mathematics

MDPI

Article

Gradient-Based Optimization Algorithm for Solving Sylvester Matrix Equation

Juan Zhang [1,*] and Xiao Luo [2]

[1] Key Laboratory of Intelligent Computing and Information Processing of Ministry of Education, Xiangtan University, Xiangtan 411105, China
[2] Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Xiangtan University, Xiangtan 411105, China; lxxiangtandaxue@163.com
* Correspondence: zhangjuan@xtu.edu.cn; Tel.: +86-131-0722-4973

Abstract: In this paper, we transform the problem of solving the Sylvester matrix equation into an optimization problem through the Kronecker product primarily. We utilize the adaptive accelerated proximal gradient and Newton accelerated proximal gradient methods to solve the constrained non-convex minimization problem. Their convergent properties are analyzed. Finally, we offer numerical examples to illustrate the effectiveness of the derived algorithms.

Keywords: Sylvester matrix equation; Kronecker product; adaptive accelerated proximal gradient method; Newton-accelerated proximal gradient method

MSC: 15A24; 65F45

Citation: Zhang, J.; Luo, X.; Gradient-Based Optimization Algorithm for Solving Sylvester Matrix Equation. *Mathematics* **2022**, *10*, 1040. https://doi.org/10.3390/math10071040

Academic Editors: Maria Isabel Berenguer and Manuel Ruiz Galán

Received: 14 February 2022
Accepted: 22 March 2022
Published: 24 March 2022

1. Introduction

Matrix equations are ubiquitous in signal processing [1], control theory [2], and linear systems [3]. Most time-dependent models accounting for the prediction, simulation, and control of real-world phenomena may be represented as linear or nonlinear dynamical systems. Therefore, the relevance of matrix equations within engineering applications largely explains the great effort put forth by the scientific community into their numerical solution. Linear matrix equations have an important role in the stability analysis of linear dynamical systems and the theoretical development of the nonlinear system. The Sylvester matrix equation was first proposed by Sylvester and produced from the research of relevant fields in applied mathematical cybernetics. It is a famous matrix equation that occurs in linear and generalized eigenvalue problems for the computation of invariant subspaces using Riccati equations [4–6]. The Sylvester matrix equation takes part in linear algebra [7–9], image processing [10], model reduction [11], and numerical methods for differential equations [12,13].

We consider the Sylvester matrix equation of the form

$$AX + XB = C, \tag{1}$$

where $A \in \mathbb{R}^{m\times m}, B \in \mathbb{R}^{n\times n}, C \in \mathbb{R}^{m\times n}$ are given matrices, and $X \in \mathbb{R}^{m\times n}$ is an unknown matrix to be solved. We discuss a special form of the Sylvester matrix equation, in which A and B are symmetric positive definite.

Recently, there has been a lot of discussion on the solution and numerical calculation of the Sylvester matrix equation. The standard methods for solving this equation are the Bartels–Stewart method [14] and the Hessenberg–Schur method [15], which are efficient for small and dense system matrices. When system matrices are small, the block Krylov subspace methods [16,17] and global Krylov subspace methods [18] are proposed. These methods use the global Arnoldi process, block Arnoldi process, or nonsymmetric block

Lanczos process to produce low-dimensional Sylvester matrix equations. More feasible methods for solving large and sparse problems are iterative methods. When system matrices are large, there are some effective methods such as the alternating direction implicit (ADI) method [19], global full orthogonalization method, global generalized minimum residual method [20], gradient-based iterative method [21], and global Hessenberg and changing minimal residual with Hessenberg process method [22]. When system matrices are low-rank, the ADI method [23], block Arnoldi method [17], preconditioned block Arnoldi method [24], and extended block Arnoldi method [25] and its variants [26,27], including the global Arnoldi method [28,29] and extended global Arnoldi method [25], are proposed to obtain the low-rank solution.

The adaptive accelerated proximal gradient (A-APG) method [30] is an efficient numerical method for calculating the steady states of the minimization problem, motivated by the accelerated proximal gradient (APG) method [31], which has wide applications in image processing and machine learning. In each iteration, the A-APG method takes the step size by using a line search initialized with the Barzilai–Borwein (BB) step [32] to accelerate the numerical speed. Moreover, as the traditional APG method is proposed for the convex problem and its oscillation phenomenon slows down the convergence, the restart scheme has been used for speeding up the convergence. For more details, one can refer to [30] and the references therein.

The main contribution is to study gradient-based optimization methods such as the A-APG and Newton-APG methods for solving the Sylvester matrix equation through transforming this equation into an optimization problem by using Kronecker product. The A-APG and Newton-APG methods are theoretically guaranteed to converge to a global solution from an arbitrary initial point and achieve high precision. These methods are especially efficient for large and sparse coefficient matrices.

The rest of this paper is organized as follows. In Section 2, we transform this equation into an optimization problem by using the Kronecker product. In Section 3, we apply A-APG and Newton-APG algorithms to solve the optimization problem and compare them with other methods. In Section 4, we focus on the convergence analysis of the A-APG method. In Section 5, the computational complexity of these algorithms is analyzed exhaustively. In Section 6, we offer corresponding numerical examples to illustrate the effectiveness of the derived methods.

Throughout this paper, let $\mathbb{R}^{n\times m}$ be the set of all $n \times m$ real matrices. I_n is the identity matrix of order n. If $A \in \mathbb{R}^{n\times n}$, the symbols A^T, A^{-1}, $\|A\|$ and $tr(A)$ express the transpose, the inverse, the 2-norm, and the trace of A, respectively. The inner product in matrix space $\mathbb{E}$ is $\langle x, y\rangle = tr(x, y), \forall x, y \in \mathbb{E}$.

2. The Variant of an Optimization Problem

In this section, we transform the Sylvester equation into an optimization problem. We recall some definitions and lemmas.

Definition 1. *Let $Y = (y_{ij}) \in \mathbb{R}^{m\times n}$, $Z \in \mathbb{R}^{p\times q}$, the Kronecker product of Y and Z be defined by*

$$Y \otimes Z = \begin{bmatrix} y_{11}Z & y_{12}Z & \cdots & y_{1n}Z \\ y_{21}Z & y_{22}Z & \cdots & y_{2n}Z \\ \vdots & \vdots & \vdots & \vdots \\ y_{m1}Z & y_{m2}Z & \cdots & y_{mn}Z \end{bmatrix}.$$

Definition 2. *If $Y \in \mathbb{R}^{m\times n}$, then the straightening operator $vec : \mathbb{R}^{m\times n} \longrightarrow \mathbb{R}^{mn}$ of Y is*

$$vec(Y) = (y_1^T, y_2^T, \ldots, y_n^T)^T.$$

Lemma 1. *Let $Y \in \mathbb{R}^{l\times m}, Z \in \mathbb{R}^{m\times n}, W \in \mathbb{R}^{n\times k}$, then*

$$vec(YZW) = (W^T \otimes Y)vec(Z).$$

From Lemma 1, the Sylvester Equation (1) can be rewritten as

$$(I_n \otimes A + B^T \otimes I_m)vec(X) = vec(C). \tag{2}$$

Lemma 2. *Let A be a symmetric positive matrix; solving the equation $Ax = b$ is equivalent to obtaining the minimum of $\varphi(x) = x^T Ax - 2b^T x$.*

According to Lemma 2 and Equation (2), define

$$\bar{A} = (I_n \otimes A + B^T \otimes I_m),\ \bar{x} = vec(X),\ \bar{b} = vec(C).$$

Therefore, Equation (2) should be $\bar{A}\bar{x} = \bar{b}$. Obviously, if A and B are symmetric positive, then $\bar{A}$ is symmetric positive. The variant of the Sylvester Equation (2) reduces to the optimization problem:

$$\begin{aligned}
\min \varphi(x) &= \min\left\{\bar{x}^T\bar{A}\bar{x} - 2\bar{b}^T\bar{x}\right\} \\
&= \min\left\{vec(X)^T(I_n \otimes A + B^T \otimes I_m)vec(X) - 2vec(X)^T vec(C)\right\} \\
&= \min\left\{vec(X)^T \cdot vec(AX) + vec(X)^T \cdot vec(XB) - 2vec(X)^T \cdot vec(C)\right\} \\
&= \min\left\{tr(X^T AX) + tr(X^T XB) - 2tr(X^T C)\right\}.
\end{aligned} \tag{3}$$

Using the calculation of the matrix differential from [33], we have the following propositions immediately.

Proposition 1. *If $A = (a_{ij}) \in \mathbb{R}^{m\times n}, X = (x_{ij}) \in \mathbb{R}^{m\times n}$, then $\frac{\partial tr(A^T X)}{\partial X} = \frac{\partial tr(X^T A)}{\partial X} = A$.*

Proposition 2. *If $A = (a_{ij}) \in \mathbb{R}^{m\times m}, X = (x_{ij}) \in \mathbb{R}^{m\times n}$, then $\frac{\partial tr(X^T AX)}{\partial X} = AX + A^T X$.*

Proposition 3. *If $B = (b_{ij}) \in \mathbb{R}^{n\times n}, X = (x_{ij}) \in \mathbb{R}^{m\times n}$, then $\frac{\partial tr(XX^T B)}{\partial X} = XB + XB^T$.*

Using Propositions 2 and 3, the gradient of the objective function (3) is

$$\bigtriangledown \varphi(X) = AX + XB + A^T X + XB^T - 2C. \tag{4}$$

By (4), the Hessian matrix is

$$\bigtriangledown^2 \varphi(X) = A + A^T + B + B^T. \tag{5}$$

3. Iterative Methods

In this section, we will introduce the adaptive accelerated proximal gradient (A-APG) method and the Newton-APG method to solve the Sylvester equation. Moreover, we compare the A-APG and Newton-APG methods with other existing methods.

3.1. APG Method

The traditional APG method [31] is designed for solving the composite convex problem:

$$\min_{x\in\mathbb{H}} H(x) = g(x) + f(x),$$

where $\mathbb{H}$ is the finite-dimensional Hilbert space equipped with the inner product $< \cdot, \cdot >$, g and f are both continuously convex, and $\bigtriangledown f$ has a Lipschitz constant L. Given initializations $x_1 = x_0$ and $t_0 = 1$, the APG method is

$$\begin{aligned} t_k &= (\sqrt{4(t-k-1)^2+1})/2, \\ Y_k &= X_k + \frac{t_{k-1}-1}{t_k}(X_k - X_{k-1}), \\ X_{k+1} &= \mathrm{Prox}_g^{\alpha}(Y_k - \alpha \nabla f(Y_k)), \end{aligned}$$

where $\alpha \in (0, L]$ and the mapping $\mathrm{Prox}_g^{\alpha}(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^n$ is defined as

$$\mathrm{Prox}_g^{\alpha}(x) = \underset{y}{\mathrm{argmin}} \left\{ g(y) + \frac{1}{2\alpha}\|y - x\|^2 \right\}.$$

Since our minimization problem is linear, we choose the explicit scheme. The explicit scheme is a simple but effective approach for the minimization problem. Given an initial value Y_0 and the step α_k, the explicit scheme is

$$Y_{k+1} = Y_k - \alpha_k \bigtriangledown \varphi(Y_k), \tag{6}$$

where Y_k is the approximation solution. The explicit scheme satisfies the sufficient decrease property using the gradient descent (GD) method.

Let X_k and X_{k-1} be the current and previous states and the extrapolation weight be w_k. Using the explicit method (6), the APG iterative scheme is

$$\begin{aligned} w_k &= k - 2/k + 1, \\ Y_k &= (1 + w_k)X_k - wX_{k-1}, \\ Y_{k+1} &= Y_k - \alpha_k \bigtriangledown \varphi(Y_k). \end{aligned} \tag{7}$$

Together with the standard backtracking, we adopt the step size α_k when the following condition holds:

$$\varphi(Y_k) - \varphi(Y_{k+1}) \geq \eta \|Y_{k+1} - Y_k\|^2, \tag{8}$$

for some $\eta > 0$.

Combining (7) and (8), the APG algorithm is summarized in Algorithm 1.

Algorithm 1 APG algorithm.

Require: $X_0, tol, \alpha_0, \eta > 0, \beta \in (0,1)$, and $k = 1$.

1: **while** the stop condition is not satisfied **do**
2: Update Y_k via Equation (7);
3: **if** Equation (8) holds **then**
4: *break*
5: **else**
6: $\alpha_k = \beta \alpha_k$;
7: Calculate Y_{k+1} via (7);
8: $k = k + 1$.

3.2. Restart APG Method

Recently, an efficient and convergent numerical algorithm has been developed for solving a discretized phase-field model by combining the APG method with the restart technique [30]. Unlike the APG method, the restart technique involves choosing $X_{k+1} = Y_{k+1}$ whenever the following condition holds:

$$\varphi(X_k) - \varphi(Y_{k+1}) \geqslant \gamma \|X_k - Y_{k+1}\|^2, \tag{9}$$

for some $\gamma > 0$. If the condition is not met, we restart the APG by setting $w_k = 0$.

The restart APG method (RAPG) is summarized in Algorithm 2.

Algorithm 2 RAPG algorithm.

Require: $X_0, tol, \alpha_0, \eta > 0, \gamma > 0, \beta \in (0,1)$, and $k = 1$.

while the stop condition is not satisfied **do**
  Calculate Y_{k+1} by APG Algorithm 1;
  if Equation (9) holds **then**
    $X_{k+1} = Y_{k+1}$ and update ω_{k+1};
  else
    $X_{k+1} = X_k$ and reset $\omega_{k+1} = 0$;
  $k = k + 1$.

3.3. A-APG Method

In RAPG Algorithm 2, we can adaptively estimate the step size α_k by using the line search technique. Define

$$s_k := X_k - X_{k-1}, g_k := \bigtriangledown\varphi(X_k) - \bigtriangledown\varphi(X_{k-1}).$$

We initialize the search step by the Barzilai–Borwein (BB) method, i.e.,

$$\alpha_k = \frac{tr(s_k^T s_k)}{tr(s_k^T g_k)} \text{ or } \frac{tr(g_k^T s_k)}{tr(g_k^T g_k)}. \tag{10}$$

Therefore, we obtain the A-APG algorithm summarized in Algorithm 3.

Algorithm 3 A-APG algorithm.

Require: $X_0, tol, \alpha_0, \eta > 0, \gamma > 0, \beta \in (0,1)$, and $k = 1$.

while the stop condition is not satisfied **do**
  Initialize α_k by BB step Equation (10);
  Update X_{k+1} by RAPG Algorithm 2.

3.4. Newton-APG Method

Despite the fast initial convergence speed of the gradient-based methods, the tail convergence speed becomes slow. Therefore, we use a practical Newton method to solve the minimization problem. We obtain the initial value from A-APG Algorithm 3, and then choose the Newton direction as the gradient in the explicit scheme in RAPG Algorithm 2. Then we have the Newton-APG method shown in Algorithm 4.

Algorithm 4 Newton-APG algorithm.

Require: $X0, \alpha 0, \gamma > 0, \eta > 0, \beta \in (0,1), \epsilon, tol$ and $k = 1$.

Obtain the initial value from A-APG Algorithm 3 by the precision ϵ;
while the stop condition is not satisfied **do**
  Initialize α_k by BB step Equation (10);
  Update X_{k+1} by RAPG Algorithm 2 using Newton direction.

3.5. Gradient Descent (GD) and Line Search (LGD) Methods

Moreover, we show gradient descent (GD) and line search (LGD) methods for comparing with the A-APG and Newton-APG methods. The GD and line search LGD methods are summarized in Algorithm 5.

Algorithm 5 GD and LGD algorithms.

Require: $X_0, tol, \alpha_0, \eta > 0, \beta \in (0,1)$, and $k = 1$.

while the stop condition is not satisfied **do**
  if the step size is fixed **then**
    Calculate X_{k+1} via $X_{k+1} = X_k - \alpha \bigtriangledown \varphi(X_k)$ using GD;
  else
    Initialize α_k by BB step Equation (10);
    if Equation (8) holds **then**
      break
    else
      $\alpha_k = \beta\alpha_k$;
    Calculate X_{k+1} via X_{k+1} via $X_{k+1} = X_k - \alpha \bigtriangledown \varphi(X_k)$ using LGD;
  $k = k + 1$.

3.6. Computational Complexity Analysis

Further, we analyze the computational complexity of each iteration of the derived algorithms.

The computation of APG is mainly controlled by matrix multiplication and addition operations in three main parts. The iterative scheme needs $4m^2n + 4mn^2 + O(mn)$ computational complexity. The backtracking linear search needs $14m^2n + 20n^2m + 6n^3 + O(mn) + O(n^2)$ computational complexity defined by Equation (8). The extrapolation needs $O(mn)$ computational complexity defined by the Equation (7). The total computational complexity is $18m^2n + 24n^2m + 6n^3 + O(mn) + O(n^2)$ in Algorithm 1.

The computation of RAPG is mainly controlled by matrix multiplication and addition operations in four main parts. The iterative scheme needs $4m^2n + 4mn^2 + O(mn)$ computational complexity. The backtracking linear search defined by Equation (8) needs $14m^2n + 20n^2m + 6n^3 + O(mn) + O(n^2)$ computational complexity. The extrapolation defined by Equation (7) needs $O(mn)$ computational complexity. The restart defined by Equation (9) needs $4m^2n + 14n^2m + 4n^3 + O(mn) + O(n^2)$ computational complexity. The total computational complexity is $22m^2n + 38n^2m + 10n^3 + O(mn) + O(n^2)$ in Algorithm 2.

The computation of A-APG is mainly controlled by matrix multiplication and addition operations in four main parts. The iterative scheme needs $4m^2n + 4mn^2 + O(mn)$ computational complexity. The BB step and the backtracking linear search defined by Equations (8) and (10) need mn, $4m^2n + 4mn^2 + 6mn$, $2n^2(2m-1) + 2n$, and $14m^2n + 20n^2m + 6n^3 + O(mn) + O(n^2)$ computational complexity. The extrapolation defined by Equation (7) needs $O(mn)$ computational complexity. The restart defined by Equation (9) needs $4m^2n + 14n^2m + 4n^3 + O(mn) + O(n^2)$ computational complexity. The total computational complexity is $26m^2n + 46n^2m + 10n^3 + O(mn) + O(n^2)$ in Algorithm 3.

The computation of Newton-APG is mainly controlled by matrix multiplication and addition operations in four main parts, different from the A-APG method. The iterative scheme needs $8n^3 + 3n^2 + O(n^2) + O(n^3)$ computational complexity. The BB step and the backtracking linear search defined by Equations (8) and (10) need n^2, $8n^3 + 6n^2$, $2n^2(2n-1) + 2n$, and $10n^2(2n-1) + 8n^3 + 3n^2 + O(n^3) + O(n^2)$ computational complexity. The extrapolation defined by Equation (7) needs $O(n^2)$ computational complexity. The restart defined by Equation (9) needs $5n^2(2n-1) + n^2 + O(n^3)$ computational complexity. The total computational complexity is $50n^3 - 10n^2 + 2n + O(n^2) + O(n^3)$ in Algorithm 4.

The computation of GD is mainly controlled by matrix multiplication and addition operations in Equations (4) and (6). It requires $mn(2m-1)$, $mn(2n-1)$, $mn(2m-1)$, $mn(2n-1)$ computational complexity to compute AX, XB, A^TX, XB^T. The total computational complexity is $4m^2n + 4mn^2 + O(mn)$ in Algorithm 5 using GD.

The computation of LGD is mainly controlled by matrix multiplication and addition operations in the calculation of s, g defined by Equation (8) and (10), and the calculation of GD, which require mn, $4m^2n + 4mn^2 + 6mn$, $2n^2(2m-1) + 2n$, $14m^2n + 20n^2m +$

$6n^3 + O(mn) + O(n^2)$, and $4m^2n + 4mn^2 + O(mn)$, respectively. The total computational complexity is $22m^2n + 32n^2m + 6n^3 + O(mn) + O(n^2)$ in Algorithm 5 using GD.

4. Convergent Analysis

In this section, we focus on the convergence analysis of A-APG Algorithm 3. The following proposition is required.

Proposition 4. *Let M be a bounded region that contains $\{\varphi \leqslant \varphi(X_0)\}$ in $\mathbb{R}^{n\times n}$, then $\nabla\varphi(X)$ satisfies the Lipschitz condition in M, i.e., there exists $L_M > 0$ such that*

$$\|\nabla\varphi(X) - \nabla\varphi(Y)\| \leqslant L_M\|X - Y\| \ for \ X, Y \in M.$$

Proof. Using the continuity of $\nabla\varphi(X)$, note that

$$\left\|\nabla^2\varphi(X)\right\| = \left\|(A + A^T) + (B + B^T)\right\|$$

defined by (5) is bounded. Then $\nabla\varphi(X)$ satisfies the Lipschitz condition in M. □

In recent years, the proximal method based on the Bregman distance has been applied for solving optimization problems. The proximal operator is

$$\mathrm{Prox}_{\varphi}^{\alpha}(y) := \underset{y}{\mathrm{argmin}}\{\varphi(y) + \frac{1}{2\alpha}\|X - X_k\|^2\}.$$

Basically, given the current estimation X_k and step size $\alpha_k > 0$, update X_{k+1} via

$$X_{k+1} = \mathrm{Prox}_0^{\alpha}(X_k - \alpha_k\nabla\varphi(X_k)) = \underset{X}{\mathrm{argmin}}\{\frac{1}{2\alpha_k}\|X - (X_k - \alpha_k\nabla\varphi(X_k)\|^2\}. \tag{11}$$

Thus we obtain

$$\frac{1}{2\alpha_k}(X_{k+1} - (X_k - \alpha_k\nabla\varphi(X_k))) = 0,$$

which implies that

$$X_{k+1} = X_k - \alpha_k\nabla\varphi(X_k).$$

This is exactly the explicit scheme in our algorithm.

4.1. Linear Search Is Well-Defined

Using the optimization from Equation (11), it is evident that

$$\begin{aligned} X_{k+1} &= \underset{X}{\mathrm{argmin}}\{\frac{1}{2\alpha_k}\|X - (X_k - \alpha_k\nabla\varphi(X_k)\|^2\} \\ &= \underset{X}{\mathrm{argmin}}\{\frac{1}{2\alpha_k}\|X - X_k\|^2 + \langle X - X_k, \nabla\varphi(X_k)\rangle\} \\ &= \underset{X}{\mathrm{argmin}}\{\frac{1}{2\alpha_k}\|X - X_k\|^2 + \langle X - X_k, \nabla\varphi(X_k)\rangle + \varphi(X_k)\}. \end{aligned}$$

Then we obtain

$$\begin{aligned} \varphi(X_k) &\geqslant \frac{1}{2\alpha_k}\|X_{k+1} - X_k\|^2 + \langle X_{k+1} - X_k, \nabla\varphi(X_k)\rangle + \varphi(X_k) \\ &\geqslant \varphi(X_{k+1}) + \frac{1}{2\alpha_k}\|X_k - X_{k+1}\|^2 - \frac{\left\|\nabla^2\varphi(X)\right\|}{2}\|X_k - X_{k+1}\|^2 \\ &\geqslant \varphi(X_{k+1}) + (\frac{1}{2\alpha_k} - \frac{L_M}{2})\|X_k - X_{k+1}\|^2, \end{aligned} \tag{12}$$

where the second inequality follows from Taylor expansion of $\varphi(X_{k+1})$. By Equation (12), set

$$0 < \alpha_k < \overline{\alpha} := \min\{\frac{1}{L_M + 2\eta}, \frac{1}{L_M + 2\gamma}\}, \tag{13}$$

the conditions in linear search Equation (8) and non-restart Equation (9) are both satisfied. Therefore, the backtracking linear search is well-defined.

4.2. Sufficient Decrease Property

In this section, we show the sufficient decrease property of the sequence generated by A-APG Algorithm 3. If α_k satisfies the condition Equation (13), then

$$\varphi(X_k) - \varphi(Y_{k+1}) \geqslant \rho_1 \|X_k - Y_{k+1}\|^2,$$

where $\rho_1 = \min\{\eta, \gamma\} > 0$. Since φ is a bounded function, then there exists φ^* such that $\varphi(X_k) \geqslant \varphi^*$ and $\varphi(X_k) \to \varphi^*$ as $k \to +\infty$. This implies

$$\rho_1 \sum_{k=0}^{\infty} \|X_{k+1} - X_k\|^2 \leqslant \varphi(X_0) - \varphi^* < +\infty,$$

which shows that

$$\lim_{k \to +\infty} \|X_{k+1} - X_k\| = 0.$$

4.3. Bounded Gradient

Define two sets $\Omega_2 = \{k : k = 2\}$ and $\Omega_1 = N \setminus \Omega_2$. Let $w_k = k - 2/k + 1$, for any $k \in \Omega_2$, then $X_{k+1} = Y_{k+1}$ when $w_k = 0$. There exists $\overline{w} = k_{max} - 2/k_{max} + 1 \in [0, 1)$ such that $w_k \leqslant \overline{w}$ as k increases. If $k \in \Omega_1$, since

$$Y_{k+1} = \underset{X}{\operatorname{argmin}}\{\frac{1}{2\alpha_k}\|X - (Y_k - \alpha_k \nabla\varphi(Y_k))\|^2\},$$

we have

$$0 = \nabla\varphi(Y_k) + \frac{1}{\alpha_k}(Y_{k+1} - Y_k).$$

Thus,

$$\nabla\varphi(Y_k) = \frac{1}{\alpha_k}(Y_k - X_{k+1}).$$

Note that $Y_k = (1 + w_k)X_k - w_k X_{k-1}$, then

$$\begin{aligned} \|\nabla\varphi(Y_k)\| &= \frac{1}{\alpha_k}\|(1 + w_k)X_k - w_k X_{k-1} - X_{k+1}\| \\ &= \frac{1}{\alpha_k}\|w_k(X_k - X_{k-1}) + (X_k - X_{k+1})\| \\ &\leqslant \frac{1}{\alpha_{min}}(\overline{w}\|X_k - X_{k-1}\| + \|X_k - X_{k+1}\|) \\ &= c_1(\|X_{k+1} - X_k\| + \overline{w}\|X_k - X_{k-1}\|), \end{aligned} \tag{14}$$

where $c_1 = \frac{1}{\alpha_{min}} > 0$.

If $k \in \Omega_2$, then

$$X_{k+1} = \underset{X}{\operatorname{argmin}}\{\frac{1}{2\alpha_k}\|X - (X_k - \alpha_k \nabla\varphi(X_k))\|^2\},$$

which implies that

$$0 = \nabla\varphi(X_k) + \frac{1}{\alpha_k}(X_{k+1} - X_k).$$

Thus

$$\|\nabla\varphi(X_k)\| = \frac{1}{\alpha_k}\|X_k - X_{k+1}\| \leqslant \frac{1}{\alpha_{min}}\|X_k - X_{k+1}\| = c_1(\|X_k - X_{k+1}\|), \tag{15}$$

Combining Equations (14) and (15), it follows that

$$\|\nabla\varphi(X_k)\| \leqslant c_1(\|X_{k+1} - X_k\| + \overline{w}\|X_k - X_{k-1}\|).$$

4.4. Subsequence Convergence

As $\{X_k\} \in M$ is compact, there exists a subsequence $\{X_{k_j}\} \subset M$ and $X^* \in M$ such that $\lim_{j\to+\infty} X_{k_j} = X^*$. Then φ is bounded, i.e., $\varphi(X) > -\infty$ and φ keeps decreasing. Hence, there exists φ^* such that $\lim_{k\to+\infty} \varphi(X_k) = \varphi^*$. Note that

$$\varphi(X_k) - \varphi(X_{k+1}) \geqslant c_0\|X_k - X_{k+1}\|^2, \ \ k = 1, 2, \ldots \tag{16}$$

Summation over k yields

$$c_0 \sum_{k=0}^{\infty} \|X_k - X_{k+1}\|^2 \leqslant \varphi(X_0) - \varphi^* < +\infty.$$

Therefore,

$$\lim_{k\to+\infty} \|X_k - X_{k+1}\| = 0.$$

Due to the property of the gradient, thus

$$\lim_{j\to+\infty} \left\|\nabla\varphi(X_{k_j})\right\| = 0.$$

Considering the continuity of φ and $\nabla\varphi$, we have

$$\lim_{j\to+\infty} \varphi(X_{k_j}) = \varphi(X^*), \lim_{j\to+\infty} \nabla\varphi(X_{k_j}) = \nabla\varphi(X^*) = 0,$$

which implies that $\nabla\varphi(X^*) = 0$.

4.5. Sequence Convergence

In this section, the subsequence convergence can be strengthened by using the Kurdyka–Lojasiewicz property.

Proposition 5. *For $\overline{x} \in dom\ \partial\varphi := \{x : \partial\varphi(x) \neq \varnothing\}$, there exists $\eta > 0$, an ϵ neighborhood of $\overline{x}$, and $\psi \in \Psi_\eta = \{\psi \in C[0, \eta) \cap C'(0, \eta)$, where ψ is concave, $\psi(0) = 0, \psi' > 0$ on $(0, \eta)\}$ such that for all $x \in \Gamma_\eta(\overline{x}, \epsilon) : U \cap \{x : \varphi(\overline{x}) < \varphi(x) < \varphi(\overline{x}) + \eta\}$, we have*

$$\psi'(\varphi(x) - \varphi(\overline{x}))\|\nabla\varphi(x)\| \geqslant 1.$$

Then we say $\varphi(x)$ satisfies the Kurdyka–Lojasiewicz property.

Theorem 1. *Assume that Propositions 4 and 5 are met. Let $\{X_k\}$ be the sequence generated by A-APG Algorithm 3. Then, there exists a point $X^* \in M$ so that $\lim_{k\to+\infty} X_k = X^*$ and $\nabla\varphi(X^*) = 0$.*

Proof. Let $\omega(X_0)$ be the set of limiting points of the sequence $\{X_k\}$. Based on the boundedness of $\{X_k\}$ and the fact that $\omega(X_0) = \overline{\cap_{q\in N} \cup_{k>q} \{X_k\}}$, it follows that $\omega(X_0)$ is a non-empty and compact set. In addition, by Equation (16), we know that $\varphi(X)$ is a constant on $\omega(X_0)$, denoted by φ^*. If there exists some k_0 such that $\varphi(X_{k_0}) = \varphi^*$, then for $\forall k > k_0$, we have $\varphi(X_k) = \varphi^*$. Next, we assume that $\forall k, \varphi(X_k) > \varphi^*$. Therefore, for

$\forall \epsilon, \eta > 0, \exists l > 0$, for $\forall k > l$ we have $dist(\omega(X_0), X_k) \leqslant \epsilon$ and $\varphi^* < \varphi(X_k) < \varphi^* + \eta$ i.e., for $\forall X^* \in \omega(X_0), X \in \Gamma_\eta(X^*, \epsilon)$. Applying Proposition 5, for $\forall k > l$, we have

$$\psi'(\varphi(X_k) - \varphi^*)\|\nabla\varphi(X_k)\| \geqslant 1.$$

Then

$$\psi'(\varphi(X_k) - \varphi^*) \geqslant \frac{1}{c_1(\|X_k - X_{k-1}\| + \overline{w}\|X_{k-1} - X_{k-2}\|)}. \tag{17}$$

By the convexity of ψ, it is obvious that

$$\psi(\varphi(X_k) - \varphi^*) - \psi(\varphi(X_{k+1}) - \varphi^*) \geqslant \psi'(\varphi(X_k) - \varphi^*)(\varphi(X_k) - \varphi(X_{k+1})). \tag{18}$$

Define

$$\triangle_{p,q} = \psi(\varphi(X_p) - \varphi^*) - \psi(\varphi(X_q) - \varphi^*), \;\; c = (1 + \overline{w})c_1/c_0 > 0.$$

Combining with Equations (16)–(18), for $\forall k > l$, we obtain

$$\begin{aligned} \triangle_{k,k+1} &\geqslant \frac{c_0\|X_{k+1} - X_k\|^2}{c_1(\|X_k - X_{k-1}\| + \overline{w}\|X_{k-1} - X_{k-2}\|)} \\ &\geqslant \frac{\|X_{k+1} - X_k\|^2}{c(\|X_k - X_{k-1}\| + \|X_{k-1} - X_{k-2}\|)}. \end{aligned} \tag{19}$$

Applying the geometric inequality to Equation (19), thus

$$2\|X_{k+1} - X_k\| \leqslant \frac{1}{2}(\|X_k - X_{k-1}\| + \|X_{k-1} - X_{k-2}\|) + 2c\triangle_{k,k+1}.$$

Therefore, for $\forall k > l$, summing up the above inequality for $i = l + 1, \ldots, k$, we obtain

$$\begin{aligned} 2\sum_{i=l+1}^{k}\|X_{i+1} - X_i\| \leqslant& \frac{1}{2}\sum_{i=l+1}^{k}(\|X_i - X_{i-1}\| + \|X_{i-1} - X_{i-2}\|) + 2c\sum_{i=l+1}^{k}\triangle_{i,i+1} \\ \leqslant& \sum_{i=l+1}^{k}\|X_{i+1} - X_i\| + \|X_{l+1} - X_l\| + \frac{1}{2}\|X_l - X_{l-1}\| \\ &+ 2c\triangle_{l+1,k+1}. \end{aligned}$$

For $\forall k > l, \psi \geqslant 0$, it is evident that

$$\sum_{i=l+1}^{k}\|X_{i+1} - X_i\| \leqslant \|X_{l+1} - X_l\| + \frac{1}{2}\|X_l - X_{l-1}\| + 2c\psi(\varphi(X_l) - \varphi^*),$$

which implies that

$$\sum_{k=1}^{\infty}\|X_{k+1} - X_k\| < \infty.$$

In the end, we have $\lim_{k\to+\infty} X_k = X^*$. □

5. Numerical Results

In this section, we offer two corresponding numerical examples to illustrate the efficiency of the derived algorithms. All code is written in Python language. Denote iteration and error by the iteration step and error of the objective function. We take the matrix order "n" as 128, 1024, 2048, and 4096.

Example 1. *Let*

$$A_1 = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}, B_1 = \begin{pmatrix} 1 & 0.5 & & & \\ 0.5 & 1 & 0.5 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 0.5 \\ & & & 0.5 & 1 \end{pmatrix}$$

be tridiagonal matrices in the Sylvester Equation (1). Set the matrix C_1 as the identity matrix. The initial step size is 0.01, which is small enough to iterate. The parameters are $\eta_1 = 0.25, \omega_1 = 0.2$ taken from (0,1) randomly. Table 1 and Figure 1 show the numerical results of Algorithms 1–5. It can be seen that the LGD, A-APG, and Newton-APG Algorithms are more efficient than other methods. Moreover, the iteration step does not increase when the matrix order increases due to the same initial value. The A-APG method has higher error accuracy compared with other methods. The Newton-APG method takes more CPU time and fewer iteration steps than the A-APG method. The Newton method needs to calculate the inverse of the matrix, while it has quadratic convergence. From Figure 1, the error curves of the LGD, A-APG, and Newton-APG algorithms are hard to distinguish. We offer another example below.

Table 1. Numerical results for Example 1.

Algorithm	n	Iteration	Error	Time(s)
GD	128	356	1.13687×10^{-13}	3.30
LGD	128	15	1.26477×10^{-12}	0.27
APG	128	374	1.4353×10^{-12}	4.31
RAPG	128	69	1.4353×10^{-12}	1.45
A-APG	128	19	3.55271×10^{-14}	0.38
Newton-APG	128	18	9.47438×10^{-11}	0.48
CG	128	19	3.49364×10^{-14}	0.42
GD	1024	356	1.02318×10^{-12}	806
LGD	1024	15	1.06866×10^{-11}	69
APG	1024	374	1.18803×10^{-11}	1261
RAPG	1024	69	2.59774×10^{-11}	367
A-APG	1024	19	2.84217×10^{-13}	113
Newton-APG	1024	18	8.95682×10^{-10}	144
CG	1024	19	3.37046×10^{-14}	71
GD	2048	356	2.04636×10^{-12}	6315
LGD	2048	15	2.13731×10^{-11}	569
APG	2048	374	2.38742×10^{-11}	9752
RAPG	2048	69	5.20686×10^{-11}	2994
A-APG	2048	19	6.82121×10^{-13}	926
Newton-APG	2048	18	8.95682×10^{-10}	1015
CG	2048	19	3.34616×10^{-14}	521
GD	4096	356	4.09273×10^{-12}	66,155
LGD	4096	15	4.27463×10^{-11}	4199
APG	4096	374	4.77485×10^{-11}	71,636
RAPG	4096	69	1.04365×10^{-10}	21,596
A-APG	4096	19	1.81899×10^{-12}	6829
Newton-APG	4096	18	3.64571×10^{-9}	7037
CG	4096	19	3.33322×10^{-14}	3553

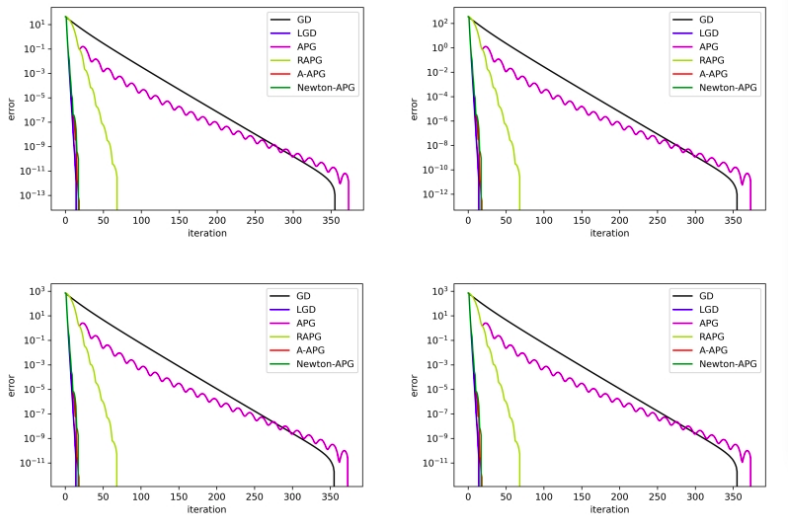

Figure 1. The error curves when n = 128, 1024, 2048, 4096 for Example 1.

Example 2. *Let* $A_2 = A_1A_1^T, B_2 = B_1B_1^T$ *be positive semi-definite matrices in the Sylvester Equation (1). Set the matrix* C_2 *as the identity matrix. The initial step size is 0.009. The parameters are* $\eta_2 = 0.28, \omega_2 = 0.25$ *taken from (0,1) randomly. Table 2 and Figure 2 show the numerical results of Algorithms 1–5. It can be seen that the LGD, A-APG, and Newton-APG algorithms take less CPU time compared with other methods. Additionally, we can observe the different error curves of the LGD, A-APG, and Newton-APG algorithms from Figure 2.*

Remark 1. *The difference of the iteration step in Examples 1 and 2 emerges due to the given different initial values. It can be seen that the LGD, A-APG, and Newton-APG algorithms have fewer iteration steps. Whether the A-APG method or Newton-APG yields fewer iteration steps varies from problem to problem. From Examples 1 and 2, we observe that the A-APG method has higher accuracy, although it takes more time and more iteration steps than the LGD method.*

Remark 2. *Moreover, we compare the performance of our methods with other methods such as the conjugate gradient method (CG) in Tables 1 and 2. We take the same initial values and set the error to* 1×10^{-14}. *From Tables 1 and 2, it can be seen that the LGD and A-APG methods are more efficient for solving the Sylvester matrix equation when the order n is small. When n is large, the LGD and A-APG methods nearly have a convergence rate with the CG method.*

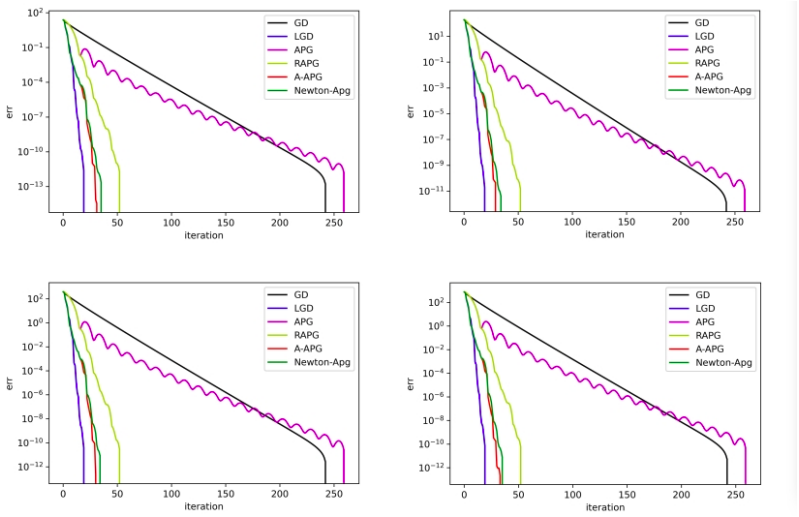

Figure 2. The error curves when n = 128, 1024, 2048, 4096 for Example 2.

Table 2. Numerical results for Example 2.

Algorithm	n	Iteration	Error	Time(s)
GD	128	243	1.63425×10^{-13}	2.38
LGD	128	20	2.45137×10^{-12}	0.47
APG	128	260	1.58096×10^{-12}	4.51
RAPG	128	53	1.90781×10^{-12}	1.46
A-APG	128	32	3.55271×10^{-15}	0.78
Newton-APG	128	36	2.30926×10^{-13}	1.26
CG	128	34	4.13025×10^{-14}	0.79
GD	1024	243	1.3074×10^{-12}	516
LGD	1024	20	1.89573×10^{-11}	95
APG	1024	260	1.25056×10^{-11}	835
RAPG	1024	53	1.51772×10^{-11}	267
A-APG	1024	32	4.61569×10^{-14}	181
Newton-APG	1024	36	4.20641×10^{-12}	214
CG	1024	34	4.29936×10^{-14}	92
GD	2048	243	2.6148×10^{-12}	4129
LGD	2048	20	3.78577×10^{-11}	814
APG	2048	260	2.48974×10^{-11}	6507
RAPG	2048	53	3.03544×10^{-11}	2193
A-APG	2048	32	2.27374×10^{-13}	1622
Newton-APG	2048	36	8.52651×10^{-12}	2125
CG	2048	34	4.22694×10^{-14}	797
GD	4096	243	5.22959×10^{-12}	29,859
LGD	4096	20	7.54881×10^{-11}	6023
APG	4096	260	4.97948×10^{-11}	48,238
RAPG	4096	53	6.07088×10^{-11}	16,482
A-APG	4096	32	2.27374×10^{-13}	12,896
Newton-APG	4096	36	7.95808×10^{-12}	14,901
CG	4096	34	4.18275×10^{-14}	5337

6. Conclusions

In this paper, we have introduced the A-APG and Newton-APG methods for solving the Sylvester matrix equation. The key idea is to change the Sylvester matrix equation to an optimization problem by using the Kronecker product. Moreover, we have analyzed the computation complexity and proved the convergence of the A-APG method. Convergence results and preliminary numerical examples have shown that the schemes are promising in solving the Sylvester matrix equation.

Author Contributions: J.Z. (methodology, review, and editing); X.L. (software, visualization, data curation). All authors have read and agreed to the published version of the manuscript.

Funding: The work was supported in part by the National Natural Science Foundation of China (12171412, 11771370), Natural Science Foundation for Distinguished Young Scholars of Hunan Province (2021JJ10037), Hunan Youth Science and Technology Innovation Talents Project (2021RC3110), the Key Project of the Education Department of Hunan Province (19A500, 21A0116).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dooren, P.M.V. *Structured Linear Algebra Problems in Digital Signal Processing*; Springer: Berlin/Heidelberg, Germany, 1991.
2. Gajic, Z.; Qureshi, M.T.J. *Lyapunov Matrix Equation in System Stability and Control*; Courier Corporation: Chicago, IL, USA, 2008.
3. Corless, M.J.; Frazho, A. *Linear Systems and Control: An Operator Perspective*; CRC Press: Boca Raton, FL, USA, 2003.
4. Stewart, G.W.; Sun, J. *Matrix Perturbation Theory*; Academic Press: London, UK, 1990.
5. Simoncini, V.; Sadkane, M. Arnoldi-Riccati method for large eigenvalue problems. *BIT Numer. Math.* **1996**, *36*, 579–594.
6. Demmel, J.W. Three methods for refining estimates of invariant subspaces. *Computing* **1987**, *38*, 43–57.
7. Chen, T.W.; Francis, B.A. *Optimal Sampled-Data Control Systems*; Springer: London, UK, 1995.
8. Datta, B. *Numerical Methods for Linear Control Systems*; Elsevier Inc.: Amsterdam, The Netherlands, 2004.
9. Lord, N. Matrix computations. *Math. Gaz.* **1999**, *83*, 556–557.
10. Zhao, X.L.; Wang, F.; Huang, T.Z. Deblurring and sparse unmixing for hyperspectral images. *IEEE Trans. Geosci. Remote Sens.* **2013**, *51*, 4045–4058.
11. Obinata, G.; Anderson, B.D.O. *Model Reduction for Control System Design*; Springer Science & Business Media: London, UK, 2001.
12. Bouhamidi, A.; Jbilou, K. A note on the numerical approximate solutions for generalized Sylvester matrix equations with applications. *Appl. Math. Comput.* **2008**, *206*, 687–694.
13. Bai, Z.Z.; Benzi, M.; Chen, F. Modified HSS iteration methods for a class of complex symmetric linear systems. *Computing* **2010**, *87*, 93–111.
14. Bartels, R.H.; Stewart, G.W. Solution of the matrix equation $AX + XB = C$. *Commun. ACM* **1972**, *15*, 820–826.
15. Golub, G.H. *A Hessenberg-Schur Method for the Problem $AX + XB = C$*; Cornell University: Ithaca, NY, USA, 1978.
16. Robbé, M.; Sadkane, M. A convergence analysis of GMRES and FOM methods for Sylvester equations. *Numer. Algorithms* **2002**, *30*, 71–89.
17. Guennouni, A.E.; Jbilou, K.; Riquet, A.J. Block Krylov subspace methods for solving large Sylvester equations. *Numer. Algorithms* **2002**, *29*, 75–96.
18. Salkuyeh, D.K.; Toutounian, F. New approaches for solving large Sylvester equations. *Appl. Math. Comput.* **2005**, *173*, 9–18.
19. Wachspress, E.L. Iterative solution of the Lyapunov matrix equation. *Appl. Math. Lett.* **1988**, *1*, 87–90.
20. Jbilou, K.; Messaoudi, A.; Sadok, H. Global FOM and GMRES algorithms for matrix equations. *Appl. Numer. Math.* **1999**, *31*, 49–63.
21. Feng, D.; Chen, T. Gradient Based Iterative Algorithms for Solving a Class of Matrix Equations. *IEEE Trans. Autom. Control* **2005**, *50*, 1216–1221.
22. Heyouni, M.; Movahed, F.S.; Tajaddini, A. On global Hessenberg based methods for solving Sylvester matrix equations. *Comput. Math. Appl.* **2019**, *77*, 77–92.
23. Benner, P.; Kürschner, P. Computing real low-rank solutions of Sylvester equations by the factored ADI method. *Comput. Math. Appl.* **2014**, *67*, 1656–1672.
24. Bouhamidi, A.; Hached, M.; Heyouni, M.J.; Bilou, K. A preconditioned block Arnoldi method for large Sylvester matrix equations. *Numer. Linear Algebra Appl.* **2013**, *20*, 208–219.
25. Heyouni, M. Extended Arnoldi methods for large low-rank Sylvester matrix equations. *Appl. Numer. Math.* **2010**, *60*, 1171–1182.
26. Agoujil, S.; Bentbib, A.H.; Jbilou, K.; Sadek, E.M. A minimal residual norm method for large-scale Sylvester matrix equations. *Electron. Trans. Numer. Anal. Etna* **2014**, *43*, 45–59.
27. Abdaoui, I.; Elbouyahyaoui, L.; Heyouni, M. An alternative extended block Arnoldi method for solving low-rank Sylvester equations. *Comput. Math. Appl.* **2019**, *78*, 2817–2830.
28. Jbilou, K. Low rank approximate solutions to large Sylvester matrix equations. *Appl. Math. Comput.* **2005**, *177*, 365–376.
29. Liang, B.; Lin, Y.Q.; Wei, Y.M. A new projection method for solving large Sylvester equations. *Appl. Numer. Math.* **2006**, *57*, 521–532.
30. Jiang, K.; Si, W.; Chen, C.; Bao, C. Efficient numerical methods for computing the stationary states of phase-field crystal models. *SIAM J. Sci. Comput.* **2020**, *42*, B1350–B1377.
31. Beck, A.; Teboulle, M. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM J. Imaging Sci.* **2009**, *2*, 183–202.
32. Bao, C.; Barbastathis, G.; Ji, H.; Shen, Z.; Zhang, Z. Coherence retrieval using trace regularization. *SIAM J. Imaging Sci.* **2018**, *11*, 679–706.
33. Magnus, J.R.; Neudecker, H. *Matrix Differential Calculus*; John Willey & Sons: Hoboken, NJ, USA, 1999.