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

Springer

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*

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

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

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

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

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

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

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 $\|(B^T \otimes A)\operatorname{Vec}(X) - \operatorname{Vec}(C)\|_F$ via Lemma 2.1(iii). So, we define the quadratic norm-error function $f : \mathbb{R}^{mn} \to \mathbb{R}$ by

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

Thus, our new iterative equation is in the form

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

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 $k = 0, 1, \dots,$ $\tilde{\phi}_{k+1}(\tilde{\tau}) := \tilde{f}(X(k+1))$. 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* $\alpha, \beta = 1, \dots, p$.

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

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

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

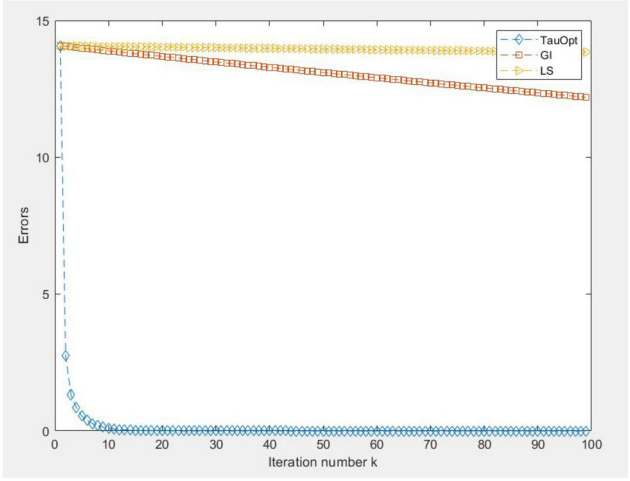

Figure 1 Errors for Example 5.1

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

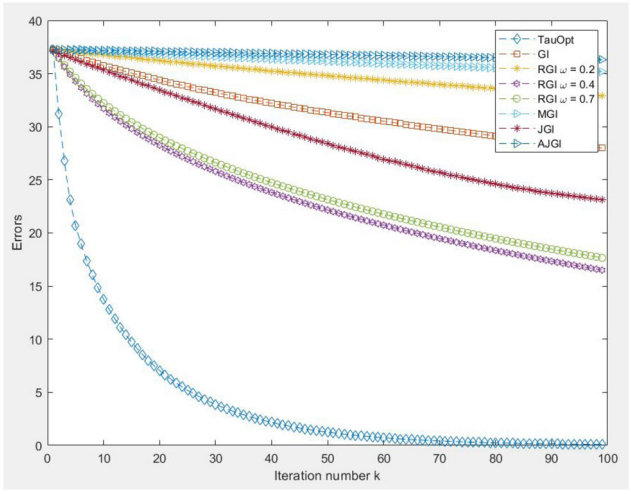

Figure 2 Errors for Example 5.2

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

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

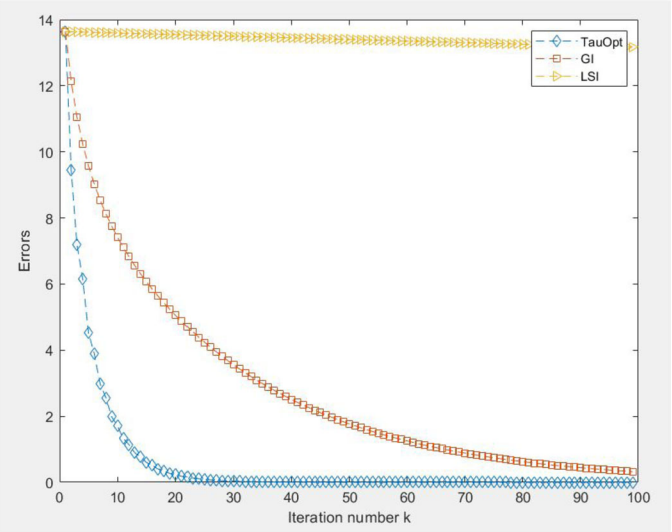

Figure 3 Errors for Example 5.3

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

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,

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

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 $\begin{bmatrix} V_1^T & V_2^T & \cdots & V_{N_y}^T \end{bmatrix}^T$ where

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

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., $f(x, y) = 0$.

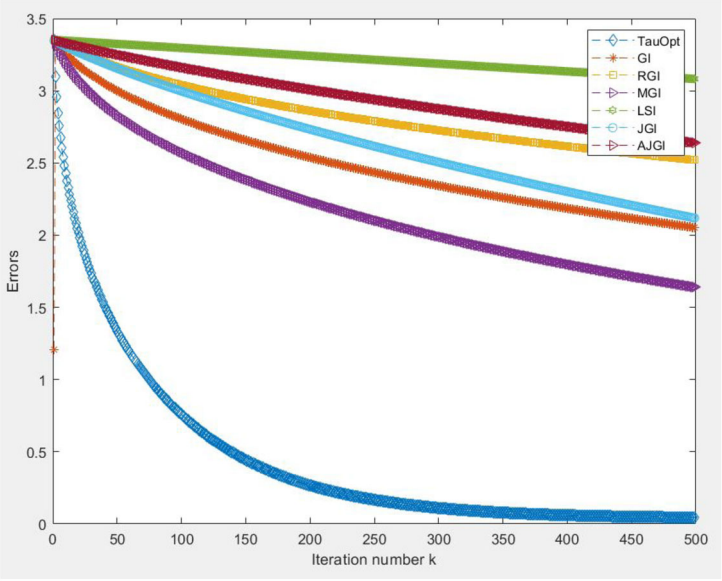

Figure 4 Errors for Example 6.2

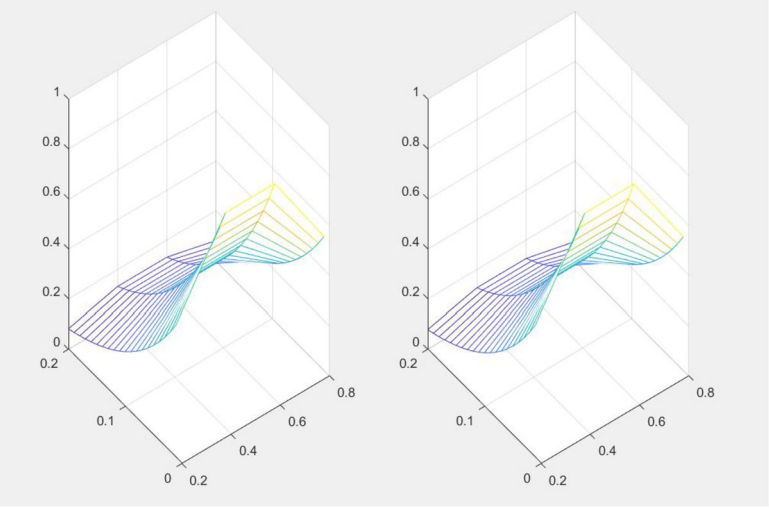

Figure 5 The 3D-plot of the analytical solution (left) and the numerical solution (right) for Example 6.2

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 $u_{11}, \dots, u_{N_x N_y}$:

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

where $U = [u_{ij}]$, T_1 has $N_y \times N_y$ blocks of the form $\operatorname{tridiag}(-1, 2, -1)$ 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:

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

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

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

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

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 $G = -h^2 \operatorname{Vec}([f_{ij}]) + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}$. 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* $c = -h^2 \operatorname{Vec}[f_{ij}] + \overline{g_u} + \overline{g_d} + \overline{g_l} + \overline{g_r}$ *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*:

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

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

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

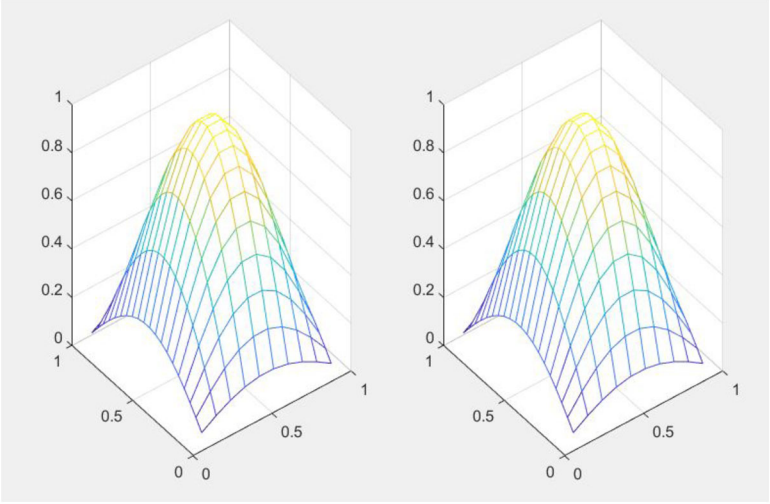

Figure 6 The 3D-plot of the analytical solution (left) and the numerical solution (right) for Example 7.3

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

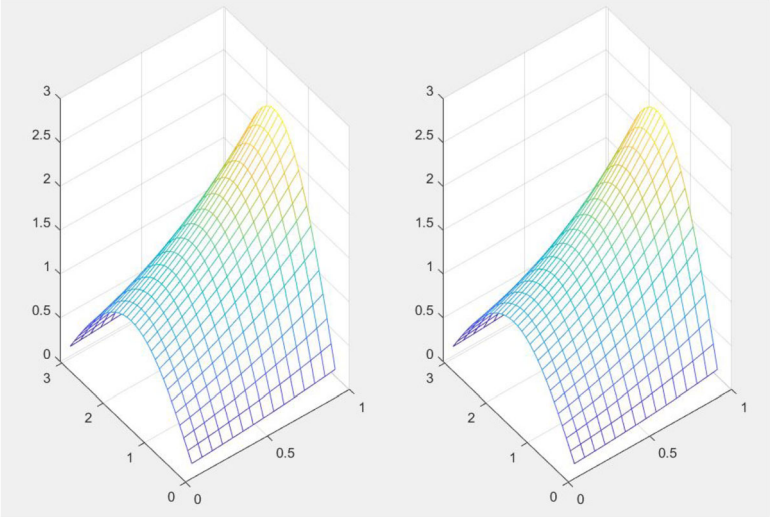

Figure 7 The 3D-plot of the analytical solution (left) and the numerical solution (right) for Example 7.4

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)