algorithms

MDPI

Article

A Parallel Two-Stage Iteration Method for Solving Continuous Sylvester Equations

Manyu Xiao [1,*], Quanyi Lv [1], Zhuo Xing [1] and Yingchun Zhang [2]

[1] Department of Applied Mathematics, Northwestern Polytechnical University, Xi'an 710072, China; luquan@nwpu.edu.cn (Q.L.); xzhuoxing0923@163.com (Z.X.)

[2] Department of Mechanical Engineering, Northwestern Polytechnical University, Xi'an 710072, China; zhangyingchun@mail.nwpu.edu.cn

* Correspondence: manyuxiao@nwpu.edu.cn; Tel.: +86-152-2931-3508

Received: 8 July 2017; Accepted: 10 August 2017; Published: 21 August 2017

Abstract: In this paper we propose a parallel two-stage iteration algorithm for solving large-scale continuous Sylvester equations. By splitting the coefficient matrices, the original linear system is transformed into a symmetric linear system which is then solved by using the SYMMLQ algorithm. In order to improve the relative parallel efficiency, an adjusting strategy is explored during the iteration calculation of the SYMMLQ algorithm to decrease the degree of the reduce-operator from two to one communications at each step. Moreover, the convergence of the iteration scheme is discussed, and finally numerical results are reported showing that the proposed method is an efficient and robust algorithm for this class of continuous Sylvester equations on a parallel machine.

Keywords: continuous Sylvester equations; SYMMLQ algorithm; two-stage iteration; parallel computing

1. Introduction

The solution of the continuous Sylvester equation

$$AX + XB = F \tag{1}$$

with large sparse matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times n}$, $X \in \mathbb{R}^{n\times n}$, $F \in \mathbb{R}^{n\times n}$ and with A, B positive definite is a common task in numerical linear algebra. It arises in many scientific computing and engineering applications, such as control theory [1,2], neural networks, model reduction [3], image processing [4], and so on. Therefore, the problem has remained an active area of research. In this context, recent methodological advances have been thoroughly discussed in many papers [5–20]. Iterative methods for solving linear or nonlinear equations have seen constant improvement in recent years to reduce the computational time; for example, two multi-step derivative-free iterative methods [5]: block Jacobi two stage method [6] and SYMMLQ algorithm [7–9]. In addition, widely-used direct methods are, for instance, the Bartels–Stewart [10] and the Hessenberg–Schur method [11]. The main idea is to transform A and B into triangular or Hessenberg form [21] by an orthogonal similarity transformation and then to solve the resulting system of linear equations directly by a back-substitution process. However, this method is not applicable in large-scale problems due to the prohibitive computational issue. In order to overcome this limitation, fast iterative methods have been developed such as the Smith method [12], the alternating direction implicit (ADI) method [13], gradient-based methods [14,15], and the Krylov subspace-based algorithm [7,16,17]. At present, the conjugate gradient (CG) method [7] and the preconditioned conjugate gradient method [18] are popularly used with the advantages of small storage and suitability for parallel computing. Typically, the SYMMLQ algorithm [7–9] is quite efficient in the case of symmetric coefficient matrices, as it has tremendous advantages in small storage capacity and stable computations. However, it is not a good option for multi-computer systems due

to the high cost of global communication. For asymmetric coefficient matrices, a modified conjugate gradient method (MCG) is useful. However, its convergence speed is slow [22,23].

Another type of iteration based on splitting methods allows us to better utilize standard methodologies. For instance, Bai et al. [24] proposed Hermitian and skew-Hermitian splitting (HSS) iteration methods for solving systems of linear equations with non-Hermitian positive definite form. This has been studied widely and generalized in [25–28]. Recently, a Hermitian and skew-Hermitian splitting (HSS) iteration method for solving large sparse continuous Sylvester equations with non-Hermitian and positive definite/semidefinite matrices was discussed in [29]. Wang et al. [30] presented a positive-definite and skew-Hermitian splitting (PSS) iteration method, and in [31] Zhou et al. applied the modified Hermitian and skew-Hermitian splitting (MHSS) iteration method to solve the continuous Sylvester equation. Zheng and Ma in [32] applied the idea of the normal and skew-Hermitian splitting (NSS) iteration method to continuous Sylvester equations.

However, these iteration methods have the common difficulty that there is no accurate formula to determine the positive value of the corresponding parameter in the iteration scheme. In many articles, a large amount of work has been done to address this issue. Unfortunately this estimation methodology is still not fully resolved in practical applications. In addition, their implementations need to solve two continuous Sylvester equations, which results in great additional computational cost.

All of these brought about the need for the development and validation of an efficient parallel algorithm. In this paper we have proposed a parallel algorithm of two-stage iteration for solving large-scale continuous Sylvester equations with the combination of the HSS iteration method and the SYMMLQ algorithm. The main idea is to split the coefficient matrices into a symmetric and an anti-symmetric matrix, respectively. Then, the original equations are transferred into symmetric matrix equations which are solved by the SYMMLQ algorithm. Furthermore, we focus on the improvement of the parallel efficiency of the SYMMLQ algorithm by adjusting the calculation steps.

The remainder of this paper is organized as follows. In Section 2, a description of the two-stage iteration method is presented based on a splitting method and the SYMMLQ algorithm for solving the continuous Sylvester Equation (1). Then the parallel implementation of the algorithm is given in Section 3 . Its convergence analysis and numerical examples are mentioned in Sections 4 and 5, respectively. We end with conclusions.

Notation in this paper: A^{T} denotes the transpose of matrix A; inner product using $[A, B] = tr(A^{\mathrm{T}}B)$; matrix norm of A induced by $\|A\| = \sqrt{[A, A]} = \sqrt{tr(A^{\mathrm{T}}A)}$ and $\rho(A)$ is the spectral radius of the matrix A. For the matrix $X = (x_1, x_2, \cdots, x_n)^{\mathrm{T}} \in \mathbb{R}^{n\times n}$, $\overline{vec}(X)$ denotes the $\overline{vec}$ operator defined as $\overline{vec}(X) = (x_1^{\mathrm{T}}, x_2^{\mathrm{T}}, \cdots, x_n^{\mathrm{T}})^{\mathrm{T}} \in \mathbb{R}^{n^2}$.

2. Description of the Two-Stage Iteration Method

The two-stage iteration method consisting of the outer and inner iterations is discussed in this section. The outer iteration is performed by splitting the coefficient matrices, while the inner iteration is computed via the SYMMLQ algorithm.

2.1. Outer Iteration Scheme

We can split A and B into symmetric and antisymmetric parts:

$$A = M_1 - N_1,\ B = M_2 - N_2 \tag{2}$$

where M_1 and M_2 are symmetric parts and N_1 and N_2 are antisymmetric parts. They can be rewritten as

$$M_1 = \frac{1}{2}(A + A^{\mathrm{T}}),\ N_1 = \frac{1}{2}(A^{\mathrm{T}} - A),$$
$$M_2 = \frac{1}{2}(B + B^{\mathrm{T}}),\ N_2 = \frac{1}{2}(B^{\mathrm{T}} - B).$$

More details can be found in [24,28,29,31].

Then, the continuous Sylvester Equation (1) can be written as the following matrix equation:

$$M_1X + XM_2 = N_1X + XN_2 + F. \tag{3}$$

Given an initial guess $X^{(0)} \in \mathbb{R}^{n\times n}$, because the initial guess has an effect on the convergence speed of the algorithm, it has little influence on the calculation results. For convenience, the initial guess is taken as $X^{(0)} = O$ in the numerical examples. For $k = 0, 1, 2, \cdots$, we use the following iteration scheme until $\{X^{(k)}\}$ converges:

$$M_1X^{(k+1)} + X^{(k+1)}M_2 = N_1X^{(k)} + X^{(k)}N_2 + F. \tag{4}$$

Let $\widetilde{F}^{(k)} = N_1X^{(k)} + X^{(k)}N_2 + F$. Then, the outer iteration can be expressed as

$$M_1X^{(k+1)} + X^{(k+1)}M_2 = \widetilde{F}^{(k)}. \tag{5}$$

We need to solve Equation (5) at each step of the iteration method so as to form the two-stage iteration method. Equation (5) is computed by the SYMMLQ algorithm, since the new coefficient matrices are symmetric.

2.2. Inner Iteration Scheme Based on the SYMMLQ Algorithm

Equation (5) is changed into the following linear system by using the $\overline{vec}$ operator

$$Hx^{(k+1)} = \widetilde{f}^{(k)}, \tag{6}$$

where $H = A \otimes I + I \otimes B^{\mathrm{T}}$, the vectors $x^{(k+1)}$ and $\widetilde{f}$ contain the concatenated columns of the matrices $X^{(k+1)}$ and $\widetilde{F}^{(k)}$, respectively, with $\otimes$ being the Kronecker product symbol.

Then, the SYMMLQ algorithm is proposed to solve the kth step iteration equation of the outer iteration scheme. For more details, we can refer to [8,9]. The description of the corresponding SYMMLQ algorithm is given roughly in the following manner.

Let $y = x^{(k+1)}$ and $b = \widetilde{f}^{(k)}$. Then, Equation (6) can be written equivalently as

$$Hy = b. \tag{7}$$

Then, transform H into an $i \times i$ symmetric tridiagonal matrix T_i by the Lanczos orthogonalization process. The coefficient matrix H is potentially unstable, and consequently T_i is not positive definite. So, the LQ decomposition (Triangular Orthogonal decomposition, where L is a lower triangular matrix, Q is an orthogonal matrix) is used to transform T_i into an $i \times i$ lower triangular matrix L_i, seen in the following flow chart: The SYMMLQ algorithm:

$$H \xrightarrow[orthogonalization]{Lanczos} T_i \xrightarrow[decomposition]{LQ} L_iP_i. \tag{8}$$

where $T_i = L_iP_i$, and

$$T_i = \begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{i-1} \\ & & & b_{i-1} & a_i \end{pmatrix}, \; L_i = \begin{pmatrix} r_1 & & & & \\ \delta_2 & r_2 & & & \\ \varepsilon_3 & \delta_3 & r_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \varepsilon_i & \delta_i & r_i \end{pmatrix},$$

$$P_i = P_{i-1} \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & c_{i-1} & s_{i-1} \\ & & & -s_{i-1} & c_{i-1} \end{pmatrix}.$$

According to the above observation, we can establish the following stationary fixed-point iteration form for Equation (7):

$$y^{(i+1)} = y^{(i)} + Q_i z_i \tag{9}$$

where $Q = (q_1, q_2, \cdots, q_i)$ is an $n \times i$ matrix and $q_1, q_2, \ldots, q_i$ are orthonormal vectors which are computed by the Lanczos orthogonalization process. Here z_i satisfies the following equation:

$$T_i z_i = \|r^{(0)}\|_2 e_1, \tag{10}$$

where $r^{(0)} = b - Hy^{(0)}$, $y^{(0)}$ is a given initial vector, and $e_1 = (1, 0, \cdots, 0)^{\mathrm{T}}$ is an i-dimensional unit vector. More details can be found in the literature [33].

3. Parallel Implementation of the Two-Stage Iteration Method

In this section we discuss the parallel implementation including data storage, and implementation of outer iteration and inner iteration.

3.1. Data Storage

For convenience, let p be the number of processors, $p_i (i = 1, 2, \ldots, p)$ represent ith processor, and l is an integer in $n = pl$.

Mark

$$\begin{aligned}
&A = (A_1^{\mathrm{T}}, A_2^{\mathrm{T}}, \cdots, A_p^{\mathrm{T}})^{\mathrm{T}},\ B = (B_1^{\mathrm{T}}, B_2^{\mathrm{T}}, \cdots, B_p^{\mathrm{T}})^{\mathrm{T}},\ F = (F_1^{\mathrm{T}}, F_2^{\mathrm{T}}, \cdots, F_p^{\mathrm{T}})^{\mathrm{T}},\\
&X^{(0)} = ((X_1^{(0)})^{\mathrm{T}}, (X_2^{(0)})^{\mathrm{T}}, \cdots, (X_p^{(0)})^{\mathrm{T}})^{\mathrm{T}},\ A^{\mathrm{T}} = (\widetilde{A}_1^{\mathrm{T}}, \widetilde{A}_2^{\mathrm{T}}, \cdots, \widetilde{A}_p^{\mathrm{T}})^{\mathrm{T}},\\
&B^{\mathrm{T}} = (\widetilde{B}_1^{\mathrm{T}}, \widetilde{B}_2^{\mathrm{T}}, \cdots, \widetilde{B}_p^{\mathrm{T}})^{\mathrm{T}},\ \widetilde{F} = (\widetilde{F}_1^{\mathrm{T}}, \widetilde{F}_2^{\mathrm{T}}, \cdots, \widetilde{F}_p^{\mathrm{T}})^{\mathrm{T}},\\
&M_1 = ((M_{1,1})^{\mathrm{T}}, (M_{1,2})^{\mathrm{T}}, \cdots, (M_{1,p})^{\mathrm{T}})^{\mathrm{T}},\ M_2 = ((M_{2,1})^{\mathrm{T}}, (M_{2,2})^{\mathrm{T}}, \cdots, (M_{2,p})^{\mathrm{T}})^{\mathrm{T}},\\
&N_1 = ((N_{1,1})^{\mathrm{T}}, (N_{1,2})^{\mathrm{T}}, \cdots, (N_{1,p})^{\mathrm{T}})^{\mathrm{T}},\ N_2 = ((N_{2,1})^{\mathrm{T}}, (N_{2,2})^{\mathrm{T}}, \cdots, (N_{2,p})^{\mathrm{T}})^{\mathrm{T}}
\end{aligned}$$

where A_i, B_i, F_i, $X_i^{(0)}$, $\widetilde{A}_i$, $\widetilde{B}_i$, $\widetilde{F}_i$, $M_{1,i}$, $M_{2,i}$, $N_{1,i}$, $N_{2,i}$ $(i = 1, 2, \cdots, p)$ are $l \times n$ sub-block matrices. These are saved in row storage. Then, store A_i, B_i, F_i, $X_i^{(0)}$, $\widetilde{A}_i$, $\widetilde{B}_i$ on the processor $p_i (i = 1, 2, \cdots, p)$. **Note:** Due to the storage method, we chose the way of matrix multiplication with block row–row matrices in parallel computing process. Detailed descriptions of parallel computing Matrix multiplication are found in References [5,23,34].

3.2. Parallel Implementation of Outer Iteration Method

(1) Splitting process: Processor $p_i (i = 1, 2, \cdots, p)$ computes

$$M_{1,i} = (A_i + \widetilde{A}_i)/2,\ N_{1,i} = (\widetilde{A}_i - A_i)/2,\ M_{2,i} = (B_i + \widetilde{B}_i)/2,\ N_{2,i} = (\widetilde{B}_i - B_i)/2.$$

(2) Cycle processor:

Step 1. Processor $p_i (i = 1, 2, \cdots, p)$ computes

$$\Delta X_i^{(k)} = X_i^{(k)} - X_i^{(k-1)} \text{ and } [\Delta X_i^{(k)}, \Delta X_i^{(k)}],$$

get $[\Delta X_i^{(k)}, \Delta X_i^{(k)}]$ after all-reduce. If $\|\Delta X^{(k)}\| < \varepsilon$ stop; otherwise, turn to step 2.

Step 2. Compute $\widetilde{F}_i = N_{1,i}X^{(k)} + X_i^{(k)}N_2 + F_i$ in each processor.

Step 3. Use the improved parallel process of SYMMLQ algorithm to solve the new equation

$$M_1X^{(k+1)} + X^{(k+1)}M_2 = \widetilde{F}. \tag{11}$$

This step, which improves the parallel efficiency and reduces the parallel time by reducing the frequency of communication, plays an important role in the whole parallel implementation of the two-stage iteration method, and the detailed implementation can be seen in Sections 3.3 and 3.4.

Step 4. Let $k := k + 1$, turn to Step 1.

3.3. Parallel Implementation of Inner Iteration Scheme

(1) Compute process:

① Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$R_i^{(0)} = F_i - (A_iX^{(0)} + X_i^{(i)}B) \text{ and } [R_i^{(0)}, R_i^{(0)}],$$

obtain $[R_i^{(0)}, R_i^{(0)}]$ after all-reduce, then compute

$$Q_i^{(1)} = R_i^{(0)}/\|R^{(0)}\|.$$

② Compute

$$G_i^{(1)} = A_iQ^{(1)} + Q_i^{(1)}B \text{ and } [G_i^{(1)}, Q_i^{(1)}],$$

obtain $a_1 = [G_i^{(1)}, Q_i^{(1)}]$ after all-reduce. Compute

$$H_i^{(1)} = G_i^{(1)} - a_1Q_i^{(1)} \text{ and } [H_i^{(1)}, H_i^{(1)}],$$

get $[H^{(1)}, H^{(1)}]$ after all-reduce and $b_1 = \sqrt{[H^{(1)}, H^{(1)}}$, then compute $\widetilde{r}_1$,$\widetilde{\xi}_1$,$\widetilde{W}_i^{(1)}$ and $X_i^{(1)} = X_i^{(0)} + \widetilde{\xi}_1\widetilde{W}_i^{(1)}$ in each processor.

③ Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$Q_i^{(2)} = H_i^{(1)}/b_1 \text{ and } G_i^{(2)} = A_iQ^{(2)} + Q_i^{(2)}B,$$

and computes the inner product $[G_i^{(2)}, Q_i^{(2)}]$, get $a_2 = [G^{(2)}, Q^{(2)}]$ after all-reduce. Compute

$$H_i^{(2)} = G_i^{(2)} - a_2Q_i^{(2)} - b_1Q_i^{(1)} \text{ and } [H_i^{(2)}, H_i^{(2)}],$$

get $[H^{(2)}, H^{(2)}]$ after all-reduce, compute $b_2 = \sqrt{[H^{(2)}, H^{(2)}]}$.

④ Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$c_1 = \frac{a_1}{(a_1^2 + b_1^2)^{\frac{1}{2}}},\ s_1 = \frac{b_1}{(a_1^2 + b_1^2)^{\frac{1}{2}}},\ \delta_2 = b_1c_1 + a_2s_1,\ \varepsilon_3 = b_2s_1,$$
$$r_1 = (a_1^2 + b_1^2)^{\frac{1}{2}},\ \xi_1 = \|r_0\|/r_1,\ \widetilde{r}_2 = b_1s_1 - a_2c_1,\ r_2 = (\widetilde{r}_2^2 + b_2^2)^{\frac{1}{2}},\ \widetilde{\delta}_3 = -b_2c_1,$$

if $\widetilde{r}_2 < 10^{-15}$ stop, otherwise compute $\widetilde{\xi}_2 = -\delta_2\xi_1/\widetilde{r}_2$.

⑤ Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$\xi_2 = -\frac{\delta_2\xi_1}{r_2},\ W_i^{(1)} = c_1\widetilde{W}_i^{(1)} + s_1Q_i^{(2)},\ \widetilde{W}_i^{(2)} = s_1\widetilde{W}_i^{(1)} - c_1Q_i^{(2)},$$
$$Y_i^{(1)} = X_i^{(0)} + \xi_1W_i^{(1)}, X_i^{(2)} = Y_i^{(1)} + \widetilde{\xi}_2\widetilde{W}_i^{(2)}.$$

(2) Cycle process:

Step 1. Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$\|R^{(k-1)}\| = |b_{k-1}(s_{k-2}\xi_{k-2} - c_{k-2}\widetilde{\xi}_{k-1})|,$$

if $\|R^{(k-1)}\| < \varepsilon$ stop, otherwise, turn to Step 2.

Step 2. Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$Q_i^{(k)} = H_i^{(k-1)}/b_{k-1} \text{ and } G_i^{(k)} = A_i Q^{(k)} + Q_i^{(k)} B,$$

then compute $[G_i^{(k)}, Q_i^{(k)}]$, obtain $a_k = [G^{(k)}, Q^{(k)}]$, after all-reduce. Compute

$$H_i^{(k)} = G_i^{(k)} - a_k Q_i^{(k)} - b_{k-1} Q_i^{(k-1)} \text{ and } [H_i^{(k)}, H_i^{(k)}],$$

Obtain $[H^{(k)}, H^{(k)}]$ after all-reduce, compute $b_k = \sqrt{[H^{(k)}, H^{(k)}]}$.

Step 3. Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$c_{k-1} = \frac{\widetilde{r}_{k-1}}{(\widetilde{r}_{k-1}^2 + b_{k-1}^2)^{\frac{1}{2}}},\ s_{k-1} = \frac{b_{k-1}}{(\widetilde{r}_{k-1}^2 + b_{k-1}^2)^{\frac{1}{2}}},\ \varepsilon_{k+1} = b_k s_{k-1},$$
$$\delta_k = \widetilde{\delta}_k c_{k-1} + a_k s_{k-1},\ \widetilde{r}_k = \widetilde{\delta}_k s_{k-1} - a_k c_{k-1},\ r_k = (\widetilde{r}_k^2 + b_k^2)^{\frac{1}{2}},\ \widetilde{\delta}_{k+1} = -b_k c_{k-1},$$

if $\widetilde{r}_k < 10^{-15}$ stop; otherwise, compute $\widetilde{\xi}_k = (\varepsilon_k \xi_{k-2} + \delta_k \xi_{k-1})/\widetilde{r}_k$.

Step 4. Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$\xi_k = -(\varepsilon_k \xi_{k-2} + \delta_k \xi_{k-1})/r_k,\ W_i^{k-1} = c_{k-1}\widetilde{W}_i^{(k-1)} + s_{k-1} Q_i^{(k)},$$
$$\widetilde{W}_i^{(k)} = s_{k-1}\widetilde{W}_i^{k-1} - c_{k-1} Q_i(k),\ Y_i^{(k-1)} = Y_i^{(k-2)} + \xi_{k-1} W_i^{(k-1)},$$
$$X_i^{(k)} = Y_i^{(k-1)} + \widetilde{\xi}_k \widetilde{W}_i^{(k)}.$$

Step 5 Let $k := k + 1$, turn to Step 1.

3.4. Improved Parallel Implementation of the SYMMLQ Algorithm

Clearly, when computing a_k, b_k in each step of the inner iteration, all processors need to apply the reduce operator twice in the parallel implementation of the SYMMLQ algorithm in Section 3.3. Therefore, we should rearrange Step 2 in the cycle process, while the remaining steps remain the same. The detailed parallel process of the algorithm can be expressed as follows.

Processor $p_i(i = 1, 2, \cdots, p)$ computes

$$Q_i^{(k)} = (H_i^{(k-1)} - a_{k-1} Q_i^{(k-1)})/b_{k-1},\ G_i^{(k)} = A_i Q^{(k)} + Q_i^{(k)} B,$$
$$H_i^{(k)} = G_i^{(k)} - b_{k-1} Q_i^{(k-1)} \text{ and } L_i^{(k)} = H_i^{(k)} - a_{k-1} Q_i^{(k)},$$

then compute $[G_i^{(k)}, Q_i^{(k)}]$ and $[L_i^{(k)}, L_i^{(k)}]$, get $a_k = [G^{(k)}, Q^{(k)}]$ and $e_k = [L^{(k)}, L^{(k)}]$ after one all-reduce, compute $b_k = \sqrt{e_k - (a_k - a_{k-1})^2}$.

In this way, computing a_k, e_k only needs to all-reduce once, so as to reduce the frequency of communication and thus reduce the parallel time. Eventually, we obtain an improved parallel implementation of the SYMMLQ algorithm.

4. Convergence Analysis

The convergence property above two-stage iteration method is mentioned here.

Lemma 1 (see [35]). *Let H be a positive definite matrix, and $H = M - N$ be a splitting, with $M = (H + H^{\mathrm{T}})/2$, and $N = (H^{\mathrm{T}} - H)/2$. Then, $\rho(M^{-1}N) < 1$ if for all $y \in C^n$, it holds that $y^{\mathrm{H}}My > |y^{\mathrm{H}}Ny|$.*

Based on the above lemma, we obtain the following conclusion.

Theorem 1. *Assume the positive definite matrix $H \in R^{n\times n}$ is split according to Lemma 1. If for all $y \in C^n$, we have that $|Re(y^{\mathrm{H}}Hy)| > |Im(y^{\mathrm{H}}Hy)|$, then $\rho(M^{-1}N) < 1$ holds.*

Proof of Theorem 1. For all $y \in C^n$, we can get

$$y^{\mathrm{H}}My = (y^{\mathrm{H}}Hy + y^{\mathrm{H}}H^{\mathrm{H}}y)/2 = Re(y^{\mathrm{H}}Hy) \tag{12}$$

and

$$y^{\mathrm{H}}Ny = (-y^{\mathrm{H}}Hy + y^{\mathrm{H}}H^{\mathrm{H}}y)/2 = Im(y^{\mathrm{H}}Hy). \tag{13}$$

According to the Lemma 1, we can obtain $|Re(y^{\mathrm{H}}Hy)| > |Im(y^{\mathrm{H}}Hy)|$. Therefore $\rho(M^{-1}N) < 1$. □

The above theorem is generalized to the continuous Sylvester equation. The convergence theorem of the matrix equation can be obtained as follows:

Theorem 2. *Let $A, B \in R^{n\times n}$ be positive definite matrices in the Sylvester Equation (1) and suppose that they are split as in the two-stage iterative format (5). If for all $Y \in C^{n\times n}$, we have*

$$|Re[Y, AY + YB]| > |Im[Y, AY + YB]|, \tag{14}$$

then

$$\rho((M_1 \otimes I + I \otimes M_2^{\mathrm{T}})^{-1}(N_1 \otimes I + I \otimes N_2^{\mathrm{T}})) < 1 \tag{15}$$

holds.

Proof of Theorem 2. By using the Kronecker product, we can transform Equation (1) into the matrix-vector form:

$$(A \otimes I + I \otimes B^{\mathrm{T}})x = f, \tag{16}$$

where $x = \overline{vec}(X)$, $f = \overline{vec}(F)$. When the coefficient matrices A and B are positive definite matrices, we split them as in Section 2.1. Then, Equation (3) can be rewritten equivalently as

$$(M_1 \otimes I + I \otimes M_2^{\mathrm{T}})x = (N_1 \otimes I + I \otimes N_2^{\mathrm{T}})x + f. \tag{17}$$

According to Theorem 1, we yield

$$|Re(y^{\mathrm{H}}(A \otimes I + I \otimes B^{\mathrm{T}})y)| > |Im(y^{\mathrm{H}}(A \otimes I + I \otimes B^{\mathrm{T}})y)|. \tag{18}$$

On the other hand, it holds that

$$y^{\mathrm{H}}(A \otimes I + I \otimes B^{\mathrm{T}})y = tr(Y^{\mathrm{H}}(AY + YB)) = [Y, AY + YB]. \tag{19}$$

Therefore,

$$|Re[Y, AY + YB]| > |Im[Y, AY + YB]|. \tag{20}$$

□

5. Numerical Examples

In order to illustrate the performance of the two-stage iteration (TS iteration) method, some examples were performed in Matlab on an Intel dual core processor (1.00 GHz, 2 GB RAM) and the parallel machine Lenovo Shen-teng 1800 cluster. All iterations were started from a zero matrix and terminated when the current iterate satisfied $\|R^{(k)}\| < 10^{-6}$, where $R^{(k)} = F - (AX^{(k)} + X^{(k)}B)$ is the residual of the kth iteration.

Here we compare the TS iteration method with the HSS iteration method proposed in [29].

Notation:

T	the computational time in seconds
ITs	the number of iteration steps
p	the total number of processors
S	speedup ratio
E_1	parallel efficiency
Δ	error $\|R^{(k)}\|$

Example 1. *Consider the continuous Sylvester Equation (1) with $m = n$ and the matrices*

$$A = B = M + 2rN + \frac{100}{(n+1)^2} I,$$

where I is the identity matrix, and $M, N \in R^{n\times n}$ are the tridiagonal matrices given by

$$M = tridiag(-1, -2, -1) \text{ and } N = tridiag(0.5, 0, -0.5).$$

The goal in this test is to compare the iteration steps and the computational time by using TS iteration method, HSS iteration method, and MCG method with $r = 0.01$, $r = 0.1$, and $r = 1.0$. The numerical results are listed in Tables 1–3, respectively. The optimal parameters α_{exp} (with $\beta_{exp} = \alpha_{exp}$) for the HSS iteration method are given in Table 4 proposed in [29].

Table 1. Comparison of the the number of iteration steps (ITs) and the computational time in seconds (T) when choosing $r = 0.01$. HSS: Hermitian and skew-Hermitian splitting; MCG: modified conjugate gradient method; TS: two-stage iteration.

n	TS		HSS		MCG	
	ITs	T	ITs	T	ITs	T
32	4	0.030	41	0.053	237	0.036
64	20	0.070	111	0.378	943	0.356
128	130	1.411	270	5.995	4091	9.526

Table 2. Comparison of ITs and T when choosing $r = 0.1$.

n	TS		HSS		MCG	
	ITs	T	ITs	T	ITs	T
32	10	0.020	55	0.055	250	0.040
64	27	0.159	96	0.464	1006	0.399
128	149	2.593	205	7.996	3864	9.010

Table 3. Comparison of ITs and T when choosing $r = 1.0$.

n	TS		HSS		MCG	
	ITs	T	ITs	T	ITs	T
32	31	0.043	48	0.061	183	0.059
64	54	0.395	125	0.443	254	0.415
128	158	5.796	247	8.872	458	8.236

Table 4. The optimal values α_{exp} for HSS.

n	HSS		
	r = 0.01	r = 0.1	r = 1.0
32	0.40	0.40	0.95
64	0.17	0.23	0.81
128	0.09	0.13	0.62

From the above tables, we see that both iteration steps and computational time by the TS method are much less than those by the HSS and MCG in all cases. The comparison between MCG and HSS is not straight-forward. Furthermore, for the above tables we observe that in some cases the number of iteration steps of MCG is larger than that of HSS, while on the contrary, for the computational time it mainly depends on the computational time of each iteration step.

Example 2. *Consider the elliptic partial differential equation*

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \sin 2\pi x \frac{\partial u}{\partial x} + \sin 2\pi y \frac{\partial u}{\partial y} + u = 0, 0 \leq x, y \leq 1$$

with boundary condition

$$\begin{cases} u|_{x=0} = u|_{x=1} = 10 + \cos \pi y \\ u|_{y=0} = u|_{y=1} = 10 + \cos \pi x \end{cases}.$$

Let the step size be $h = 1/101$ and $h = 1/1001$. That means that the size of the linear system is 100×100 and 1000×1000, respectively. The equation is discretized by using five-point difference format, and then is transformed into a Sylvester equation. The numerical results are shown in Tables 5 and 6. This numerical experiment is performed on the parallel machine Lenovo Shen-teng 1800 cluster. Here we focus on comparing the parallel performance with the TS iteration method and the MCG method.

Table 5. Numerical results of Example 2 with $h = 1/101$.

	TS				MCG			
p	1	2	4	6	1	2	4	6
T	140.25	77.62	39.83	30.45	297.47	160.89	84.29	63.56
ITs	1067	1067	1067	1067	4992	4992	4992	4992
S		1.81	3.52	4.61		1.85	3.53	4.68
E_1	1.00	0.90	0.88	0.77	0.47	0.44	0.42	0.37
Δ	9.66×10^{-7}	9.66×10^{-7}	9.66×10^{-7}	9.14×10^{-7}	9.01×10^{-7}	9.13×10^{-7}	9.11×10^{-7}	9.80×10^{-7}

Table 6. Numerical results of Example 2 with $h = 1/1001$.

	TS				MCG			
p	20	25	30	35	20	25	30	35
T	641.12	569.88	491.28	463.74	1563.70	1374.68	1184.62	1116.93
ITs	7194	7194	7194	7194	91,332	91,332	91,332	91,336
S		22.50	26.10	27.65		22.75	26.40	28.00
E_1	1.00	0.90	0.87	0.79	0.16	0.37	0.36	0.33
Δ	9.72×10^{-7}	9.72×10^{-7}	9.72×10^{-7}	9.72×10^{-7}	9.83×10^{-7}	9.83×10^{-7}	9.93×10^{-7}	9.54×10^{-7}

From the results Tables 5 and 6, we observe that both iteration steps and computational time using TS are much less than those with the MCG. Furthermore, parallel efficiency with the TS method is higher than with the MCG. In addition, the advantages of the TS method increase over the MCG method with increasing scale-size of equations from 10^4 ($n = 100$) to 10^6 ($n = 1000$).

Example 3. *Let* $A = (a_{ij})_{n\times n}$ *and* $B = (b_{ij})_{n\times n}$ *where*

$$a_{ij} = \begin{cases} -n + \sin(i+j), & i = j \\ i, & others \end{cases}, \quad b_{ij} = \begin{cases} 3 + 2\sin(i+j), & i = j \\ \cos(i+j), & j - i = 1. \\ \sin(i+j), & i - j = 1 \end{cases}$$

In the Sylvester matrix equation $AX + XB = F$, let $n = 1000$ and F is an any given matrix. The numerical results are listed in Table 7.

Table 7. Numerical results of Example 3.

	TS			
p	16	20	24	28
T	2480.19	2351.59	2296.47	2241.46
ITs	509	509	509	509
S		16.87	17.28	17.92
E_1	1.00	0.84	0.72	0.64
Δ	8.83×10^{-7}	8.83×10^{-7}	8.83×10^{-7}	8.83×10^{-7}

From Table 7 we observe that the two-stage iteration method is still efficient in the case that the coefficient matrices are indefinite matrices. This indicates that the condition for the convergence is only a sufficient condition in Theorem 1.

6. Conclusions

In this paper we have proposed a two-stage iteration parallel method for solving the continuous Sylvester equations. The outer iteration scheme is based on the coefficient matrices' splitting. Furthermore, an inner iteration scheme is obtained using the SYMMLQ algorithm. Its parallel implementation and its improved strategy have been explained in detail. Moreover, we have proved the convergence of the proposed iteration method under certain conditions. Numerical results show that the new proposed algorithm is better than both MCG and HSS methods with regard to computational storage, computational time, and iteration steps. Crucially, these advantages become more significant for large-scale systems of continuous Sylvester equations.

Acknowledgments: This work has been supported by the National Natural Science Foundation of China (Grants No. 11302173)and the Natural Science Foundation of Shaanxi Province (Grants No. 2017JQ1037). The computation of examples was executed in the Center for High Performance Computing of Northwestern Polytechnical University.

Author Contributions: Manyu Xiao, Quanyi Lv and Zhuo Xing conceived the key ideas, designed the algorithm, performed the examples and drafted the initial manuscript; Manyu Xiao, Quanyi Lv and Yingchun Zhang revised the final manuscript.

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

References

1. Datta, B.N. *Numerical Methods for Linear Control Systems*; Elsevier Academic Press: San Diego, CA, USA, 2004.
2. Schaft, A.J.V.D. *L_2-Gain and Passivity Techniques in Nonlinear Control*, 2nd ed.; Springer-Verlag: London, UK, 2000.
3. Baur, U.; Benner, P. Cross-gramian based model reduction for data-sparse systems. *Electron. Trans. Numer. Anal.* **2008**, *31*, 256–270.
4. Calvetti, D.; Reichel, L. Application of ADI iterative methods to the restoration of noisy images. *SIAM J. Matrix Anal. Appl.* **1996**, *17*, 165–186.
5. Wang, X.; Fan, X. Two efficient derivative-free iterative methods for solving nonlinear systems. *Algorithms* **2016**, *9*, 14.
6. Allahviranloo, T.; Ahmady, E.; Ahmady, N.; Shams, A.K. Block Jacobi two-stage method with Gauss-Sidel inner iterations for fuzzy system of linear equations. *Appl. Math. Comput.* **2006**, *175*, 1217–1228.
7. Wu, J.P.; Wang, Z.H.; Li, X.M. *The Efficient Solution and Parallel Algorithm of Spare Linear Equations*; Science and Technology Press of Hunan: Changsha, China, 2004, pp. 269–273.
8. Paige, C.C.; Saunders, M.A. Solution of sparse indefinite system of linear equations. *SIAM J. Number Anal.* **1975**, *12*, 617–629.
9. Jiang, E.X. *Matrix Calculations*; Science Press: Beijing, China, 2008.
10. Bartels, R.H.; Stewart, G.W. Solution of the matrix equation $AX + XB = C$: [F4]. *Commun. ACM* **1972**, *15*, 820–826.
11. Golub, G.H.; Nash, S.; Loan, C.V. A Hessenberg-Schur method for the problem $AX + XB = C$. *IEEE Trans. Automat. Control.* **1979**, *24*, 909–913.
12. Smith, R.A. Matrix equation $XA + BX = C$. *SIAM J. Appl. Math.* **1968**, *16*, 198–201.
13. Levenberg, M.; Reichel, L. A generalized ADI iterative method. *Numer. Math.* **1993**, *66*, 215–233.
14. Niu, Q.; Wang, X.; Lu, L.Z. A relaxed gradient based algorithm for solving Sylvester equations. *Asian J. Control.* **2011**, *13*, 461–464.
15. Wang, X.; Dai, L.; Liao, D. A modified gradient based algorithm for solving Sylvester equations. *Appl. Math. Comput.* **2012**, *218*, 5620–5628.
16. Bao, L.; Lin, Y.Q.; Wei, Y.M. A new projection method for solving large Sylvester equations. *Appl. Numer. Math.* **2007**, *57*, 521–532.
17. Druskin, V.; Knizhnerman, L.; Simoncini, V. Analysis of the rational Krylov subspace and ADI methods for solving the Lyapunov equation. *SIAM J. Numer.* **2011**, *49*, 1857–1898.
18. Evans, D.J.; Galligani, E. A parallel additive preconditioner for conjugate gradient method for $AX + XB = C$. *Parallel Comput.* **1994**, *20*, 1055–1064.
19. Kelley, C.T. *Iterative Methods for Linear and Nonlinear Equations*; Society For Industrial And Applied Mathematics: Philadelphia, PA, USA, 1995.
20. Herisanu, N.; Marinca, V. An iteration procedure with application to van der pol oscillator. *Int. J. Nonlinear Sci. Numer. Simul.* **2009**, *10*, 353–362.
21. Cheng, Y.P.; Zhang, K.Y.; Xu, Z. *Matrix Theory*, 3rd ed.; Northwestern Polytechnical University Press: Xi'an, China, 2006, pp. 215–216.
22. Hou, J.X.; Lv, Q.Y.; Man, Y.X. A Parallel Preconditioned Modified Conjugate Gradient Method for Large Sylvester Matrix Equation. *Math. Prob. Eng.* **2014**, *2014*, 1–7.
23. Chen, G.L.; Hong, A.; Chen, J.; Zheng, Q.L.; Shan, J.L. *The Parallel Algorithm*; Higher Education Press: Beijing, China, 2004.
24. Bai, Z.Z.; Golub, G.H.; Ng, M.K. Hermitian and Skew-Hermitian splitting methods for non-Hermitian positive definite linear systems. *SIAM J. Matrix Anal. Appl.* **2003**, *24*, 603–626.
25. Bai, Z.Z. Optimal parameters in the HSS-like methods for saddle-point problems. *Numer. Linear Algebra Appl.* **2009**, *16*, 447–479.

26. Bai, Z.Z.; Golub, G.H.; Ng, M.K. On inexact Hermitian and skew-Hermitian splitting methods for non-Hermitian positive definite linear systems. *Numer. Linear Algebra Appl.* **2008**, *428*, 413–440.
27. Huang, Y.M. A practical formula for computing optimal parameters in the HSS iteration methods. *J. Comput. Math.* **2014**, *255*, 142–149.
28. Wang, X.; Li, Y.; Dai, L. On Hermitian and skew-Hermitian splitting iteration methods for the linear matrix equation $AXB = C$. *Comput. Math. Appl.* **2013**, *65*, 657–664.
29. Bai, Z.Z. On Hermitian and skew-Hermitian splitting iteration methods for continuous Sylvester equations. *J. Comput. Math.* **2011**, *29*, 185–198.
30. Wang, X.; Li, W.W.; Mao, L.Z. On positive-definite and skew-Hermitian splitting iteration methods for continuous Sylvester equation $AX + XB = C$. *Comput. Math. Appl.* **2013**, *66*, 2352–2361.
31. Zhou, D.M.; Chen, G.L.; Cai, Q.Y. On modified HSS iteration methods for continuous Sylvester equations. *Appl. Math. Comput.* **2015**, *263*, 84–93.
32. Zheng, Q.Q.; Ma, C.F. On normal and skew-Hermitian splitting iteration methods for large sparse continuous Sylvester equations. *J. Comput. Appl. Math.* **2014**, *268*, 145–154.
33. Golub, G.H.; Loan, C.F.V. *Matrix Computations*, 3rd ed.; The Johns Hopkins University Press: Baltimore, MD, USA; London, UK, 1996.
34. Li, X.M.; Wu, J.P. *Numerical Parallel Algorithm and Software*; Science Press: Beijing, China, 2007.
35. Wang, C.L.; Bai, Z.Z. Sufficient conditions for the convergent splittings of non-Hermitian positive definite matrices. *Linear Algebra Appl.* **2001**, *330*, 215–218.