On ADI Method for Sylvester Equations

Ninoslav Truhar, Ren-Cang Li
On ADI Method for Sylvester Equations

Ninoslav Truhar * Ren-Cang Li †

February 8, 2007

Abstract

This paper is concerned with numerical solutions of large scale Sylvester equations $AX - XB = C$, Lyapunov equations as a special case in particular included, with $C$ having very small rank. For stable Lyapunov equations, Penzl (2000) and Li and White (2002) demonstrated that the so called Cholesky factored ADI method with decent shift parameters can be very effective. In this paper we present a generalization of Cholesky factored ADI for Sylvester equations. We also demonstrate that often much more accurate solutions than ADI solutions can be gotten by performing Galerkin projection via the column space and row space of the computed approximate solutions.

1 Introduction

An $m \times n$ Sylvester equation takes the form

$$AX - XB = C,$$  \hspace{1cm} (1.1)

where $A$, $B$, and $C$ are $m \times m$, $n \times n$, and $m \times n$, respectively, and unknown matrix $X$ is $m \times n$. A Lyapunov equation is a special case with $m = n$, $B = -A^*$, and $C = C^*$, where the star superscript takes complex conjugate and transpose. Equation (1.1) has a unique solution if $A$ and $B$ have no common eigenvalues, which will be assumed throughout this paper.

Sylvester equations appear frequently in many areas of applied mathematics, both theoretically and practically. We refer the reader to the elegant survey by Bhatia and Rosenthal [6] and references therein for a history of the equation and many interesting and important theoretical results. Sylvester equations play vital roles in a number of applications such as matrix eigen-decompositions [13], control theory and model reduction [2, 32], numerical solutions to matrix differential Riccati equations [9, 10], and image processing [7].

*Department of Mathematics, J.J. Strossmayer University of Osijek, Trg Ljudevita Gaja 6, 31000 Osijek, Croatia. Email: ntruhar@mathos.hr. Supported in part by the National Science Foundation under Grant No. 235-2352818-1042. Part of this work was done while this author was a visiting professor at the Department of Mathematics, University of Texas at Arlington, Arlington, TX, USA.

†Department of Mathematics, University of Texas at Arlington, P.O. Box 19408, Arlington, TX 76019-0408, USA. Email: rcli@uta.edu. Supported in part by the National Science Foundation under Grant No. DMS-0510664 and DMS-0702335.
This paper is concerned with numerical solutions of Sylvester equations, Lyapunov equations as a special case included. There are several numerical algorithms for that purpose. The standard ones are due to Bartels and Stewart [4], and Golub, Nash, and van Loan [12]. Both are efficient for dense matrices $A$ and $B$. However, recent interests are directed more towards large and sparse matrices $A$ and $B$, and $C = GF^*$ with very low rank, where $G$ and $F$ have only a few columns. In these cases, the standard methods are often too expensive to be practical, and some iterative solutions become more viable choices. Common ones are Krylov subspace based Galerkin algorithms [16, 17, 18, 19, 25, 28] and Alternating-Direction-Implicit (ADI) iterations [15, 20, 22, 24, 30]. Advantages of Krylov subspace based Galerkin algorithms over ADI iterations are that no knowledge about the spectral information on $A$ and $B$ is needed and no linear systems of equations with shifted $A$ and $B$ have to be solved. But ADI iterations often enable faster convergence if (sub)optimal shifts to $A$ and $B$ can be effectively estimated. So for a problem for which linear systems with shifted $A$ and $B$ can be solved at modest cost, ADI iterations may turn out to be better alternatives. This is often true for stable Lyapunov equations (all of $A$’s eigenvalues have negative real parts) from control theory [15, 20, 22].

In this paper, we shall first extend the Cholesky factored ADI for Lyapunov equations to solve Sylvester equations. Then, we argue that often much more accurate solutions than the ADI solutions can be gotten by performing Galerkin projection via the row and column subspaces of the computed solutions. The improvement is often more dramatic with not-so-good shifts. Indeed, except for stable Lyapunov equations with Hermitian $A$ [11, 31, 21] and for Sylvester equations with Hermitian $A$ and $B$ [26], currently there is no provably ways to select good shifts, and existing practices are more heuristic than rigorously justifiable.

The rest of this paper is organized as follows. Section 2 reviews ADI and derives factored ADI iterations for Sylvester equations. An extension of Penzl’s shift strategy to Sylvester equations is explained in Section 3. Projection ADI subspace methods via Galerkin projection or the minimal residual condition are presented in Section 4. Section 5 explains connection between the new algorithm and CF-ADI for Lyapunov equations. We report several numerical tests in Section 6 and finally present our conclusions in Section 7.

**Notation.** Throughout this paper, $\mathbb{C}^{n\times m}$ is the set of all $n \times m$ complex matrices, $\mathbb{C}^n = \mathbb{C}^{n\times 1}$, and $\mathbb{C} = \mathbb{C}^{1\times 1}$. Similarly define $\mathbb{R}^{n\times m}$, $\mathbb{R}^n$, and $\mathbb{R}$ except replacing the word complex by real. $I_n$ (or simply $I$ if its dimension is clear from the context) is the $n \times n$ identity matrix, and $e_j$ is its $j$th column. The superscript “*$” takes conjugate transpose while “.T” takes transpose only. For scalars, $\bar{\alpha}$ is the complex conjugate of $\alpha$, and $\Re(\alpha)$ takes the real part of $\alpha$. We shall also adopt MATLAB-like convention to access the entries of vectors and matrices. $i : j$ is the set of integers from $i$ to $j$ inclusive and $i : i = \{i\}$. For a vector $u$ and a matrix $X$, $u_{(j)}$ is $u$'s $j$th entry, $X_{(i,j)}$ is $X$’s $(i,j)$th entry; $X$’s submatrices $X_{(k:\ell,i:j)}$, $X_{(k:\ell,i)}$, and $X_{(:,i:j)}$ consist of intersections of row $k$ to row $\ell$ and column $i$ to column $j$, row $k$ to row $\ell$, and column $i$ to column $j$, respectively.
2 ADI for Sylvester Equation

Given two sets of parameters \( \{\alpha_i\} \) and \( \{\beta_j\} \), Alternating-Directional-Implicit (ADI) iteration for iteratively solving (1.1) goes as follows \(^{[30]}\): For \( i = 0, 1, \ldots, \)

1. solve \((A - \beta_i I)X_{i+1/2} = X_i(B - \beta_i I) + C\) for \( X_{i+1/2} \);
2. solve \(X_{i+1}(B - \alpha_i I) = (A - \alpha_i I)X_{i+1/2} - C\) for \( X_{i+1} \),

where initially \( X_0 \) is given and in this paper it is the zero matrix. A rather straightforward implementation for ADI can be given based on this.

Expressing \( X_{i+1} \) in terms of \( X_i \), we have\(^{[2]}\)

\[
X_{i+1} = (\beta_i - \alpha_i)(A - \beta_i I)^{-1}C(B - \alpha_i I)^{-1} + (A - \alpha_i I)(A - \beta_i I)^{-1}X_i(B - \beta_i I)(B - \alpha_i I)^{-1},
\]

and the error equation

\[
X_{i+1} - X = (A - \alpha_i I)(A - \beta_i I)^{-1}(X_i - X)(B - \beta_i I)(B - \alpha_i I)^{-1}
\]

\[
= \left[ \prod_{j=0}^{i}(A - \alpha_j I)(A - \beta_j I)^{-1} \right] (X_0 - X) \left[ \prod_{j=0}^{i}(B - \beta_j I)(B - \alpha_j I)^{-1} \right],
\]

where \( X \) denotes the exact solution. Equation (2.1) leads naturally to another implementation for ADI.

Equation (2.3) implies that if \( \{\alpha_j\}_{j=0}^{i} \) contains all of \( A \)'s eigenvalues (multiple eigenvalues counted as many times as their algebraic multiplicities) or if \( \{\beta_j\}_{j=0}^{i} \) contains all of \( B \)'s eigenvalues, then \( X_{i+1} - X \equiv 0 \). This is because, by the Cayley-Hamilton theorem, \( p(A) \equiv 0 \) for \( A \)'s characteristic polynomial \( p(\lambda) \) defined as \( \det(\lambda I - A) \) and \( q(B) \equiv 0 \) for \( q(\lambda) \) defined as \( \det(\lambda I - B) \).

Error equation (2.3) in a way also sheds light on parameter choosing, namely the parameters \( \alpha_j \) and \( \beta_j \) should be chosen to achieve

\[
\min_{\alpha_j, \beta_j} \left\| \prod_{j=0}^{i}(A - \alpha_j I)(A - \beta_j I)^{-1} \right\| \cdot \left\| \prod_{j=0}^{i}(B - \beta_j I)(B - \alpha_j I)^{-1} \right\|,
\]

where \( \| \cdot \| \) is usually taken to be the spectral norm, but any other matrix norm would serve the purpose just fine. This in general is very hard to do and has not been solved and

\(^{[1]}\)Wachspress referred (1.1) a Lyapunov equation, while traditionally it is called a Sylvester equation and a Lyapunov equation refers to the one with \( B = -A^* \) and \( C \) being Hermitian. We shall adopt the traditional nomenclature.

\(^{[2]}\)It can be also gotten from the identity

\[
(A - \beta I)X(B - \alpha I) - (A - \alpha I)X(B - \beta I) = (\beta - \alpha)C.
\]
unlikely will be solved any time soon. But for stable Lyapunov equations: \( A \) is Hermitian and its eigenvalues have negative real parts and \( B = -A^* \), as well as for Sylvester equations when \( A \) and \( B \) are Hermitian and their eigenvalues are located in two separate intervals, Problem (2.4) is solved in [31] (see also [20] for the Lyapunov equation case) with the help of the solution by Zolotarev in 1877 for a best rational approximation problems of his own [1, 31]. It has been just heuristic when it comes to the shifts selection for the general cases.

Suppose now that \( C = GF^* \) is in the factored form, where \( G \) is \( m \times r \) and \( F \) is \( n \times r \), respectively. If \( r \ll \min\{m, n\} \) and if convergence occurs much earlier in the sense that it takes much fewer than \( \min\{m, n\}/r \) steps, then ADI in the factored form as below would be a more economic way to do. Let \( X_i = Z_i D_i Y_i^* \). We have

\[
X_{i+1} = \left( (A - \beta_i I)^{-1} G \ (A - \alpha_i I) (A - \beta_i I)^{-1} Z_i \right) \times \\
\left( (\beta_i - \alpha_i) I \\ D_i \right) \times \\
\left( F^* (B - \alpha_i I)^{-1} \\ Y_i^* (B - \beta_i I) (B - \alpha_i I)^{-1} \right) = Z_{i+1} D_{i+1} Y_{i+1}^*,
\]

(2.5)

where

\[
Z_{i+1} = \left( (A - \beta_i I)^{-1} G \ Z_i + (\beta_i - \alpha_i) (A - \beta_i I)^{-1} Z_i \right),
\]

(2.6)

\[
D_{i+1} = \left( (\beta_i - \alpha_i) I \\ D_i \right),
\]

(2.7)

\[
Y_{i+1}^* = \left( Y_i^* + (\alpha_i - \beta_i) Y_i^* (B - \alpha_i I)^{-1} \right).
\]

(2.8)

Equations (2.5) – (2.8) yield the third implementation for ADI but in the factored form. In this implementation, every column of \( Z_i \) and every column of \( Y_i \) evolve with the iteration. We collectively call those ADI that compute the solution of a Sylvester equation (Lyapunov equation included) in the factored form \textit{Factored Alternating-Directional-Implicit iterations} or \textit{fADI} for short.

Suppose that \( Z_0 = Y_0 = 0 \). Then both \( X_i \) and \( Y_i \) have \( ir \) columns and thus the rank of \( X_i \) is no more than \( ir \). Therefore, a necessary condition for ADI iteration in this factored form to be numerically more efficient than the previous two versions is that the exact solution \( X \) must be numerically of low-rank, i.e., it has much fewer than \( \min\{m, n\} \) dominant singular values and the rest singular values are tiny relative to these dominant ones. This is provably true by Penzl [23] for stable Lyapunov equations with Hermitian \( A \). But Penzl’s argument can be combined with Zolotarev’s solution to Problem (2.4) to yield an optimal \( X \)’s eigenvalue decay bound among all \( A \) with given spectral condition number. This is done in [26]. Also in [26], one finds \( X \)’s singular value decay bound for Sylvester equations with Hermitian \( A \) and \( B \) whose eigenvalues are located in two separate intervals. The case for diagonalizable \( A \) was investigated by Antoulas, Sorensen, and Zhou [3], Truhar and Veselić [29] (in addition [29] contains the bound for \( X \)’s low rankness for \( A \) with a special Jordan structure), and in general by Grasedyck [14].
ADI’s convergence is crucially dependent on the selected parameters \(\{\alpha_i\}\) and \(\{\beta_i\}\). It is not a trivial matter to select “good” parameters to steer \(X_i\) towards the exact solution. This is especially true for the general case. Existing practices are all heuristically. Later in Section 3, we’ll present an extension of Penzl’s shift strategy.

Iterate (2.6) one more time to get the column blocks of \(Z_{i+1}\) as

\[
\begin{align*}
(A - \beta_i I)^{-1}G, \\
(A - \alpha_i I)(A - \beta_i I)^{-1}(A - \beta_{i-1} I)^{-1}G, \\
(A - \alpha_i I)(A - \beta_i I)^{-1}(A - \alpha_{i-1} I)(A - \beta_{i-1} I)^{-1}Z_{i-1}.
\end{align*}
\]

Keep on iterating to get the column blocks of \(Z_{i+1}\), after reorder the factors due to commutativity, as

\[
\begin{align*}
(A - \beta_i I)^{-1}G, \\
(A - \alpha_i I)(A - \beta_i I)^{-1}(A - \beta_{i-1} I)^{-1}G, \\
(A - \alpha_{i-1} I)(A - \beta_{i-2} I)^{-1}(A - \alpha_i I)(A - \beta_{i-1} I)^{-1}(A - \beta_i I)^{-1}G, \\
\ldots
\end{align*}
\]

Similarly we have the row blocks of \(Y_{i+1}^*\) as

\[
\begin{align*}
F^*(B - \alpha_i I)^{-1}, \\
F^*(B - \alpha_i I)^{-1}(B - \alpha_{i-1} I)^{-1}(B - \beta_i I), \\
F^*(B - \alpha_i I)^{-1}(B - \alpha_{i-1} I)^{-1}(B - \beta_i I)(B - \alpha_{i-2} I)^{-1}(B - \beta_{i-1} I), \\
\ldots
\end{align*}
\]

At the same time, the diagonal blocks of \(D_{i+1}\) are

\[
(\beta_i - \alpha_i)I, (\beta_{i-1} - \alpha_{i-1})I, \ldots.
\]

To proceed, we re-name the parameters \(\{\alpha_i\}\) and \(\{\beta_i\}\) so that the lists in (2.9) and (2.10) read as

\[
\begin{align*}
(A - \beta_1 I)^{-1}G, \\
(A - \alpha_1 I)(A - \beta_2 I)^{-1}(A - \beta_1 I)^{-1}G, \\
(A - \alpha_2 I)(A - \beta_3 I)^{-1}(A - \alpha_1 I)(A - \beta_2 I)^{-1}(A - \beta_1 I)^{-1}G, \\
\ldots
\end{align*}
\]

and

\[
\begin{align*}
F^*(B - \alpha_1 I)^{-1}, \\
F^*(B - \alpha_1 I)^{-1}(B - \alpha_2 I)^{-1}(B - \beta_1 I), \\
F^*(B - \alpha_1 I)^{-1}(B - \alpha_2 I)^{-1}(B - \beta_1 I)(B - \alpha_3 I)^{-1}(B - \beta_2 I), \\
\ldots
\end{align*}
\]

respectively. Finally we can write

\[
Z_k = \begin{pmatrix} Z^{(1)} & Z^{(2)} & \cdots & Z^{(k)} \end{pmatrix},
\]

with

\[
\begin{align*}
Z^{(1)} &= (A - \beta_1 I)^{-1}G, \\
Z^{(i+1)} &= (A - \alpha_i I)(A - \beta_{i+1} I)^{-1}Z^{(i)}, \\
&= Z^{(i)} + (\beta_{i+1} - \alpha_i)(A - \beta_{i+1} I)^{-1}Z^{(i)},
\end{align*}
\]

(2.13)
and
\[ Y_k = \begin{pmatrix} Y^{(1)} & Y^{(2)} & \cdots & Y^{(k)} \end{pmatrix}, \]
with
\[
\begin{aligned}
Y^{(i)*} & = F^*(B - \alpha_i I)^{-1}, \\
Y^{(i+1)*} & = Y^{(i)*}(B - \alpha_{i+1} I)^{-1}(B - \beta_i I) \\
& = Y^{(i)*} + (\alpha_{i+1} - \beta_i)Y^{(i)*}(B - \alpha_{i+1} I)^{-1},
\end{aligned}
\]  
(2.14)
and
\[ X_k = Z_k D_k Y_k^*, \quad D_k = \text{diag}((\beta_1 - \alpha_1)I_r, \ldots, (\beta_k - \alpha_k)I_r). \]  
(2.15)

Formulas (2.13) – (2.15) yields a new fADI which is a natural extension of CF-ADI [20] and LR-ADI [22, 24] for stable Lyapunov equations.

It is obvious that formulas (2.13) – (2.15) offers two significant advantages over the one by (2.5) – (2.8) for computing \( X_k \):

1. Since \( Z_i \) has many times more columns than \( Z^{(i)} \) and the same is true for \( Y_i \) and \( Y^{(i)} \), each step by (2.5) – (2.8) solves many times more linear systems than by (2.13) – (2.15). Specifically, \( k \) step fADI by (2.5) – (2.8) solves
\[ 2[r + 2r + \cdots + kr] = k(k + 1)r \]
linear systems, while \( k \) step fADI by (2.13) – (2.15) solve \( 2kr \) linear systems. Assume that it costs \( \tau \) unit to solve a shifted linear system \( (A - \beta I)x = b \) or \( (B - \alpha I)y = c \) the first time and subsequently it costs 1 unit for a linear system with the same \( A - \beta I \) or \( B - \alpha I \). It is reasonable to expect that the linear system solving part dominates the rest as far as computational expense is concerned. So \( k \) step fADI based on (2.5) – (2.8) costs
\[ 2 \sum_{i=1}^{k} (\tau + ir - 1) = 2k(\tau - 1) + k(k + 1)r, \]
while ADI based on (2.13) – (2.15) costs
\[ 2k(\tau + r - 1) = 2k(\tau - 1) + 2kr. \]
Conceivably \( \tau \geq 1 \); but for \( A \) and \( B \) with structures such as banded, Toeplitz, Hankel, or very sparse as often the case in practice, \( \tau \approx 1 \). When that happens, fADI based on (2.13) – (2.15) can be up to \( (k + 1)/2 \) times faster.

2. Each step by (2.13) – (2.15) computes a column block of \( Z_i \) and a row block of \( Y_i \) for all \( i \geq k \), while by (2.5) – (2.8), all column blocks of \( Z_i \) and all row blocks of \( Y_i \) change as \( i \) does. This is especially important as far as numerical efficiency is concerned for any of our ADI subspace methods to be proposed because orthonormal bases can be expanded as soon as a column or row block is computed if formulas (2.13) – (2.15) are used.
Algorithm 2.1 below summarizes the fADI by (2.13) – (2.15). Naturally it applies to Lyapunov equations, too, yielding ADIs for Lyapunov equation $AX + AX^* + GG^* = 0$ which gives the CF-ADI [20] and LR-ADI [22, 24] in a slightly different form. Detail for such a specialization will be discussed separately in Section 5 because of the particularly important role of Lyapunov equations in practice.

Algorithm 2.1 fADI for Sylvester equation $AX - XB = GF^*$:

Input:  
(a) $A_{(m \times m)}$, $B_{(n \times n)}$, $G_{(m \times r)}$, and $F_{(n \times r)}$;  
(b) ADI shifts $\{\beta_1, \beta_2, \ldots\}$, $\{\alpha_1, \alpha_2, \ldots\}$;  
(c) $k$, the number of ADI steps;  
Output: $Z_{(m \times kr)}$, $D_{(kr \times kr)}$, and $Y_{(n \times kr)}$ such that $ZDY^*$ approximately solves Sylvester equation $AX - XB = GF^*$;  
1. $Z_{(:,1:r)} = (A - \beta_1 I)^{-1}G$; $(Y^*)_{(1:r,:)} = F^*(B - \alpha_1 I)^{-1}$;  
2. for $i = 1, 2, \ldots, k$ do  
3. $Z_{(:,i:r+1:(i+1)r)} = Z_{(:,(i-1)r+1:ir)} + (\beta_{i+1} - \alpha_i)(A - \beta_i I)^{-1}Z_{(:,(i-1)r+1:ir)}$;  
4. $(Y^*)_{(ir+1:(i+1)r,:)} = (Y^*)_{((i-1)r+1:ir,:)} + (\alpha_{i+1} - \beta_i)(Y^*)_{((i-1)r+1:ir,:)} (B - \alpha_{i+1} I)^{-1}$;  
5. end for;  
6. $D = \text{diag}((\beta_1 - \alpha_1)I_r, \ldots, (\beta_k - \alpha_k)I_r)$.  

3 A shift strategy

ADI shifts determine the speed of the convergence of the method. There are a number of strategies out there, and most of them are based on heuristic arguments, except in the Hermitian cases. In his thesis, Sabino [26] presented a quite complete review of the existing strategies. Since this paper, however, is not about looking for yet another shift strategy, for testing purpose we shall simply discuss an easily implementable extension of Penzl’s [22, 24] who did it for Lyapunov equations.

When $A$ and $B$ are Hermitian (this is in fact true for normal $A$ and $B$), (2.4) is equivalent to the following ADI minimax problem for $k$ ADI steps with $\mathbb{E} = \text{eig}(A)$ and $\mathbb{F} = \text{eig}(B)$.

\[
\text{Find } \alpha_j \text{ and } \beta_j, \ j = 1, \ldots, k, \text{ such that } \min_{\alpha_j \in \mathbb{C}} \max_{\beta_j \in \mathbb{C}} \prod_{j=1}^{k} \frac{|(x - \alpha_j)(y - \beta_j)|}{|x - \beta_j(y - \alpha_j)|}. \tag{3.1}
\]

In practice since $\text{eig}(A)$ and $\text{eig}(B)$ are not known a priori, $\mathbb{E}$ and $\mathbb{F}$ are often replaced by intervals that contain the eigenvalues of $A$ and $B$, respectively. In the case for Lyapunov equations, $B = -A^*$, $\beta_j = -\bar{\alpha}_j$ (the complex conjugate of $\alpha_j$), and $\mathbb{F} = -\mathbb{E}$, Problem
(3.1) reduces to

\[
\text{Find } \alpha_j, j = 1, \ldots, k, \text{ such that } \min_{\alpha_i \in C} \max_{x \in E} \prod_{j=1}^{k} \frac{|x - \alpha_j|}{|x + \alpha_j|}.
\]  

(3.2)

Regardless of whether \( A \) is Hermitian or not, for stable Lyapunov equations Penzl [24] proposed a heuristic shift-selection strategy by solving a much simplified (3.2): Find \( \alpha_j, j = 1, \ldots, k \), such that

\[
\min_{\alpha_i \in E} \max_{x \in F} \prod_{j=1}^{k} \frac{|x - \alpha_j|}{|x + \beta_j|}.
\]  

(3.3)

with \( E \) set to be a collection of certain estimates of the extreme eigenvalues of \( A \). The strategy usually works very well. In obtaining \( E \), Penzl proposed to run a pair of Arnoldi processes. The first process delivers \( k_+ \) Ritz values that tend to approximate well “outer” eigenvalues, which are generally not close to the origin. The second process is used to get \( k_- \) Ritz values to approximate those eigenvalues near the origin. His algorithm then chooses a set of shift parameters out of \( E \) by solving (3.3). The shifts delivered by the heuristic are ordered in such a way that shifts, which should reduce the ADI error most, are applied first.

Penzl’s strategy can be naturally extended to the case for Sylvester equations. Now we need to compute two sets \( \{\alpha_1, \ldots, \alpha_k\} \) and \( \{\beta_1, \ldots, \beta_k\} \) of presumed good shift parameters. We start by generating two discrete sets \( E \) and \( F \) which “well” approximates parts of the spectra of \( A \) and \( B \), respectively, and then solve a much simplified (3.1): Find \( \alpha_j \) and \( \beta_j \), \( j = 1, \ldots, k \), such that

\[
\min_{\alpha_i \in E} \max_{\beta_j \in F} \prod_{j=1}^{k} \frac{|x - \alpha_j(y - \beta_j)|}{|x - \beta_j(y - \alpha_j)|}.
\]  

(3.4)

Again the selected shifts are ordered in such a way that shifts, which should reduce the ADI error most, are applied first. This is summarized in Algorithm 3.1.

**Algorithm 3.1** ADI parameters by Ritz values (ADIpR).

**Input:** \( A, F, B, G, k \);

**Output:** ADI parameters \( \{\alpha_1, \ldots, \alpha_k\} \) and \( \{\beta_1, \ldots, \beta_k\} \);

1. Run Arnoldi process with \( A \) on \( G \) to give the set \( E_A^+ \) of Ritz values;
2. Run Arnoldi process with \( A^{-1} \) on \( G \) to give the set \( E_A^- \) of Ritz values;
3. \( E = E_A^+ \cup (1/E_A^-) \);
4. Run Arnoldi process with \( B^* \) on \( F \) to give the set \( F_B^+ \) of Ritz values;
5. Run Arnoldi process with \( B^{-*} \) on \( F \) to give the set \( F_B^- \) of Ritz values;
6. \( F = \text{conj}(F_B^+) \cup \text{conj}(1/F_B^-) \);
We now offer a few suggestions as to how this algorithm can be efficiently implemented.

1. We suggest to run the Arnoldi processes in Steps 1, 2, 4, and 5 a few more steps than the number of Ritz values sought in each run and pick up those Ritz values with the smallest residual errors. A reasonable choice would be to run each Arnoldi process \( \min \{2k, k + 10\} \) steps and pick out \( k \) best Ritz values. Doing so makes both \( \mathbb{E} \) and \( \mathbb{F} \) with \( 2k \) values. More values in \( \mathbb{E} \) and \( \mathbb{F} \) increase computational work in Steps 7 – 10. Conceivably one could pick any number between \( k/2 \) and \( k \) of best Ritz values in each run to reduce the work.

2. Step 7 can be realized as follows. Suppose that \( \mathbb{E} \) and \( \mathbb{F} \) have \( k_a \) and \( k_b \) values, respectively, and of course \( k_a, k_b > k \). Let \( W \) be the \( k_a \)-by-\( k_b \) matrix whose \((p, q)\)th entry is

\[
\max_{x \in \mathbb{E}} \frac{(x - \alpha)(y - \beta)}{(x - \beta)(y - \alpha)} = \max_{x \in \mathbb{E}} \frac{x - \alpha}{x - \beta} \cdot \max_{y \in \mathbb{F}} \frac{y - \beta}{y - \alpha}
\]

for the \( p \)th value \( \alpha \in \mathbb{E} \) and the \( q \)th value \( \beta \in \mathbb{F} \). Then we find the smallest entry of \( W \), and if the smallest entry is \( W \)'s \((p_0, q_0)\)th entry, then \( \alpha_1 \) is the \( p_0 \)th value in \( \mathbb{E} \) and \( \beta_1 \) is the \( q_0 \)th value in \( \mathbb{F} \). If \( p_0 \neq k_a \), i.e., \( \alpha_1 \) is not the last value in \( \mathbb{E} \), swap the \( p_0 \)th and \( k_a \)th value of \( \mathbb{E} \). Likewise if \( q_0 \neq k_b \), swap the \( q_0 \)th and \( k_b \)th value of \( \mathbb{F} \). Define column vectors \( r_a \) and \( r_b \) whose \( p \)th entry and \( q \)th entry are

\[
\begin{bmatrix}
  x - \alpha_1 \\
  x - \beta_1 \\
\end{bmatrix}, \quad \begin{bmatrix}
  y - \alpha_1 \\
  y - \beta_1 \\
\end{bmatrix}
\]

respectively, where \( x \) is the \( p \)th value of \( \mathbb{E} \) and \( y \) is the \( q \)th value of \( \mathbb{F} \). The cost in this step is \( O(k_a k_b (k_a + k_b)) \) since each entry \((3.5)\) of \( W \) costs \( O(k_a + k_b) \) to compute.

3. Suppose \( \alpha_j \) and \( \beta_j \) for \( j \leq i - 1 \) are already selected. We need to pick out \( \alpha_i \) and \( \beta_i \). Note at this point we have column vectors \( r_a \) and \( r_b \) whose \( p \)th entry and \( q \)th entry are

\[
\prod_{j=1}^{i-1} \frac{x - \alpha_j}{x - \beta_j}, \quad \prod_{j=1}^{i-1} \frac{y - \beta_j}{y - \alpha_j}
\]
respectively, where $x$ is the $p$th value of $E$ and $y$ is the $q$th value of $F$. Let $W$ be the $(k_a - i + 1)$-by-$(k_b - i + 1)$ matrix whose $(p,q)$th entry is

$$
\max_{x \in E} \left| \frac{(x - \alpha)(y - \beta)}{(x - \beta)(y - \alpha)} \right| \prod_{j=1}^{i-1} \left| \frac{(x - \alpha_j)(y - \beta_j)}{(x - \beta_j)(y - \alpha_j)} \right|
$$

$$
= \max_{x \in E} \left| \frac{x - \alpha}{x - \beta} \right| \prod_{j=1}^{i-1} \left| \frac{x - \alpha_j}{x - \beta_j} \right| \cdot \max_{y \in F} \left| \frac{y - \beta}{y - \alpha} \right| \prod_{j=1}^{i-1} \left| \frac{y - \beta_j}{y - \alpha_j} \right|
$$

for the $p$th value $\alpha \in E'$ and the $q$th value $\beta \in F'$. Now we find the smallest entry of $W$, and if the smallest entry is $W$’s $(p_0,q_0)$th entry, then $\alpha_i$ is the $p_0$th value in $E$ and $\beta_i$ is the $q_0$th value in $F$. If $p_0 \neq k_a - i + 1$, swap the $p_0$th and $(k_a - i + 1)$st value of $E$ and the $p_0$th and $(k_a - i + 1)$st entry of $r_a$. Likewise if $q_0 \neq k_b - i + 1$, swap the $q_0$th and $(k_b - i + 1)$st value of $F$ and the $q_0$th and $(k_b - i + 1)$st entry of $r_b$. Update the $p$th entry and $q$th entry of $r_a$ and $r_b$ by

$$(r_a)_{(p)} \frac{x - \alpha_i}{x - \beta_i}, \quad (r_b)_{(q)} \frac{y - \beta_i}{y - \alpha_i}$$

respectively, where $x$ is the $p$th value of $E$ and $y$ is the $q$th value in $F$. The cost in this step is $O((k_a - i + 1)(k_b - i + 1)(k_a + k_b))$ since each entry (3.6) of $W$ costs $O(k_a + k_b)$ to compute, given $r_a$ and $r_b$.

This completes implementing the loop by Steps 7 – 9 for each $i$.

The overall cost is thus in the order of

$$
\sum_{i=1}^{k} (k_a - i + 1)(k_b - i + 1)(k_a + k_b) = O(k_a k_b (k_a + k_b)).
$$

4 Projection ADI Subspace Methods for Sylvester Equation

Given parameters $\{\alpha_i\}$ and $\{\beta_i\}$, we define the $k$th ADI column subspace to be the column space of the $k$th ADI solution $X_k = Z_k D_k V_k^*$ and the $k$th ADI row subspace to be the row space of $X_k^*$. Equivalently the $k$th ADI column subspace is the column space of $Z_k$, and the $k$th ADI row subspace is the row space of $V_k^*$.

Our numerical experiments strongly suggest often these ADI subspaces are quite good in the sense that the ADI column subspaces come very close to the column space of $X$, the exact solution, and ADI row subspaces come very close to the row space of $X$. This is true even for not so good parameters $\{\alpha_i\}$ and $\{\beta_i\}$. Our numerical experiments also suggest that one single poor shift can effectively offset all previous good shifts and thus degrade ADI approximations enormously for the next many iterations.

Given that it is so hard to select optimal, sometime even decent, parameters in general for ADI solutions to be any good, perhaps we should seek instead solutions having form

$$
\tilde{X}_k = U_k W_k V_k^*
$$

(4.1)
under the Galerkin condition or the minimal residual condition, where $U_k$ has the same columns space as $X_k$ and $V^*_k$ has the same row space as $Y^*_k$. We call a method as such an \textit{projection ADI subspace method}.

Besides being presented primarily for our purpose of describing projection ADI subspace methods, the idea of using Galerkin projection or minimizing the residual is not new. Previously it was used when columns of $U_k$ and $V_k$ span a Krylov subspace of $A$ on $G$ and a Krylov subspace of $B^*$ on $F^*$, respectively [16, 17, 19, 27] and more recently for \textit{Lyapunov equations} for which $U_k = V_k$ spans a direct sum of Krylov subspaces of $A$ on $G$ and $A^{-1}$ on $G$ [28].

\section*{4.1 Galerkin projection}
Suppose $U_k$ and $V_k$ have orthonormal columns. Let residual $R_k = A\tilde{X}_k - \tilde{X}_kB - C$ for an approximation solution $\tilde{X}_k$. The Galerkin condition enforces $U_k^*R_kV_k = 0$. Thus
\begin{equation}
(U_k^*AU_k)W_k - W_k(V_k^*BV_k) = U_k^*CV_k \tag{4.2}
\end{equation}
which is a Sylvester equation but of a much smaller size and can be solved by, e.g., Bartel-Stewart algorithm [4], or Golub-Nash-Van Loan algorithm [12].

Note that $U_k$ and $V_k$ do not necessarily have to have the same number of columns. When they don’t, $W_k$ will not be a square matrix. Also this projection idea is not limited to $U_k$ and $V_k$ whose columns span the ADI subspaces.

\section*{4.2 A minimal residual method}
The minimal residual condition requires to solve
\begin{equation}
\min_{W_k} ||R_k||_F \equiv \min_{W_k} \|A(U_kW_kV_k^*) - (U_kW_kV_k^*)B - C\|_F. \tag{4.3}
\end{equation}
It turns out going from the simple Galerkin projection to this minimal residual condition is utterly nontrivial computationally. The novel idea due to Hu and Reichel [16, p.293] can be modified to work, thanks to Theorem 4.1 below. But the amount of increased work makes it less attractive. Nevertheless, we still present Theorem 4.1 which may be of independent interest in its own right. Adopt the notation of Section 2 in its entirety. By (2.13) and (2.14), the $k$th ADI column and row spaces are
\begin{align*}
\mathcal{C}_k & \overset{\text{def}}{=} \text{colspan}\{Z^{(1)}, Z^{(2)}, \ldots, Z^{(k)}\}, \\
\mathcal{R}_k & \overset{\text{def}}{=} \text{rowspan}\{Y^{(1)*}, Y^{(2)*}, \ldots, Y^{(k)*}\},
\end{align*}
respectively.

\textbf{Theorem 4.1} We have for $i \geq 1$
\begin{align*}
AZ^{(i)} & = G + \prod_{j=1}^{i-1}(\beta_j - \alpha_j)Z^{(j)} + \beta_iZ^{(i)}, \tag{4.4} \\
Y^{(i)*}B & = F^* + \prod_{j=1}^{i-1}(\alpha_j - \beta_j)Y^{(j)*} + \alpha_iY^{(i)*}, \tag{4.5}
\end{align*}
where $\prod_{j=1}^{0}(\cdots)$ is taken to be 0. Therefore

\[
A\mathcal{C}_k \subseteq \text{colspan}\{G, Z^{(1)}, Z^{(2)}, \ldots, Z^{(k)}\} = \text{colspan}\{G\} + \mathcal{C}_k, \quad (4.6)
\]

\[
\mathcal{R}_k B \subseteq \text{rowspan}\{F^*, Y^{(1)*}, Y^{(2)*}, \ldots, Y^{(k)*}\} = \text{rowspan}\{F^*\} + \mathcal{R}_k. \quad (4.7)
\]

**Proof** By (2.13), we have

\[
AZ^{(1)} = A(A - \beta_1 I)^{-1}G
\]

\[
= G + \beta_1(A - \beta_1 I)^{-1}G
\]

\[
= G + \beta_1 Z^{(1)},
\]

\[
AZ^{(i+1)} = AZ^{(i)} + (\beta_{i+1} - \alpha_i)A(A - \beta_{i+1} I)^{-1}Z^{(i)}
\]

\[
= AZ^{(i)} + (\beta_{i+1} - \alpha_i)Z^{(i)} + (\beta_{i+1} - \alpha_i)\beta_{i+1}(A - \beta_{i+1} I)^{-1}Z^{(i)}
\]

\[
= AZ^{(i)} + (\beta_{i+1} - \alpha_i)Z^{(i)} + \beta_{i+1}(Z^{(i+1)} - Z^{(i)})
\]

\[
= AZ^{(i)} - \alpha_i Z^{(i)} + \beta_{i+1}Z^{(i+1)}
\]

\[
= \cdots
\]

\[
= AZ^{(1)} - \alpha_1 Z^{(1)} + \prod_{j=2}^{i}(\beta_j - \alpha_j)Z^{(j)} + \beta_{i+1}Z^{(i+1)}
\]

\[
= G + \prod_{j=1}^{i}(\beta_j - \alpha_j)Z^{(j)} + \beta_{i+1}Z^{(i+1)}
\]

which proves (4.4). Similarly

\[
Y^{(1)*}B = F^*(B - \alpha_1 I)^{-1}B
\]

\[
= F^* + \alpha_1 F^*(B - \alpha_1 I)^{-1}
\]

\[
= F^* + \alpha_1 Y^{(1)*},
\]

\[
Y^{(i+1)*}B = Y^{(i)*}B + (\alpha_{i+1} - \beta_i)Y^{(i)*}(B - \alpha_{i+1} I)^{-1}B
\]

\[
= Y^{(i)*}B + (\alpha_{i+1} - \beta_i)Y^{(i)*} + (\alpha_{i+1} - \beta_i)(\alpha_{i+1}Y^{(i)*}(B - \alpha_{i+1} I)^{-1})
\]

\[
= Y^{(i)*}B + (\alpha_{i+1} - \beta_i)Y^{(i)*} + \alpha_{i+1}(Y^{(i+1)*} - Y^{(i)*})
\]

\[
= Y^{(i)*}B - \beta_iY^{(i)*} + \alpha_{i+1}Y^{(i+1)*}
\]

\[
= F^* + \prod_{j=1}^{i}(\alpha_j - \beta_j)Y^{(j)*} + \alpha_{i+1}Y^{(i+1)*}
\]

which proves (4.5). \(\square\)

5 **Application to Lyapunov equation**

fADI in Section 2 is a natural extension of the LR-CF ADI [20, 22, 24] for the Lyapunov Equation

\[
AX + XA^* = C, \quad (5.1)
\]
where $A$, $C$, and unknown $X$ are all $n \times n$, and $C$ is Hermitian. Since, Lyapunov Equation (5.1) is a special case of Sylvester equation (1.1) with $B = -A^*$, previous developments apply upon substituting $B = -A^*$ and $\beta_i = -\bar{\alpha}_i$, and most expressions can be much simplified, too.

Equations (2.1) – (2.3) become

$$X_{i+1} = 2\Re(\alpha_i)(A + \bar{\alpha}_i)\text{inv}(A^* + \alpha_i I)^{-1} + (A - \alpha_i I)(A + \bar{\alpha}_i)\text{inv}(A^* - \bar{\alpha}_i I)(A^* + \alpha_i I)^{-1},$$

(5.2)

where $\Re(\cdot)$ takes the real part of a complex number, and the error equation

$$X_{i+1} - X = (A - \alpha_i I)(A + \bar{\alpha}_i)\text{inv}(X_i - X)(A^* - \bar{\alpha}_i I)(A^* + \alpha_i I)^{-1}\left[\prod_{j=0}^{i}(A^* - \bar{\alpha}_j I)(A^* + \alpha_j I)^{-1}\right].$$

(5.3)

Now suppose also $C = -GG^*$, i.e., take $F = -G^*$ in (2.5) – (2.8) to get $Y_{i+1} = Z_{i+1}^*$ and $X_{i+1} = Z_{i+1}D_{i+1}Z_{i+1}^*$ with

$$Z_{i+1} = \left((A + \bar{\alpha}_i)\text{inv}G, (A - \alpha_i I)(A + \bar{\alpha}_i)\text{inv}Z_i\right),$$

(5.5)

$$D_{i+1} = \begin{pmatrix} -2\Re(\alpha_i) & 0 \\ 0 & D_i \end{pmatrix}.$$  

(5.6)

Upon re-naming the parameters $\{\alpha_i\}$ as we did to get (2.11) – (2.14), we get the column blocks of $Z_{i+1}$ as

$$\begin{align*}
(A + \bar{\alpha}_1 I)^{-1}G, \\
(A - \alpha_1 I)(A + \bar{\alpha}_2 I)^{-1}(A + \bar{\alpha}_1 I)^{-1}G, \\
(A - \alpha_2 I)(A + \bar{\alpha}_3 I)^{-1}(A - \alpha_1 I)(A + \bar{\alpha}_2 I)^{-1}(A + \bar{\alpha}_1 I)^{-1}G, \\
\cdots
\end{align*}$$

(5.7)

Finally we can write

$$Z_k = \left(Z^{(1)} \quad Z^{(2)} \quad \cdots \quad Z^{(k)}\right),$$

with

$$\begin{align*}
Z^{(1)} &= (A + \bar{\alpha}_1 I)^{-1}G, \\
Z^{(i+1)} &= (A - \alpha_i I)(A + \bar{\alpha}_{i+1} I)^{-1}Z^{(i)} \\
&= Z^{(i)} - (\bar{\alpha}_{i+1} + \alpha_i)(A + \bar{\alpha}_{i+1} I)^{-1}Z^{(i)}.
\end{align*}$$

(5.8)

and

$$X_k = Z_kD_kZ_k^*, \quad D_k = -2\text{diag}(\Re(\alpha_1)I_r, \ldots, \Re(\alpha_k)I_r).$$

(5.9)

Based on (5.8) and (5.9), an fADI for Lyapunov equation $AX + XA^* + GG^* = 0$ is obtained as in Algorithm 5.1.

**Algorithm 5.1** fADI for Lyapunov equation $AX + XA^* + GG^* = 0$:
Figure 6.1: Relative residual errors for ADI solutions and solutions by Galerkin projection via ADI subspaces for two different runs for Example 6.1 with (6.1).

**Input:**
(a) $A_{(m \times m)}$, and $G_{(m \times r)}$;
(b) ADI shifts $\{\alpha_1, \alpha_2, \ldots\}$;
(c) $k$, the number of ADI steps;

**Output:** $Z_{(m \times kr)}$ and $D_{(kr \times kr)}$ such that $ZDZ^*$ approximately solves Lyapunov equation $AX + XA^* + GG^* = 0$;

1. $Z_{(:,1:r)} = (A + \bar{\alpha}_1 I)^{-1}G$;
2. for $i = 1, 2, \ldots, k$ do
3. $Z_{(:,ir+1:(i+1)r)} = Z_{(:,(i-1)r+1:ir)} - (\bar{\alpha}_{i+1} + \alpha_i)(A + \bar{\alpha}_{i+1})^{-1}Z_{(:,(i-1)r+1:ir)}$;
4. end for;
5. $D = -2 \text{diag} (\Re(\alpha_1)I_r, \ldots, \Re(\alpha_k)I_r)$.

For a stable Lyapunov equation, this essentially gives the so-called Cholesky Factored ADI (CF-ADI) of Li and White [20] and Low Rank ADI of Penzl [22], except that in CF-ADI/LR-ADI matrices $D_i$ are embedded into $Z_i$.

### 6 Numerical Experiments

In this section, we shall report several numerical examples to demonstrate that Galerkin projection via ADI subspaces can lead to more accurate solutions than ADI alone.

**Example 6.1** This is essentially [5, Example 1], except for $C$ which will be set to some random rank-1 matrix. Depending on parameters $a, b, s$ and the dimension $n$, matrices $A$, $B$, and $C$ are generated as follows. First, set

\[
\hat{A} = \text{diag}(-1, -a, -a^2, \ldots, -a^{n-1}),
\]

\[
\hat{B} = \text{diag}(1, b, b^2, \ldots, b^{n-1}),
\]

\[
\hat{C} = \hat{G}\hat{F}^*.
\]
Figure 6.2: Relative residual errors for ADI solutions and solutions by Galerkin projection via ADI subspaces for two different runs for Example 6.1 with (6.2).

where \( \tilde{G} \) and \( \tilde{F} \) are \( n \times 1 \) and generated randomly as by \texttt{randn}(n,1) in MATLAB. Parameters \( a \) and \( b \) regulate the distribution of the spectra of \( A \) and \( B \), respectively, and therefore their separation. The entries of the solution matrix to \( \hat{A}\hat{X} - \hat{X}\hat{B} = \hat{C} \) are then given by

\[
\hat{X}_{(i,j)} = \frac{\hat{C}_{(i,j)}}{A_{(i,i)} - B_{(j,j)}}.
\]

Next we employ a transformation matrix to define

\[
A = T^{-T}\tilde{A}T^T, \quad B = T^{-T}\tilde{B}T^{-1}, \quad G = T^{-T}\tilde{G}, \quad F = \tilde{F}T^{-1},
\]

where \( T = H_2SH_1 \in \mathbb{C}^{n \times n} \) is defined through

\[
H_1 = I_n - \frac{2}{n}h_1 h_1^T, \quad h_1 = (1, 1, \ldots, 1)^T,
\]

\[
H_2 = I_n - \frac{2}{n}h_2 h_2^T, \quad h_2 = (1, -1, \ldots, (-1)^{n-1})^T,
\]

\[
S = \text{diag}(1, s, \ldots, s^{n-1}).
\]

The scalar \( s \) is used here to regulate the conditioning of \( T \). Because of the way they are constructed, each linear system with shifted \( A \) or \( B \) costs \( O(n) \) flops to solve. In all our tests reported here, \( k = 25 \) and \( n = 500 \). We tested Algorithms 2.1 and 3.1 on two sets of parameter values:

\[
a = 1.03, \quad b = 1.008, \quad s = 1.001; \quad (6.1)
\]

\[
a = 1.03e^{i\theta}, \quad b = 1.008e^{i\theta}, \quad s = 1.001, \quad (6.2)
\]

where \( i = \sqrt{-1} \) and \( \theta = \pi/(2n) \). The values given in (6.1) were the ones used in [5]. In applying Algorithm 3.1, each Arnoldi run takes 35 steps and 17 best Ritz values are
taken, and thus both $E$ and $F$ have 34 values among which 25 are selected in the end. Our fADI produces an approximation $X_k = Z_k D_k Y_k^*$, along with intermediate approximations $X_i = Z_i D_i Y_i^*$ for $i \leq k$. For two runs with different random $F$ and $G$, Figures 6.1 and 6.2 plot the relative residual errors $\frac{\|AX_i - X_i B - GF^*\|_F}{\|GF^*\|_F}$ (marked as “ADI”), as well as the relative residual errors for the approximations $U_i W_i V_i^*$ by Galerkin projection (4.2) (marked as “ADI+Galerkin”). We have run tests on each parameter set many times with different random $F$ and $G$, and the residual behaviors are all similar to those plotted in Figures 6.1 and 6.2. In both figures, Galerkin projection via ADI subspaces produces better approximations after $i \geq 7$ and the improvements are up to more than 2 decimal digits.

Example 6.2 Chahlaoui and Van Dooren [8] compiled a collection of benchmark examples for model reduction. Except those for descriptor systems, these examples give rise to Lyapunov equations $AX + XA^* + GG^* = 0$. Simplified versions of Algorithms 2.1 and 3.1 upon substituting $B = -A^*$ and $F = -G^*$ can be applied. We have tested ADI and Galerkin projection via ADI subspaces on these equations, and found out both perform badly when $A$ is highly non-normal in the sense that $\|A\|_F^2$ and $\|A\|_F^2 - \sum |\lambda_j|^2$ are of the same magnitude and $\sum |\lambda_j|^2 \ll \|A\|_F^2$, where $\{\lambda_j\}$ consists of all $A$’s eigenvalues. In the collection, there are two other examples whose $A$ are in fact normal, i.e., $\|A\|_F^2 = \sum |\lambda_j|^2$. We now report our numerical results on them. The first example is FOM: $A = \text{diag}(A_1, A_2, A_3, A_4)$ with

$$A_1 = \begin{pmatrix} -1 & 100 \\ -100 & -1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} -1 & 200 \\ -200 & -1 \end{pmatrix}, \quad A_3 = \begin{pmatrix} -1 & 400 \\ -400 & -1 \end{pmatrix},$$

Figure 6.3: Relative residual errors for ADI solutions and solutions by Galerkin projection via ADI subspaces. Left: FOM in [8]; Right: HEAT in [8].
$A_4 = \text{diag}(-1, -2, \ldots, -1000)$, and $G = (10, \ldots, 10, 1, \ldots, 1)^T$. So $n = 1006$ and each linear system with shifted $A$ or $A^*$ takes $O(n)$ flops to solve. Apply Algorithms 2.1 and 3.1 with $k = 66$, where for Algorithm 3.1, each Arnoldi run takes 76 steps and 38 best Ritz values are taken, and thus $E$ has 76 values among which 66 are selected in the end. The left of Figure 6.3 plots the relative residual errors for ADI solutions and solutions by Galerkin projection via ADI subspaces, as explained in Example 6.1. For this example, ADI barely does anything while projection ADI subspace method does extremely well.

The other example is from discretizing the 1-D heat equation. $A$ is real symmetric tridiagonal. Thus each linear system with shifted $A$ or $A^*$ takes $O(n)$ flops to solve. The size of $A$ can be made as large as one wishes. As in [8], we take $n = 200$, and $A$ has diagonal entries $-808$ and off-diagonal entries $404$, and all of $G$’s entries are zero, except $G_{(67)} = 1$. With $k = 20$ for Algorithm 3.1, each Arnoldi run takes 30 steps and 15 best Ritz values are taken, and thus $E$ has 30 values among which 20 are selected in the end. The right of Figure 6.3 plots the relative residual errors for ADI solutions and for solutions by Galerkin projection via projection ADI subspaces method.

Example 6.3 This is a Sylvester equation $AX - XB = GF^*$ with real symmetric $A$ and $B$, both taken from the Harwell-Boeing Collection. In fact, $A(675 \times 675)$ is NOS6 and $B(468 \times 468)$ is negative NOS5 from Set LANPRO\(^3\). $G$ and $F$ are taken to be random vectors. Sylvester equations so constructed are solely for our testing purpose because there is no physical background yet for combining the two matrices together in one Sylvester equation. Both $A$ and $B$ are sparse and in fact very narrow-banded, and each linear system with shifted $A$ or $B$ costs $O(m)$ or $O(n)$ flops to solve, respectively.

Figure 6.4 plots the relative residual errors for ADI solutions and solutions by Galerkin projection via ADI subspaces for $k = 34$ (left) and $k = 44$ (right) by applying Algorithms 2.1 and 3.1. For $k = 34$, in Algorithm 3.1 each Arnoldi run takes 44 steps and 22 best Ritz values are taken, and thus both $E$ and $F$ have 44 values among which 34 are selected in the end; and for $k = 44$, in Algorithm 3.1 each Arnoldi run takes 54 steps and 27 best Ritz values are taken, and thus both $E$ and $F$ have 54 values among which 44 are selected in the end.

7 Conclusions

We have presented a factored ADI for Sylvester equation $AX - XB = GF^*$, Lyapunov equation as a special case included. It is based on a set of formulas which generalize corresponding ones in the CF-ADI for Lyapunov equation. They enable one to compute the columns of the left factor and the rows of the right factor one block per step. We also demonstrate that often much more accurate solutions than the ADI solutions can be gotten by performing Galerkin projection using the column spaces and row spaces of the computed approximate solutions.

\(^3\)http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/lanpro/lanpro.html.
Figure 6.4: Relative residual errors for ADI solutions and solutions by Galerkin projection via ADI subspaces for Example 6.3. Left: $k = 34$; Right: $k = 44$.

References


