

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{ij} + u_{i,j+1}}{h_y^2}, \tag{24}$$

or equivalently,

$$h_y^2(2u_{ij} - u_{i-1,j} - u_{i+1,j}) + h_x^2(2u_{ij} - u_{i,j+1} - u_{i,j+1}) = -h_x^2 h_y^2 f_{ij},$$

for $1 \le i \le N_x$, $1 \le j \le N_y$. Now, we can convert the differential equation (22) to a linear system of $N_x N_y$ equations in $N_x N_y$ unknowns $u_{11}, \ldots, u_{N_x N_y}$:

$$(h_y^2 T_1 + h_x^2 T_2)\,\mathrm{Vec}(U) = -h_x^2 h_y^2\,\mathrm{Vec}[f_{ij}] + h_x^2(\overline{g_u} + \overline{g_d}) + h_y^2(\overline{g_l} + \overline{g_r}), \tag{25}$$

where $U = [u_{ij}]$, T_1 has $N_y \times N_y$ blocks of the form tridiag$(-1, 2, -1)$ of $N_x \times N_x$ on its diagonal and T_2 also has $N_y \times N_y$ blocks of the form $2I_{N_x}$ on its diagonal and $-I_{N_x}$ blocks on its off-diagonals. The boundary conditions produce constant vectors $\overline{g_u}, \overline{g_d}, \overline{g_l}, \overline{g_r}$ at the RHS of (25) as follows:

$$\overline{g_u} = \begin{bmatrix} g_{u_{x_1},\beta_y} & g_{u_{x_2},\beta_y} & \cdots & g_{u_{x_{N_x}},\beta_y} & 0 & \cdots & 0 \end{bmatrix}^T,$$

$$\overline{g_d} = \begin{bmatrix} 0 & \cdots & 0 & g_{d_{x_1},\alpha_y} & g_{d_{x_2},\alpha_y} & \cdots & g_{d_{x_{N_x}},\alpha_y} \end{bmatrix}^T,$$

$$\overline{g_l} = \begin{bmatrix} g_{\alpha_x,y_{N_y}} & 0 & \cdots & 0 & g_{\alpha_x,y_{N_y-1}} & 0 & \cdots & g_{\alpha_x,y_1} & 0 & \cdots & 0 \end{bmatrix}^T,$$

$$\overline{g_r} = \begin{bmatrix} 0 & \cdots & 0 & g_{\beta_x,y_{N_y}} & 0 & \cdots & 0 & g_{\beta_x,y_{N_y-1}} & 0 & \cdots & g_{\beta_x,y_1} \end{bmatrix}^T.$$

Note that, for the Laplace's equation, equation (25) will be reduced to

$$\left(h_y^2 T_1 + h_x^2 T_2\right) \operatorname{Vec}(U) = h_x^2(\overline{g_u} + \overline{g_d}) + h_y^2(\overline{g_l} + \overline{g_r}).$$

Equation (25) is formed as $AXB = C$ where $A = h_y^2 T_1 + h_x^2 T_2$, $X = \operatorname{Vec}(U)$, $B = I$ and $C = -h_x^2 h_y^2 \operatorname{Vec}([f_{ij}]) + h_x^2(\overline{g_u} + \overline{g_d}) + h_y^2(\overline{g_l} + \overline{g_r})$. According to Algorithm 3.1, we obtain an algorithm for the rectangular-grid case as follows:

Algorithm 7.1 *The gradient-descent iterative algorithm for solving two-dimensional Poisson's equation.*

Input step. *Input $N_x, N_y \in \mathbb{N}$ as numbers of partition.*

Initialization step. *Let h_x and h_y be as in (23) and for each $i = 1, \dots, N_x$ and $j = 1, \dots, N_y$, $f_{ij} = f(x_i, y_j)$ where $x_i = \alpha_x + ih_x$ and $y_j = \alpha_y + jh_y$. Compute $c = -h_x^2 h_y^2 \operatorname{Vec}[f_{ij}] + h_x^2(\overline{g_u} + \overline{g_d}) + h_y^2(\overline{g_l} + \overline{g_r})$, $s = T_N c$, $S = T_N^2$, $t = T_N s$, and $T = T_N S$ where $T_N = h_y^2 T_1 + h_x^2 T_2$. Choose $u(0) \in \mathbb{R}^{N_x N_y}$ and set $k := 0$.*

Updating step. *Compute*

$$\tau_{k+1} = \frac{\sum_{p=1}^{N_x N_y} (s_p - \sum_{q=1}^{N_x N_y} S_{pq} u_q(k))^2}{\sum_{p=1}^{N_x N_y} (t_p - \sum_{q=1}^{N_x N_y} T_{pq} u_q(k))^2},$$

$$u(k+1) = u(k) + \tau_{k+1}\big(s - Su(k)\big).$$

Set $k := k + 1$ and repeat the Updating step.

Here, we denote by s_p the pth entry of a vector s and by S_{pq} the (p, q)-entry of S. In case of solving the two-dimensional Laplace's equation, initially compute $c = h_x^2(\overline{g_u} + \overline{g_d}) + h_y^2(\overline{g_l} + \overline{g_r})$. To stop the algorithm, a reasonable stopping rule is $\|c - T_N u(k)\|_F^2 < \epsilon$ where ϵ is a small positive number. Since the coefficient matrix T_N is sparse, the error norm can be described more precisely:

$$\begin{aligned}\left\|c - T_N u(k)\right\|_F^2 &= \|c\|_F^2 - 2\operatorname{tr}\left(c^T T_N u(k)\right) + \left\|T_N u(k)\right\|_F^2 \\ &= \|c\|_F^2 - 2h_x^2 h_y^2 \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} h_y^2 f_{ij}(-u_{i-1,j} + 2u_{ij} - u_{i+1,j}) \\ &\quad + h_x^2 f_{ij}(-u_{i,j+1} + 2u_{ij} - u_{i,j-1}) + \left\|T_N u(k)\right\|_F^2.\end{aligned}$$

7.2 Discretization with square grid

Now, we consider the Poisson's equation (22) on the square $[\alpha, \beta] \times [\alpha, \beta]$ with the boundary condition $u = 0$ on the boundary of the square. In this case, $h := h_x = h_y$ and $N := N_x = N_y$ and hence

$$T_N = I_N \otimes T_r + T_r \otimes I_N,$$

where $T_r = \operatorname{tridiag}(-1, 2, -1) \in M_N$. Thereby, (25) can be transformed into

$$T_N \operatorname{Vec}(U) = -h^2 \operatorname{Vec}\left([f_{ij}]\right) + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}, \tag{26}$$

or equivalently, $T_rU + UT_r = G$ where $G = -h^2\,\mathrm{Vec}([f_{ij}]) + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}$. Thus (26) can be solved by Algorithm 4.1 where $P = T_N$.

To have the condition number of T_N, we consider the smallest and largest eigenvalues of T_r which are given respectively by (see, e.g., [42])

$$\lambda_1 = 2\left(1 - \cos\frac{\pi}{N+1}\right) \approx \left(\frac{\pi}{N+1}\right)^2, \qquad \lambda_N = 2\left(1 - \cos\frac{N\pi}{N+1}\right) \approx 4.$$

Since $T_N = I_N \otimes T_r + T_r \otimes I_N$, the eigenvalue of T_N is $\lambda_i + \lambda_j$ where $\lambda_i, \lambda_j \in \sigma(T_r)$. Thus, the condition number of T_N for large N is

$$\kappa_{T_N} = \frac{\lambda_N + \lambda_N}{\lambda_1 + \lambda_1} \approx \frac{4}{\pi^2}(N+1)^2. \tag{27}$$

Corollary 7.2 *The discretization (26) of the Poisson's equation (22) can be solved by using Algorithm 7.1 in which* $c = -h^2\,\mathrm{Vec}[f_{ij}] + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}$ *so that the approximate solution* $u(k)$ *converges to the exact solution* u^* *for any initial vector* $u(0)$*. The convergence rate of the algorithm is governed by* $\sqrt{1 - \kappa_{T_N}^{-2}}$*, where* κ_{T_N} *is given by (27). Moreover, the error estimates are given as follows*:

$$\|u(k) - u^*\|_F \le \kappa_{T_N}\left(1 - \kappa_{T_N}^{-2}\right)^{\frac{1}{2}}\|u(k-1) - u^*\|_F,$$
$$\|u(k) - u^*\|_F \le \kappa_{T_N}\left(1 - \kappa_{T_N}^{-2}\right)^{\frac{k}{2}}\|u(0) - u^*\|_F.$$

7.3 Numerical simulations for the Poisson's equation

Example 7.3 We consider an application of our algorithm to the two-dimensional Poisson's equation (22) with

$$f(x,y) = -2\pi^2 \sin(\pi x)\sin(\pi y), \quad 0 < x < 1, 0 < y < 1,$$

and the boundary condition $u = 0$ on the boundary of the rectangle. It is called a *Dirichlet problem*. We choose an initial vector $u(0) = 10^{-6}[1\ \cdots\ 1]^T$. We run Algorithm 7.1 with the rectangular grid of 10×20 which causes the size of the matrix T_N to be 200×200. The analytical solution is

$$u^*(x,y) = \sin(\pi x)\sin(\pi y).$$

In this example, we provide only a comparison of numerical and analytical solutions in Table 6, and a 3D-plot of both solutions in Fig. 6.

Example 7.4 Consider the two-dimensional Laplace's equation on $[0,1] \times [0,\pi]$ with the boundary conditions:

$$u(0,y) = \sin y, \qquad u(1,y) = e\sin y, \qquad u(x,0) = 0, \qquad u(x,\pi) = 0.$$

We run Algorithm 7.1 with the initial vector $u(0) = [1\ \cdots\ 1]^T$. We choose two grid partitions: one has $h_x = 0.25$, $h_y = \pi/4$ and the other has $h_x = 0.0625$, $h_y = \pi/32$. So the sizes

Table 6 Comparison of numerical and analytical results for Example 7.3

y	$u(x,y)$							
	$x = 0.3636$		$x = 0.5454$		$x = 0.7272$		$x = 0.9090$	
	Numer	Exact	Numer	Exact	Numer	Exact	Numer	Exact
0.1904	0.5136	0.5124	0.5588	0.5576	0.4267	0.4257	0.1587	0.1591
0.3808	0.8488	0.8468	0.9236	0.9214	0.7052	0.7035	0.2629	0.2623
0.5712	0.8889	0.8868	0.9673	0.9650	0.7386	0.7368	0.2753	0.2747
0.7616	0.6202	0.6187	0.6749	0.6732	0.5153	0.5140	0.1921	0.1916
0.9520	0.1359	0.1356	0.1479	0.1475	0.1129	0.1126	0.0423	0.0420

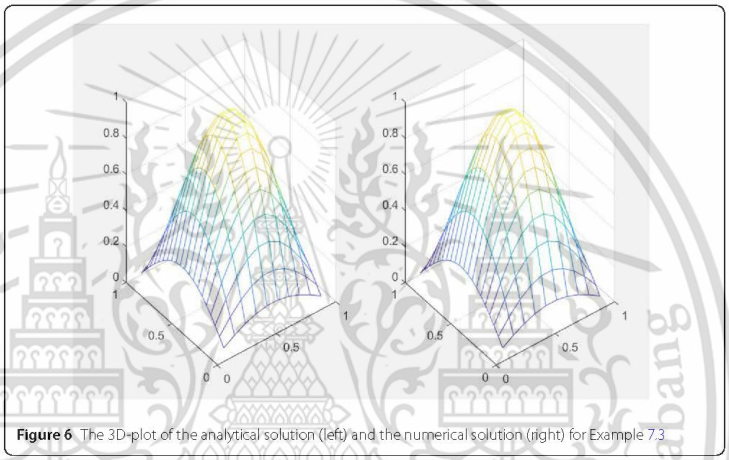

Figure 6 The 3D-plot of the analytical solution (left) and the numerical solution (right) for Example 7.3

Table 7 comparison of numerical and analytical results for Example 7.4

	Exact	$h_x = 0.25, h_y = \pi/4$		$h_x = 0.0625, h_y = \pi/32$	
		Numerical	Error (%)	Numerical	Error (%)
$u(0.25, \pi/4)$	0.9079	0.9131	0.57	0.9080	0.01
$u(0.50, \pi/2)$	1.6487	1.6593	0.64	1.6489	0.01
$u(0.75, 3\pi/4)$	1.4969	1.5031	0.41	1.4971	0.01

of the matrix T_N are 9×9 and 465×465, respectively. A comparison of numerical and analytical results is shown in Table 7. Figure 7 displays a 3D-plot of the numerical and the analytical results for the latter grid partition. Note that the analytical solution is

$$u^*(x, y) = e^x \sin y.$$

8 Conclusion

The proposed gradient-descent based iterative algorithm is well suited for solving the generalized Sylvester matrix equation, $\sum_{t=1}^{p} A_t X B_t = C$. Such matrix equation can be reduced to a class of well-known linear matrix equations such as the Sylvester equation, the Kalman–Yakubovich equation, and so on. The proposed algorithm is applicable for any problems as long as A_t and B_t have full column-rank and full row-rank, respectively, for all t. The convergence rate of the algorithm is governed by $\sqrt{1-\kappa^{-2}}$ where κ is the

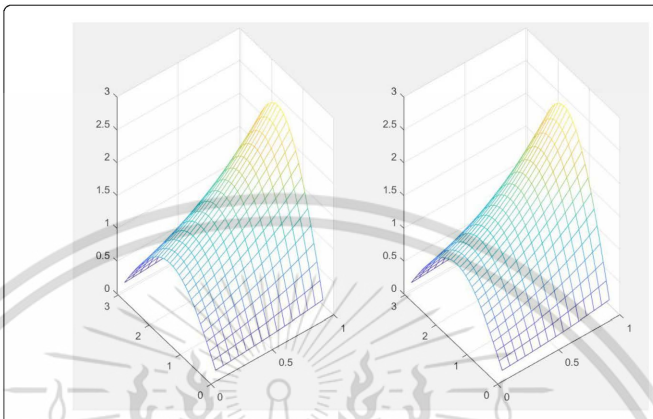

Figure 7 The 3D-plot of the analytical solution (left) and the numerical solution (right) for Example 7.4

condition number of $\sum_{t=1}^{p} B_t^T \otimes A_t$. As applications, our algorithm can be adapted to the discretization of the one-dimensional heat equation and the two-dimensional Poisson's equation. According to numerical simulations, our algorithms converge fast to the exact solution in spite of a little more computational time compared to other methods. The numerical examples for heat and Poisson's equations in Sects. 6 and 7 guarantee the capability and adaptability of our proposed algorithms.

Acknowledgements
This work was supported by King Mongkut's Institute of Technology Ladkrabang.

Funding
The first author received financial support from KMITL Doctoral Scholarships, grant no. KDS 2019/022 during his PhD study.

Availability of data and materials
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors contributed equally and significantly in writing this article. All authors read and approved the final manuscript.

Publisher's Note

Received: 20 March 2020 Accepted: 19 June 2020 Published online: 01 July 2020

References

1. Wimmer, H.K.: Consistency of a pair of generalized Sylvester equations. IEEE Trans. Autom. Control **39**(5), 1014–1016 (1994). https://doi.org/10.1109/9.284883
2. Wu, A., Duan, G., Xue, Y.: Kronecker maps and Sylvester-polynomial matrix equations. IEEE Trans. Autom. Control **52**(5), 905–910 (2007). https://doi.org/10.1109/TAC.2007.895906
3. Wu, A., Duan, G., Zhou, B.: Solution to generalized Sylvester matrix equations. IEEE Trans. Autom. Control **53**(3), 811–815 (2008). https://doi.org/10.1109/TAC.2008.919562
4. Geir, E.D., Fernando, P.: A Course in Robust Control Theory: A Convex Approach. Springer, New York (1999)
5. Benner, P., Quintana-Ortí, E.S.: Solving stable generalized Lyapunov equations with the matrix sign function. Numer. Algorithms **20**(1), 75–100 (1999). https://doi.org/10.1023/A:1019191431273

6. Isak, J., Bo, K.: Recursive blocked algorithms for solving triangular systems—part I: one-sided and couple Sylvester-type matrix equations. ACM Trans. Math. Softw. **28**(4), 392–415 (2002). https://doi.org/10.1145/592843.592845
7. Isak, J., Bo, K.: Recursive blocked algorithms for solving triangular systems—part II: two-sided and generalized Sylvester and Lyapunov matrix equations. ACM Trans. Math. Softw. **28**(4), 416–435 (2002). https://doi.org/10.1145/592843.592846
8. Amer, K., Asghar, K., Faezeh, T.: A new version of successive approximations method for solving Sylvester matrix equations. Appl. Math. Comput. **186**(1), 638–645 (2006). https://doi.org/10.1016/j.amc.2006.08.007
9. Li, Y.-Q.: Implicity restarted global FOM and GMRES for nonsymmetric matrix equations and Sylvester equations. Appl. Math. Comput. **167**(2), 1004–1025 (2005). https://doi.org/10.1016/j.amc.2004.06.141
10. Bai, Z.: On Hermitian and skew-Hermitian splitting iteration methods for continuous Sylvester equation. J. Comput. Math. **29**(2), 185–198 (2011). https://doi.org/10.4208/jcm.1009-m3152
11. Dehghan, M., Shirilord, A.: A generalized modified Hermitian and skew-Hermitian splitting (GMHSS) method for solving complex Sylvester matrix equation. Appl. Math. Comput. **348**, 632–651 (2019). https://doi.org/10.1016/j.amc.2018.11.064
12. Dehghan, M., Hajarian, M.: An iterative algorithm for the reflexive solutions of the generalized coupled Sylvester matrix equations and its optimal approximation. Appl. Math. Comput. **202**(2), 571–588 (2008). https://doi.org/10.1016/j.amc.2008.02.035
13. Dehghan, M., Hajarian, M.: An iterative method for solving the generalized coupled Sylvester matrix equations over generalized bisymmetric matrices. Appl. Math. Model. **34**(3), 639–654 (2010). https://doi.org/10.1016/j.apm.2009.06.018
14. Dehghan, M., Shirilord, A.: The double-step scale splitting method for solving complex Sylvester matrix equation. Comput. Appl. Math. **38**, 146 (2019). https://doi.org/10.1007/s40314-019-0921-6
15. Dehghan, M., Shirilord, A.: Solving complex Sylvester matrix equation by accelerated double-step scale splitting (ADSS) method. Eng. Comput. (2019). https://doi.org/10.1007/s00366-019-00838-6
16. Ding, F., Chen, T.: Hierarchical gradient-based identification of multivariable discrete-time systems. Automatica **41**(2), 315–325 (2005). https://doi.org/10.1016/j.automatica.2004.10.010
17. Ding, F., Chen, T.: Hierarchical least squares identification methods for multivariable systems. IEEE Trans. Autom. Control **50**(3), 397–402 (2005). https://doi.org/10.1109/TAC.2005.843856
18. Ding, F., Chen, T.: Iterative least-squares solutions of coupled Sylvester matrix equations. Syst. Control Lett. **54**(2), 95–107 (2005). https://doi.org/10.1016/j.sysconle.2004.06.008
19. Ding, F., Chen, T.: On iterative solutions of general coupled matrix equations. SIAM J. Control Optim. **44**(6), 2269–2284 (2006). https://doi.org/10.1137/S0363012904441350
20. Xie, L., Ding, J., Ding, F.: Gradient based iterative solutions for general linear matrix equations. Comput. Math. Appl. **58**(7), 1441–1448 (2009). https://doi.org/10.1016/j.camwa.2009.06.047
21. Xie, L., Liu, Y., Yang, H.: Gradient based and least squares based iterative algorithms for matrix equations $AXB + CX^TD = F$. Appl. Math. Comput. **217**(5), 2191–2199 (2010). https://doi.org/10.1016/j.amc.2010.07.019
22. Ding, J., Liu, Y., Ding, F.: Iterative solutions to matrix equations of the form $A_iXB_i = F_i$. Comput. Math. Appl. **59**(11), 3500–3507 (2010). https://doi.org/10.1016/j.camwa.2010.03.041
23. Ding, F., Zhang, H.: Brief paper—gradient-based iterative algorithm for a class of the coupled matrix equations related to control systems. IET Control Theory Appl. **8**(15), 1588–1595 (2014). https://doi.org/10.1049/iet-cta.2013.1044
24. Xie, Y.-J., Ma, C.-F.: The accelerated gradient based iterative algorithm for solving a class of generalized Sylvester-transpose matrix equation. Appl. Math. Comput. **273**, 1257–1269 (2016). https://doi.org/10.1016/j.amc.2015.07.022
25. 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**, 1–8 (2017). https://doi.org/10.1155/2017/1624969
26. Zhu, M., Zhang, G., Qi, Y.: On single-step HSS iterative method with circulant preconditioner for fractional diffusion equations. Adv. Differ. Equ. **2019**, 422 (2019). https://doi.org/10.1186/s13662-019-2353-4
27. Sun, M., Wang, Y., Liu, J.: Two modified least-squares iterative algorithms for the Lyapunov matrix equations. Adv. Differ. Equ. **2019**, 305 (2019). https://doi.org/10.1186/s13662-019-2253-7
28. Kittisopaporn, A., Chansangiam, P.: The steepest descent of gradient-based iterative method for solving rectangular linear systems with an application to Poisson's equation. Adv. Differ. Equ. **2020**, 259 (2020). https://doi.org/10.1186/s13662-020-02715-9
29. Ding, F., Zhang, X., Xu, L.: The innovation algorithms for multivariable state-space models. Int. J. Adapt. Control Signal Process. **33**, 1601–1618 (2019). https://doi.org/10.1002/acs.3053
30. Ding, F., Lv, L., Pan, J., et al.: Two-stage gradient-based iterative estimation methods for controlled autoregressive systems using the measurement data. Int. J. Control. Autom. Syst. **18**, 886–896 (2020). https://doi.org/10.1007/s12555-019-0140-3
31. Ding, F., Xu, L., Meng, D., et al.: Gradient estimation algorithms for the parameter identification of bilinear systems using the auxiliary model. J. Comput. Appl. Math. **369**, 112575 (2020). https://doi.org/10.1016/j.cam.2019.112575
32. Ding, F., Chen, T.: Gradient based iterative algorithms for solving a class of matrix equations. IEEE Trans. Autom. Control **50**(8), 1216–1221 (2005). https://doi.org/10.1109/TAC.2005.852558
33. Ding, F., Liu, X.P., Ding, J.: Iterative solutions of the generalized Sylvester matrix equations by using the hierarchical identification principle. Appl. Math. Comput. **197**(1), 41–50 (2008). https://doi.org/10.1016/j.amc.2007.07.040
34. Niu, Q., Wang, X., Lu, L.: A relaxed gradient based algorithm for solving Sylvester equation. Asian J. Control **13**(3), 461–464 (2011). https://doi.org/10.1002/asjc.328
35. Wang, X., Dai, L., Liao, D.: A modified gradient based algorithm for solving Sylvester equation. Appl. Math. Comput. **218**(9), 5620–5628 (2012). https://doi.org/10.1016/j.amc.2011.11.055
36. Tian, Z., Tian, M., et al.: An accelerated Jacobi-gradient based iterative algorithm for solving Sylvester matrix equations. Filomat **31**(8), 2381–2390 (2017). https://doi.org/10.2298/FIL1708381T
37. Slavko, V., Petar, S.: 2D BEM analysis of power cables thermal field. Int. J. Eng. Model. **19**(1–4), 87–94 (2006)
38. Yildirim, S.: Exact and numerical solutions of Poisson equation for electrostatic potential problems. Math. Probl. Eng. **2008**, 578723 (2008). https://doi.org/10.1155/2008/578723

39. Horn, R., Johnson, C.: Topics in Matrix Analysis. Cambridge University Press, New York (1991)
40. Horn, R., Johnson, C.: Matrix Analysis. Cambridge University Press, New York (1990)
41. Stephen, P.B., Lieven, V.: Convex Optimization. Cambridge University Press, Cambridge (2004)
42. James, W.D.: Applied Numerical Linear Algebra. Society for Industrial and Applied Mathematics, Philadelphia (1997)

Appendix B

The research paper 2

Kittisopaporn and Chansangiam *Advances in Difference Equations* (2021) 2021:266
https://doi.org/10.1186/s13662-021-03427-4

Advances in Difference Equations
a SpringerOpen Journal

RESEARCH Open Access

Approximated least-squares solutions of a generalized Sylvester-transpose matrix equation via gradient-descent iterative algorithm

Adisorn Kittisopaporn[1] and Pattrawut Chansangiam[1*]

*Correspondence:
pattrawut.ch@kmitl.ac.th
[1]Department of Mathematics, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, 10520 Bangkok, Thailand

Abstract

This paper proposes an effective gradient-descent iterative algorithm for solving a generalized Sylvester-transpose equation with rectangular matrix coefficients. The algorithm is applicable for the equation and its interesting special cases when the associated matrix has full column-rank. The main idea of the algorithm is to have a minimum error at each iteration. The algorithm produces a sequence of approximated solutions converging to either the unique solution, or the unique least-squares solution when the problem has no solution. The convergence analysis points out that the algorithm converges fast for a small condition number of the associated matrix. Numerical examples demonstrate the efficiency and effectiveness of the algorithm compared to renowned and recent iterative methods.

MSC: 15A60; 15A69; 26B25; 65F45

Keywords: Generalized Sylvester-transpose matrix equation; Gradient descent; Iterative method; Least-squares solution

1 Introduction

In differential equations and control engineering, there has been much attention for the following linear matrix equations:

$$AXB = C, \tag{1}$$

$$AX + XA^T = B: \quad \text{Lyapunov equation,} \tag{2}$$

$$AX + XB = C: \quad \text{Sylvester equation,} \tag{3}$$

$$AXB + CXD = E: \quad \text{a generalized Sylvester equation,} \tag{4}$$

$$AXB + CX^TD = E: \quad \text{a generalized Sylvester-transpose equation,} \tag{5}$$

$$X + AXB = C: \quad \text{Stein equation,} \tag{6}$$

$$X + AX^TB = C: \quad \text{Stein-transpose equation.} \tag{7}$$

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

$$\sum_{t=1}^{p} A_t X B_t + \sum_{s=1}^{q} C_s X^T D_s = E, \tag{8}$$

where, for each $t = 1, \dots, p$, $A_t \in \mathbb{R}^{l \times m}$, $B_t \in \mathbb{R}^{n \times r}$, for each $s = 1, \dots, q$, $C_s \in \mathbb{R}^{l \times n}$, $D_s \in \mathbb{R}^{m \times r}$, $E \in \mathbb{R}^{l \times r}$ are known matrices whereas $X \in \mathbb{R}^{m \times n}$ is the matrix to be determined. These equations play important roles in control and system theory, robust simulation, neural network, and statistics; see e.g. [1–4].

A traditional method of finding their exact solutions is to use the Kronecker product of a matrix and the vectorization to reduce the matrix equation to a linear system; see e.g. [5, Ch. 4]. However, the dimension of the linear system can be very large due to the Kronecker multiplication, so that the step of finding the inversion of the associated matrix will result in excessive computer storage memory. For that reason, iterative approaches have received much attention. The conjugate gradient (CG) is an interesting idea to formulate finite-step iterative procedures to obtain the exact solution at the final step. There are variants of CG method for solving linear matrix equations, namely, the generalized conjugate direction method (GCD) [6], the conjugated gradient least-squares method (CGLs) [7], generalized product-type methods based on a bi-conjugate gradient (GPBi) [8]. Another interesting idea to create an iterative method is to use Hermitian and skew-Hermitian splitting (HSS); see e.g. [9].

A group of methods, called gradient-based iterative methods, aim to construct a sequence of approximated solutions that converges to the exact solution for any given initial matrices. These methods are derived from the minimization of associated norm-error functions using gradients, and the hierarchical identification. Such techniques have stimulated and have played a role in many pieces of research in a few decades. In 2005, Ding and Chen [10] proposed a gradient-based iterative (GI) method for solving Eqs. (3), (4), and (6). Ding et al. [11] proposed the GI and the least-squares iterative (LSI) methods for solving $\sum_{j=1}^{p} A_j X B_j = F$ which includes Eqs. (1) and (4). Niu et al. [12] developed a relaxed gradient-based iterative (RGI) method for solving Eq. (3) by introducing a weighted factor. The MGI method, developed by Wang et al. [13], is a half-step-update modification of the GI method. Zhaolu et al. [14] presented two methods for solving Eq. (3). The first method is based on the GI method and called the Jacobi gradient iterative (JGI) method. Furthermore, they introduced relaxation factors to accelerate the speed of convergence and called the accelerated Jacobi gradient iterative (AJGI) method. Recently, Sun et al. (2019, [15]) proposed two modified least-squares iterative algorithms namely, LSIA1 [15, Theorem 2.3] and LSIA2 [15, Theorem 3.1] for the Lyapunov equation (2). See more algorithms in [16–24]. The developed iterative methods can be applied to state-space models [25], controlled autoregressive systems [26], and parameter estimation in signal processing [27].

Let us focus on gradient-based iterative methods for solving Eqs. (5) and (8). A recent gradient iterative method for Eq. (5) is AGBI method, developed in [28]. The following two methods were proposed to produce the sequence $X(k)$ of approximated solutions converging to the exact solution X^* of Eq. (8).

Method 1.1 ([29]) *Gradient iterative (GI) method.*

$$X(k) = \frac{1}{p+q}\left(\sum_{j=1}^{p} X_j(k) + \sum_{l=1}^{q} X_{p+l}(k)\right),$$

$$X_j(k) = X(k-1) + \mu A_j^T\left(E - \sum_{i=1}^{p} A_i X(k-1) B_i - \sum_{i=1}^{q} C_i X^T(k-1) D_i\right) B_j^T,$$

$$X_{p+l}(k) = X(k-1) + \mu D_l\left(E - \sum_{i=1}^{p} A_i X(k-1) B_i - \sum_{i=1}^{q} C_i X^T(k-1) D_i\right)^T C_l.$$

A conservative choice of the convergence factor μ is

$$0 < \mu < 2\left(\sum_{j=1}^{p} \lambda_{\max}\left(A_j A_j^T\right)\lambda_{\max}\left(B_j^T B_j\right) + \sum_{l=1}^{q} \lambda_{\max}\left(C_l C_l^T\right)\lambda_{\max}\left(D_l^T D_l\right)\right).$$

Method 1.2 ([29]) *Least-squares iterative (LSI) method.*

$$R(k) = E - \sum_{i=1}^{p} A_i X(k-1) B_i - \sum_{i=1}^{q} C_i X(k-1) D_i,$$

$$X_j(k) = X(k-1) + \mu\left(A_j^T A_j\right)^{-1} A_j^T R(k) B_j^T\left(B_j B_j^T\right)^{-1},$$

$$X_{p+l}(k) = X(k-1) + \mu\left(D_l D_l^T\right)^{-1} D_l R(k) C_l\left(C_l^T C_l\right)^{-1},$$

$$X(k) = \frac{1}{p+q}\left(\sum_{j=1}^{p} X_j(k) + \sum_{l=1}^{q} X_{p+l}(k)\right), \quad 0 < \mu < 2(p+q).$$

In this work, we introduce a new iterative algorithm based on gradient-descent for solving Eq. (8). The techniques of gradient and steepest descent let us obtain the search direction and the step sizes. Indeed, our varied step sizes are the optimal convergence factors that guarantee the algorithm to have a minimum error at each iteration. Our convergence analysis proves that, when Eq. (8) has a unique solution, the algorithm constructs a sequence of approximated solutions converging to the exact solution. On the other hand, when Eq. (8) has no solution, the generated sequence converges to the unique least-squares solution. We provide the convergence rate to show that the speed of convergence depends on the condition number of the associated certain matrix. In addition, we have an error analysis that gives an error estimation comparing the current iteration with the preceding and the initial iterations. Finally, we provide numerical simulations to guarantee the efficiency and effectiveness of our algorithm. The illustrative examples show that our algorithm is applicable to both Eq. (8) and its certain interesting special cases.

The organization of this paper is as follows. In Sect. 2, we recall the criterion for the matrix equation (8) to have a unique solution or a unique least-squares solution, via the Kronecker linearization. We propose the gradient-descent algorithm to solve Eq. (8) in Sect. 3. The proof of convergence criteria, convergence rates, and error estimation for the proposed algorithm are provided in Sect. 4. In Sect. 5, we present the comparison of the efficiency of our proposed algorithm to well-known and recent iterative algorithms.

In the remainder of this paper, all vectors and matrices are real. Denote the set of n columns vectors by $\mathbb{R}^n$ and the set of $m \times n$ matrices by $\mathbb{R}^{m\times n}$. The (i,j)th entry of a matrix

A is denoted by $A(i,j)$ or a_{ij}. To perform a convergence analysis, we use the Frobenius norm, the spectral norm, and the (spectral) condition number of $A \in \mathbb{R}^{m\times n}$, which are, respectively, defined by

$$\|A\|_F = \sqrt{\operatorname{tr}(A^TA)}, \qquad \|A\|_2 = \sqrt{\lambda_{\max}(A^TA)}, \qquad \kappa(A) = \left(\frac{\lambda_{\max}(A^TA)}{\lambda_{\min}(A^TA)}\right)^{1/2}.$$

2 Exact and least-squares solutions of the matrix equation by the Kronecker linearization

In this section, we explain how to solve the generalized Sylvester-transpose matrix equation (8) directly using the Kronecker linearization.

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 $A \otimes B = [a_{ij}B] \in \mathbb{R}^{mp\times nq}$. The vector operator $\operatorname{Vec}(\cdot)$ turns each matrix $A = [a_{ij}] \in \mathbb{R}^{m\times n}$ to the vector

$$\operatorname{Vec}(A) = \begin{bmatrix} a_{11}\dots a_{m1} & a_{12}\dots a_{m2} & \dots & a_{1n}\dots a_{mn} \end{bmatrix}^T \in \mathbb{R}^{mn}.$$

Lemma 2.1 (e.g. [5]) *For compatible matrices A, B, and C, we have the following properties of the Kronecker product and the vector operator.*

(i) $(A \otimes B)^T = A^T \otimes B^T$,

(ii) $\operatorname{Vec}(ABC) = (C^T \otimes A)\operatorname{Vec}(B)$.

Recall also that there is a permutation matrix $P(m,n) \in \mathbb{R}^{mn\times mn}$ such that

$$\operatorname{Vec}(X^T) = P(m,n)\operatorname{Vec}(X) \quad \text{for all } X \in \mathbb{R}^{m\times n}. \tag{9}$$

This matrix depends only on the dimensions m and n and is given by

$$P(m,n) = \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij} \otimes E_{ij}^T,$$

where E_{ij} has entry 1 in (i,j)th position and all other entries are 0.

Now, we can transform Eq. (8) to an equivalent linear system by applying the vector operator and utilizing Lemma 2.1(ii) and the property (9). Indeed, we get the linear system

$$Q\operatorname{Vec}(X) = \operatorname{Vec}(E), \tag{10}$$

where

$$Q = \sum_{t=1}^{p}(B_t^T \otimes A_t) + \sum_{s=1}^{q}(D_s^T \otimes C_s)P(m,n) \in \mathbb{R}^{rl\times mn}. \tag{11}$$

Thus Eq. (8) has a (unique) solution if and only if Eq. (10) does. We impose the assumption that Q is of full column-rank, or equivalently, Q^TQ is invertible.

If Eq. (8) has a solution, then we obtain the exact (vector) solution to be

$$\operatorname{Vec}(X^*) = (Q^TQ)^{-1}Q^T\operatorname{Vec}(E). \tag{12}$$

If Eq. (8) has no solution, then we can seek for a least-squares solution, i.e. a matrix X^* that minimizes the squared Frobenius norm $\|Q\,\mathrm{Vec}(X) - \mathrm{Vec}(E)\|_F^2$. The assumption on Q implies that the least-squares solution for Eq. (8) is uniquely determined by the solution of the associated normal equation, and it is also given by Eq. (12). In this case, the least-squares error is given by

$$\begin{aligned}\left\|Q\,\mathrm{Vec}\left(X^*\right) - \mathrm{Vec}(E)\right\|_F^2 &= \left\|\mathrm{Vec}(E)\right\|_F^2 - \mathrm{Vec}^T(E)Q\,\mathrm{Vec}\left(X^*\right)\\ &= \|E\|_F^2 - \mathrm{Vec}^T(E)Q\left(Q^TQ\right)^{-1}Q^T\,\mathrm{Vec}(E). \end{aligned} \tag{13}$$

We denote both the exact and the least-squares solutions of Eq. (8) by X^*.

3 Gradient-descent iterative solutions for the matrix equation

This section is intended to propose a new iterative algorithm for creating a sequence $\{X_k\}$ of well-approximated solutions of Eq. (8) that converges to the exact or least-squares solution X^*. This algorithm will be applicable if the matrix Q is of full column-rank, no matter Eq. (8) has a solution or not.

Our aim is to generate a sequence $\{x_k\}$, starting from an initial vector x_0, using the recurrence

$$x_{k+1} = x_k + \tau_{k+1}d_k, \quad k = 0, 1, \ldots,$$

where x_k is the kth approximation, $\tau_{k+1} > 0$ is the step size, and d_k is the search direction. To obtain the search direction, we consider the Frobenius-norm error $\|\sum_{t=1}^{p} A_tXB_t + \sum_{s=1}^{q} C_sX^TD_s - E\|_F$ which is then transformed into $\|Qx - \mathrm{Vec}(E)\|_F$ via Lemma 2.1(ii) and $x = \mathrm{Vec}(X)$. Let $f : \mathbb{R}^{mn} \to \mathbb{R}$ be the norm-error function defined by

$$f(x) := \frac{1}{2}\left\|Qx - \mathrm{Vec}(E)\right\|_F^2.$$

It is easily seen that f is convex. Hence, the gradient-descent iterative method can be shown as the following recursive equation:

$$x_{k+1} = x_k - \tau_{k+1}\nabla f(x_k).$$

To find the gradient of the function f, the following properties of the matrix trace will be used:

$$\frac{d}{dX}\,\mathrm{tr}(AX) = A^T,$$
$$\frac{d}{dX}\,\mathrm{tr}\left(XAX^TB\right) = BXA + B^TXA^T.$$

By letting $\tilde{e} = \mathrm{Vec}(E)$, we compute the derivative of f as follows:

$$\begin{aligned}\nabla f(x) &= \frac{1}{2}\frac{d}{dx}\,\mathrm{tr}\left((Qx - \tilde{e})^T(Qx - \tilde{e})\right)\\ &= \frac{1}{2}\frac{d}{dx}\,\mathrm{tr}\left(Qxx^TQ^T - \tilde{e}x^TQ^T - Qx\tilde{e}^T + \tilde{e}\tilde{e}^T\right)\end{aligned}$$

$$= \frac{1}{2}\left(Q^TQx + Q^TQx - Q^T\tilde{e} - Q^T\tilde{e}\right)$$
$$= Q^T(Qx - \tilde{e}). \tag{14}$$

Thus, we have the new form of the iterative equation as follows:

$$x_{k+1} = x_k + \tau_{k+1}Q^T(\tilde{e} - Qx_k).$$

The above equation can be transformed into matrix form via Lemma 2.1(ii), i.e.,

$$X_{k+1} = X_k + \tau_{k+1}\left(\sum_{t=1}^{p} A_t^T R_k B_t^T + \sum_{s=1}^{q} D_s R_k^T C_s\right)$$

where $R_k = E - \sum_{t=1}^{p} A_t X_k B_t - \sum_{s=1}^{q} C_s X_k^T D_s$.

To choose a step size, we define $\phi_{k+1} : [0,\infty) \to \mathbb{R}$ by for each $k \in \mathbb{N} \cup \{0\}$,

$$\phi_{k+1}(\tau) = f(x_{k+1}) = \frac{1}{2}\|Qx_{k+1} - \tilde{e}\|_F^2$$
$$= \frac{1}{2}\left\|\tau QQ^T(\tilde{e} - Qx_k) + Qx_k - \tilde{e}\right\|_F^2.$$

We differentiate ϕ_{k+1} by using the properties of a matrix trace and obtain

$$\frac{d}{d\tau}\phi_{k+1}(\tau) = \tau_{k+1}\left\|QQ^T(\tilde{e} - Qx_k)\right\|_F^2 - \left\|Q^T(\tilde{e} - Qx_k)\right\|_F^2.$$

It is obvious that the second-order derivative of ϕ_{k+1} is $\|QQ^T(\tilde{e} - Qx_k)\|_F^2$ which is a positive constant. So when $\frac{d}{d\tau}\phi_{k+1}(\tau) = 0$, we get the minimizer of ϕ_{k+1}, i.e.

$$\tau_{k+1} = \frac{\|Q^T(\tilde{e} - Qx_k)\|_F^2}{\|QQ^T(\tilde{e} - Qx_k)\|_F^2}$$
$$= \frac{\|\operatorname{Vec}(W_k)\|_F^2}{\|\operatorname{Vec}(\sum_{t=1}^{p} A_t W_k B_t + \sum_{s=1}^{q} C_s W_k^T D_s)\|_F^2}$$
$$= \frac{\|W_k\|_F^2}{\|\sum_{t=1}^{p} A_t W_k B_t + \sum_{s=1}^{q} C_s W_k^T D_s\|_F^2}.$$

Here $W_k = \sum_{t=1}^{p} A_t^T R_k B_t^T + \sum_{s=1}^{q} C_s^T R_k D_s^T$.

An implementation of the gradient-descent iterative algorithm for solving Eq. (8) is given by the following algorithm where the search direction and the step size are taken into account. To terminate the algorithm, one can alternatively set the stopping rule to be $\|R_k\|_F - \delta < \epsilon$ where $\epsilon > 0$ is a small error and δ is the least-squares error described in Eq. (13).

4 Convergence analysis of the proposed algorithm

In this section, Algorithm 1 will be proved to converge to the exact solution or the unique least-squares solution. Recall the next lemma.

Algorithm 1: The gradient-descent iterative algorithm for Eq. (8)

$A_t, B_t, C_s, D_s, E, X_0$;
for $k = 1, \dots, n$ **do**
  $R_k = E - \sum_{t=1}^{p} A_t X_k B_t - \sum_{s=1}^{q} C_s X_k^T D_s$;
  $W_k = \sum_{t=1}^{p} A_t^T R_k B_t^T + \sum_{s=1}^{q} C_s^T R_k D_s^T$;
  $\tau_{k+1} = \|W_k\|_F^2 / \| \sum_{t=1}^{p} A_t W_k B_t + \sum_{s=1}^{q} C_s W_k^T D_s \|_F^2$;
  $X_{k+1} = X_k + \tau_{k+1}(\sum_{t=1}^{p} A_t^T R_k B_t^T + \sum_{s=1}^{q} D_s R_k^T C_s)$
end

Lemma 4.1 ([30]) *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a strongly convex function, i.e. there exist two non-negative constants ψ, Ψ such that $\psi I \le \nabla^2 f(x) \le \Psi I$ for all $x \in \mathbb{R}^n$. Then, for any $x, y \in \mathbb{R}^n$,*

$$f(y) \ge f(x) + \nabla f(x)^T (y - x) + \frac{\psi}{2} \|y - x\|_F^2, \tag{15}$$

$$f(y) \le f(x) + \nabla f(x)^T (y - x) + \frac{\Psi}{2} \|y - x\|_F^2. \tag{16}$$

The following definition is an extension of the Frobenius norm and will be used in the convergence analysis.

Definition 4.2 Given a full-column-rank matrix $P \in \mathbb{R}^{k \times n}$, we define the P-weighted Frobenius norm of $A \in \mathbb{R}^{m \times n}$ by

$$\|A\|_P := \|PA\|_F = \sqrt{\operatorname{tr}(A^T P^T P A)}. \tag{17}$$

Theorem 4.3 *Consider Eq. (8). Assume that Q is of full column-rank.*

(i) *Suppose Eq. (8) has a solution (and thus, the solution is unique). Then, for any initial matrix X_0, the sequence X_k of approximated solutions generated by Algorithm 1 converges to the exact solution X^*.*

(ii) *Suppose Eq. (8) has no solution (and thus, it has the unique least-squares solution X^*). Then $\|X_k\|_Q \to \|X^*\|_Q$ for any initial matrix X_0. Here, $\|\cdot\|_Q$ is the Q-weighted Frobenius norm defined by Eq. (17).*

Proof Since $x^* = \operatorname{Vec}(X^*)$ is the optimal solution of $\min_{x \in \mathbb{R}^{mn}} f(x)$, we denote the minimum value, $\inf_{x \in \mathbb{R}^{mn}} f(x) = f(x^*)$ as δ. Note that δ is equal to the least-squares error determined by Eq. (13) and is zero if X^* is the unique exact solution. If there exists $k \in \mathbb{N}$ such that $\nabla f(x_k) = 0$, then $X_k = X^*$ and the result holds. To investigate the convergence of the algorithm, we assume that $\nabla f(x_k) \neq 0$ for all k. Considering the strong convexity of f, we have from Eq. (14) $\nabla^2 f(x_k) = Q^T Q$. Let $\lambda_{\min}$ ($\lambda_{\max}$) be the minimum (maximum) eigenvalue of $Q^T Q$, respectively. Since $Q^T Q$ is symmetric, we have

$$\lambda_{\min} I \le \nabla^2 f(x_k) \le \lambda_{\max} I.$$

Thus, f is strongly convex. From (15), substituting $y = x_{k+1}$ and $x = x_k$ yields

$$f(y) \ge f(x_k) - \tau \left\| \nabla f(x_k) \right\|_F^2 + \frac{\lambda_{\min} \tau^2}{2} \left\| \nabla f(x_k) \right\|_F^2.$$

We minimize the RHS by taking $\tau = 1/\lambda_{\min}$, so that

$$f(y) \geq f(x_k) - \frac{1}{2\lambda_{\min}} \left\| \nabla f(x_k) \right\|_F^2.$$

Since the above equation is true for all $y \in \mathbb{R}^{mn}$, we have

$$\delta \geq f(x_k) - \frac{1}{2\lambda_{\min}} \left\| \nabla f(x_k) \right\|_F^2. \tag{18}$$

Similarly, from (16), we have

$$f(x_{k+1}) \leq f(x_k) - \tau \left\| \nabla f(x_k) \right\|_F^2 + \frac{\lambda_{\max} \tau^2}{2} \left\| \nabla f(x_k) \right\|_F^2.$$

Minimizing the RHS by taking $\tau = 1/\lambda_{\max}$ yields

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2\lambda_{\max}} \left\| \nabla f(x_k) \right\|_F^2. \tag{19}$$

Subtracting each side of (19) by δ and combining with $\|\nabla f(x_k)\|_F^2 \geq 2\lambda_{\min}(f(x_k) - \delta)$ (from (18)), we get

$$\begin{aligned} f(x_{k+1}) - \delta &\leq f(x_k) - \delta - \frac{1}{2\lambda_{\max}} \left\| \nabla f(x_k) \right\|_F^2 \\ &\leq \big(f(x_k) - \delta\big) - \frac{2\lambda_{\min}}{2\lambda_{\max}} \big(f(x_k) - \delta\big) \\ &\leq \left(1 - \frac{\lambda_{\min}}{\lambda_{\max}}\right) \big(f(x_k) - \delta\big). \end{aligned}$$

Putting $\alpha := 1 - \lambda_{\min}/\lambda_{\max}$, we have

$$f(x_{k+1}) - \delta \leq \alpha \big(f(x_k) - \delta\big). \tag{20}$$

By induction, we obtain

$$f(x_k) - \delta \leq \alpha^k \big(f(x_0) - \delta\big). \tag{21}$$

Since $Q^T Q$ is assumed to be invertible, $Q^T Q > 0$, it follows that $\lambda_{\min} > 0$ and hence $0 < \alpha < 1$. Thus, $f(x_k) - \delta \to 0$, or equivalently, $f(x_k) \to \delta$ as $k \to \infty$.

Consider the case of X^* is the unique exact solution, i.e., $\delta = 0$. We have $f(x_k) \to 0$, or equivalently $Qx_k - \mathrm{Vec}(E) \to 0$ as $k \to \infty$. Now, the assumption that Q is of full column-rank implies that

$$x_k \to \big(Q^T Q\big)^{-1} Q^T \,\mathrm{Vec}(E).$$

Therefore, $X_k = \mathrm{Vec}^{-1}(x_k) \to X^*$ as $k \to \infty$.

The other case is that X^* is the unique least-squares solution, i.e., $\delta > 0$. We have $f(x_k) \to \delta$ or $\frac{1}{2}\|Qx_k - \mathrm{Vec}(E)\|_F^2 \to \|\mathrm{Vec}(E)\|_F^2 - \mathrm{Vec}(E)^T Qx^*$. Then

$$\frac{1}{2} \mathrm{tr}\Big(\big(Qx_k - \mathrm{Vec}(E)\big)^T \big(Qx_k - \mathrm{Vec}(E)\big)\Big) \to \mathrm{tr}\big(\mathrm{Vec}(E)^T \mathrm{Vec}(E)\big) - \mathrm{Vec}(E)^T Qx^*.$$

We omit some algebraic operations and hence immediately write

$$\|x_k\|_Q^2 = \operatorname{tr}\left(x_k^T Q^T Q x_k\right) \to \operatorname{tr}\left(\operatorname{Vec}(E)^T \operatorname{Vec}(E)\right) = \left\|x^*\right\|_Q^2.$$

Therefore, $\|X_k\|_Q \to \|X^*\|_Q$ as $k \to \infty$. □

We denote the condition number of Q by $\kappa = \kappa(Q)$. Observe that $\alpha = 1-\kappa^{-2}$. The relation between the quadratic norm-error $f(x_k)$ and the norm of residual error $\|R_k\|$ is given by

$$f(x_k) = \frac{1}{2}\|R_k\|_F^2.$$

Making use of Lemma 2.1(ii), the inequalities (20) and (21) become the following estimation:

$$\|R_k\|_F^2 \le \alpha\|R_{k-1}\|_F^2 + 2\delta\kappa^{-2}, \tag{22}$$

$$\|R_k\|_F^2 \le \alpha^k\|R_0\|_F^2 + 2\delta\left(1-\alpha^k\right). \tag{23}$$

In the case of Eq. (8) having a unique exact solution ($\delta = 0$), the error estimations (22) and (23) reduce to (24) and (25), respectively.

$$\|R_k\|_F \le \alpha^{\frac{1}{2}}\|R_{k-1}\|_F, \tag{24}$$

$$\|R_k\|_F \le \alpha^{\frac{k}{2}}\|R_0\|_F. \tag{25}$$

Since $0 < \alpha < 1$, it follows that, if $\|R_{k-1}\|_F$ are nonzero, then

$$\|R_k\|_F < \|R_{k-1}\|_F. \tag{26}$$

The above discussion is summarized in the following theorem.

Theorem 4.4 *Assume that Q is of full column-rank.*

(i) *Suppose Eq. (8) has a unique solution. The error estimation $\|R_k\|_F$ compared with $\|R_{k-1}\|_F$ (the preceding iteration) and $\|R_0\|_F$ (the initial iteration) are given by (24) and (25), respectively. Particularly, the relative error $\|R_k\|_F$ gets smaller than the preceding (nonzero) error, as in (26).*

(ii) *When Eq. (8) has a unique least-squares solution, the error estimation (22) and (23) hold.*

In both cases, the convergence rate of Algorithm 1 (regarding the error $\|R_k\|_F$) is governed by $\sqrt{1-\kappa^{-2}}$.

Remark 4.5 The relative errors (22) and (23) do not seem to decrease every step of iteration since the terms $2\delta\kappa^{-2}$ and $2\delta(1-\alpha^k)$ are positive. However, the inequality (19) implies that $\{\|R_k\|_F\}_{k=1}^{\infty}$ is a strictly decreasing sequence converging to δ.

We recall the following properties.

Lemma 4.6 (e.g. [5]) *For any compatible matrices A and B, we have*

(i) $\|A^TA\|_2 = \|A\|_2^2$,

(ii) $\|A^T\|_2 = \|A\|_2$,

(iii) $\|AB\|_F \le \|A\|_2\|B\|_F$.

Theorem 4.7 *Suppose that Q is of full column-rank and Eq. (8) has a unique exact solution. We have the error estimation $\|X_k - X^*\|_F$ compared with the preceding iteration and the initial iteration of Algorithm 1 are provided by*

$$\|X_k - X^*\|_F \le \kappa\sqrt{\kappa^2 - 1}\,\|X_{k-1} - X^*\|_F, \tag{27}$$

$$\|X_k - X^*\|_F \le \kappa^2\left(1 - \kappa^{-2}\right)^{\frac{k}{2}}\|X_0 - X^*\|_F. \tag{28}$$

Particularly, the convergence rate of the algorithm is governed by $\sqrt{1-\kappa^{-2}}$.

Proof Utilizing (25) and Lemma 4.6, we have

$$\begin{aligned}
\|X_k - X^*\|_F &= \|x_k - x^*\|_F \\
&= \left\|(Q^TQ)^{-1}(Q^TQ)x_k - (Q^TQ)^{-1}(Q^TQ)x^*\right\|_F \\
&\le \left\|(Q^TQ)^{-1}\right\|_2\|Q^T\|_2\|Qx_k - Qx^*\|_F \\
&\le \left(1-\kappa^{-2}\right)^{\frac{k}{2}}\left\|(Q^TQ)^{-1}\right\|_2\|Q^T\|_2\|Qx_0 - \tilde{e}\|_F \\
&\le \left(1-\kappa^{-2}\right)^{\frac{k}{2}}\left\|(Q^TQ)^{-1}\right\|_2\|Q^T\|_2\|Q\|_2\|X_0 - X^*\|_F \\
&= \left(1-\kappa^{-2}\right)^{\frac{k}{2}}\frac{\lambda_{\max}(Q^TQ)}{\lambda_{\min}(Q^TQ)}\|X_0 - X^*\|_F \\
&= \kappa^2\left(1-\kappa^{-2}\right)^{\frac{k}{2}}\|X_0 - X^*\|_F.
\end{aligned}$$

As the limiting behavior of $\|X_k - X^*\|_F$ depends on $(1-\kappa^{-2})^{\frac{k}{2}}$, the convergence rate for Algorithm 1 is governed by $\sqrt{1-\kappa^{-2}}$. Similarly, using (24), it follows that

$$\begin{aligned}
\|X_k - X^*\|_F &\le (1-\kappa^{-2})^{\frac{1}{2}}\left\|(Q^TQ)^{-1}\right\|_2\|Q^T\|_2\|Qx_{k-1} - \tilde{e}\|_F \\
&\le (1-\kappa^{-2})^{\frac{1}{2}}\left\|(Q^TQ)^{-1}\right\|_2\|Q^T\|_2\|Q\|_2\|X_{k-1} - X^*\|_F \\
&= \kappa^2(1-\kappa^{-2})^{\frac{1}{2}}\|X_{k-1} - X^*\|_F,
\end{aligned}$$

and hence (28) is obtained. □

Theorem 4.8 *Suppose Q is of full column-rank and Eq. (8) has a unique least-squares solution. The error estimation $\|X_k - X^*\|_F^2$ compared to the preceding iteration and the initial iteration of Algorithm 1 are provided by*

$$\|X_k - X^*\|_F^2 \le \alpha\kappa^4\|X_{k-1} - X^*\|_F^2 + 2\delta\lambda_{\min}^{-1}, \tag{29}$$

$$\|X_k - X^*\|_F^2 \le \alpha^k\kappa^4\|X_0 - X^*\|_F^2 + 2\delta\kappa^2\left(1-\alpha^k\right)\lambda_{\min}^{-1}. \tag{30}$$

Proof The proof is similar to that of Theorem 4.7 and carried out by (22) and (23). We, therefore, omit the proof. □

Consequently, our convergence analysis indicates that the proposed algorithm always converges to the unique (exact or least-squares) solution for any initial matrices and small condition numbers. Moreover, the algorithm will converge fast when the condition number is close to 1.

5 Numerical experiments for the generalized Sylvester-transpose matrix equation and its special cases

In this section, we provide numerical results to show the efficiency and effectiveness of Algorithm 1. We perform the experiments in the following cases:

- a large-scaled square generalized Sylvester-transpose equation,
- a small-scaled rectangular generalized Sylvester-transpose equation,
- a small-scaled square Sylvester-transpose equation,
- a large-scaled square Sylvester equation,
- a moderate-scaled square Lyapunov equation.

Each example contains some comparisons of the proposed algorithm (denoted by *TauOpt*) with the mentioned existing algorithms as well as the direct method Eq. (12). CT stands for the computational time (in seconds) and is measured by the *tic toc* function in MATLAB. The relative error $\|R_k\|_F$ is used to measure error at the kth step of the iteration. All iterations have been evaluated by MATLAB R2020b, on a PC (2.60-GHz intel(R) Core(TM) i7 processor, 8 Gbyte RAM).

Example 5.1 Consider a generalized Sylvester-transpose matrix equation

$$\sum_{t=1}^{2} A_t X B_t + \sum_{s=1}^{3} C_s X^T D_s = E$$

with 100×100 coefficient matrices:

$$\begin{aligned}
&A_1 = \text{tridiag}(-0.242, 0.217, 0.109), && A_2 = \text{tridiag}(0.539, 0.253, -0.835),\\
&B_1 = \text{tridiag}(0.098, -0.793, 0.561), && B_2 = \text{tridiag}(0.001, 0.533, 0.212),\\
&C_1 = \text{tridiag}(0.586, 0.462, -0.688), && C_2 = \text{tridiag}(-0.245, -0.937, 0.687),\\
&C_3 = \text{tridiag}(-0.930, 0.471, -0.813), && D_1 = \text{tridiag}(0.440, -0.762, 0.008),\\
&D_2 = \text{tridiag}(0.995, 0.075, 0.169), && D_3 = \text{tridiag}(0.514, -0.779, 0.358),\\
&\text{and} \quad E = \text{septdiag}(-0.427, -0.158, -1.181, 1.182, -0.452, -0.014, -0.158).
\end{aligned}$$

We choose an initial matrix $X_0 = \text{zero}(100)$, where $\text{zero}(n)$ is the $n \times n$ zero matrix. In fact, this equation has the unique solution

$$X^* = \text{tridiag}(0.293, 0.152, 0.905).$$

Table 1 shows that the direct method consumes a big amount of time to get the exact solution, while Algorithm 1 produces a small-error solution in a small time (0.1726 sec-

Table 1 Relative errors and CTs for Ex. 5.1

Method	Iterations	Relative error	CT
TauOpt	100	0.0014	0.1726
GI	100	4.0260	0.3252
LSI	100	17.8265	2.0425
direct	–	0	5.4274e+03

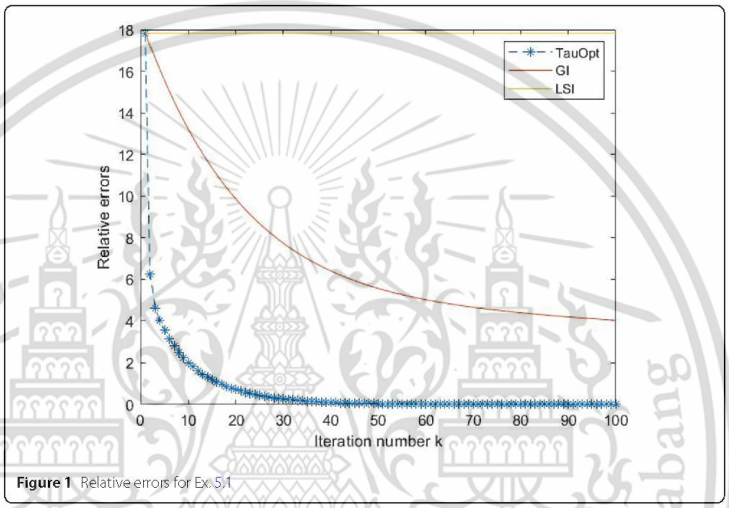

Figure 1 Relative errors for Ex. 5.1

onds after 100 iterations). We compare the efficiency of Algorithm 1 with another existing gradient-based iterative algorithms, namely, GI (Method 1.1) and LSI (Method 1.2). Figure 1 displays the error plot which supports the theoretical results i.e., the sequence of errors generated by Algorithm 1 is monotone decreasing. Table 1 indicates that our algorithm performs well in computational time.

Example 5.2 Consider the equation

$$\sum_{t=1}^{3} A_t X B_t + \sum_{s=1}^{2} C_s X^T D_s = E$$

with the rectangular coefficient matrices as follows:

$$A_1 = \begin{bmatrix} 0.491 & 0.064 \\ 0.071 & 0.436 \\ 0.887 & 0.826 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0.394 & 0.886 \\ 0.613 & 0.931 \\ 0.818 & 0.190 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 0.258 & 0.503 \\ 0.897 & 0.612 \\ 0.593 & 0.819 \end{bmatrix},$$

$$B_1 = \begin{bmatrix} 0.531 & 0.453 & 0.966 \\ 0.202 & 0.427 & 0.620 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0.695 & 0.346 & 0.556 \\ 0.720 & 0.517 & 0.156 \end{bmatrix},$$

Table 2 Errors and CTs for Ex. 5.2

Method	Iterations	Error	CT
TauOpt	100	7.3178e−04	0.0015
GI	100	0.6164	0.0025
LSI	100	0.8453	0.0076
direct	–	0	0.0020

$$B_3 = \begin{bmatrix} 0.562 & 0.426 & 0.731 \\ 0.694 & 0.836 & 0.360 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 0.454 & 0.734 \\ 0.386 & 0.430 \\ 0.775 & 0.693 \end{bmatrix},$$

$$C_2 = \begin{bmatrix} 0.945 & 0.109 \\ 0.784 & 0.389 \\ 0.705 & 0.590 \end{bmatrix}, \quad D_1 = \begin{bmatrix} 0.459 & 0.228 & 0.015 \\ 0.050 & 0.834 & 0.863 \end{bmatrix},$$

$$D_2 = \begin{bmatrix} 0.078 & 0.500 & 0.571 \\ 0.669 & 0.218 & 0.122 \end{bmatrix} \quad \text{and} \quad E = \begin{bmatrix} 0.671 & 0.056 & 0.435 \\ 0.599 & 0.152 & 0.832 \\ 0.056 & 0.019 & 0.617 \end{bmatrix}.$$

We find that $4 = \operatorname{rank} Q \neq \operatorname{rank}[Q \ \operatorname{Vec}(E)] = 5$, i.e., the matrix equation does not have an exact solution. However, the size of Q is 9×4, i.e., Q is of full-column rank. Hence, according to Theorem 4.3, Algorithm 1 will converge to the least-squares solution in which the least-squares error (13) is equal to 0.0231. We choose an initial matrix $X_0 = \operatorname{zero}(2)$. Algorithm 1 is compared with GI (Method 1.1), LSI (Method 1.2) and the direct method Eq. (12). In this case, we consider the error $\|X^* - X_k\|_F$ where X^* is the least-squares solution. Figure 2 displays the error plot, and Table 2 shows the errors and CTs for TauOpt, GI, LSI and the direct method. We see that the errors converge monotonically to zero, i.e., the approximate solutions X_k generated by Algorithm 1 converge to X^*. Moreover, Algorithm 1 consumes less computational time than other methods.

Next, we will consider the Sylvester-transpose equation (5) which is a special case of the generalized Sylvester-transpose equation (8). From Algorithm 1, the optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|AW_kB + CW_k^TD\|_F^2},$$

where $W_k = A^TR_kB^T + C^TR_kD^T$ and $R_k = E - AX_kB - CX_k^TD$.

Example 5.3 Let us consider the Sylvester-transpose equation (5) with

$$A = \begin{bmatrix} 6 & -4 & -7 & -8 \\ 9 & -4 & 5 & 2 \\ -9 & 6 & -5 & 4 \\ 8 & -3 & 3 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} 6 & -5 & 4 & -2 \\ 9 & -7 & -5 & 6 \\ 6 & 2 & -8 & 2 \\ 7 & 3 & -1 & -1 \end{bmatrix},$$

$$C = \begin{bmatrix} -8 & -5 & -4 & 7 \\ 2 & 7 & -4 & 6 \\ 4 & 8 & -9 & -7 \\ 3 & 1 & 5 & 6 \end{bmatrix}, \quad D = \begin{bmatrix} 3 & -5 & 1 & 2 \\ 6 & 6 & 3 & 1 \\ 4 & -8 & -5 & 4 \\ 3 & -5 & -1 & 9 \end{bmatrix} \quad \text{and}$$

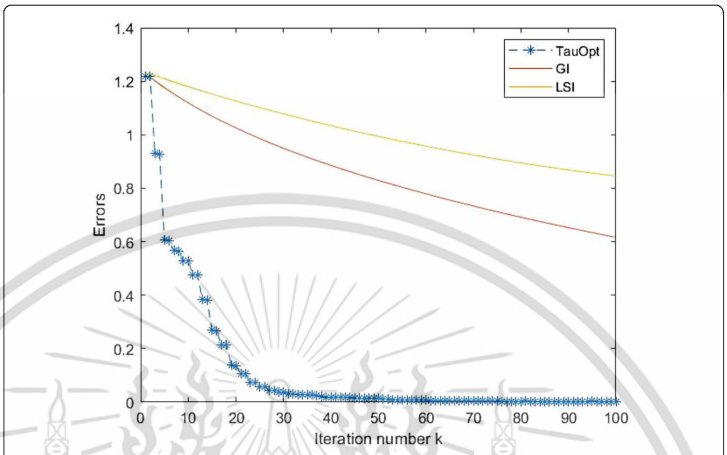

Figure 2 Errors for Ex. 5.2

Table 3 Relative errors and CTs for Ex. 5.3

Method	Iterations	Relative error	CT
TauOpt	100	0.3368	0.0011
GI	100	302.3879	0.0021
LSI	100	562.8838	0.0034
AGBI	100	80.7919	0.0015
direct	–	0	0.0051

$$E = \begin{bmatrix} -284 & 13 & 74 & -93 \\ 248 & -47 & -103 & 109 \\ -54 & 92 & 85 & -112 \\ 326 & -98 & -127 & 167 \end{bmatrix}.$$

Choosing $X_0 = \text{zero}(4)$, then the sequence of numerical solutions generated by Algorithm 1 converges to the exact solution,

$$X^* = \begin{bmatrix} 0.3342 & 0.3443 & 0.4843 & 0.7574 \\ 0.9568 & 0.7485 & 0.4250 & 0.2941 \\ 0.0177 & 0.8061 & 0.6380 & 0.6972 \\ 0.4516 & 0.1859 & 0.7069 & 0.6669 \end{bmatrix}.$$

We report the comparison of Algorithm 1 with GI (Method 1.1), LSI (Method 1.2), AGBI ([28]) and the direct method Eq. (12) by Fig. 3 and Table 3. Both of them imply that Algorithm 1 outperforms other algorithms.

Next, we will consider the Sylvester equation (3) which is also a special case of Eq. (8). For this equation, the optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|AW_k + W_kB\|_F^2},$$

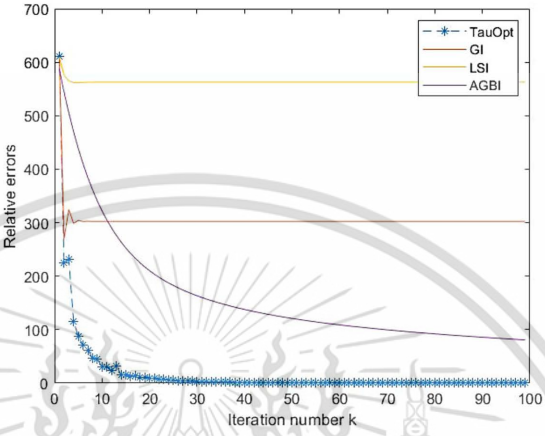

Figure 3 Relative errors for Ex. 5.3

Table 4 Relative errors and CTs for Ex. 5.4

Method	Iterations	Relative error	CT
TauOpt	100	0.1457	0.0681
GI	100	183.4122	0.0731
RGI	100	33.0116	0.0661
MGI	100	115.7206	0.0640
AGBI	100	57.8981	0.0839
JGI	100	515.9767	0.0385
AJGI	100	469.0704	0.0547
direct	-	0	5.4606e+03

where $W_k = A^T R_k + R_k B^T$ and $R_k = C - AX_k - X_k B$.

Example 5.4 Suppose that the Sylvester equation (3) has large-scaled tridiagonal coefficient matrices, i.e.,

$$A = \text{tridiag}(10, -2, 9), \qquad B = \text{tridiag}(-1, 2, -5), \quad \text{and} \quad C = \text{tridiag}(-45, 13, -20),$$

where $A, B, C \in \mathbb{R}^{100\times 100}$. We choose an initial matrix $X_0 = \text{zero}(100)$. Here, the symmetric exact solution is given by $X^* = \text{tridiag}(1, -5, 1)$, so that AGBI algorithm can be applicable. We compare Algorithm 1 with GI (Method 1.1), AGBI ([28]), RGI [12], MGI [13], JGI [14], and AJGI [14]. Although Table 4 tells us that our algorithm takes a slightly more time than some other algorithms, Fig. 4 illustrates that Algorithm 1 reaches the fastest convergence.

The last example presents another special case of Eq. (8) that is the Lyapunov equation (2). The optimal step size τ is described by

$$\tau_{k+1} = \frac{\|W_k\|_F^2}{\|AW_k + W_k A^T\|_F^2},$$

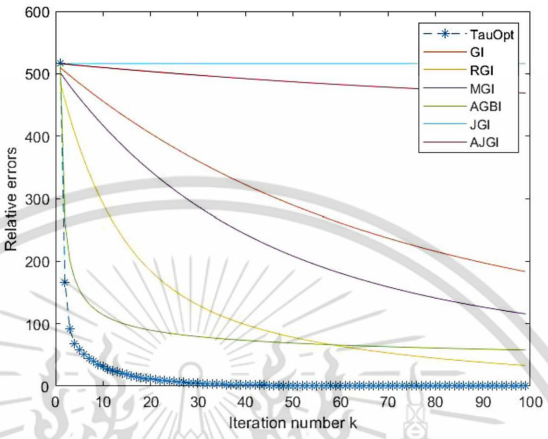

Figure 4 Relative errors for Ex. 5.4

Table 5 Relative errors and CTs for Ex. 5.5

Method	Iterations	Relative error	CT
TauOpt	50	2.4121e−06	0.0056
GI	50	8.9274	0.0056
RGI	50	6.8604	0.0063
MGI	50	2.5820	0.0061
AGBI	50	10.0789	0.0072
JGI	50	8.3219	0.0056
AJGI	50	0.5565	0.0058
LSIA 1	50	5.1063	0.0080
LSIA 2	50	5.1708	0.0041
direct	–	0	0.4636

where $W_k = A^T R_k + R_k A$ and $R_k = B - AX_k - X_k A^T$.

Example 5.5 We consider the Lyapunov equation (2) with medium-scale coefficient matrices

$$A = -\operatorname{triu}(\operatorname{rand}(n), 1) + \operatorname{diag}(8 - \operatorname{diag}(\operatorname{rand}(n))), \qquad B = \operatorname{rand}(n).$$

We choose $n = 20$ and set $X_0 = \operatorname{zero}(20)$. Algorithm 1 is compared with GI, RGI, MGI, AGBI, JGI, AJGI, LSIA1, and LSIA2 methods. We report the results in Fig. 5 and Table 5. In conclusion, Algorithm 1 takes a slightly more computational time than some other algorithms but still outperforms distinctly in performance of convergence.

6 Concluding remarks

We properly establish a gradient-descent iterative algorithm for solving the generalized Sylvester-transpose matrix equation (8). We show that the proposed algorithm is useful and applicable for wide range of problems, even though the problem has no solution, as

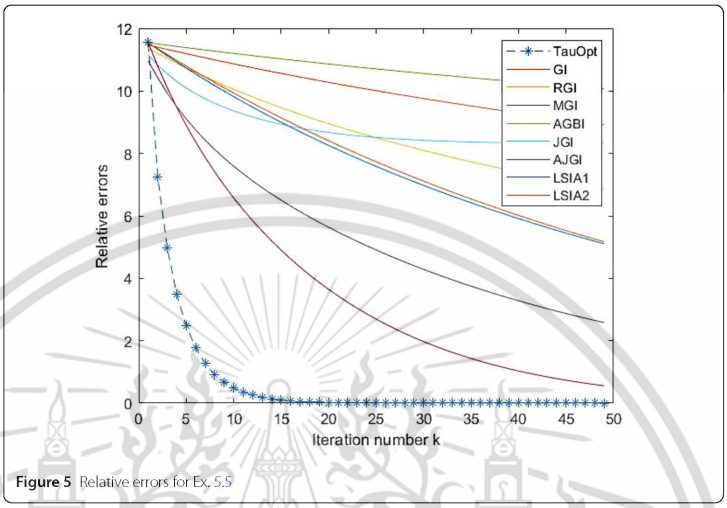

Figure 5 Relative errors for Ex. 5.5

long as the associated matrix Q, defined by Eq. (11), is of full column-rank. If the problem has the unique exact solution, then the approximate solutions converge to the exact solution. In the case of a no-solution problem, we have $\|X\|_Q \to \|X^+\|_Q$ where X^+ is the unique least-squares solution. The convergence rate is described in terms of κ, the matrix condition number of Q, that is, $\sqrt{1-\kappa^{-2}}$. Moreover, the analysis shows that the sequence of errors generated by our algorithm is monotone decreasing. Numerical examples are provided to verify our theoretical findings.

Acknowledgements
The first author received financial support from KMITL Doctoral Scholarships, grant no. KDS 2019/022 during his Ph.D. study.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
Writing–original draft preparation, A.K.; writing–review and editing, P.C.; data curation, A.K.; supervision, P.C. All authors contributed equally and significantly in writing this article. All authors have read and approved the manuscript.

Publisher's Note

Received: 30 January 2021 Accepted: 17 May 2021 Published online: 21 May 2021

References

1. Geir, E.D., Fernando, P.: A Course in Robust Control Theory: A Convex Approach. Springer, New York (1999)
2. Varga, A.: Robust pole assignment via Sylvester equation based state feedback parametrization. In: Proceedings of the 2000 IEEE International Symposium on Computer-Aided Control System, pp. 13–18. Design, Alsaka (2000)
3. Magnus, J.R., Neudecker, H.: Matrix Differential Calculus with Applications in Statistics and Econometries, 3rd edn. Wiley, Chichester (2007)

4. Nouri, K., Beik, S.P.A., et al.: An iterative algorithm for robust simulation of the Sylvester matrix differential equations. Adv. Differ. Equ. **2020**(1), Article ID 287 (2020). https://doi.org/10.1186/s13662-020-02757-z
5. Horn, R., Johnson, C.: Topics in Matrix Analysis. Cambridge University Press, New York (1991)
6. Hajarian, M.: Generalized conjugate direction algorithm for solving the general coupled matrix equations over symmetric matrices. Numer. Algorithms **73**(3), 591–609 (2016)
7. Hajarian, M.: Extending the CGLS algorithm for least squares solutions of the generalized Sylvester-transpose matrix equations. J. Franklin Inst. **353**(5), 1168–1185 (2016)
8. Dehghan, M., Mohammadi-Arani, R.: Generalized product-type methods based on Bi-conjugate gradient(GPBiCG) for solving shifted linear systems. Comput. Appl. Math. **36**(4), 1591–1606 (2017)
9. Bai, Z.: On Hermitian and skew-Hermitian splitting iteration methods for continuous Sylvester equation. J. Comput. Math. **29**(2), 185–198 (2011). https://doi.org/10.4208/jcm.1009-m3152
10. Ding, F., Chen, T.: Gradient based iterative algorithms for solving a class of matrix equations. IEEE Trans. Autom. Control **50**(8), 1216–1221 (2005). https://doi.org/10.1109/TAC.2005.852558
11. Ding, F., Liu, X.P., Ding, J.: Iterative solutions of the generalized Sylvester matrix equations by using the hierarchical identification principle. Appl. Math. Comput. **197**(1), 41–50 (2008). https://doi.org/10.1016/j.amc.2007.07.040
12. Niu, Q., Wang, X., Lu, L.Z.: A relaxed gradient based algorithm for solving Sylvester equation. Asian J. Control **13**(3), 461–464 (2011). https://doi.org/10.1002/asjc.328
13. Wang, X., Dai, L., Liao, D.: A modified gradient based algorithm for solving Sylvester equation. Appl. Math. Comput. **218**(9), 5620–5628 (2012). https://doi.org/10.1016/j.amc.2011.11.055
14. Tian, Z., Tian, M., et al.: An accelerated Jacobi-gradient based iterative algorithm for solving Sylvester matrix equations. Filomat **31**(8), 2381–2390 (2017). https://doi.org/10.2298/FIL1708381T
15. Sun, M., Wang, Y., Liu, J.: Two modified least-squares iterative algorithms for the Lyapunov matrix equations. Adv. Differ. Equ. **2019**, 305 (2019). https://doi.org/10.1186/s13662-019-2253-7
16. Ding, F., Chen, T.: Hierarchical gradient-based identification of multivariable discrete-time systems. Automatica **41**(2), 315–325 (2005). https://doi.org/10.1016/j.automatica.2004.10.010
17. Ding, F., Chen, T.: Hierarchical least squares identification methods for multivariable systems. IEEE Trans. Autom. Control **50**(3), 397–402 (2005). https://doi.org/10.1109/TAC.2005.843856
18. Wu, A., Duan, G., Zhou, B.: Solution to generalized Sylvester matrix equations. IEEE Trans. Autom. Control **53**(3), 811–815 (2008). https://doi.org/10.1109/TAC.2008.919562
19. Xie, L., Liu, Y., Yang, H.: Gradient based and least squares based iterative algorithms for matrix equations $AXB + CX^TD = F$. Appl. Math. Comput. **217**(5), 2191–2199 (2010). https://doi.org/10.1016/j.amc.2010.07.019
20. 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**, 1–8 (2017). https://doi.org/10.1155/2017/1624969
21. Kittisopaporn, A., Chansangiam, P.: The steepest descent of gradient-based iterative method for solving rectangular linear systems with an application to Poisson's equation. Adv. Differ. Equ. **2020**(1), Article ID 259 (2020). https://doi.org/10.1186/s13662-020-02715-9
22. Boonruangkan, N., Chansangiam, P.: Gradient iterative method with optimal convergent factor for solving a generalized Sylvester matrix equation with applications to diffusion equations. Symmetry **12**(10), Article ID 1732 (2020). https://doi.org/10.3390/sym12101732
23. Sasaki, N., Chansangiam, P.: Modified Jacobi–gradient iterative method for generalized Sylvester matrix equation. Symmetry **12**(11), Article ID 1831 (2020). https://doi.org/10.3390/sym12111831
24. Kittisopaporn, A., Chansangiam, P., Lewkeeratiyutkul, W.: Convergence analysis of gradient–based iterative algorithms for a class of rectangular Sylvester matrix equations based on Banach contraction principle. Adv. Differ. Equ. **2021**(1), Article ID 17 (2021). https://doi.org/10.1186/s13662-020-03185-9
25. Ding, F., Zhang, X., Xu, L.: The innovation algorithms for multivariable state-space models. Int. J. Adapt. Control Signal Process. **33**, 1601–1618 (2019). https://doi.org/10.1002/acs.3053
26. Ding, F., Lv, L., Pan, J., et al.: Two-stage gradient-based iterative estimation methods for controlled autoregressive systems using the measurement data. Int. J. Control. Autom. Syst. **18**, 886–896 (2020). https://doi.org/10.1007/s12555-019-0140-3
27. Ding, F., Xu, L., Meng, D., et al.: Gradient estimation algorithms for the parameter identification of bilinear systems using the auxiliary model. J. Comput. Appl. Math. **369**, 112575 (2020). https://doi.org/10.1016/j.cam.2019.112575
28. Xie, Y.-J., Ma, C.-F.: The accelerated gradient based iterative algorithm for solving a class of generalized Sylvester-transpose matrix equation. Appl. Math. Comput. **273**, 1257–1269 (2016). https://doi.org/10.1016/j.amc.2015.07.022
29. Xie, L., Ding, J., Ding, F.: Gradient based iterative solutions for general linear matrix equations. Comput. Math. Appl. **58**(7), 1441–1448 (2009). https://doi.org/10.1016/j.camwa.2009.06.047
30. Stephen, P.B., Lieven, V.: Convex Optimization. Cambridge University Press, Cambridge (2004)

Author Biography

Name	Mr. Adisorn Kittisopaporn
Date of Birth	8 February 1995
Address	116/59 Moo 4, Bangmuang, Muang, Samutprakarn, 10270
Education	2013-2016 Bachelor of Science in Applied Mathematics. GPA 3.92 (First Class Honors) King Mongkut's Institute of Technology Ladkrabang
	2017-2018 Master of Science in Applied Mathematics. GPA 4.00 King Mongkut's Institute of Technology Ladkrabang
	2019-2021 Doctor of Philosophy in Applied Mathematics. GPA 4.00 King Mongkut's Institute of Technology Ladkrabang
Scholarship	2013-2016 King Mongkut's Institute of Technology Ladkrabang Academic Excellent Scholarship
	2017-2018 Faculty of science, King Mongkut's Institute of Technology Ladkrabang Graduate Scholarship
	2019-2021 KMITL Doctoral Scholarship

Academic Publication(s)

1. Kittisopaporn, A. and Chansangiam, P. 2020. "Gradient-descent iterative algorithm for solving a class of linear matrix equations with applications to heat and Poisson equations." Advances in Difference Equations, vol. 2020, no.1, Article no. 324.
2. Kittisopaporn, A. and Chansangiam, P. 2021. "Approximated least-squares solutions of a generalized Sylvester-transpose matrix equation via gradient descent iterative algorithm." Advances in Difference Equations, vol. 2021, no.1, Article no. 266.