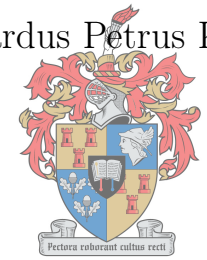
Comparison of methods for solving Sylvester systems

by

Gerhardus Petrus Kirsten



UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY

Thesis presented in partial fulfilment of the requirements for the degree of
Master of Science
in the Faculty of Science at Stellenbosch University

Supervisor: Dr Nick Hale
Co-supervisor: Prof André Weideman

December 2018

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2018

Abstract

This thesis serves as a comparative study of numerical methods for solving Sylvester equations, which are linear matrix equations of the form $AX + XB + C = 0$. These equations have important applications in many areas of science and engineering, such as signal processing, control theory, and systems engineering, and their efficient solution is therefore of practical significance.

As with standard linear systems (i.e., those of the form $Ax = b$), algorithms for the efficient solution of Sylvester equations typically fall into two categories, namely direct and iterative methods. As a naive approach, one can convert a Sylvester equation to a standard linear system (of larger size) using Kronecker operations, and then apply standard methods from numerical linear algebra. We shall see, however, that unless the matrix is very sparse and structured, this approach is usually inefficient.

Instead, modern algorithms for solving Sylvester equations are applied directly to the equation in Sylvester form. When the matrices A and B are small and dense, direct methods such as Bartels–Stewart and Hessenberg–Schur, which are based on suitable factorisations of A and B, are efficient. As the matrices become larger, however, one typically switches to a projection-based or some other iterative method. The projection methods considered in this thesis use Krylov subspace techniques to project the system onto a much smaller subspace, which can be solved efficiently using one of the direct methods mentioned above as an internal solver. In this thesis we consider two different subspaces for the comparison of projection methods, namely the standard Krylov subspace and an enriched approximation space known as the extended Krylov subspace. We shall see that when the matrix C is of low rank, then the extended Krylov subspace method is competitive with direct methods, even when the system size is relatively small.

Each of the methods discussed above are compared, both theoretically by consideration of floating point operation counts and numerically by computational efficiency and accuracy, when used to solve several example problems arising in applications. Based on the results of these experiments, it is concluded that a method based on the eigenvalue decompositions of A and B is the most efficient direct method, although to some degree at the expense of numerical stability. In the class of projection methods, we find that the extended Krylov subspace to be the most efficient approximation space.

Samevatting

Hierdie tesis is 'n vergelykende studie van numeriese metodes om Sylvester-vergelykings op te los, wat lineêre matriksvergelykings is met die vorm $AX + XB + C = 0$. Die vergelykings het belangrike toepassings in verskeie wetenskaplike en ingenieursvelde soos seinverwerking, beheerteorie en stelselingenieurswese, en die doeltreffende oplos daarvan is dus van praktiese belang.

Soos wat die geval is met gewone lineêre stelsels (met die vorm $Ax = b$), bestaan daar normaalweg twee kategorieë vir die doeltreffende oplos van Sylvester-vergelykings, naamlik direkte en iteratiewe metodes. As 'n naïewe benadering kan 'n mens 'n Sylvester-vergelyking in 'n gewone lineêre stelsel omskakel deur die gebruik van Kronecker-bewerkings en dan standaardmetodes van numeriese lineêre algebra toepas. Tensy die matriks (wat nou veel groter is) baie yl en gestruktureerd is, is hierdie benadering egter selde doeltreffend.

In plaas van bogenoemde benadering word moderne algoritmes vir die oplos van Sylvester-vergelykings direk in Sylvester-vorm toegepas. Wanneer die matrikse A en B klein is, is direkte metodes soos Bartels–Stewart en Hessenberg–Schur, wat op gepaste faktoriserings van A en B gebaseer is, doeltreffend. Wanneer die matrikse egter vergroot, word daar normaalweg na 'n projeksie-gebaseerde of ander iteratiewe metode oorgeskakel. Die projeksiemetodes wat in hierdie tesis bespreek word, gebruik Krylov-subruimtetegnieke om die stelsel op 'n kleiner subruimte te projekteer, wat dan doeltreffend opgelos kan word deur een van die direkte metodes hierbo genoem as 'n interne oplosser te gebruik. In die tesis word twee verskillende subruimtes vir die vergelyking van projeksiemetodes oorweeg, naamlik die gewone Krylov-subruimte en die verrykte benaderingsruimte bekend as die uitgebreide Krylov-subruimte. Ons sien dat wanneer die matriks C van lae rang is, die uitgebreide Krylov-subruimte kompeterend met direkte metodes is, selfs wanneer die stelselgrootte relatief klein is.

Elke metode wat hierbo bespreek word, word vergelyk — teoreties, deur middel van wisselpuntbewerkingstellings, sowel as numeries, deur berekenings-doeltreffendheid en -akkuraatheid — wanneer dit gebruik word om verskeie voorbeeldprobleme op te los wat in toepassings voorkom. Die bevindings van hierdie eksperimente toon dat 'n metode gebaseer op die eiewaarde-ontbindings van A en B die doeltreffendste direkte metode is, hoewel dit in 'n mate ten koste van numeriese stabiliteit is. In die geval van projeksiemetodes word daar bevind dat die uitgebreide Krylov-subruimte die doeltreffendste benaderingsruimte is.

Acknowledgements

The author wishes to acknowledge the following people for their various contributions towards the completion of this work:

- My supervisors, Dr Nick Hale and Prof André Weideman for their professional scientific guidance, perfectionism and the effort that they put into supporting my thesis by meeting with me on a weekly basis.
- My roommate JC Landman and my friend Rachel Theron for always being willing to listen to another story about 'Sylvester'. The two of you played an important role in helping me find a balance.
- Prof. Valeria Simoncini for some valuable advice and bringing inspiration to the topic I worked on.
- My parents, for inspiring me to perform as an academic and always being willing to advise me on how to make it through the process of writing a thesis. Also, for providing some valuable scientific input.
- My grandfather, Prof. Sampie Terreblanche, for being an absolute academic role model. The influence you had in inspiring me academically is ineffable.

Table of Contents

Matrix Decompositions

Cholesky decomposition A factorisation of an $n \times n$ real symmetric matrix A into the form $A = LL^T$, where the $n \times n$ matrix L is lower triangular. This factorisation exists if and only if A is positive definite.

Eigenvalue decomposition A factorisation of an $n \times n$ matrix A into a canonical form containing the eigenvalues and eigenvectors, given that A is diagonalisable. The factorisation is expressed as $A = V\Lambda V^{-1}$, where V is an $n \times n$ matrix containing the eigenvectors and Λ an $n \times n$ diagonal matrix containing the eigenvalues.

Hessenberg decomposition A factorisation of an $n \times n$ matrix A into the form $A = QHQ^T$. The $n \times n$ matrix Q is orthogonal, while the $n \times n$ matrix H is in upper Hessenberg form.

***LU* decomposition** A factorisation of an $n \times n$ matrix A into a product of an $n \times n$ lower triangular and an $n \times n$ upper triangular matrix, such that $A = LU$.

***QR* decomposition** A factorisation of an $n \times m$ matrix A into a product $A = QR$. The $n \times n$ matrix Q is orthogonal, and the $n \times m$ matrix R is upper triangular.

***QR* decomposition (economy-size)** If A $(n \times m)$ is overdetermined $(n > m)$, the full QR decomposition forms an upper triangular matrix with $n - m$ zero rows at the bottom. This can be avoided by factorising A such that $A = QR$, where Q is has size $n \times m$ and R is upper triangular and square.

Schur decomposition (complex) A factorisation of an $n \times n$ matrix A into the form $A = QUQ^H$. The matrix $Q \in \mathbb{C}^{n \times n}$ is unitary and $U \in \mathbb{C}^{n \times n}$ is upper triangular. The upper triangular matrix U is known as the complex Schur form of A, and since A and U are similar, the eigenvalues of A are contained on the diagonal of U.

Schur decomposition (real) A factorisation of an $n \times n$ matrix A into the form $A = QUQ^T$. The matrix $Q \in \mathbb{R}^{n \times n}$ is orthogonal and $U \in \mathbb{R}^{n \times n}$ is quasi-upper triangular. The quasi-upper triangular matrix U is known as the real Schur form of A, and since A and U are similar, the eigenvalues of A are contained on the diagonal or block-diagonal of U.

Singular value decomposition (SVD) A factorisation of an $n \times m$ matrix A into the form $A = U\Sigma V^T$. The matrices $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{m \times m}$ are orthogonal and $\Sigma \in \mathbb{R}^{n \times m}$ is a diagonal matrix containing the singular values in non-increasing order. The singular values are representative of the square root of the eigenvalues of either $A^T A$ or AA^T.

Truncated singular value decomposition The singular values of the SVD are contained in non-increasing order on the diagonal of Σ. Some of these singular values are small enough to be negligible in an approximation. If the ℓ smallest singular values are removed from Σ and the respective ℓ columns of U and V relating to them, the truncated SVD is formed as $A = U_\ell \Sigma_\ell V_\ell^T$. Here $U_\ell \in \mathbb{R}^{n \times \ell}$, $\Sigma_\ell \in \mathbb{R}^{\ell \times \ell}$ and $V_\ell \in \mathbb{R}^{m \times \ell}$.

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

The efficient solution of the Sylvester matrix equation

$$AX + XB + C = 0, \quad A \in \mathbb{R}^{n\times n}, \ B \in \mathbb{R}^{m\times m}, \ C \in \mathbb{R}^{n\times m} \tag{1.1}$$

is of great importance, since it arises naturally in many areas of science and engineering such as decoupling techniques for ordinary and partial differential equations, the numerical solution of Riccati equations, and image restoration. An extensive review of applications is presented in [9]. As such, the solution of large-scale linear matrix equations is a well-researched topic, with the Sylvester equation, in particular, possessing a rich set of literature. However, the literature is still in need of a thorough numerical comparison of the respective algorithms [60]. Such a comparison is the objective of this thesis.

Since the operator $S(X) = AX + XB$ is a linear function of the unknown X, we say that (1.1) is a linear matrix equation. The Sylvester equation can, therefore, be expressed and solved as a linear system in standard form (i.e., $Ax = b$), at the expense of catastrophically increasing the size of the coefficient matrix. This is, nevertheless, still taken into consideration as a potential solution method.

The first appearance of the Sylvester equation is usually attributed to the work of J.J. Sylvester [63]. The special case where $B = A^T$, is known as the Lyapunov matrix equation, and it plays a key role in control and communications theory [1, 5]. This equation, named in honour of A.M. Lyapunov for his early contributions to the stability problem of motion, is defined as

$$AX + XA^T + C = 0, \quad A, C \in \mathbb{R}^{n\times n}. \tag{1.2}$$

See [3] for a bibliography of Lyapunov's work. Whilst solutions to (1.2) can be trivially obtained from algorithms designed to solve (1.1), we shall see that several advantages can be exploited by considering the special structure of (1.2).

We shall see that the best methods for solving (1.1) and (1.2) are dependent on the structure of the problem, especially regarding the size and the sparsity of the coefficient matrices A and B. Therefore, a distinction will be made between dense and sparse matrices in this thesis. We will follow Wilkinson's definition of sparsity, stating that a matrix is sparse if it has enough zeros that it pays to take advantage of them. This means that the matrix can be reordered in such a way that the LU or Cholesky factorisation of the matrix will contain enough zero values such that a sparse triangular solver can be used to solve the system in a reduced number of steps.

Matrices without these properties will be referred to as dense matrices. Wilkinson states this definition in negated form [68]:

> The matrix may be sparse, either with the nonzero elements concentrated on a narrow band centred on the diagonal or alternatively they may be distributed in a less systematic manner. We shall refer to a matrix as dense if the percentage of zero elements or its distribution is such as to make it uneconomic to take advantage of their presence.

Two examples of sparse matrices are shown in Figure 1.1. The structure of matrices will be presented by means of spy plots, a MATLAB visualisation tool for displaying the sparsity pattern of a matrix. Spy plots present a figure of a matrix with nonzero elements represented by dots, while zero elements remain blank. The sparsity of the Sylvester equation is relevant, since special techniques can be used to exploit this sparsity in order to solve the system in a reduced number of operations. These techniques are discussed in Chapter 4. This can be achieved using the original coefficient matrices A and B or after conversion to a linear system in standard form.

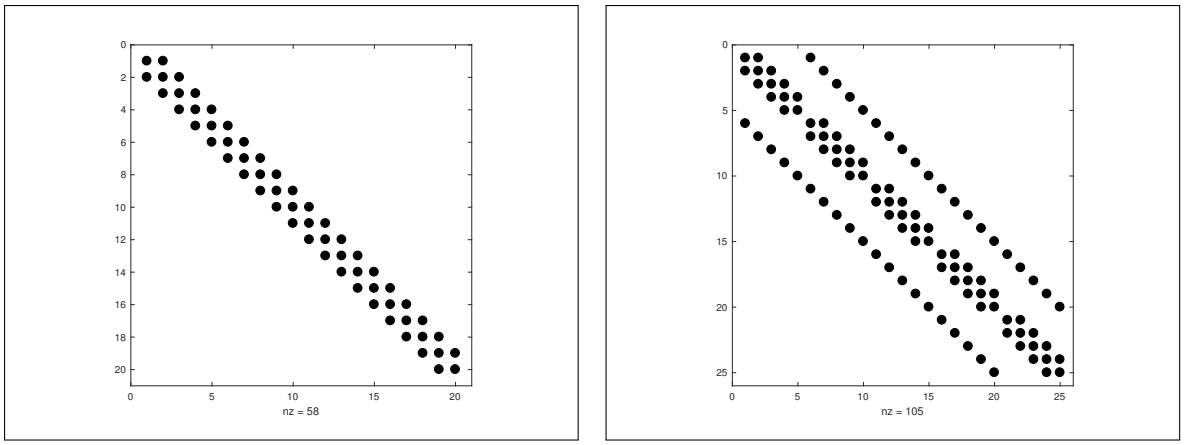


FIGURE 1.1: *Spy plots with examples of sparse matrices. On the left a tridiagonal matrix is shown and a banded matrix on the right.*

For standard linear systems of the form $Ax = b$, the trend of algorithmic development is such that direct methods are used for a small or structured coefficient matrix A and projection or iterative methods when the matrix becomes large and sparse. Iterative methods are typically not beneficial if the coefficient matrix is dense, unless the matrix has a special structure such that fast matrix-vector products can be exploited.

Similarly, for the solution of (1.1) and (1.2), when the coefficient matrices are small or structured, transformation methods based on the Schur and Hessenberg decomposition are used to solve the equations directly [4, 21, 24]. A comparison of these methods, both theoretical and computational, is presented in Chapter 4. However, as the dimension of the coefficient matrices increase, direct methods become slow and inefficient. The attention is subsequently turned to projection and iterative methods.

The development of the projection algorithms for Sylvester and Lyapunov equations has mainly been focused on using Krylov subspace techniques to project the large problem onto a smaller subspace, containing enough spectral information for an accurate approximation. Subsequently, the reduced system is solved using one of the direct methods as an internal solver. This is another

reason why the efficiency of the direct methods is important. The projection of the Sylvester equation onto a smaller subspace can be achieved either by projecting each of the matrices A and B separately or by applying projection algorithms to the operator $S(X) = AX + XB$. The latter is simpler in the sense that standard projection techniques are applied, but that the orthogonalisations are done with respect to the operator in stead of a matrix [47]. In this thesis we only consider algorithms projecting the matrices A and B separately (see [14, 34, 35, 52, 58, 60] and the references therein). These techniques allow for more flexibility as the structure of each of the matrices A and B can be exploited separately. Consider, for example, the case where one of the matrices A or B are Toeplitz[1]. Applying a standard projection method to this matrix allows for the use of fast matrix-vector products to speed up convergence.

It should be noted that the literature on projection methods is largely focused on the situation where the right-hand side C from (1.1) is low rank such that $C = EF^T$, with E and F tall and skinny matrices. This will be discussed in Chapter 6. The case where the right-hand side is of full rank is still an open question in the field. Palitta and Simoncini recently began addressing this question in [44], where they consider the case where all matrices are of full rank, but have a banded structure.

We see then that there is no shortage of methods for solving these Sylvester type systems; however the scientific community is not always in agreement over which methods are more suitable for different settings. Simoncini states [60]

> Despite a lot of intense work in the past 15-20 years, the community has not entirely agreed upon the best approaches for all settings, hence the need for an overview that aims to analyse where the field stands at this point.

This statement raises the main research question for this thesis. Which of the existing direct and projection methods for solving the Sylvester and Lyapunov equations are the most efficient in certain settings arising from realistic problems in science and engineering? These methods are algebraically and numerically compared using different model problems coming from certain applications described in Chapter 3.

The algebraic comparisons are made by means of floating-point operation (flop) counts. A flop is defined as any algebraic operation $(+, -, \times, \div)$ and acts as a theoretical proxy for how long an algorithm can be expected to execute. It should be noted however that there are many other factors (i.e., memory access) affecting the actual execution time. This is why we also consider numerical examples to compare how well the algorithms perform in practice.

We will choose the model problems in such a way that the comparisons are done on systems of different sizes and sparsities, in order to conclude which methods work best in which setting. All algorithms derived in this thesis will be compared using the author's own implementations of them in MATLAB. Comparisons of algorithms not derived will be done using their respective built-in MATLAB functions (e.g., *lyap*, *pcg*). All computations are done in MATLAB, on a Macbook Pro with a 2.4 GHz intel core i5 processor and 16 GB of RAM.

The outline of the thesis is as follows:

- Chapter 2 contains a discussion of the elementary theory of Sylvester- and Lyapunov equations, and existence and uniqueness theorems.

[1] A matrix in which each descending diagonal, from left to right, is constant

- Chapter 3 contains some of the well-known applications of the Sylvester and Lyapunov equations as well as a derivation of the model problems that will be considered in later chapters.
- Chapter 4 reviews four different direct methods for solving the Sylvester equation and compares these methods algebraically and numerically using model problems derived in Chapter 3. The methods are also compared to the built-in MATLAB solver *lyap*.
- Chapter 5 serves as a review of some well-known Krylov subspace methods for standard linear systems $Ax = b$.
- Chapter 6 describes the development of projection methods for solving the Sylvester- and Lyapunov equations with a comparison of these methods by using model problems derived in Chapter 3.
- The conclusions and potential future work are discussed in Chapter 7.

CHAPTER 2

Introductory theory of Sylvester equations

Contents

The aim of this chapter is to discuss the necessary background information regarding the Sylvester equation.

2.1 The Kronecker product

The Sylvester equation (1.1) can be restated as a standard linear system

$$\widetilde{A}x = c, \tag{2.1}$$

where we define

$$\widetilde{A} = I_m \otimes A + B^T \otimes I_n, \quad x = \text{vec}(X), \quad c = -\text{vec}(C), \tag{2.2}$$

with $\otimes$ and vec defined below.

Definition 2.1 ([33]). For some $A \in \mathbb{R}^{n_1 \times m_1}$ and $B \in \mathbb{R}^{n_2 \times m_2}$, the Kronecker product is defined as

$$A \otimes B = \begin{bmatrix} (A)_{1,1}B & (A)_{1,2}B & \cdots & (A)_{1,m_1}B \\ (A)_{2,1}B & (A)_{2,2}B & \cdots & (A)_{2,m_1}B \\ \vdots & & & \vdots \\ (A)_{n_1,1}B & (A)_{n_1,2}B & \cdots & (A)_{n_1,m_1}B \end{bmatrix} \in \mathbb{R}^{n_1 n_2 \times m_1 m_2}.$$

Definition 2.2. Suppose a matrix $A \in \mathbb{R}^{n \times m}$ has entries $(A)_{i,j}$, then

$$\text{vec}(A) = [(A)_{1,1}, (A)_{2,1}, \ldots, (A)_{n,1}, (A)_{1,2}, \ldots, (A)_{n,2}, \ldots, (A)_{1,m}, \ldots, (A)_{n,m}]^T.$$

Denoting the set of eigenvalues of a matrix A by $\Lambda(A)$, we have the following Lemma.

Lemma 2.1 ([33]). The Kronecker product satisfies the following properties:

1. $\text{vec}(AXB) = (B^T \otimes A)\text{vec}(X)$;
2. If $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$ and $\Lambda(A) = \{\lambda_i\}_{i=1}^n$, $\Lambda(B) = \{\phi_j\}_{j=1}^m$, then

$$\Lambda(A \otimes B) = \{\lambda_i \phi_j \mid i = 1, \ldots, n, j = 1, \ldots, m\}, \tag{2.3}$$

and

$$\Lambda(A \otimes I_m + I_n \otimes B) = \{\lambda_i + \phi_j \mid i = 1, \ldots, n, j = 1, \ldots, m\}. \tag{2.4}$$

These results will be important when discussing the existence and uniqueness of the solution X in the following section.

2.2 Existence and uniqueness

Consider again the Sylvester equation (1.1). The existence and uniqueness of the solution X can be guaranteed in differing ways. One of the first conditions was given by Roth [48], stating that a solution to (1.1) exists if and only if the matrices

$$\begin{bmatrix} A & C \\ 0 & -B \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} A & 0 \\ 0 & -B \end{bmatrix}, \tag{2.5}$$

are similar. If the solution does exist, then the similarity transform is given by

$$\begin{bmatrix} I & X \\ 0 & I \end{bmatrix}, \tag{2.6}$$

with X the solution to (1.1).

Alternatively, a simpler condition comes from [50], stating that (1.1) admits a unique solution if and only if $\Lambda(A) \cap \Lambda(-B) = \emptyset$. The informal proof comes from property 2 in Lemma 2.1. If (1.1) is restated as the linear system (2.1), it will admit a unique solution if and only if the matrix $\widetilde{A}$ is nonsingular. Since the determinant of a matrix can be expressed as a product of its eigenvalues, it is sufficient to state that the matrix will be nonsingular if and only if none of the eigenvalues are 0. Considering (2.4), this is equivalent to requiring $\lambda_i \neq -\phi_j$ $(i = 1, \ldots, n, j = 1, \ldots, m)$ if $\Lambda(A) = \{\lambda_i\}_{i=1}^n$ and $\Lambda(B) = \{\phi_j\}_{j=1}^m$. This is equivalent to $\Lambda(A) \cap \Lambda(-B) = \emptyset$. This result can be extended to the Lyapunov equation such that $\lambda_i \neq -\lambda_j$ for all $i, j = 1, \ldots, n$. It is therefore sufficient to require that the coefficient matrix A of the Lyapunov equation should be stable (i.e., all the eigenvalues in the open left half plane), since A and A^T have the same eigenvalues. Therefore if the eigenvalues of A are in the left half plane, the eigenvalues of $-A^T$ will be on the right. It will be assumed throughout the thesis that the conditions above are satisfied, such that the solution to (1.1) exists.

2.3 Closed-form solutions

Several representations of the solution X to (1.1) in closed form exist [39]. We mention some of the main ones.

1. *Integral of resolvents.*

$$X = \frac{1}{4\pi^2} \int_{S_1} \int_{S_2} \frac{(\lambda I_n - A)^{-1} C (\phi I_m - B)^{-1}}{\lambda + \phi} d\phi \, d\lambda, \tag{2.7}$$

where the closed curves S_1 and S_2 respectively contain the spectra of A and B.

2. *Integral of exponentials.* Assuming that $\Lambda(A)$ and $\Lambda(B)$ are separated by a straight line, then

$$X = -\int_0^\infty e^{At} C e^{Bt} dt, \tag{2.8}$$

where e^{At} and e^{Bt} represent the matrix exponential of the matrices At and Bt respectively.

3. *Finite power sum.* Let $C = C_1 C_2^T$ and let a_m of degree m be the monic polynomial of smallest degree such that $a_m(A)C_1 = 0$. Similarly, let b_k of degree k be the smallest degree monic polynomial such that $b_k(B)C_2 = 0$, then

$$X = -\sum_{i=0}^{m-1} \sum_{j=0}^{k-1} \gamma_{i,j} A^i C B^j = -[C_1, AC_1, \ldots, A^{m-1}C_1](\gamma \otimes I) \begin{bmatrix} C_2^T \\ C_2^T B \\ \vdots \\ C_2^T B^{k-1} \end{bmatrix},$$

with γ the solution of the Sylvester equation if the coefficient matrices are given by the companion matrices of a_m and b_k and the right-hand side is given by $[1, 0, \ldots, 0]^T [1, 0, \ldots, 0]$.

Some early computational methods relied on these closed-form representations for solving the Sylvester equation. See [19] for a review of these computational methods. Despite the fact that these explicit forms are not commonly used to solve the Sylvester equation, it has inspired the development of many algorithms, with (2.8) in particular motivating the use of projection methods [52].

CHAPTER 3

Applications

The solution of linear matrix equations is important in signal processing, control theory, and systems engineering [60]. In this chapter, we review the most significant applications relating to Sylvester and Lyapunov equations. The applications discussed are from the governing fields of partial differential equations, model reduction, generalised eigenvalue problems, non-linear system analysis, and image processing.

First, consider the Poisson partial differential equation

$$U_{xx} + U_{yy} = f(x, y), \qquad U(\pm 1, 0) = U(0, \pm 1) = 0. \tag{3.1}$$

The discretisation of this differential operator (be it finite difference or spectral) on a square domain typically results in a Lyapunov equation

$$AX + XA^T = F, \qquad F_{ij} = f(x_i, y_j), \tag{3.2}$$

with X containing the solution values at the interior nodes of the discretisation grid, namely (x_i, y_j). This specific, rather simple, model problem is discussed in more detail in Section 4.5.3. This type of application can be generalised to more complex differential operators, with some resulting in Sylvester or even generalised Sylvester equations. In [15], a model problem for the Sylvester equation is derived in the solution of elliptic boundary value problems. As a final application to differential equations, we refer the reader to [16], where the Sylvester equation is used to implement implicit Runge-Kutta integration formulae.

Turning our attention to stability analysis and model reduction, and using the derivation from [60], consider the continuous-time linear system

$$\frac{dx(t)}{dt} = Ax(t) + C_1 u(t), \qquad y(t) = C_2^T x(t). \tag{3.3}$$

Here x is the model state, u is the input, y is the output and A, C_1 and C_2 are invariant under time. Using the matrices in (3.3), one defines the controllability and observability Gramians P and Q of the system as the solution of the Lyapunov equations [1]

$$AP + PA^T + C_1 C_1^T = 0, \qquad AQ + QA^T + C_2 C_2^T = 0. \tag{3.4}$$

These Gramians are used to describe how the energy is distributed over the coordinates of the state space [20]. If B_1 and B_2 have the same number of columns we can find the *cross-Gramian*, W, by solving a single Lyapunov equation of the form [18]

$$AW + WA^T + C_1 C_2^T = 0. \tag{3.5}$$

For more information about the *cross-Gramian* and its relation to the controllability and observability Gramians and Hankel singular values, the reader is referred to [17, 61]. A model problem coming from a continuous-time linear system is discussed in Section 6.5.

Another interesting application comes from the field of digital image processing [8]. In this application, a Sylvester equation is solved to recover an approximation of an image corrupted by noise. Suppose $f := \text{vec}(F)$ is a long vector containing all the pixel values of an image F. Define then the vector g as the image corrupted by white Gaussian noise such that $g = f + \eta$, with η being the noise vector with variance σ_η^2. We would like to approximate the original image by applying a linear filter L to g such that

$$\widehat{f} = Lg. \tag{3.6}$$

The linear filter of choice for this application is known as the Wiener filter [26, p. 374], defined as

$$L = \Phi_f(\Phi_f + \Phi_\eta)^{-1}, \tag{3.7}$$

with Φ_f and Φ_η the respective covariance matrices of f and η, such that (3.6) becomes

$$(I + \Phi_\eta \Phi_f^{-1})\widehat{f} = g. \tag{3.8}$$

If the variability of the image in the horizontal and vertical directions is unrelated and the noise is white and Gaussian with variance σ_η^2 (i.e., $\Phi_\eta = \sigma_\eta^2 I$), then (3.8) can be expressed as

$$(I + \sigma_\eta^2 \Phi_y^{-1} \otimes \Phi_x^{-1})\widehat{f} = g. \tag{3.9}$$

By definition of the Kronecker product, (3.9) can be rewritten as a Sylvester equation such that

$$\sigma_\eta^2 \Phi_y^{-1} \widehat{F} + \widehat{F}\Phi_x = G\Phi_x, \tag{3.10}$$

where the solution $\widehat{F}$ is a noise-free approximation to the original image F. It turns out that the covariance matrices can be expressed as $\Phi_z = \sigma_z^2 R_z$, where

$$R_z = \begin{bmatrix} 1 & \rho_z & \rho_z^2 & \cdots & \\ \rho_z & 1 & \rho_z & \ddots & \vdots \\ \rho_z^2 & \rho_z & \ddots & \ddots & \rho_z^2 \\ \vdots & \ddots & \ddots & 1 & \rho_z \\ & \cdots & \rho_z^2 & \rho_z & 1 \end{bmatrix}, \qquad z \in \{x, y\}, \tag{3.11}$$

and ρ_z and σ_z^2 represent, respectively, the adjacent element corellation and variance in the z-direction [8]. This matrix is particularly efficient to work with, since its inverse is known explicitly and moreover tridiagonal, namely

$$R_z^{-1} = -(1-\rho_z)^{-1}(1+\rho_z)^{-1} \begin{bmatrix} -1 & \rho_z & & & & \\ \rho_z & -d & \rho_z & & & \\ & \rho_z & -d & \rho_z & & \\ & & \ddots & \ddots & \ddots & \\ & & & \rho_z & -d & \rho_z \\ & & & & \rho_z & -1 \end{bmatrix}, \tag{3.12}$$

with $d = 1 + \rho_z^2$. This essentially means that a Sylvester equation with tridiagonal A and Toeplitz B has to be solved. This is considered as a model problem in Section 4.5.3 to test the

efficiency of the direct methods, since the matrices under consideration are not particularly large.

Another application comes from the generalised eigenvalue problem. The Lyapunov equation appears in the solution of the algebraic Riccati equation, which is used for the computation of invariant subspaces. A well-known solution method for the Riccati equation is the Newton-Kleinman iteration, requiring the solution of a Lyapunov equation at each iteration [7].

The efficient solution of the Sylvester and Lyapunov equations arising in these application is crucial, especially when the matrices are large. In the following chapters we compare existing solution methods for solving these systems, by using relevant model problems coming from the above-mentioned applications. We first consider the small scale case, where direct methods can be implemented in order to recover the solution.

CHAPTER 4

Direct methods for Sylvester systems

Contents

A method is referred to as direct if the desired solution is yielded by a finite number of operations [57]. The solution might contain the usual rounding errors, but if none are present, the exact solution is obtained. Direct methods are often preferred because of their robustness and predictable behaviour. The aim of this chapter is to determine the most efficient direct solver of (1.1) for dense and sparse systems respectively. We shall compare efficiency in terms of both the theoretical number of floating-point operations (flops) and the computational time needed to solve the system in practice. Note that the Schur and eigenvalue decomposition methods are technically not direct methods, but these will still be considered as direct methods for consistency in this section.

4.1 Solving $AX + XB + C = 0$ as a linear system

Recall from Section 2.1 that the Sylvester equation (1.1) is equivalent to a linear system in Kronecker form

$$\widetilde{A}x = c. \tag{4.1}$$

By definition, a solution to (4.1) exists and is unique if $\widetilde{A}$ is not singular, but the question remains whether or not this method is efficient. Solving a linear system for a solution vector x consists of three steps, namely

- LU factorisation
- Forward substitution

- Back substitution

The LU factorisation can be viewed as the matrix form of the process of Gaussian elimination, appearing in Figure 4.1. The blue blocks represent possible nonzero entries, and the grey blocks represent entries that have become zero.

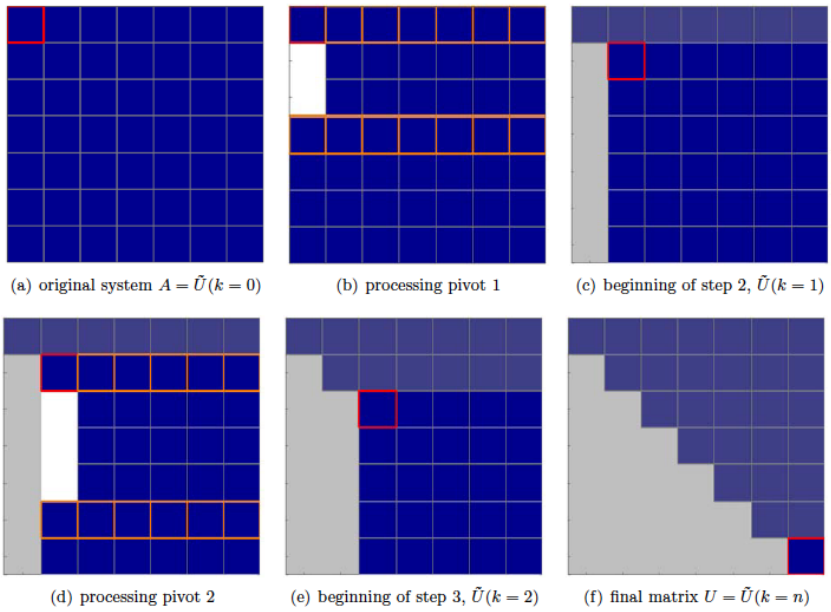


FIGURE 4.1: *A 9×9 visualisation of the process of Gaussian elimination [38].*

It turns out that an appropriate permutation of rows can increase the stability of the process of Gaussian elimination. This process is referred to as partial pivoting and can be represented as

$$P\widetilde{A}x = \widetilde{L}\widetilde{U}x, \tag{4.2}$$

with P the permutation matrix, which interchanges the rows. It is also possible that both the rows and the columns can be reordered in order to improve numerical stability even further. This is referred to as full pivoting and can be represented as

$$P\widetilde{A}Qx = \widetilde{L}\widetilde{U}x, \tag{4.3}$$

with Q a permutation matrix, interchanging the columns. In the special case where the coefficient matrix $\widetilde{A}$ is symmetric positive definite (i.e., all the eigenvalues are positive), $\widetilde{A}$ can be factorised instead as

$$\widetilde{A}x = \widetilde{L}\widetilde{L}^T x. \tag{4.4}$$

This is known as the Cholesky factorisation, which is typically more efficient and stable than the standard LU factorisation, and no pivoting is required.

This is the basis of the default algorithm implemented by *backslash* (denoted by `A\b`) in MATLAB. The algorithm follows the following general outline:

- If the matrix is not square, the least squares problem is solved.

- If the matrix is triangular, or a permutation of it, then forward or back substitution is performed.
- If the matrix is symmetric, a Cholesky factorisation is attempted. If successful, this is followed by forward and back substitution.
- If the matrix is nonsymmetric, or the Cholesky factorisation is unsuccessful, an LU factorisation (with pivoting) is computed. This is then followed by forward and back substitution.

In the case where the matrix $\widetilde{A}$ is sparse, MATLAB implements *sparse backslash*. The original design and implementation of sparse backslash were presented by Gilbert, Moler and Schreiber in 1992 [23]. The sparse matrix solver is outlined as follows:

- If the matrix is not square, the least squares problem is solved.
- If the matrix is triangular, or a permutation of it, a sparse triangular solve is performed. The sparse triangular solve follows the same concept as back substitution, except that it takes advantage of the zero values.
- If the matrix is symmetric, a symmetric approximate minimum degree (AMD) ordering [1] with permutation vector p is found (see Figure 4.2), after which a Cholesky factorisation of $A(p, p)$ is attempted. If successful, this is followed by forward and back substitution.
- If the matrix is nonsymmetric, or the Cholesky factorisation is unsuccessful, a column AMD ordering with permutation vector p is found, and an LU factorisation of $A(p, p)$ is now computed. This is then followed by forward and back substitution.

The reason for the reordering at the bottom left of Figure 4.2 is to minimise the number of nonzero values in the lower and upper triangular matrices after LU or Cholesky factorisation. In order to see this we include a visualisation of the Cholesky factorisation of the original sparse matrix without reordering in Figure 4.2.

The upper triangular matrix now contains 737 nonzero values, where the upper triangular Cholesky factor at the bottom right of Figure 4.2 contains only 506. This is not a big difference, but this example is based on an 80×80 matrix. This ratio gets bigger as the size of the matrix increases.

We now turn our attention to solution methods used when (1.1) is in Sylvester form.

4.2 The Bartels–Stewart method

The first numerically stable way to systematically solve (1.1), when not in Kronecker form, was introduced by Bartels and Stewart in 1972 [4]. Five decades later the method is still widely used.

The method is based on orthogonally reducing the matrices A and B to their real Schur forms (as defined on page ix), which in turn transforms equation (1.1) into a triangular system easily solved by substitution. The algorithm is visualised in Figure 4.3.

[1] An approximate minimum degree ordering performs a permutation of the columns and rows of a sparse symmetric matrix, in an attempt to make the respective Cholesky factor as sparse as possible

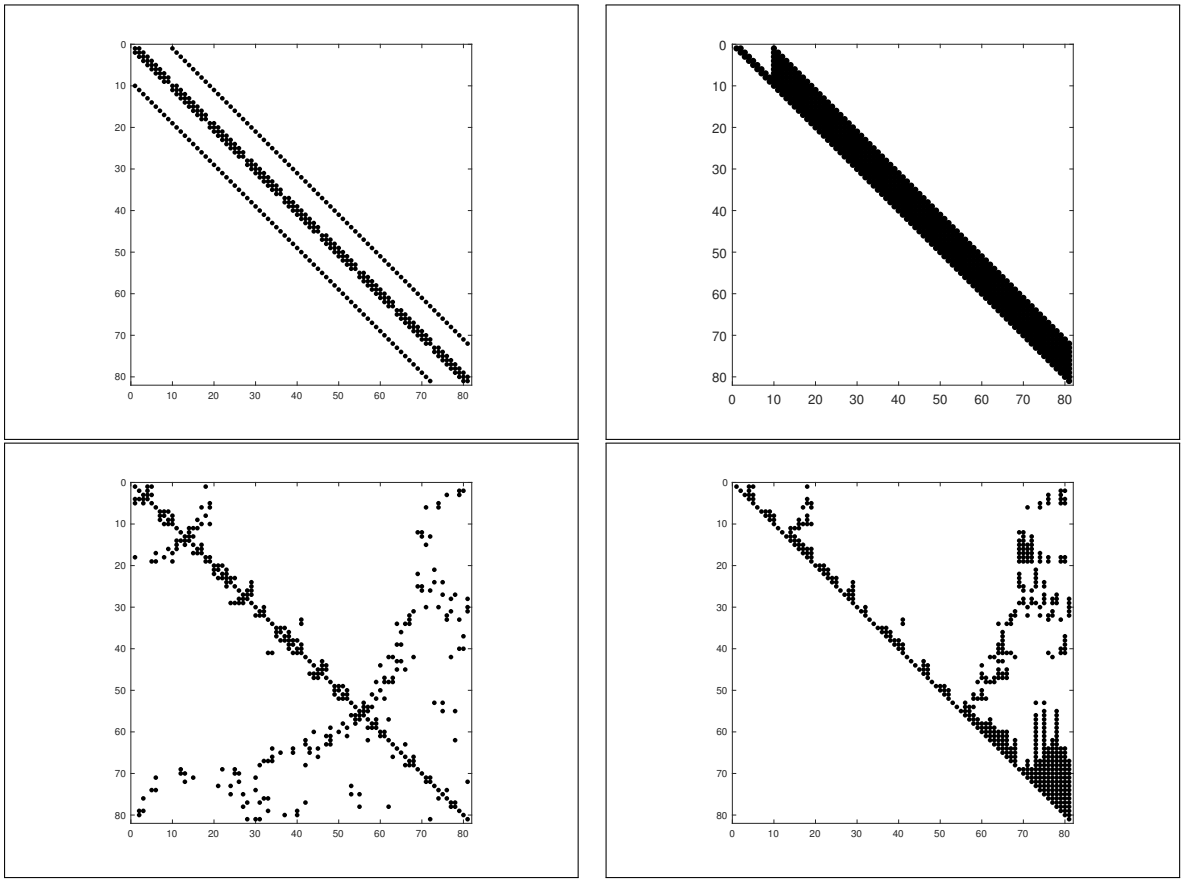


FIGURE 4.2: *Summary of the process of sparse backslash for a symmetric positive definite matrix. At the top left, the original sparse matrix A. At the bottom left, the matrix $A(p,p)$ with symmetric AMD ordering p, and the Cholesky factorisation of $A(p,p)$ at the bottom right. The figure at the top right depicts the Cholesky factorisation of the original image without reordering.*

The matrices A and B are transformed into their respective Schur forms by using the QR algorithm [25], such that

$$A = Q_1 U Q_1^T \qquad B = Q_2 R Q_2^T, \tag{4.5}$$

with both U and R in upper quasi-triangular form. If a matrix is in upper quasi-triangular form, the diagonal elements can be either 1×1 or 2×2 blocks, depending on whether the eigenvalues are real or complex. Figure 4.4 depicts the difference between these two cases.

It should be noted that the matrices A and B can also be reduced into complex Schur form. This means that the diagonal elements of the matrices will only be 1×1 blocks, but these elements may be complex in the case of complex eigenvalues.

The matrices A and B are now replaced by (4.5), which transforms (1.1) to

$$Q_1 U Q_1^T X + X Q_2 R Q_2^T = C. \tag{4.6}$$

Allowing $\widetilde{C} = Q_1^T C Q_2$, equation (4.6) is simplified to the triangular system

$$U\widetilde{X} + \widetilde{X} R = \widetilde{C}, \tag{4.7}$$

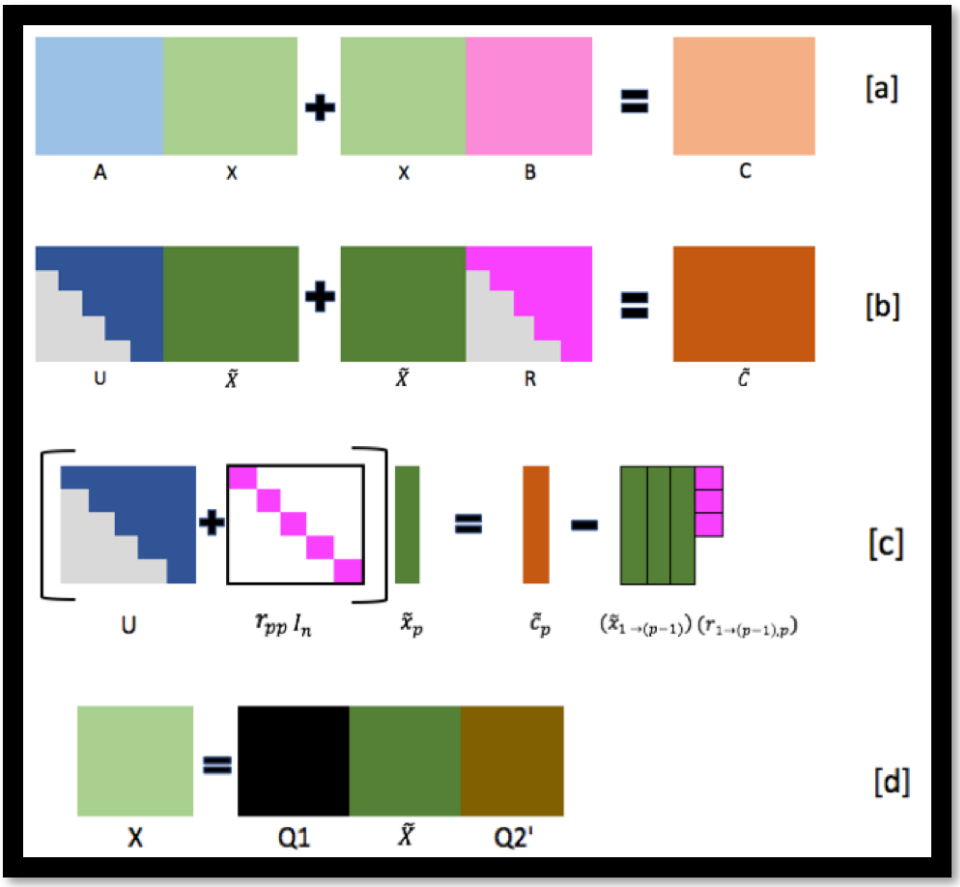


FIGURE 4.3: *A 5×5 visualisation of the Bartels–Stewart algorithm. From top to bottom,* ***[a]*** *represents (1.1) before the first step of Bartels–Stewart,* ***[b]*** *represents the triangular system described by (4.7),* ***[c]*** *shows how (4.9) can be used to recover the entries of $\widetilde{X}$ and* ***[d]*** *shows how X is recovered from $\widetilde{X}$.*

where $\widetilde{X} = Q_1^T X Q_2$.

Once $\widetilde{X}$ is solved, the solution X can readily be recovered by $X = Q_1 \widetilde{X} Q_2^T$. Since it is assumed throughout the paper that $\Lambda(A) \cap \Lambda(-B) = \emptyset$, it is clear that (4.7) will have a unique solution. The eigenvalues of A and B are contained on the block diagonals of U and R respectively. Therefore, since A and $-B$ have no common eigenvalues, U and $-R$ will have no common eigenvalues, i.e., $\Lambda(U) \cap \Lambda(-R) = \emptyset$.

For the solution of $\widetilde{X}$, we first consider the case where A and B have real eigenvalues, hence the case where there are only 1×1 blocks on the diagonals of the matrices U and R. If we denote the matrices $\widetilde{X}$ and $\widetilde{C}$ in (4.7) by their columns as

$$\widetilde{X} = [\tilde{x}_1\ \tilde{x}_2 \dots\ \tilde{x}_n] \qquad \widetilde{C} = [\tilde{c}_1\ \ \tilde{c}_2 \dots\ \ \tilde{c}_n] \tag{4.8}$$

and the matrix R by its entries $R = [r_{ij}]$ with $i, j = 1 \dots n$, then (4.7) can be expressed as

$$(U + r_{pp} I_n)\tilde{x}_p = \tilde{c}_p - \sum_{i=1}^{p-1} \tilde{x}_i r_{ip}, \quad p = 1, 2 \dots n, \tag{4.9}$$

by comparison of columns. The solution matrix $\widetilde{X}$ can now be formed using (4.9), by means of n linear solves with an upper triangular matrix; see Figure 4.3(c).

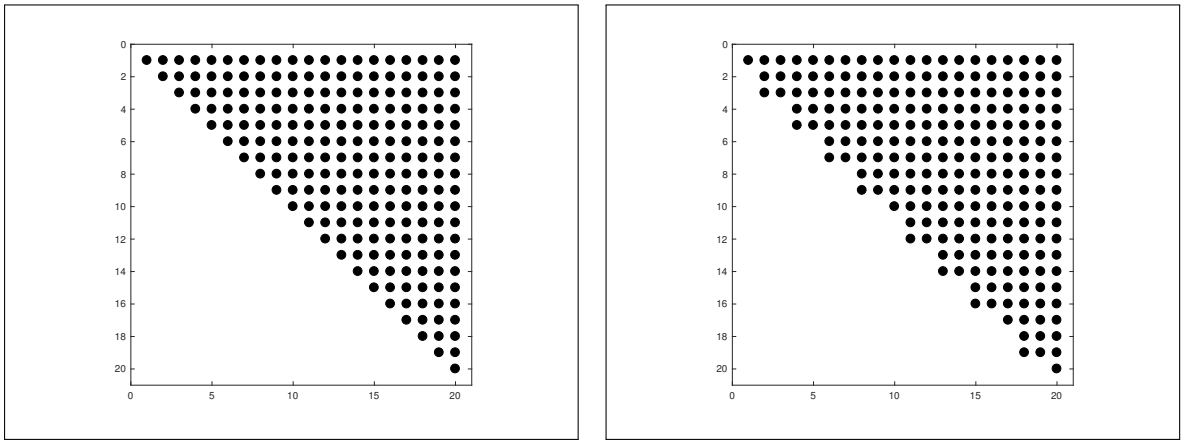


FIGURE 4.4: *Spy plots depicting two different upper quasi-triangular matrices. On the left a normal upper triangular matrix resulting from either a complex Schur decomposition or from a real Schur decomposition of a matrix with real eigenvalues. On the right a block upper triangular matrix resulting from the real Schur decomposition of a matrix with complex eigenvalues.*

In the case where 2×2 blocks are contained on the diagonal (i.e., $r_{p+1,p} \neq 0$ for some values of p), a bit more work is required. The columns $\tilde{x}_p$ and $\tilde{x}_{p+1}$ are now found simultaneously by solving the $2n \times 2n$ system

$$\begin{pmatrix} U + r_{pp}I_n & r_{mp}I_n \\ r_{pm}I_n & U + r_{mm}I_n \end{pmatrix} \begin{pmatrix} \tilde{x}_p \\ \tilde{x}_m \end{pmatrix} = \begin{pmatrix} \tilde{c}_p \\ \tilde{c}_m \end{pmatrix} - \sum_{i=1}^{p-1} \begin{pmatrix} \tilde{x}_i r_{ip} \\ \tilde{x}_i r_{im} \end{pmatrix}, \tag{4.10}$$

with $m = p+1$. It should be noted that the system of equations in (4.10) can be reordered by the permutation $(1, n+1, 2, n+2, \ldots, n, 2n)$ in order to form a banded system that is solved in $\mathcal{O}(n^2)$ flops. Further discussions about the operation counts of the algorithms derived in this chapter appear in Section 4.5. The method described in this section is summarised in Algorithm 4.1.

Algorithm 4.1: Bartels–Stewart

input : $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{n\times m}$
output: $X \in \mathbb{R}^{n\times m}$, the solution of (1.1)

1 Compute the Schur decompositions $A = Q_1 U Q_1^T$ and $B = Q_2 R Q_2^T$;
2 Compute $\widetilde{C} = Q_1^T C Q_2$;
3 **if** $r_{p+1,p} = 0$ *for all* p **then**
4 | Find $\widetilde{X}$ using (4.9)
5 **else**
6 | Find $\widetilde{X}$ using (4.10)
7 **end**
8 Compute $X = Q_1 \widetilde{X} Q_2^T$;

Further improvements on the substitution methods are made in [62]. One of the most effective improvements to the substitution step was introduced by Jonsson and Kågström in 2002 [36, 37] for the case where A is much larger than B. Consider the system already in triangular form, as in (4.7), the larger, upper triangular matrix U can then be split into blocks, such that (4.7) is written as

$$\begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix} \begin{pmatrix} \widetilde{X}_1 \\ \widetilde{X}_2 \end{pmatrix} + \begin{pmatrix} \widetilde{X}_1 \\ \widetilde{X}_2 \end{pmatrix} B = \begin{pmatrix} \widetilde{C}_1 \\ \widetilde{C}_2 \end{pmatrix}, \tag{4.11}$$

with U_{11} and U_{22} being upper triangular matrices of size $n/2$. The latter equation, $U_{22}\widetilde{X}_2 + \widetilde{X}_2 B = \widetilde{C}_2$, represents a Sylvester equation of a smaller scale, which can once again be split using the blocks of U_{22}. The solution $\widetilde{X}_2$ is found recursively in this manner. The first solution block of $\widetilde{X}$ is then recovered by recursively solving with the updated right-hand side in the first equation of (4.11). More implementation details are given in [36].

The most well known modification to the Bartels–Stewart method, especially for the case where one matrix is larger than the other, was brought about by Golub, Nash and Van Loan in 1979 [24]. This method is discussed in the following section.

4.3 The Hessenberg–Schur method

The Hessenberg–Schur method is also a transformation method used for the direct solving of the Sylvester equation. The key difference between this method and the Bartels–Stewart algorithm, described in the preceding section, lies in the name. Instead of decomposing both A and B into Schur form, only the latter will be decomposed into Schur form, and A (the larger of the two matrices) will be decomposed into upper Hessenberg form. This means that one can find a matrix $H = Q_3^T A Q_3$, with entries h_{ij}, such that $h_{ij} = 0$ for all $i > j + 1$. The shape of an upper Hessenberg matrix is shown in Figure 4.5. The matrix A can be orthogonally reduced

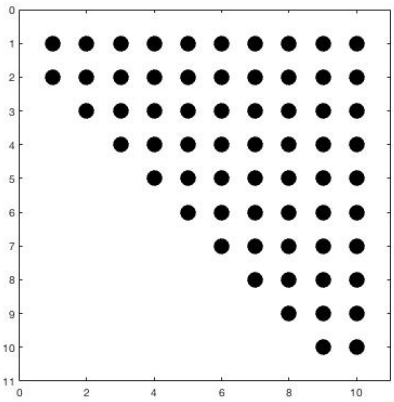


FIGURE 4.5: *Spy plot representation of an upper Hessenberg matrix.*

to upper Hessenberg form using Householder matrices, or some other orthogonal transformation [25]. The motivation behind this adaptation is the fact that a Hessenberg reduction requires fewer operations than a Schur decomposition. In fact, the Hessenberg decomposition is a direct algorithm, where the Schur decomposition is iterative. Unfortunately, the upper Hessenberg structure could be at the expense of efficient back substitution. This will be discussed in more detail in Section 4.5.

In order to keep the notation standard, we once again denote the Schur decomposition of B by $B = Q_2 R Q_2^T$ and let $A = Q_3 H Q_3^T$. After the transformations have been made, a 'nearly-triangular' system remains, similar to equation (4.7). This equation is expressed as

$$H\widetilde{X} + \widetilde{X}R = \widetilde{C}. \tag{4.12}$$

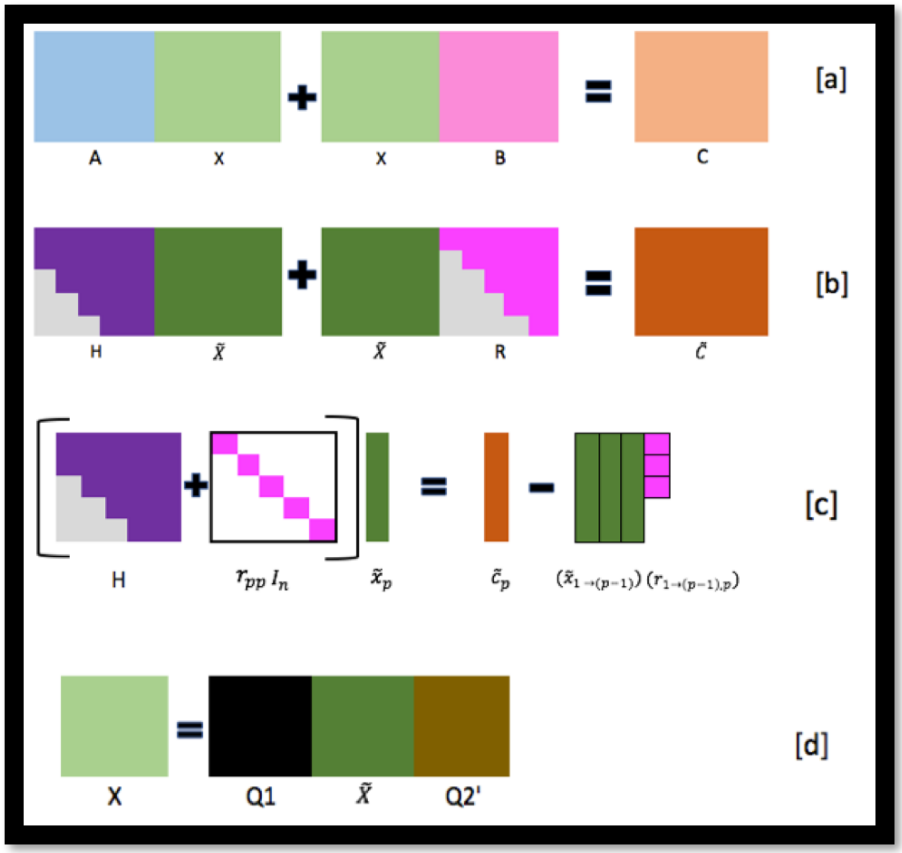


FIGURE 4.6: *A 5×5 visualisation of the Hessenberg–Schur Algorithm. From top to bottom,* ***[a]*** *represents (1.1) before the first step of Hessenberg–Schur,* ***[b]*** *represent the nearly triangular system described by (4.12),* ***[c]*** *shows how (4.13) can be used to recover the entries of* $\widetilde{X}$ *and* ***[d]*** *shows how* X *is recovered from* $\widetilde{X}$*. This visualisation can be followed in the description of the algorithm. This figure shows how similar this algorithm is to the Bartels–Stewart algorithm.*

The substitution method to find $\widetilde{X}$ is similar to (4.9) and (4.10) in the respective cases. In particular, when $r_{p+1,p} = 0$ for all p, the matrix $\widetilde{X}$ is found by solving

$$(H + r_{pp}I_n)\tilde{x}_p = \tilde{c}_p - \sum_{i=1}^{p-1} \tilde{x}_i r_{ip}, \quad p = 1, 2 \dots n. \tag{4.13}$$

In the case where $r_{p+1,p} \neq 0$ for some values of p, $\widetilde{X}$ is found by solving

$$\begin{pmatrix} H + r_{pp}I_n & r_{mp}I_n \\ r_{pm}I_n & H + r_{mm}I_n \end{pmatrix} \begin{pmatrix} \tilde{x}_p \\ \tilde{x}_m \end{pmatrix} = \begin{pmatrix} \tilde{c}_p \\ \tilde{c}_m \end{pmatrix} - \sum_{i=1}^{p-1} \begin{pmatrix} \tilde{x}_i r_{ip} \\ \tilde{x}_i r_{im} \end{pmatrix}, \tag{4.14}$$

with $\widetilde{X} = [\tilde{x}_1\ \tilde{x}_2 \dots\ \tilde{x}_n]$ and $\widetilde{C} = [\tilde{c}_1\ \tilde{c}_2 \dots\ \tilde{c}_n]$. Once again the system of equations can be reordered by the permutation $(1, n+1, 2, n+2, \dots, n, 2n)$ in order to form a banded system that is solved in $\mathcal{O}(n^2)$ flops.

Note that, in the case of the Lyapunov equation, only one Schur decomposition is needed for both A and A^T, when using Bartels–Stewart, but it will still be necessary to compute both a

Hessenberg and a Schur decomposition for Hessenberg–Schur. The Hessenberg–Schur method will therefore not be advantageous for this case. The method is summarised in Algorithm 4.2.

The Bartels–Stewart and Hessenberg–Schur methods form the basis for the direct solving of the Sylvester equation by software. In MATLAB, the Sylvester equation is solved by the function *lyap.m*. MATLAB calls the SLICOT subroutine [66]. The algorithms are also included in LAPACK [11].

Algorithm 4.2: Hessenberg–Schur

input : $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{n\times m}$
output: $X \in \mathbb{R}^{n\times m}$, the solution of (1.1)
Compute the Hessenberg reduction $A = Q_3 H Q_3^T$ and the Schur decomposition $B = Q_2 R Q_2^T$;
Compute $\widetilde{C} = Q_3^T C Q_2$;
if $r_{p+1,p} = 0$ *for all* p **then**
  Find $\widetilde{X}$ using (4.13)
else
  Find $\widetilde{X}$ using (4.14)
end
Compute $X = Q_3 \widetilde{X} Q_2^T$;

4.4 Eigenvalue method

The final method discussed in this chapter is also a transformation method. The key difference here is that an eigenvalue decomposition is used instead of a Schur or Hessenberg decomposition, provided the matrices A and B are diagonalisable. If a matrix is symmetric its eigenvalue decomposition is equivalent to a Schur decomposition, then this method could be very efficient. The motivation behind this is that an eigenvalue decomposition turns the Sylvester equation into a diagonal system for which an explicit solution can be found. This method has had little attention in the literature since the 1980's, due to stability issues arising from the fact that the matrices containing the eigenvectors are not orthogonal, unless the coefficient matrices are symmetric. We consider the method nevertheless.

Suppose the matrices A and B in (1.1) are diagonalisable, such that

$$A = P_1 \Lambda P_1^{-1}, \quad B = P_2 D P_2^{-1}, \tag{4.15}$$

with $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2 \dots \lambda_n)$ and $D = \operatorname{diag}(d_1, d_2 \dots d_n)$. Substituting (4.15) into (1.1), the Sylvester system is transformed such that

$$P_1 \Lambda P_1^{-1} X + X P_2 D P_2^{-1} = C. \tag{4.16}$$

After simplification, a diagonal system remains such that

$$\Lambda \widetilde{X} + \widetilde{X} D = P_1^{-1} C P_2 \tag{4.17}$$

with $\widetilde{X} = P_1^{-1} X P_2$.

The solution to (4.17) can now be expressed as

$$\widetilde{X} = L \circ (P_1^{-1} C P_2), \tag{4.18}$$

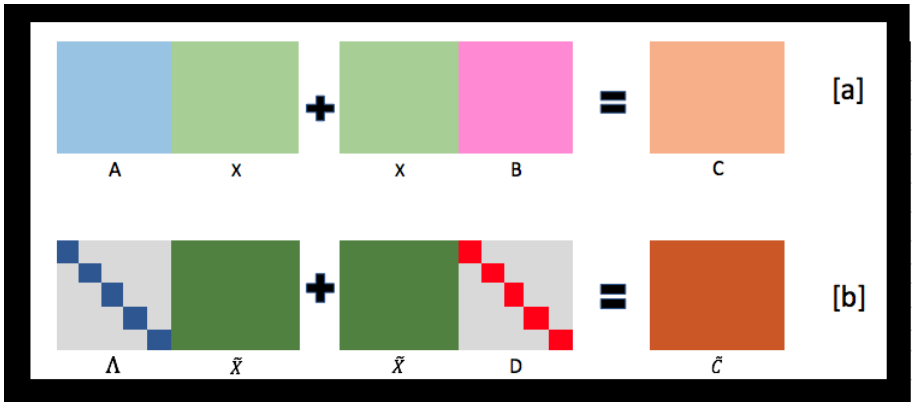


FIGURE 4.7: *A 5×5 visualisation of the eigenvalue method. From top to bottom,* ***[a]*** *represents (1.1) before the first step of the Eigenvalue method and* ***[b]*** *represent the diagonal system described by (4.17). An explicit solution for X can now be found due to this diagonal structure, using (4.19).*

with $(A \circ B)_{ij} = A_{ij}B_{ij}$ and $L_{ij} = \frac{1}{\lambda_i + d_j}$. Therefore, an explicit solution to (1.1) is given by

$$X = P_1 \left[L \circ (P_1^{-1} C P_2) \right] P_2^{-1}. \tag{4.19}$$

The matrices containing the eigenvectors and eigenvalues can therefore simply be substituted into (4.19) for the solution to be obtained. In the case where A and B are symmetric, (4.19) will be simplified since $P_1^{-1} = P_1^T$ and $P_2^{-1} = P_2^T$.

4.5 Comparison of methods

The preceding methods will be compared for sparse, banded systems and dense systems separately. In both sections, the methods will be compared by an elementary algebraic operation count, after which the methods are compared by actual computational performance.

4.5.1 Dense systems

Consider the case where the matrices $\widetilde{A}$, A and B are all dense, which by our definition refers to matrices not having enough zero values such that special techniques can be used to take advantage of them. The operation counts for solving the linear system (4.1) is calculated first. This is based on the operation counts for the process of Gaussian elimination, since the LU factorisation described in Section 4.1 is the matrix representation of Gaussian elimination. The reader is reminded that a flop is defined here as any algebraic operation $(+, -, \times, \div)$. Flop counts are merely an indication of how a method should perform numerically, but actual numerical performance is influenced by many other factors, such as memory access, when implemented.

Suppose the matrix $\widetilde{A}$ is of size N. The first step of the solution process is Gaussian elimination. Eliminating the elements of the first column requires $2N^2$ flops, after which the same process is applied to the remaining $(N-1) \times (N-1)$ system and so on, until the final 2×2 system is reached. The total number of flops for Gaussian elimination is therefore

$$2N^2 + 2(N-1)^2 + \cdots + 2 \cdot 3^2 + 2 \cdot 2^2 \sim \frac{2}{3}N^3, \qquad N \to \infty. \tag{4.20}$$

Part of the Gaussian elimination process is updating the right-hand side (equivalent to forward substitution). This requires 2 operations per row, one for multiplication and one for addition, therefore approximately $2N$ per step, keeping in mind that one fewer row is used for each step. The total number of flops for updating the right hand side is therefore

$$2N + 2(N-1) + \cdots + 2 \cdot 3 + 2 \cdot 2 \sim N^2, \tag{4.21}$$

but since this will make such a small difference in comparison to the $\frac{2}{3}N^3$, when N increases, it becomes negligible.

The final step is back substitution. The recovery of the k^{th} element of x requires $2(N-k)$ multiplication-subtraction pairs and one division. The total number of flops for back substitution is therefore

$$1 + (1+2) + \cdots + (1 + 2(N-1)) \sim N^2, \tag{4.22}$$

which also becomes negligible in comparison to the N^3 term when N increases. The total number of flops for Gaussian elimination with forward and back substitution, for a matrix of size N, is summarised in Table 4.1.

TABLE 4.1: *Total flops for solving a dense linear system of size N*

Step	**Number of flops**
Gaussian Elimination	$\frac{2}{3}N^3$
Forward substitution	N^2
Back substitution	N^2

Notice that in these specific applications, where the linear system arises from the Kronecker product of an $n \times n$ and $m \times m$ matrix, the matrix is of size $nm \times nm$. If $n = m$, then $N = n^2$, and the total flops for solving a linear system is $\mathcal{O}(n^6)$. The question remains whether it is more efficient to solve the system in Sylvester form. The Bartels–Stewart method is investigated first.

The first step of the Bartels–Stewart method is the Schur decomposition of the matrices A and B. This will require approximately $25n^3$ and $25m^3$ flops, respectively, to compute both the eigenvalues and the Schur vectors [25, p. 359], ultimately being the most expensive step in the algorithm. Next, the right-hand side should be updated in order to form equation (4.7). This results in left multiplication of the $n \times m$ matrix by a $n \times n$ matrix and right multiplication by a $m \times m$ matrix, requiring $n^2m + nm^2$ flops. The next step is the substitution represented by equations (4.9) and (4.10). Calulating the right hand side of these equations requires $\frac{1}{2}nm^2$ flops, after which the left hand side is solved in a total of $\frac{1}{2}n^2m$ flops. Finally, the solution X is recovered from $\widetilde{X}$, with a similar left-right multiplication to step 2, therefore this step will also require $n^2m + nm^2$ flops. The total operation counts for the Bartels–Stewart method is summarised in Table 4.2.

The Hessenberg–Schur method is very similar in approach to the Bartels–Stewart method and therefore expected to be very similar in operation counts. We will, therefore, only highlight where it differs from the Bartels–Stewart method. The first difference in required flops comes when the larger matrix A is only reduced to Hessenberg form. This is indeed cheaper, requiring $\frac{14}{3}n^3$ flops [25, p. 345] compared to the $25n^3$ required for a Schur decomposition. The other difference comes in step 3, when the left hand side is now solved with an upper Hessenberg matrix in stead of an upper triangular matrix. This requires $3n^2m$ flops compared to the $\frac{1}{2}n^2m$

TABLE 4.2: *Total flops for solving a dense system in Sylvester form by the Bartels–Stewart method*

STEP	Number of flops
Reduce $A \in \mathbb{R}^{n\times n}$ to UT form	$25n^3$
Reduce $B \in \mathbb{R}^{m\times m}$ to UT form	$25m^3$
Update RHS	$n^2m + nm^2$
Back Substitution	$\frac{1}{2}n^2m + \frac{1}{2}nm^2$
Recover X	$n^2m + nm^2$

for Bartels–Stewart [24]. The required flops for the Hessenberg–Schur method is summarised in Table 4.3, with the steps differing from Bartels–Stewart highlighted in red. As can be deduced from Tables 4.2 and 4.3, the flop count of Hessenberg–Schur in comparison to Bartels–Stewart becomes better as the ratio n/m increases, with n the size of the larger matrix. Advantage will therefore be drawn from Hessenberg–Schur when the matrices differ in size.

TABLE 4.3: *Total flops for solving a dense system in Sylvester form by the Hessenberg–Schur method*

STEP	Number of flops
Reduce $A \in \mathbb{R}^{n\times n}$ to UH form	$\frac{14}{3}n^3$
Reduce $B \in \mathbb{R}^{m\times m}$ to UT form	$25m^3$
Update RHS	$n^2m + m^2n$
Back Substitution	$3n^2m + \frac{1}{2}nm^2$
Solve for X	$n^2m + nm^2$

Finally, we consider the operation counts for the eigenvalue method with dense matrices A and B. The method only consists of two steps, with the first being the eigenvalue decompositions and the second being the multiplication of (4.19). The number of operations needed for producing all the eigenpairs differ for symmetric and nonsymmetric matrices, but this does not come as a surprise, since it was mentioned in Section 4.4 that the eigenvalue decomposition of a symmetric matrix is merely a Schur decomposition. The Schur decomposition of a symmetric matrix only requires $9n^3$ flops [25, p. 421]. The exact complexity of computing both the eigenvalues and eigenvectors of a nonsymmetric matrix is still an open problem according to Golub and Van Loan [25]. It is mentioned in [45] that the computation of a basis for the eigenspace when the eigenvalues are known requires a complexity of $\mathcal{O}(n^3)$. Therefore if the eigenvalues of a nonsymmetric matrix are calculated in $25n^3$ operations using the Schur decomposition, it is enough for our application to know that the computation of the eigenvalues and eigenvectors of a nonsymmetric matrix will require more then $25n^3$ operations.

The second step is the solving of (4.19), which consists of multiplications as well as linear solves. The matrix-matrix multiplications each require n^3 flops. The dot product requires n^2 flops. The operation counts for a linear solve with a right-hand side of size $n \times 1$ is summarised in Table 4.1, but in this case the right-hand sides have dimension $n \times n$. One would expect that this will result in $\mathcal{O}(n^4)$ operations, but only one Gaussian elimination is needed, after which n backward substitutions are performed instead of 1. This results in the linear solves each requiring $\frac{5}{3}n^3$ flops. Adding all the solves and multiplications together, the solution step requires $\frac{16}{3}n^3$ flops. The flop counts for the eigenvalue method are summarised in Table 4.4. The table represents the most general case. There are special cases where the flop count can decrease drastically. One of these will be discussed in the computational comparison section.

TABLE 4.4: *Total flops for solving a Sylvester system using the eigenvalue method*

Step	Number of flops
Eigenvalue Decomposition of symmetric A	$9n^3$
Eigenvalue Decomposition of nonsymmetric A	$> 25n^3$
Eigenvalue Decomposition of symmetric B	$9n^3$
Eigenvalue Decomposition of nonsymmetric B	$> 25n^3$
Multiplication and solving to find the solution	$\frac{16}{3}n^3$

4.5.2 Sparse, banded systems

Consider now the case where A and B are sparse and banded with respective bandwidths b_A and b_B. This results in $\widetilde{A}$ also being banded, with the bandwidth b dependent on b_A and b_B. Once again the process of Gaussian elimination and back substitution is analysed first. Recall that the first step of eliminating the entries of the first column required $2N^2$ flops for a dense matrix. This number now reduces to approximately $2(b+1)^2$ flops per column since the elimination is only applied to the first b rows. This means that the process of Gaussian elimination applied to N columns will require $2N(b+1)^2$ flops. Note that this has now decreased from $\mathcal{O}(N^3)$ to $\mathcal{O}(N)$ for the sparse, banded case. The updating of the right-hand side is also only performed on b rows at the same time, requiring a maximum of $2N(b+1)$ flops. Finally, the process of back substitution requires twice the number of flops as there are nonzero values remaining [38]. Therefore, in the worst case of a densely banded upper triangular matrix remaining, there will be a total of $N(b+1)$ nonzero values, resulting in $2N(b+1)$ flops needed for back substitution. A summary of the required flops for Gaussian elimination and back substitution of a sparse, banded system appears in Table 4.5. Note that in the case where $\widetilde{A}$ is in Kronecker form, $\widetilde{A} \in \mathbb{R}^{N\times N}$,

TABLE 4.5: *Sparse linear system flops*

Step	Number of flops
Gaussian Elimination	$2N(b+1)^2$
Update right hand side	$2N(b+1)$
Back substitution	$2N(b+1)$

where $N = n^2$. This is the operation count for sparse Gaussian elimination, but this is not a fair prediction for *sparse backslash*, due to the AMD reordering and other improvements mentioned in Section 4.1. Heath [27] predicts $\mathcal{O}(N^{1.5})$ for *sparse backslash*, therefore $\mathcal{O}(n^3)$ for the matrix in Kronecker form.

In the case where the system is in Sylvester form, we assume both matrices A and B are sparse and banded. Unfortunately, all three transformation methods take a full matrix as input[2], resulting in the method being unable to take advantage of the sparsity of the matrices A and B. This results in the algebraic operation counts being very similar for the sparse and dense case for these three methods. It should be mentioned that there are some special examples of tridiagonal symmetric matrices where the operation counts for the eigenvalue method can be decreased to $\mathcal{O}(n^2 \log n)$. This is discussed in Section 4.5.3.

[2]The eigenvalue decomposition function *eig* used by MATLAB can actually take a sparse matrix as input, but returns only the eigenvalues, not the eigenvectors. This is therefore not applicable to the eigenvalue method, since the entire eigenvalue decomposition is required.

4.5.3 Numerical comparison

The aim of this section is to compare the described methods to each other numerically, using four different model problems. All model problems used will result in either Lyapunov equations or Sylvester equations where $n = m$. The first model problem considered for the computational comparison of the described methods will be the discretisation of the well-known Poisson equation on the unit square

$$U_{xx} + U_{yy} = f(x, y), \qquad (x, y) \in [-1, 1], \tag{4.23}$$

with homogeneous Dirichlet boundary conditions, such that

$$U(\pm 1, 0) = U(0, \pm 1) = 0, \tag{4.24}$$

and the right-hand side f chosen such that the reference solution $U(x, y)$ is given by

$$U(x, y) = (1 - x^2)(1 - y^2)\cos(20xy), \tag{4.25}$$

as visualised in Figure 4.8. The most common approach is the discretisation by a five-point

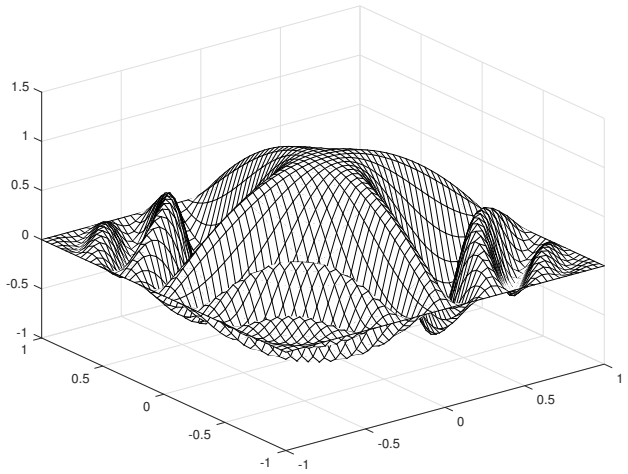


FIGURE 4.8: *The reference solution* $U(x, y) = (1 - x^2)(1 - y^2)\cos(20xy)$

stencil on an $(n + 1) \times (n + 1)$ equispaced grid, therefore this will be considered first. The discretisation results in a sparse Lyapunov equation

$$AX + XA = F, \qquad A = \frac{-1}{h^2}\begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}, \tag{4.26}$$

with $h = \frac{1}{n+1}$ and $F \in \mathbb{R}^{n \times n}$ representing the evaluation of $f(x, y)$ at the interior grid points. The matrix X represents the solution values at the interior nodes of the equispaced discretisation grid. Therefore, Figure 4.8 can also be interpreted as a visualisation of the values of the matrix X on $[-1, 1] \times [-1, 1]$ when n is large enough.

As with any Sylvester or Lyapunov equation, (4.26) can be represented using the Kronecker product, resulting in the linear system

$$\widetilde{A}x = h^2 c. \tag{4.27}$$

Here, $\widetilde{A}$ is a sparse, banded matrix ($b = n$) with structure as shown in Figure 4.9. The diagonal entries are -4 and all other entries are 1. The right-hand side c is now a vector chosen such that (4.25) is satisfied. The dimension of the linear system (4.27) is $N = n^2$, where the Lyapunov

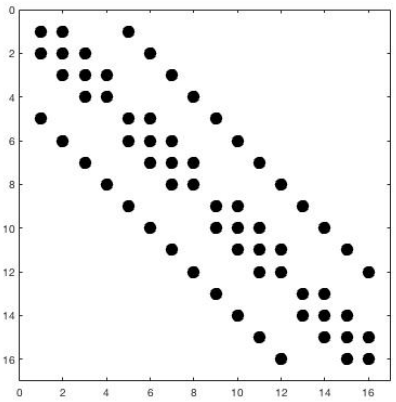


FIGURE 4.9: *Structure of the sparse, banded matrix* $\widetilde{A}$

system (4.26) is of dimension n. This is solved using *sparse backslash*, as described in Section 4.1.

As mentioned in the previous section, there are special cases where the eigenvalue method can be used to do computations in Sylvester form in $\mathcal{O}(n^2 \log n)$ operations. The key element in the decrease in operations is the fact that the eigenvalues of A are known explicitly, resulting in the elimination of the eigenvalue decomposition. For this specific example the eigenvalues are explicitly given by $\lambda_k = -\frac{4}{h^2}\sin^2(\frac{\pi k}{2n})$ for $1 \leq k \leq n$ [40]. The matrix of eigenvectors, Q, is the normalised discrete sine transform matrix [40]. Since A is symmetric, Q is orthogonal. This simplifies (4.19) to

$$X = Q\left(L \circ (Q^T C Q)\right) Q^T, \qquad L_{jk} = \frac{1}{\lambda_j + \lambda_k}. \tag{4.28}$$

The matrix Q is also symmetric in this case. Since Q is the discrete sine transform matrix, all matrix-vector products with Q can be done in $\mathcal{O}(n \log n)$ operations using the Fast Fourier Transform (FFT), resulting in a total of $\mathcal{O}(n^2 \log n)$ operations for solving (4.26). For further implementation details the reader is referred to [28]. Note that this is merely a description of what could be done in these special cases, but for the numerical comparison shown in Figure 4.10, the time needed for calculating the eigenvalue decomposition was still taken into account.

The methods were timed in MATLAB and compared to the built-in MATLAB function *lyap.m*. The figures confirm the predicted asymptotic order of complexity $\mathcal{O}(n^3)$ for the Bartels–Stewart and Hessenberg–Schur methods. Both Bartels–Stewart and Hessenberg–Schur only require one Schur decomposition in this example, since the matrices are symmetric and already in Hessenberg form. Despite this, Bartels–Stewart is still the more efficient of the two. This comes down to the substitution step being trivial for the diagonal matrices resulting from the Schur decomposition in Bartels–Stewart. It is interesting that *sparse backslash* does not reach the order of 3, predicted

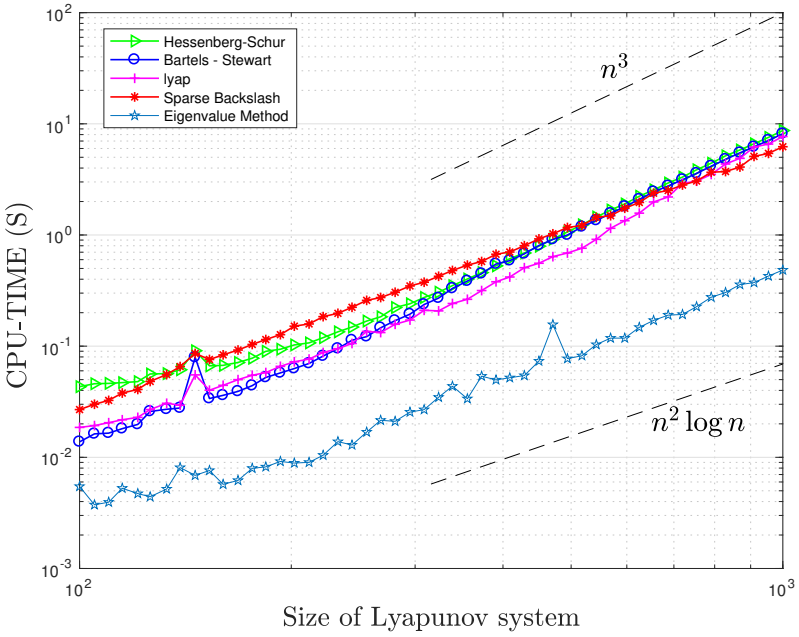


FIGURE 4.10: *Comparison in computational time for solving the sparse Lyapunov system (4.26), using the described methods as well as lyap and the fast Fourier solver. The methods were timed for matrices up to dimension 1200 in Sylvester form (1440000 in Kronecker form). The fast Fourier method dominates in efficiency, but only works for very specific examples.*

by Heath [27]. In the book by George and Liu [22], they describe the operation counts for solving by nested dissection (very closely related to the AMD reordering used by *sparse backslash*) by $\frac{829}{84}n^3 + \mathcal{O}(n^2 \log n)$. The $n^2 \log n$ term comes from the number of nonzero values remaining after reordering. By the computational results it seems that *sparse backslash* increases like $\mathcal{O}(n^2 \log n)$. We suspect that this could be due to the constant of the $n^2 \log n$ term in the prediction by George and Liu being so large that the n^3 term is dominated by the $\mathcal{O}(n^2 \log n)$ term, until n becomes very large. As n increases this method is more efficient than Bartels–Stewart and Hessenberg–Schur and also surpasses the Matlab function *lyap*. The eigenvalue method is by far the fastest method for this example, and since the coefficient matrix A is symmetric it should also be stable. This is checked by plotting the relative residuals for the respective methods in Figure 4.11. As predicted, the eigenvalue method is just as stable as Bartels–Stewart and Hessenberg–Schur for this example.

Notice the $\mathcal{O}(n^2 \log n)$ behaviour of the eigenvalue method, after a growth of $\mathcal{O}(n^3)$ has been predicted. We conclude here that the size of the matrices considered here are relatively small and that the $\mathcal{O}(n^3)$ growth should be reached asymptotically. The experiment was also carried out using the known eigenvalues and the FFT solver. For the size of matrices considered the improvement in computational time and stability is minimal and, therefore, not necessary to add to Figures 4.10 and 4.11.

We now turn our attention to the case where the discretisation will result in a set of dense matrices. This is done by using Chebyshev differentiation matrices in order to obtain a pseudo-spectral discretisation of (4.23), satisfying (4.25). Details on how to obtain the differentiation matrices

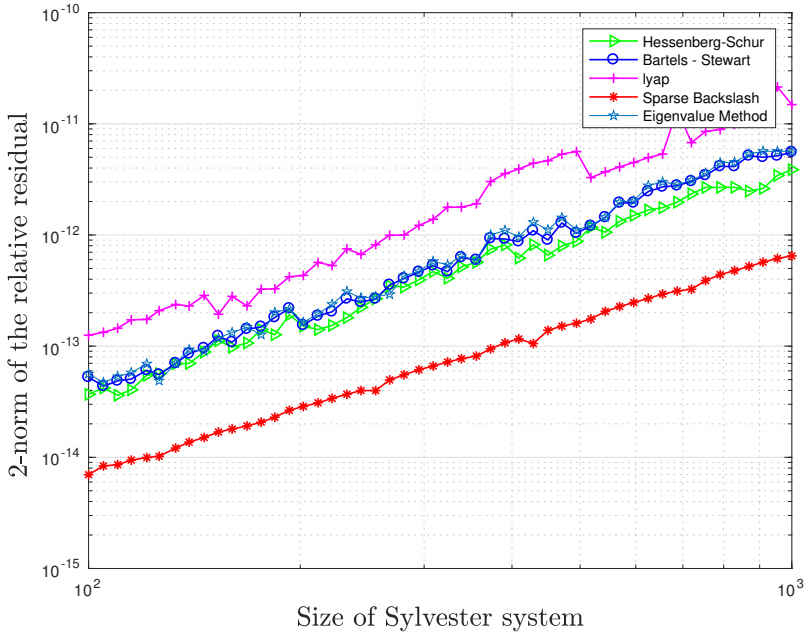


FIGURE 4.11: *Relative residual for the described methods used to solve (4.26). It can be seen that all the methods produce the correct solution and that sparse backslash is the most accurate method.*

can be found in [65]. The discretisation once again results in a Lyapunov system

$$AX + XA^T = F, \tag{4.29}$$

with A nonsymmetric and dense in this case. The matrix F is now an evaluation of $f(x, y)$ from (4.23) on the spectral discretisation grid. Despite containing many zero values, the matrix $\widetilde{A}$, when (4.29) is in Kronecker form, still contains enough nonzero values to be referred to as dense. This statement is supported by Figure 4.12, showing how *sparse backslash* cannot take much advantage of the sparsity of the matrix.

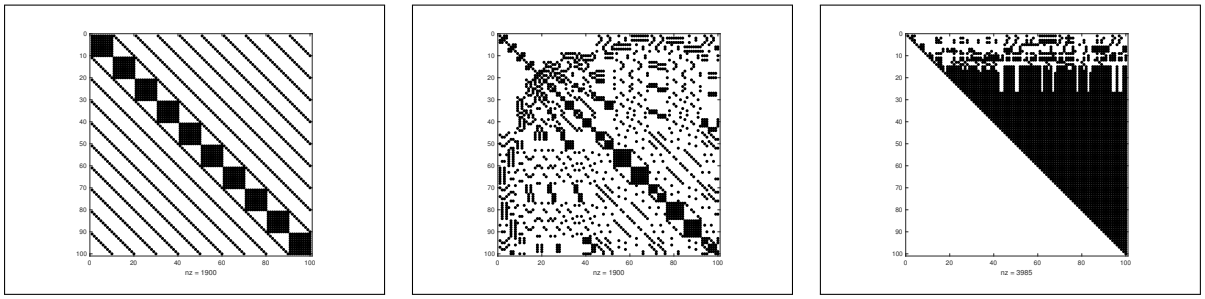


FIGURE 4.12: *The steps of sparse backslash for the system (4.29) in Kronecker form. The figure on the left shows the shape of the matrix in Kronecker form. In the middle we see $A(p, p)$ permuted by an AMD permutation vector, p. The figure on the right shows the LU factorisation of $A(p, p)$, which is almost dense, showing that sparsity cannot be exploited for this matrix.*

The computations for this example were done for matrices up to size 200 in Sylvester form (40000 in Kronecker form) using MATLAB. The results are shown in Figure 4.13. It is confirmed by the computational results, that in the case where the matrices are dense the use of the Kronecker for-

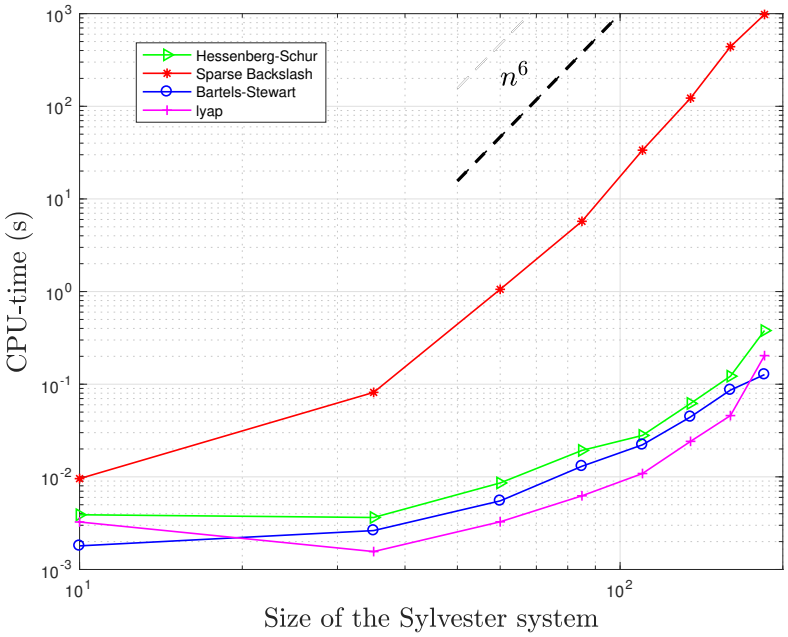


FIGURE 4.13: *Comparison in computational time for solving the dense Lyapunov system (4.29), using the described methods as well as lyap. The methods were timed for matrices up to dimension 200 in Sylvester form (40000 in Kronecker form).*

mulation and *backslash* becomes extremely inefficient, with a confirmed increase in computational time of $\mathcal{O}(n^6)$. Depending on the nature of the system, Bartels–Stewart and Hessenberg–Schur will trade off being the most efficient solver. In this particular case Bartels–Stewart has the upper hand, since it is a Lyapunov system and only one Schur decomposition is necessary, while the Hessenberg–Schur method still computes a Hessenberg and a Schur decomposition.

It is clear that solving (1.1) as a linear system in Kronecker form cannot compete when the matrices are dense. After eliminating this, we compare the three transformation methods to each other and *lyap* for Lyapunov systems up to size 1000. The results appear in Figure 4.14. The relative residual for this problem appears in Figure 4.15.

Despite a higher flop count, the eigenvalue method still appears to be the fastest method in the nonsymmetric, dense case. This can be due to the fact that the eigenvalue function in MATLAB is very well optimised, but it is still interesting to see that it beats the compiled function *lyap*.

In order to understand why one would use a discretisation resulting in dense matrices instead of sparse matrices, the reader is referred to Figure 4.16. It is illustrated here that much larger matrices (grid sizes) are needed in the case of finite difference methods for the differential approximation to be accurate, as can be seen on the left of Figure 4.16. Due to the small matrix dimension needed for accuracy, the spectral discretisation also requires less computational time for convergence, despite the coefficient matrices being dense, as can be seen on the right of Figure 4.16. It should be noted that this example problem has a very smooth solution, so it might be different in other cases. The spectral discretisation also only works on a rectangle, where the finite difference method can be used on more general domains. This explains why one would still consider the finite difference discretisation despite the fact that the spectral discretisation seems

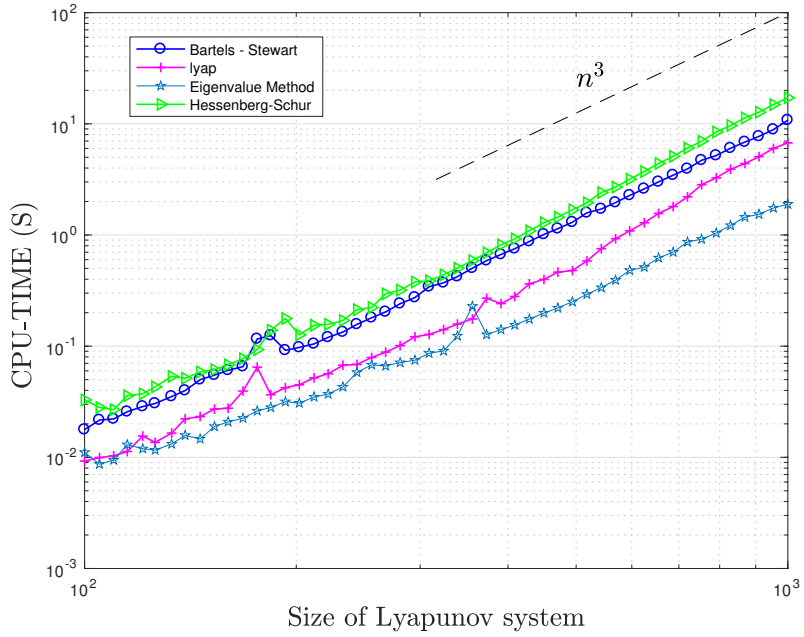


FIGURE 4.14: *Comparison in computational time for solving the dense Lyapunov system (4.29), using the described transformation methods and lyap. The methods were timed for matrices up to dimension 1000 in Sylvester form. The eigenvalue decomposition appears to be the most efficient method for this example.*

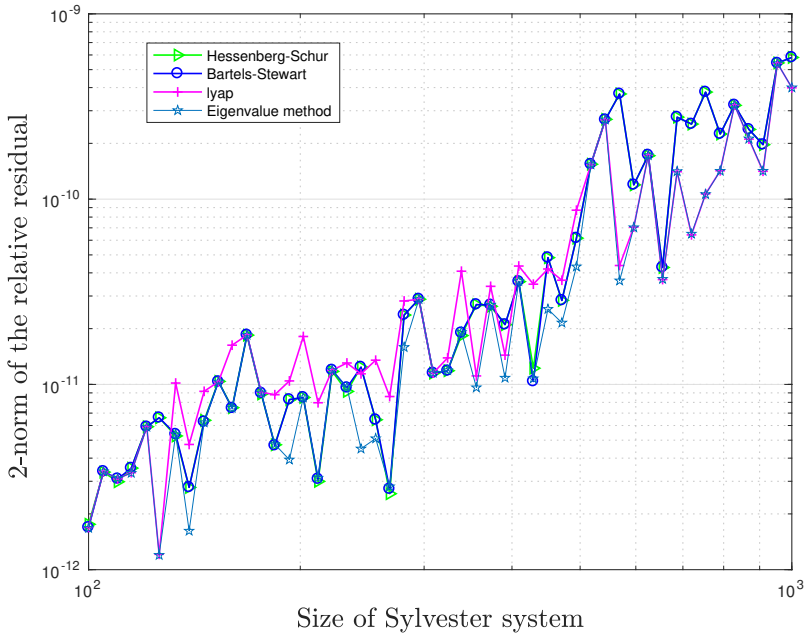


FIGURE 4.15: *Relative residual of the described methods and lyap used to solve the dense Lyapunov system (4.29).*

to be the dominant one in this example.

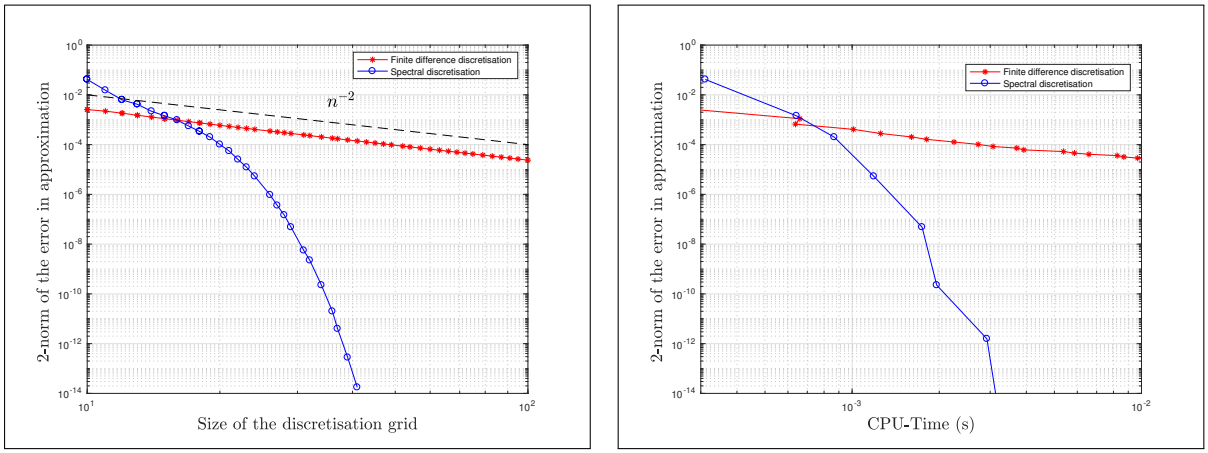


FIGURE 4.16: *Convergence of finite difference discretisations versus spectral discretisations. On the left the comparison in terms of grid size, and the comparison in terms of CPU-time on the right. The spectral discretisation requires smaller grid sizes, as well as less computational time for convergence. The error was measured for the Bartels–Stewart algorithm.*

The discretisation of the Poisson equation results in a Lyapunov equation in both the finite difference and spectral discretisation methods. In order to test the efficiency of the methods on a system where $B \neq A^T$, we consider a modified example problem. The finite difference and spectral discretisation of the Poisson equation with non-constant coefficients given by

$$U_{xx} + e^y U_{yy} = g(x, y), \tag{4.30}$$

are once again performed on the unit square with similar homogeneous boundary conditions to the previous example. The right hand side $g(x, y)$ is once again chosen such that the reference solution (4.25) is satisfied. The discretisations result in a Sylvester equation

$$AX + XB = G, \tag{4.31}$$

where the matrices A and B are tridiagonal in the case of the finite difference discretisation and dense in the case of the spectral discretisation. The tridiagonal case is considered first. Note that in this case the coefficient matrix A is symmetric and B is nonsymmetric, which can result in stability problems for the eigenvalue method. The results appear in Figure 4.17.

Sparse backslash still performs well and appears to beat *lyap* for $n > 1000$. The performance of Bartels–Stewart, Hessenberg–Schur and *lyap* remains very similar to the Lyapunov example. The eigenvalue method still appears to be the fastest method in practice, despite the fact that two eigenvalue decompositions have to be performed. In this case it could be due to the fact that the matrix A is symmetric and tridiagonal, for which the eigenvalue decomposition could be more efficient than a normal symmetric matrix. For more information the reader is referred to [10]. The relative residuals of the methods are plotted in Figure 4.18, showing that the eigenvalue method loses some stability due to the fact that the coefficient matrix B is nonsymmetric.

Finally we consider the efficiency of the methods when the Sylvester system (4.31) consists of dense matrices A and B. The computations in Kronecker form using *backslash* are omitted in this case, since it was shown in Figure 4.13 that this will not compete for dense matrices. The timings were done in MATLAB for Sylvester systems of size up to 1000. The results appear

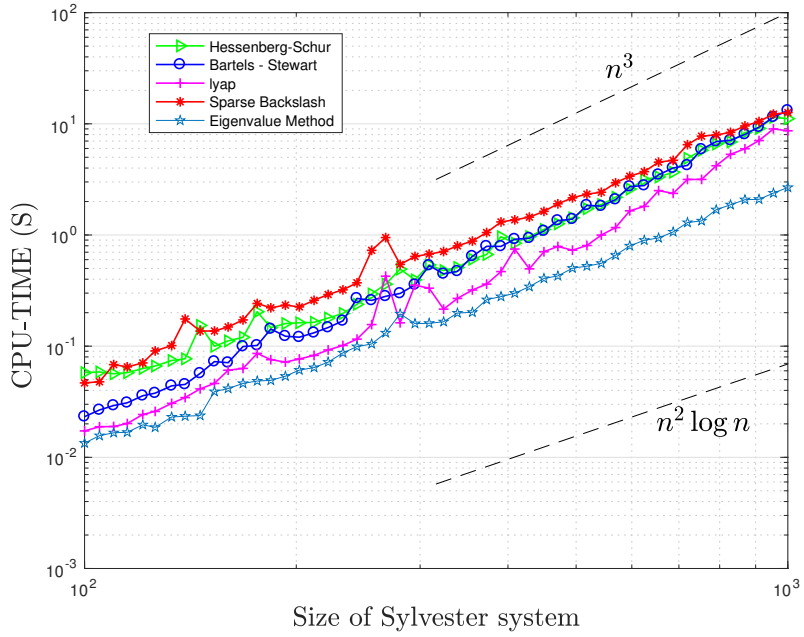


FIGURE 4.17: *Comparison in computational time for solving the sparse Sylvester system (4.31), using the described methods as well as lyap. The methods were timed for matrices up to dimension 1500 in Sylvester form (2250000 in Kronecker form). The eigenvalue method still appears to be the most efficient method.*

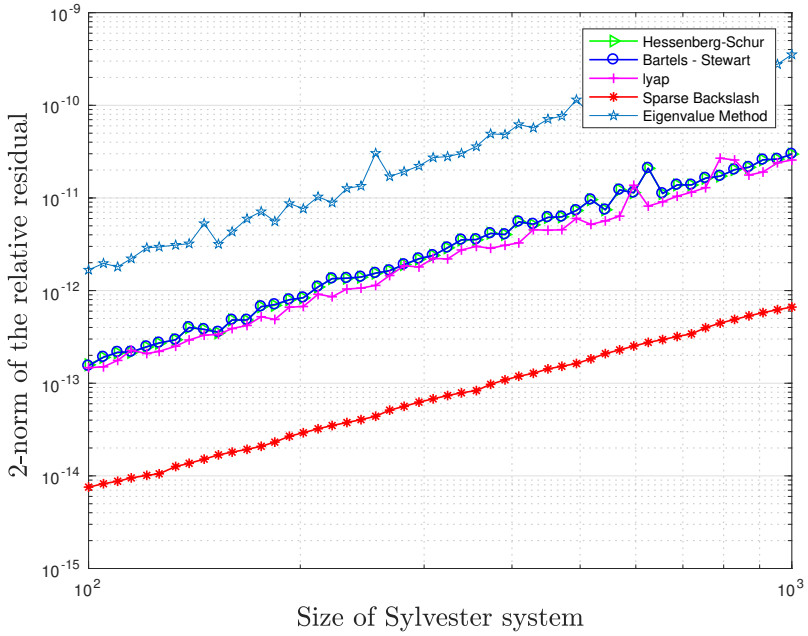


FIGURE 4.18: *Respective 2-norms of the relative residuals of the described methods, as well as lyap, when used to solve the sparse Sylvester equation. This illustrates that the eigenvalue method loses stability due to B having nonsymmetric entries.*

in Figure 4.19.

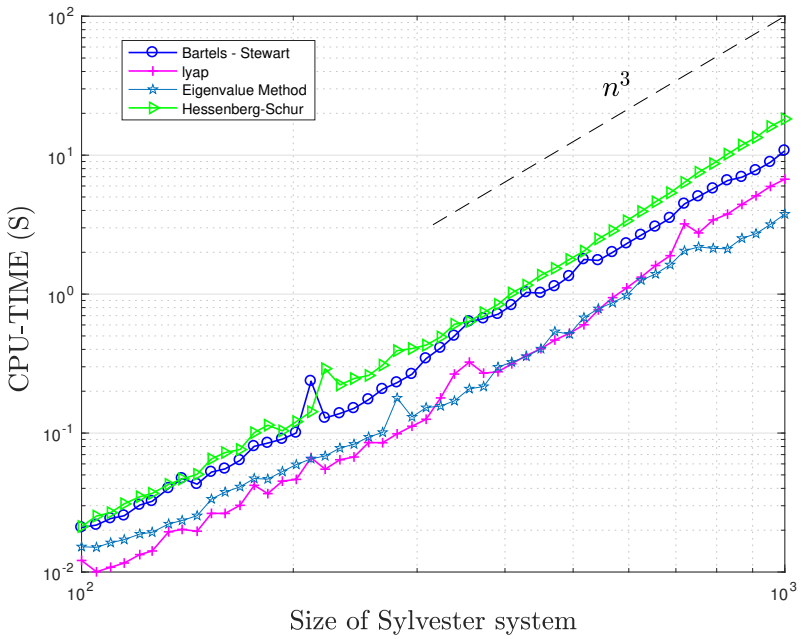


FIGURE 4.19: *Comparison in computational time for solving the dense Sylvester system (4.31), using the described transformation methods as well as lyap. The methods were timed for matrices up to size 1000 in Sylvester form. The eigenvalue method still appears to be the fastest.*

The results are very similar to the Lyapunov example, except for the eigenvalue method becoming a bit slower due to the extra eigenvalue decomposition. Despite this, the eigenvalue method still remains the fastest method for this example, when actually implemented in MATLAB.

In Chapter 3, the idea of solving a Sylvester equation for noise removal in images was introduced. Consider the original image in Figure 4.20, which has size 2686×2686, variance $\sigma^2 = 5163$ and adjacent element correlations $\rho_x = 0.9630$ and $\rho_y = 0.9697$. The image is corrupted by white Gaussian noise with variance $\sigma_\eta^2 = 1634$, such that the signal to noise ratio (SNR) is 5dB. The corrupted image appears on the left of Figure 4.21. The Sylvester equation introduced in Chapter 3 has to be solved in order to recover an approximation of the original image. The matrices A and B each have dimension 2686, where A is tridiagonal (sparse) and B is Toeplitz (dense). Notice from the derivation in Chapter 3 that both A and B are symmetric. The time needed for the discussed direct methods to solve this Sylvester equation is summarised in Table 4.6.

TABLE 4.6: *Time needed for the respective direct methods to solve the Sylvester equation of dimension 2686 in order to remove noise from an image*

	Bartels–Stewart	Hessenberg–Schur	*lyap*	Eigenvalue method
time (s)	240	229	219	25

The methods differ in efficiency for this specific example, with the eigenvalue method requiring the least time. All methods produce the required approximation to the image, but in terms of efficiency, the eigenvalue method dominates. Moreover, the symmetry of the coefficient matrices

FIGURE 4.20: *The original image, with dimension* 2686*, before being corrupted by white Gaussian noise. Photo credit: Viktoria Tollinger.*

also guarantees the stability of this method. Note that this is also the first model problem where Hessenberg–Schur is faster than Bartels–Stewart. The recovered approximation $\widehat{F}$ is shown on the right of Figure 4.21.

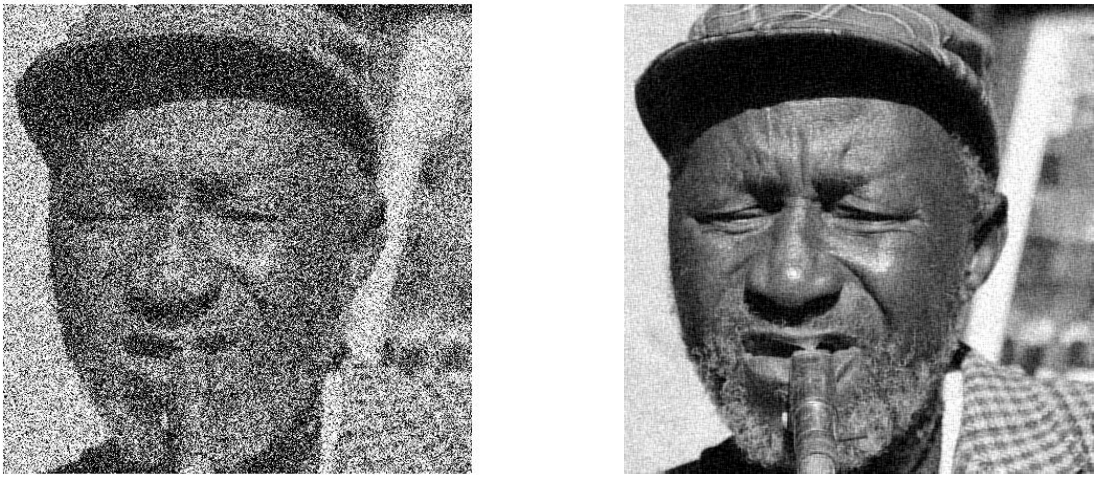


FIGURE 4.21: *The image on the left represents the original image after being corrupted by white Gaussian noise, with SNR* $:= 5$*dB. The image on the right is the recovered approximation* $\widehat{F}$*, found by solving a Sylvester equation.*

4.6 Concluding remarks

In conclusion, the aim of this chapter was to establish which of the described methods will be the most efficient direct solver of equation (1.1), be it dense or sparse. Computations were done, using a spectral- and finite difference discretisation of the well known Poisson equation, as well as

a Poisson equation with non-constant coefficient matrices, on the unit square. It was conjectured that the solving of (1.1) as a linear system in Kronecker form could be inefficient, which was confirmed in the case of dense matrices, where the computational time increased like $\mathcal{O}(n^6)$. On the other hand, in the case of sparse matrices, this solution method appeared to be very competitive, increasing like $\mathcal{O}(n^2 \log n)$, for the size of n under consideration. The Bartels–Stewart and Hessenberg–Schur methods remain very similar in performance, despite the systems changing from sparse to dense. These two methods would trade off in being more efficient, depending on the nature of the system. It was confirmed that for the examples used in this chapter, the method of eigenvalue decomposition is the most efficient, for both dense and sparse Lyapunov and Sylvester equations. This is unfortunately to the expense of stability in comparison to the other methods when the coefficient matrices are nonsymmetric. A comparison of computational time versus relative error was done using the Bartels–Stewart method, showing that the spectral discretisation is the more efficient discretisation method for these type of problems. Finally, a model problem coming from the application of noise removal in images was implemented in order to test the efficiency of the methods when one matrix is sparse and the other dense. Once again, the eigenvalue method required the least time to solve the Sylvester equation, showing that it is the most efficient method in this setting, since the coefficient matrices are symmetric.

CHAPTER 5

Krylov subspace methods for linear systems

Contents

The aim of this chapter is to give some insight on the basic background information for the use of Krylov subspace methods to solve standard linear systems. Krylov subspace methods fall under the general class of *projection methods* for solving a linear system of the form

$$Ax = b, \tag{5.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a nonsingular matrix. A projection method extracts an approximate solution to (5.1) from a subspace of $\mathbb{R}^n$, which will be denoted by $\mathscr{K}$ and referred to as the *search subspace*. If $\mathscr{K}$ has dimension k, it implies that k constraints must be imposed, for example, k independent orthogonality conditions. This space will be denoted by $\mathscr{K}_k$. We also define the left subspace of dimension k, denoted by $\mathscr{L}_k$, as the subspace that arises when the residual vector $b-Ax$ is also made orthogonal to k independent vectors. These are referred to as Petrov-Galerkin conditions. A projection method is referred to as *orthogonal* when $\mathscr{K}_k = \mathscr{L}_k$. In this case the Petrov-Galerkin conditions are simply referred to as Galerkin conditions. When $\mathscr{K}_k$ differs from $\mathscr{L}_k$, it is referred to as an *oblique* projection method.

5.1 The Krylov subspace

The general idea of projection methods is to extract an approximate solution x_k to (5.1) from an affine subspace $\mathscr{K}_k$. This is done by imposing the Petrov-Galerkin condition

$$b - Ax_k \perp \mathscr{L}_k. \tag{5.2}$$

A method is referred to as a Krylov subspace method when the search subspace $\mathscr{K}_k$ is of the form

$$\mathscr{K}_k(A, v) = \text{span}\left\{v, Av, A^2v, \ldots, A^{k-1}v\right\}, \tag{5.3}$$

with v usually selected as the original residual vector $r_0 = b - Ax_0$, where x_0 is an arbitrary initial guess to the solution. Saad [53] shows that when viewed from the angle of approximation theory, the approximations found using Krylov subspace methods are of the form

$$x_k = q_{k-1}(A)v, \tag{5.4}$$

with q_{k-1} a polynomial of degree $k-1$. The choice of iterative method used will depend on the choice of subspace $\mathscr{L}_k$. Two broad choices for the subspace $\mathscr{L}_k$ embracing the most well-known techniques are the choice of orthogonal projection $\mathscr{L}_k = \mathscr{K}_k$ (Full Orthogonalisation Method (FOM)) and the minimum- residual variant $\mathscr{L}_k = A\mathscr{K}_k$ (GMRES, MINRES etc.). These methods will be discussed in more detail at a later stage. It should be noted that the dimension of the Krylov subspace increases by one after each step of the process of approximation, since it is a nested subspace, such that

$$\mathscr{K}_k \subseteq \mathscr{K}_{k+1}. \tag{5.5}$$

This will become clear in the discussion of the projection algorithms in Section 5.3.

5.2 Arnoldi algorithms

An orthogonal projection method for general nonsymmetric matrices was introduced by Arnoldi [2] in 1951. The method was first introduced as a means to reduce dense matrices to upper Hessenberg form, after which it also proved to be an efficient technique for approximating the eigenvalues of large, sparse matrices.

5.2.1 The Arnoldi algorithm

The Arnoldi algorithm takes as input a matrix $A \in \mathbb{R}^{n\times n}$ and an initial vector $v \in \mathbb{R}^n$ and produces as output an upper Hessenberg matrix $\widetilde{H}_k \in \mathbb{R}^{(k+1)\times k}$ and an orthonormal matrix $V_{k+1} \in \mathbb{R}^{n\times(k+1)}$, with column vectors $v_1, v_2, \ldots, v_{k+1}$. The matrix $\widetilde{H}_k$, with its last row deleted, is denoted by H_k and the matrix V_{k+1} with its last column deleted is denoted by V_k. The entries of the output matrices will now be defined.

As proved in [53], the Arnoldi iteration relates the matrix A to the matrix $\widetilde{H}_k$ by

$$AV_k = V_{k+1}\widetilde{H}_k, \tag{5.6}$$

shown in Figure 5.1. The matrix $\widetilde{H}_k$ is the restriction of A to the Krylov subspace $\mathscr{K}_k(A, v)$. The derivation is done by comparison of columns. When we compare the first column on both sides of the equality, it remains to solve

$$Av_1 = v_1h_{11} + v_2h_{21}. \tag{5.7}$$

At this point the only known values are the entries of A and v_1, but since we impose that the entries of V_{m+1} are orthonormal, we multiply both sides of (5.7) by v_1^T, such that

$$h_{11} = v_1^T Av_1, \tag{5.8}$$

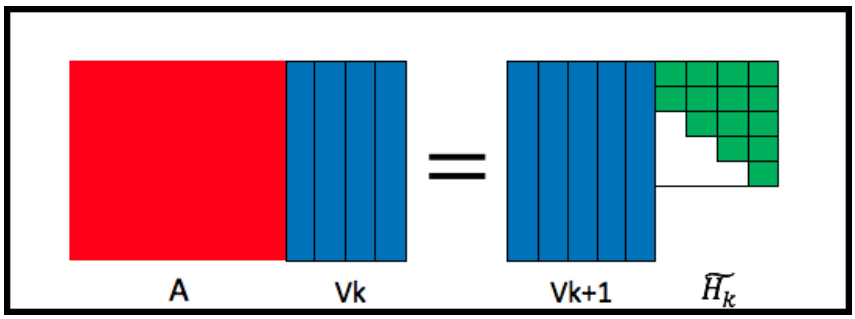


FIGURE 5.1: *A visualisation of the transformation resulting from the Arnoldi iteration. In this case,* $n = 8$ *and* $k = 4$.

since $v_1^T v_2 = 0$. Now that h_{11} is known it remains to solve

$$v_2 h_{21} = Av_1 - v_1 h_{11}, \tag{5.9}$$

for v_2 and h_{21}. However, $||v_2|| = 1$, therefore

$$h_{21} = ||Av_1 - v_1 h_{11}||. \tag{5.10}$$

Finally, the entries of v_2 are recovered by a simple division in (5.9). This simple derivation for the first column is generalised to all k columns in order to generate the Arnoldi algorithm (Algorithm 5.1), which also defines the entries of the matrices $\widetilde{H}_k$ and V_{k+1}.

Algorithm 5.1: Arnoldi

input : $A \in \mathbb{R}^{n\times n}$, $v \in \mathbb{R}^{n\times 1}$, k
output: $\widetilde{H}_k = [h_{i,j}]$, $V_{k+1} = [v_1, v_2, \ldots, v_{k+1}]$

$v_1 = \frac{v}{||v||_2}$;
for $j = 1 : k$ **do**
  $w_j = Av_j$;
  for $i = 1 : j$ **do**
    $h_{ij} = v_i^T w_j$;
    $w_j := w_j - h_{ij} v_i$;
  end
  $h_{j+1,j} = ||w_j||_2$;
  if $h_{j+1,j} = 0$ **then**
    Stop
  else
    $v_{j+1} = \frac{w_j}{h_{j+1,j}}$;
  end
end

Two more properties resulting from the Arnoldi iteration can be derived. Firstly, (5.6) can equivalently be expressed with a rank one error such that

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T, \tag{5.11}$$

where e_k represents the last column of the $k \times k$ identity matrix. This is one property that will be used often when applying Arnoldi's iteration to linear systems. Finally, multiplying both sides of (5.11) with V_k^T and taking the orthonormality of the matrix V_k into account, a final property

is given by

$$V_k^T A V_k = H_k. \tag{5.12}$$

Notice that by (5.12), if the matrix A is symmetric, the upper Hessenberg matrix H_k will also be symmetric, hence tridiagonal. This essentially means that only short-recurrences are needed for the formation of the basis of the Krylov subspace. This is better known as the Lanczos algorithm. All Krylov subspace based algorithms are implemented in such a way that short recurrences are used when the coefficient matrix is symmetric.

5.2.2 The block-Arnoldi algorithm

Note that the Arnoldi algorithm is defined for an input vector $v \in \mathbb{R}^n$. If the right-hand side of (5.1) is not a vector, but a matrix $U \in \mathbb{R}^{n\times s}$, Algorithm 5.1 will not suffice in forming an orthonormal basis for

$$\mathscr{K}_k(A, U) = \text{span}\left\{U, AU, A^2U, \ldots, A^{k-1}U\right\}. \tag{5.13}$$

The block-Arnoldi algorithm [56] is therefore considered for these cases. We say that the block-Arnoldi algorithm forms a basis for the block Krylov subspace, given by

$$\mathbf{K}_k^{\square}(A, U) = \text{span}\left\{U, AU, A^2U, \ldots, A^{k-1}U\right\}. \tag{5.14}$$

Consider the matrix $A \in \mathbb{R}^{n\times n}$ from (5.1) and a rectangular input matrix $U \in \mathbb{R}^{n\times s}$, then the block-Arnoldi algorithm is described in Algorithm 5.2.

Algorithm 5.2: Block-Arnoldi

input : $A \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{n\times s}$, k
output: $\widetilde{\mathcal{H}}_k = [H_{i,j}]$, $\mathcal{U}_{k+1} = [U_1, U_2, \ldots, U_{k+1}]$

```
Compute the QR decomposition of U such that U = Q_1 R_1;
Let U_1 = Q_1;
for j = 1 : k do
    W_j = A U_j;
    for i = 1 : j do
        H_ij = U_i^T W_j;
        W_j := W_j − U_i H_ij;
    end
    Compute the QR decomposition of W_j such that W_j = Q_j R_j ;
    Let U_{j+1} = Q_j and H_{j+1,j} = R_j;
end
```

The block-Arnoldi algorithm produces the orthogonal basis $\mathcal{U}_{k+1} \in \mathbb{R}^{n\times(k+1)s}$ and the restriction of the matrix A onto $\mathbf{K}_k^{\square}(A, U)$ given by

$$\mathcal{H}_k = \mathcal{U}_k^T A \mathcal{U}_k = \begin{bmatrix} H_{1,1} & H_{1,2} & & \cdots & H_{1,k} \\ H_{2,1} & H_{2,2} & & & H_{2,k} \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & & & \\ 0 & \cdots & 0 & H_{k,k-1} & H_{k,k} \end{bmatrix}, \tag{5.15}$$

such that $\mathcal{H}_k$ is in block upper Hessenberg form, with each block $H_{i,j} \in \mathbb{R}^{s\times s}$. This gives rise to the block generalisation of (5.11), such that

$$A\mathcal{U}_k = \mathcal{U}_k\mathcal{H}_k + U_{k+1}H_{k+1,k}E_k^T, \tag{5.16}$$

where E_k represents the last s columns of the $ks \times ks$ identity matrix I_{ks}. The algorithm is actually defined to produce the block upper Hessenberg-like matrix $\widetilde{\mathcal{H}}_k \in \mathbb{R}^{(k+1)s\times ms}$, given by

$$\widetilde{\mathcal{H}}_k = \mathcal{U}_{k+1}^T A\mathcal{U}_k = \begin{bmatrix} H_{1,1} & H_{1,2} & & \cdots & H_{1,k} \\ H_{2,1} & H_{2,2} & & & H_{2,k} \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & & & \\ & & & H_{k,k-1} & H_{k,k} \\ 0 & \cdots & & 0 & H_{k+1,k} \end{bmatrix}, \tag{5.17}$$

where $\mathcal{H}_k$ is then obtained by removing the last s rows of $\widetilde{\mathcal{H}}_k$. The block-Arnoldi algorithm will be of importance in Chapter 6 when deriving the projection algorithms for the Sylvester equation. For now we consider the most well known methods for solving standard linear systems, based on the standard Arnoldi algorithm (i.e., Algorithm 5.1).

5.3 Krylov subspace methods

As mentioned in Section 5.1, Krylov subspace methods attempt to extract an approximate solution to (5.1) from an affine subspace of $\mathbb{R}^n$, with the subspace being of the form

$$\mathscr{K}_k(A, v) = \text{span}\left\{v, Av, A^2v, \ldots, A^{k-1}v\right\}. \tag{5.18}$$

This subspace can be either oblique or orthogonal, depending on the choice of the left subspace $\mathscr{L}_k$. We first discuss the application of the Arnoldi algorithm to an orthogonal projection method ($\mathscr{L}_k = \mathscr{K}_k$), known as the full orthogonalisation method (FOM), derived by Saad in 1981 [51].

5.3.1 FOM

Consider an arbitrary initial guess x_0 to the solution of the equation $Ax = b$ and define r_0, the initial residual, by

$$r_0 = b - Ax_0, \tag{5.19}$$

then the search subspace $\mathscr{K}_k$ for FOM is defined by

$$\mathscr{K}_k(A, v_1) = \text{span}\left\{v_1, Av_1, A^2v_1, \ldots, A^{k-1}v_1\right\}, \tag{5.20}$$

where $v_1 = r_0/\beta$, with $\beta = ||r_0||_2$. The algorithm is derived for the most common case, where $x_0 = \mathbf{0}_k$, such that $r_0 = b$. If residual after k iterations is given by

$$r_k = b - Ax_k, \tag{5.21}$$

then the Galerkin orthogonality condition can be imposed, such that

$$V_k^T r_k = 0. \tag{5.22}$$

This is equivalent to

$$V_k^T A(V_k V_k^T)x_k = V_k^T r_0. \tag{5.23}$$

By imposing (5.12), (5.23) reduces to

$$H_k y_k = V_k^T r_0, \tag{5.24}$$

where $y_k = V_k^T x_k$. Since $r_0 = \beta v_1$, (5.24) can be simplified to

$$H_k y_k = \beta e_1, \tag{5.25}$$

by the orthonormality of V_k. Here e_1 represent the first column of the $k \times k$ identity matrix I_k. An appropriate direct method can now be used to solve the reduced system (5.25) for y_k. Once y_k is known, x_k can be recovered by $x_k = V_k y_k$. The summarised full orthogonalisation method appears in Algorithm 5.3.

Algorithm 5.3: FOM

input : $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^{n\times 1}$, k
output: An approximation x_k to the solution of (5.1)
1 Compute $\beta = ||b||_2$ and $v_1 = b/\beta$;
2 Generate V_k and H_k using Algorithm 5.1 with v_1 as starting vector;
3 Compute $y_k = H_k^{-1}\beta e_1$;
4 Compute $x_k = V_k y_k$;

In this version of the algorithm, the value k has to be decided explicitly beforehand, which is not useful in practice. We would like to recover the residual norm at each step inexpensively in order to impose a stopping criterion for conversion.

Proposition 5.1 ([53]). *The explicit residual of the approximate solution x_k calculated by FOM is given by*

$$r_k = -h_{k+1,k}(e_k^T y_k)v_{k+1}, \tag{5.26}$$

therefore

$$||r_k||_2 = h_{k+1,k}|e_k^T y_k|. \tag{5.27}$$

Proof. We define $r_k = b - Ax_k$ such that

$$\begin{aligned} b - Ax_k &= b - A(V_k y_k) \\ &= r_0 - AV_k y_k, \end{aligned}$$

therefore, using (5.11)

$$r_k = \beta v_1 - V_k H_k y_k - h_{k+1,k} e_k^T y_k v_{k+1},$$

and therefore the result follows, since $\beta v_1 - V_k H_k y_k = 0$ by (5.25). □

This result can therefore be implemented into Algorithm 5.3 in order to dynamically determine k.

In the next section we discuss one of the most well known oblique projection methods.

5.3.2 GMRES

The generalised minimal residual method (GMRES) was introduced by Saad and Schultz in 1986 [54]. It is based on solving the least squares problem $||b-Ax||_2$. For this oblique projection method the left subspace $\mathscr{L}_k$ is chosen such that $\mathscr{L}_k = A\mathscr{K}_k$. A simple derivation of the method is done in this section.

Any vector x in the affine subspace $x_0 + \mathscr{K}_k$ can be expressed as

$$x = x_0 + V_k y, \tag{5.28}$$

for some vector $y \in \mathbb{R}^k$. Then, by making use of (5.6),

$$\begin{aligned} b - Ax &= b - A(x_0 + V_k y) \\ &= r_0 - AV_k y \\ &= \beta v_1 - V_{k+1}\widetilde{H}_k y \\ &= V_{k+1}(\beta e_1 - \widetilde{H}_k y). \end{aligned} \tag{5.29}$$

Therefore, by the orthonormality of V_{k+1}, we define the function

$$M(y) \equiv ||b - A(x_0 + V_k y)||_2 = ||\beta e_1 - \widetilde{H}_k y||_2. \tag{5.30}$$

The approximate solution x_k returned by GMRES is given by $x_k = x_0 + V_k y_k$, where y_k minimises the function $M(y) = ||\beta e_1 - \widetilde{H}_k y||_2$. In summary,

$$\begin{aligned} x_k &= x_0 + V_k y_k \\ y_k &= \min_y ||\beta e_1 - \widetilde{H}_k y||_2. \end{aligned} \tag{5.31}$$

GMRES is summarised in Algorithm 5.4.

Algorithm 5.4: GMRES

input : $A \in \mathbb{R}^{n\times n}$, $x_0 \in \mathbb{R}^{n\times 1}$, k
output: An approximation x_k to the solution of (5.1)
1 Compute $r_0 = Ax_0 - b$, $\beta = ||r_0||_2$ and $v_1 = r_0/\beta$;
2 Generate V_k and $\widetilde{H}_k$ using Algorithm 5.1 using v_1 as starting vector;
3 Compute $y_k = \min_y ||\beta e_1 - \widetilde{H}_k y||_2$;
4 Compute $x_k = x_0 + V_k y_k$;

5.4 Preconditioning

In theory, the preceding methods are well defined, but in practice some of these methods might suffer from slow convergence. This is a common drawback of projection methods. Preconditioning is the technique used to improve the efficiency in convergence and computational time for iterative solvers. The basic concept relies on changing the linear system to one with the same solution, but with better spectral properties, such that convergence happens for a smaller subspace. The biggest challenge related to preconditioning lies in choosing a good preconditioner for the system being solved. Choices of preconditioners include incomplete LU factorisations, successive over-relaxation (SOR) and many more.

Consider a preconditioning matrix, say P, which is nonsingular. The two most important requirements for P, in the case of Krylov subspace methods, is that it should be inexpensive to solve $Px = b$, since a linear solve using P will be required at each iteration, and that

$$||I - P^{-1}A||_2 \ll 1. \tag{5.32}$$

Depending on the preconditioner P, there are three ways in which the preconditioner can be applied to the linear system. If the preconditioner is applied to the left, it leads to the system

$$P^{-1}Ax = P^{-1}b. \tag{5.33}$$

This is referred to as left preconditioning. Alternatively, it can be applied to the right such that

$$AP^{-1}\tilde{x} = b, \qquad x \equiv P^{-1}\tilde{x}, \tag{5.34}$$

referred to as right preconditioning. Finally, if the preconditioner can be factored such that

$$P = P_1 P_2, \tag{5.35}$$

then left-right preconditioning can be applied such that

$$P_1^{-1}AP_2^{-1}\tilde{x} = b, \qquad x \equiv P_2^{-1}\tilde{x}. \tag{5.36}$$

For a better understanding, we show how left-preconditioning is applied to GMRES for some preconditioner P. In this case the Arnoldi iteration will form an orthonormal basis for the Krylov subspace

$$\text{span}\left\{v, P^{-1}Av, (P^{-1}A)^2v, \ldots, (P^{-1}A)^{k-1}v\right\}. \tag{5.37}$$

It is important to note that $P^{-1}A$ is not explicitly computed, but rather solved as a linear system at each step. Algorithm 5.5 summarises the preconditioned GMRES method.

Algorithm 5.5: PGMRES

input : $A \in \mathbb{R}^{n\times n}$, $P \in \mathbb{R}^{n\times n}$ $x_0 \in \mathbb{R}^{n\times 1}$, k
output: An approximation x_k to the solution of (5.1)

1 Solve $Pr_0 = Ax_0 - b$ and compute $\beta = ||r_0||_2$ and $v_1 = r_0/\beta$;
2 Generate V_k and $\widetilde{H}_k$ using Algorithm 5.1, using v_1 as starting vector, and solving $Pw_j = Av_j$ for each j instead of computing $w_j = Av_j$;
3 Compute $y_k = \min_y ||\beta e_1 - \widetilde{H}_k y||_2$;
4 Compute $x_k = x_0 + V_k y_k$;

In the following chapter it is shown how these algorithms can be extended as solution methods for the Sylvester equation.

CHAPTER 6

Krylov subspace methods for Sylvester systems

Contents

This chapter is a survey of the development of projection methods to solve the Sylvester equation when the coefficient matrices are large and sparse. We consider Krylov subspace techniques that project the large problem onto a smaller subspace, where the resulting low-dimensional system is solved using one of the direct methods discussed in Chapter 4. The key to success in the use of these projection methods lies in projecting the problem onto a subspace containing enough spectral information about the coefficient matrices, such that the solution can accurately be approximated, without having to form bases of extensive sizes. The two main Krylov subspaces discussed will be the standard subspace as well as the extended one. Algorithms arising from these two subspaces will be compared using applicable model problems in order to test efficiency as measured by computational time.

6.1 The low-rank phenomenon

Despite the fact that the coefficient matrices may be sparse, the solution X will typically be dense. To support this statement, consider the very simple Lyapunov equation

$$AX + XA^T = I_n, \tag{6.1}$$

where the right-hand side is the identity matrix, which is sparse (diagonal) but of full rank. If A is symmetric and nonsingular, the unique solution to (6.1) is given by

$$X = \frac{1}{2}A^{-1}. \tag{6.2}$$

Suppose A is tridiagonal, then X is dense, as the spy plots of Figure 6.1 show. This is a draw-

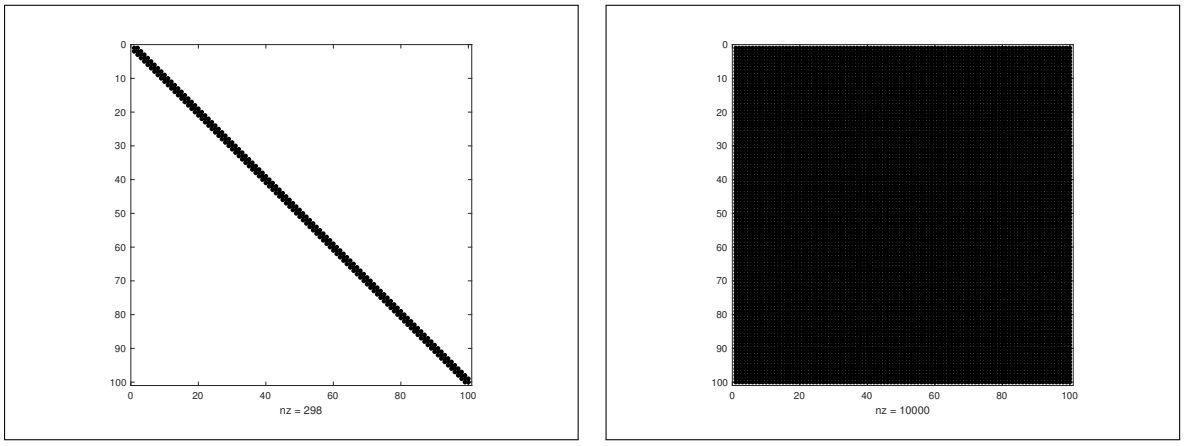


FIGURE 6.1: *Spy plots showing the tridiagonal matrix A on the left and the dense solution X on the right, showing that the solution will be dense despite the coefficient matrix being sparse.*

back, since it will be impractical to store the solution when the coefficient matrices are large. One way around this problem is that the solution matrix can be stored as a product of two low-rank matrices when it contains rapidly decaying singular values [60]. Unfortunately, this is not always the case as can be seen in the plot of singular value decay of the solution X from (6.2) in Figure 6.2. Notice that for this specific example the dense matrix on the right of Figure 6.1 has a special structure, such that every off-diagonal block is of rank ≤ 1. It is also well-known that the inverse of banded matrices can easily be stored (i.e., the solution in (6.2)), but this is not true for general sparse matrices. This example is, therefore, merely indicative of a more general situation.

From results in the thesis by Sabino [55], we conclude that the possibility of obtaining a good low-rank approximation to the solution X is dependent on the ability to express the right-hand side C as a product of two tall and skinny matrices (which implies low rank), such that $C = C_1 C_2^T$.

In order to explain this notion of a low-rank approximation, it is visualised by an application from computer vision. The grayscale image from Figure 4.20 can be represented as a matrix of size 2686×2686, where each entry represents a pixel value between 0 and 255, with 0 being black and 255 being white. This matrix has rank 2686. The matrix representing the image can be represented as a product of low-rank matrices, with the accuracy of the approximation dependent on the rate of singular value decay. The approximations of the image as a product of rank 50 and rank 20 matrices, respectively, appear in Figure 6.3 on the left and on the right.

The original image does not have particularly rapidly decaying singular values, as can be seen in Figure 6.4, hence the loss of accuracy in the rank 20 approximation. If this were not the case, the approximation will be more accurate for an even lower rank. It is therefore essential that the solution X of the Sylvester equation has rapidly decaying singular values.

For these cases, projection methods are attractive, since the approximate solution is of the

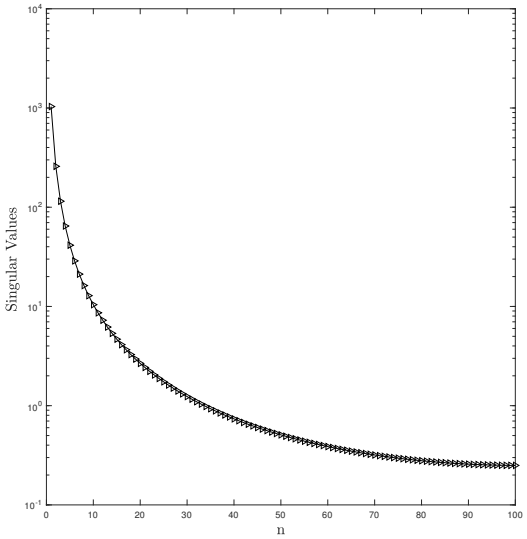


FIGURE 6.2: *Slow singular value decay of the solution X, which will not admit a good low-rank approximation.*

form $X_k = V_k Y_k W_k \approx X$, where V_k and W_k have far fewer columns than A and B, respectively. It is possible to form an approximation where V and W have a different number of columns (i.e., $\widetilde{X} = V_{k1} Y W_{k2} \approx X$), but we only consider the case where they have the same size. The projection methods considered in this chapter are an extension of the Krylov subspace methods derived in Chapter 5. The application of the low-rank approximation is discussed within the respective algorithms.

6.2 Solving in Kronecker form

As in the direct case, a first idea would be to write (1.1) in Kronecker form such that

$$(I_m \otimes A + B^T \otimes I_n)x = -c, \tag{6.3}$$

with $x, c \in \mathbb{R}^{nm \times 1}$ defined in Section 2.1. The large, sparse linear system (6.3) can then be solved iteratively using one of the projection methods derived in Chapter 5. This is conjectured to be an unattractive method due to the fact that it becomes impractical to store a Krylov subspace $\mathscr{K}_m(A \otimes I_m + I_n \otimes B, r_0)$ in fast computer memory when dimensions are too large [34]. Despite this, this method will still be compared in this chapter to the iterative methods derived for the Sylvester system (1.1), for moderate dimensions. The following methods all consider the application of Krylov subspace methods to the system in Sylvester form.

6.3 The standard Krylov subspace

Consider (1.1) with a rank s right-hand side, such that $AX + XB + C_1 C_2^T = 0$ and use Algorithm 5.2 to generate orthonormal bases

$$\mathcal{V}_{k+1} = [V_1, V_2 \dots V_{k+1}] \quad \text{and} \quad \mathcal{W}_{k+1} = [W_1, W_2 \dots W_{k+1}] \tag{6.4}$$

FIGURE 6.3: *Low-rank approximations of Figure 4.20. On the left a rank* 50 *approximation and a rank* 20 *approximation on the right. The image loses its smoothness, but the man can still clearly be identified.*

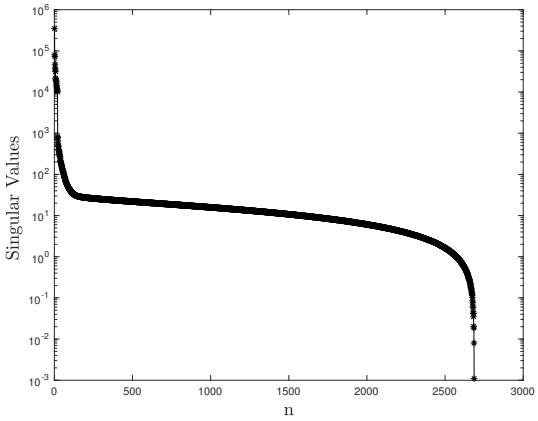


FIGURE 6.4: *Slow singular value decay of the image in Figure 4.20*

for $\mathscr{K}_k = \mathbf{K}_k^{\square}(A, C_1)$ and $\mathscr{C}_k = \mathbf{K}_k^{\square}(B^T, C_2)$ respectively. Algorithm 5.2 will also produce, respectively, the block upper Hessenberg-like matrices $\widetilde{\mathcal{H}}_k^A$ and $\widetilde{\mathcal{H}}_k^B$, defined in Section 5.2. The block upper Hessenberg matrices $\mathcal{H}_k^A$ and $\mathcal{H}_k^B$ can be obtained by removing the last s rows of $\widetilde{\mathcal{H}}_k^A$ and $\widetilde{\mathcal{H}}_k^B$. This gives the relations

$$A\mathcal{V}_k = \mathcal{V}_{k+1}\widetilde{\mathcal{H}}_k^A, \qquad B^T\mathcal{W}_k = \mathcal{W}_{k+1}\widetilde{\mathcal{H}}_k^B \tag{6.5}$$

and

$$\mathcal{H}_k^A = \mathcal{V}_k^T A\mathcal{V}_k, \qquad \mathcal{H}_k^B = \mathcal{W}_k^T B^T\mathcal{W}_k. \tag{6.6}$$

As mentioned in Section 6.1, the aim is to obtain an approximate solution of the form $X_k = \mathcal{V}_k Y_k \mathcal{W}_k^T$. If we define the residual after a certain number of iterations as

$$R_k := AX_k + X_k B + C_1 C_2^T, \tag{6.7}$$

then a Galerkin orthogonality condition is imposed on R_k such that

$$\mathcal{V}_k^T R_k \mathcal{W}_k = 0. \tag{6.8}$$

This results in a system of the form

$$\mathcal{V}_k^T A X_k \mathcal{W}_k + \mathcal{V}_k^T X_k B \mathcal{W}_k + \mathcal{V}_k^T C_1 C_2^T \mathcal{W}_k = 0, \tag{6.9}$$

which can be rewritten as

$$\mathcal{V}_k^T A \mathcal{V}_k (\mathcal{V}_k^T X_k \mathcal{W}_k) + (\mathcal{V}_k^T X_k \mathcal{W}_k) \mathcal{W}_k^T B \mathcal{W}_k + \mathcal{V}_k^T C_1 C_2^T \mathcal{W}_k = 0 \tag{6.10}$$

such that, by (6.6),

$$\mathcal{H}_k^A Y_k + Y_k (\mathcal{H}_k^B)^T + \mathcal{V}_k^T C_1 C_2^T \mathcal{W}_k = 0. \tag{6.11}$$

It can become rather expensive to compute $V_k^T C_1 C_2^T \mathcal{W}_k$ at each iteration. This can be avoided, as we now show. Suppose the matrices C_1 and C_2 can, respectively, be factored as $C_1 = V_1 R_1$ and $C_2 = W_1 R_2$ ($R_1, R_2 \in \mathbb{R}^{s \times s}$), using economy-size QR. Then, V_1 and W_1 are the first blocks of the orthonormal matrices $\mathcal{V}_k$ and $\mathcal{W}_k$, respectively. Taking this orthonormality into account, (6.11) can be reduced to

$$\mathcal{H}_k^A Y_k + Y_k (\mathcal{H}_k^B)^T + E_1 R_1 R_2^T E_1^T = 0, \tag{6.12}$$

where $E_1 \in \mathbb{R}^{ks \times s}$ represents the first s columns of I_{ks}. This system is now much smaller than the original and an appropriate direct method can be applied to find Y_k, after which the approximate solution is recovered by

$$X_k \;=\; \mathcal{V}_k Y_k \mathcal{W}_k^T. \tag{6.13}$$

Notice that a Galerkin (orthogonality) condition has been imposed on the residual vector in order to form (6.11). Other conditions, like minimal residual have also been considered at this step, but we only consider the orthogonal case. The reader is referred to [31, 34, 42] for more information. Note that it is inefficient to calculate the residual in (6.7) at each iteration, since the matrices A and B are large. Fortunately, it is possible to perform the residual computation using matrices of smaller dimension. The proof of Proposition 6.1 is adapted from [29].

Proposition 6.1. *Let R_k be the residual matrix defined in (6.7), then*[1]

$$||R_k||_F^2 = ||H_{k+1,k}^A E_k^T Y_k||_F^2 + ||Y_k E_k (H_{k+1,k}^B)^T||_F^2 \tag{6.14}$$

Proof. Consider (6.7) and (6.13), then

$$||R_k||_F^2 = ||A \mathcal{V}_k Y_k \mathcal{W}_k^T + \mathcal{V}_k Y_k \mathcal{W}_k^T B + C_1 C_2^T||_F^2.$$

By applying (5.16), this is equivalent to

$$\begin{aligned} ||R_k||_F^2 = ||&\mathcal{V}_k \mathcal{H}_k^A Y_k \mathcal{W}_k^T + V_{k+1} H_{k+1,k}^A E_k^T Y_k \mathcal{W}_k^T \\ &+ \mathcal{V}_k Y_k (\mathcal{H}_k^B)^T \mathcal{W}_k^T + \mathcal{V}_k Y_k E_k (H_{k+1,k}^B)^T W_{k+1}^T + C_1 C_2^T||_F^2. \end{aligned}$$

Making use of (6.11), this can be reduced to

$$||R||_F^2 = ||V_{k+1} H_{k+1,k}^A E_k^T Y_k \mathcal{W}_k^T + \mathcal{V}_k Y_k E_k (H_{k+1,k}^B)^T W_{k+1}^T||_F^2,$$

[1]The notation $||\cdot||_F$ refers to the Frobenius norm, defined by $||M||_F^2 = \text{trace}(MM^T)$.

which is equivalent to

$$||R||_F^2 = ||V_{k+1}H^A_{k+1,k}E_k^T Y_k \mathcal{W}_k^T||_F^2 + ||\mathcal{V}_k Y_k E_k (H^B_{k+1,k})^T W_{k+1}^T||_F^2,$$

since $\langle V_{k+1}H^A_{k+1,k}E_k^T Y_k \mathcal{W}_k^T,\ \mathcal{V}_k Y_k E_k (H^B_{k+1,k})^T W_{k+1}^T \rangle = 0$. Finally, taking into account that the Frobenius norm is invariant under multiplication by orthogonal matrices, the residual formula is given by

$$||R_k||_F^2 = ||H^A_{k+1,k}E_k^T Y_k||_F^2 + ||Y_k E_k (H^B_{k+1,k})^T||_F^2,$$

which completes the proof. □

6.3.1 Computing the low-rank factors

It was mentioned in Section 6.1 that instead of explicitly computing the solution X using (6.13), the solution can be stored as a product of low-rank matrices, since the right-hand side $C = C_1 C_2^T$ has low rank. Therefore, after convergence, Y_k is factorised as $Y_k = \widehat{Y}_1 \widehat{Y}_2^T$, with $\widehat{Y}_1$ and $\widehat{Y}_2$ calculated using the truncated singular value decomposition given that Y_k is numerically rank deficient (i.e., has rapidly decaying singular values). Suppose the singular value decomposition of Y_k can be written as $Y_k = \widetilde{V}\Sigma\widetilde{W}^T$, with $\Sigma = \text{diag}[\sigma_1, \sigma_2, \ldots, \sigma_{ks}]$ the matrix of singular values sorted in decreasing order. Suppose $\widetilde{V}_\ell$ and $\widetilde{W}_\ell$ represent, respectively, the first ℓ columns of $\widetilde{V}$ and $\widetilde{W}$, corresponding to the first ℓ singular values of Σ (i.e., Σ_ℓ), greater than some tolerance ϵ. The negligible singular values are then discarded such that $Y_k \approx \widetilde{V}_\ell \Sigma_\ell \widetilde{W}_\ell^T$. The factors $\widehat{Y}_1$ and $\widehat{Y}_2$ are then, respectively, given by $\widehat{Y}_1 = \widetilde{V}_\ell \Sigma_\ell^{1/2}$ and $\widehat{Y}_2 = \widetilde{W}_\ell \Sigma_\ell^{1/2}$. The solution X is then factorised as $X = Z_1 Z_2^T$, with $Z_1 = \mathcal{V}_k \widehat{Y}_1$ and $Z_2 = \mathcal{W}_k \widehat{Y}_2$. When calculating the solution, only the factors $Z_1 \in \mathbb{R}^{n\times\ell}$ and $Z_2 \in \mathbb{R}^{m\times\ell}$ are stored, with $\ell \ll n, m$.

Due to the application of a Galerkin orthogonality condition, the algorithm is referred to as the Sylvester full orthogonalisation method (SFOM). A summary of the algorithm appears in Algorithm 6.1. A visualisation of one iteration of SFOM appears in Appendix A.

Algorithm 6.1: SFOM

input : $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$, $C_1 \in \mathbb{R}^{n\times s}$, $C_2 \in \mathbb{R}^{m\times s}$, ϵ
output: $Z_1 \in \mathbb{R}^{n\times\ell}$, $Z_2 \in \mathbb{R}^{m\times\ell}$

Let $\beta_1 = ||C_1||_F$ and $\beta_2 = ||C_2||_F$;
Perform economy-size QR such that $C_1 = V_1 R_1$, $C_2 = W_1 R_2$;
Set $\mathcal{V}_1 \equiv V_1$, $\mathcal{W}_1 \equiv W_1$;
for $k = 2, 3 \ldots$ **do**
    Compute the next basis blocks V_k, W_k using Algorithm 5.2;
    Let $\mathcal{V}_k = [\mathcal{V}_{k-1}, V_k]$ and $\mathcal{W}_k = [\mathcal{W}_{k-1}, W_k]$;
    Update $H_k^A = \mathcal{V}_k^T A \mathcal{V}_k$ and $H_k^B = \mathcal{W}_k^T B \mathcal{W}_k$;
    Solve $\mathcal{H}_k^A Y_k + Y_k (\mathcal{H}_k^B)^T + E_1 R_1 R_2^T E_1^T = 0$ for Y_k;
    Compute $||R_k||_F$ using (6.14);
    if $||R_k||_F / (\beta_1 \beta_2) < \epsilon$ **then**
        Stop
    end
end
Compute $Y_k = \widehat{Y}_1 \widehat{Y}_2^T$ using the truncated SVD;
Set $Z_1 = \mathcal{V}_k \widehat{Y}_1$ and $Z_2 = \mathcal{W}_k \widehat{Y}_2$;

6.3.2 A note on preconditioning

As in the case of standard linear systems, depending on the spectral properties of A and B, the standard Krylov subspace might require very large bases V_k and W_k in order to accurately approximate the solution X [46]. Unfortunately, these bases are full matrices and when they are large storage will once again be a problem. The common solution to this problem for general linear systems is to precondition the coefficient matrix in order to improve its spectral properties such that convergence occurs for smaller bases. The question is whether or not the same concept can be applied to Sylvester systems.

Naturally, the system can be restated in Kronecker form and normal preconditioning can be applied [30], but this is not equivalent to preconditioning in Sylvester form. Suppose there exist invertible matrices P_1 and P_2 such that $P_1^{-1}A$ and $P_2^{-1}B^T$ exhibit more attractive spectral properties than A and B. In the symmetric case, better spectral properties refers to a better clustering of the eigenvalues. Applying P_1 and P_2, we are left with

$$P_1^{-1}AXP_2^{-T} + P_1^{-1}XBP_2^{-T} + P_1^{-1}C_1C_2^TP_2^{-T} = 0. \tag{6.15}$$

The coefficient matrices now have better spectral properties, but we are not solving a Sylvester equation any more. By adapting (6.15) accordingly, it can be changed to a Sylvester equation, such that

$$P_1^{-1}AP_1(P_1^{-1}XP_2^{-T}) + (P_1^{-1}XP_2^{-T})P_2^TBP_2^{-T} + P_1^{-1}C_1C_2^TP_2^{-T} = 0, \tag{6.16}$$

which can be simplified to

$$(P_1^{-1}AP_1)X_p + X_p(P_2^TBP_2^{-T}) + P_1^{-1}C_1C_2^TP_2^{-T} = 0, \tag{6.17}$$

with $X_p = P_1^{-1}XP_2^{-T}$. By doing this, the coefficient matrices are merely similarity transformations of A and B, resulting in the eigenvalues remaining unchanged.

This means that another method for accelerating the convergence has to be considered. Simoncini [58] introduced the idea that instead of enriching the spectral properties of the coefficient matrices, rather enrich the approximation space. This has become known as the extended Krylov subspace method for Sylvester equations.

6.4 The extended Krylov subspace

The extended Krylov subspace method projects the problem onto an approximation space generated by a combination of Krylov subspaces in A and A^{-1}. The inspiration for this enrichment comes from a similar acceleration applied to the approximation of matrix functions in [12]. The enrichment of the subspace is due to the fact that it is simultaneously expanding in the A and A^{-1} directions. It falls under the general class of rational Krylov subspaces, first introduced by Ruhe [49] and denoted by

$$\mathscr{K}_k^R(A, C) = \text{span}\left\{(A+\sigma_1 I)^{-1}C, (A+\sigma_2 I)^{-1}(A+\sigma_1 I)^{-1}C \dots\right\}, \tag{6.18}$$

for some chosen sequence $\{\sigma_j\}$, $(j = 1, 2 \dots)$. For the Extended Krylov subspace, the poles have been selected at 0 and ∞. It can therefore be defined blockwise as

$$\begin{aligned}\mathbf{EK}_k^{\square}(A, C) &= \text{span}\left\{C, A^{-1}C, AC, A^{-2}C, A^2C \dots\right\} \\ &= \mathbf{K}_k^{\square}(A, C) + \mathbf{K}_k^{\square}(A^{-1}, A^{-1}C),\end{aligned} \tag{6.19}$$

which is why the method is also sometimes referred to as Krylov-plus-inverted-Krylov (K-PIK). Before deriving the method for solving the Sylvester equation using the extended Krylov subspace, we first consider the extended Arnoldi algorithm. This method is not considered for standard linear systems, since the product $A^{-1}C$ appears in the algorithm, which implies that the system could have been solved by some other method for the same computational expense needed for one step in this algorithm.

6.4.1 The extended Arnoldi algorithm

The extended Arnoldi algorithm uses a similar modified Gram–Schmidt orthogonalisation procedure to the standard block-Arnoldi algorithm (Algorithm 5.2), except for the fact that two blocks are now being orthogonalised simultaneously, starting with $[C, A^{-1}C]$.

The algorithm produces the orthonormal matrix $\mathbb{V}_k = [V_1 \dots V_k] \in \mathbb{R}^{n\times 2ks}$, where $V_i \in \mathbb{R}^{n\times 2s}$ $(i = 1, 2 \dots k)$. The block upper Hessenberg matrix $\mathbb{H}_k \in \mathbb{R}^{2ks\times 2ks}$ is also defined, with each block $H_{i,j} \in \mathbb{R}^{2s\times 2s}$. Unfortunately, due to the fact that every second column is spanned by A^{-1}, the relation $\mathbb{H}_k = \mathbb{V}_k^T A\mathbb{V}_k$ does not hold as in the standard (block) Arnoldi algorithm. To resolve this, the block upper Hessenberg matrix $\mathbb{T}_k$ is introduced, such that

$$\mathbb{T}_k = \mathbb{V}_k^T A\mathbb{V}_k, \tag{6.20}$$

is the restriction of A to the extended Krylov subspace $\mathbf{EK}_k^{\square}(A, C)$ [58].

When k becomes large it will be inefficient to calculate $\mathbb{T}_k$ at each iteration using matrix-matrix products, therefore a cheap iteration for calculating $\mathbb{T}_k$ is derived. The iteration is given for the case $s = 1$ for purposes of simplicity, but this can easily be extended to the general case, where constants will now become $s \times s$ blocks.

Proposition 6.2 ([58]). *Let $\ell^{(q)} = (\ell_{ij})$ be the 2×2 matrix such that $\widehat{V}_q = V_q\ell^{(q)}$ $(q = 1 \dots k)$ and V_q has orthogonal columns. Let*

$$\mathbb{T}_k = (t_{i,j})_{i=1:2k+2,j=1:2k} \text{ and } \mathbb{H}_k = (h_{i,j})_{i=1:2k+2,j=1:2k}.$$

Then, for the odd columns

$$t_{:,2q-1} = h_{:,2q-1} \quad (q = 1, \dots, k),$$

while for the even columns

$$\begin{aligned}
(q = 1) \quad & t_{:,2} = h_{:,1}(\ell_{11}^{(1)})^{-1}\ell_{12}^{(1)} + e_1(\ell_{11}^{(1)})^{-1}\ell_{22}^{(1)} \qquad t_{:,4} = (e_2 - \mathbb{T}_1 h_{1:2,2})\ell_{22}^{(2)}, \\
& \rho^{(2)} = (\ell_{11}^{(2)})^{-1}\ell_{12}^{(2)} \\
(1 < q \le k) \quad & t_{:,2q} = t_{:,2q} + t_{2q-1}\rho^{(q)} \\
& t_{2q+2} = (e_{2q} - \mathbb{T}_q h_{1:2q,2q})\ell_{22}^{(q+1)} \qquad \rho^{(q+1)} = (\ell_{11}^{(q+1)})^{-1}\ell_{12}^{(q+1)}.
\end{aligned}$$

It can be noted from the iteration that the lower-diagonal blocks of $\mathbb{T}_k$ have zero second row, such that $\mathbb{T}_k$ has the shape depicted in Figure 6.5. The derivation of the matrix $\mathbb{T}_k$ will be of importance when applying the extended Arnoldi algorithm to the Sylvester equation. A summary of the extended Arnoldi algorithm appears in Algorithm 6.2.

It is important to notice that A^{-1} appearing in the algorithm is never explicitly calculated. The product $A^{-1}V_j$ is rather determined by solving $Ax = V_j$ using an LU-factorisation based solver

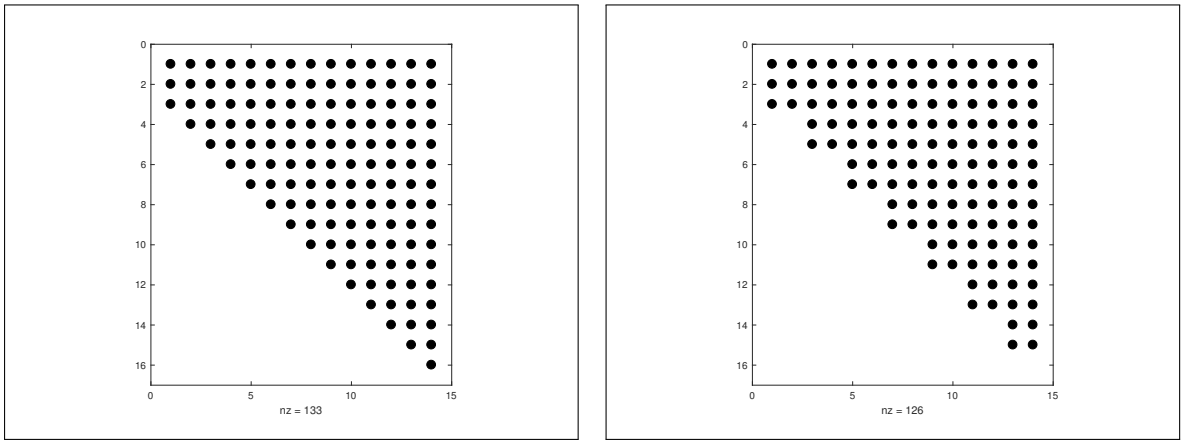


FIGURE 6.5: *The shape of the matrices $\mathbb{H}_k$ and $\mathbb{T}_k$. On the left is the block upper Hessenberg matrix $\mathbb{H}_k$ and on the right is the block upper Hessenberg matrix $\mathbb{T}_k$, with the second row of the lower diagonal blocks being zero. In the case where $s > 1$, each black dot will represent an $s \times s$ block.*

or an appropriate preconditioned iterative method, when A is large and sparse. This appears to be inefficient due to the high cost of solving a system at each iteration, but in [59] it is shown that the excellent convergence properties of the method compensates for this high cost.

Algorithm 6.2: Extended Arnoldi

input : $A \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{n\times s}$, k
output: $\mathbb{H}_k = [H_{i,j}]$, $\mathbb{V}_k = [V_1, V_2, \ldots, V_k]$
Compute the QR factorisation of $[C, A^{-1}C]$ s.t $[C, A^{-1}C] = V_1\Lambda$;
Let $\mathbb{V}_0 = \emptyset$;
for $j = 1 : k$ **do**
    Let $V_j^{(1)}$ be the first s columns of V_j and $V_j^{(2)}$ the last s;
    Then $\mathbb{V}_j = [\mathbb{V}_{j-1}, V_j]$ and $U = [AV_j^{(1)}, A^{-1}V_j^{(2)}]$;
    for $i = 1 : j$ **do**
        $H_{ij} = V_i^T U$;
        $U := U - V_i H_{ij}$;
    end
    Compute the QR factorisation of U such that $U = V_{j+1}H_{j+1,j}$;
end

6.4.2 The extended Krylov subspace method

The derivation of the extended Krylov subspace method is also based on applying a Galerkin orthogonality condition, similar to Sylvester FOM. An investigation of the extended Krylov subspace method for the Sylvester equation, in particular, appears in [29].

Consider once again the Sylvester equation with low-rank right hand side, such that

$$AX + XB + C_1C_2^T = 0. \tag{6.21}$$

The aim is to determine an approximation to the solution X of the form $X_k = \mathbb{V}_k Y_k \mathbb{W}_k^T \approx X$. The matrices $\mathbb{V}_k$ and $\mathbb{W}_k$ are formed after respectively applying k steps of Algorithm 6.2 to the pairs (A, C_1) and (B^T, C_2). Before commencing the derivation, first notice that the first blocks

of $\mathbb{V}_k$ and $\mathbb{W}_k$ are given by the respective economy size QR decompositions of $[C_1, A^{-1}C_1]$ and $[C_2, B^{-T}C_2]$, such that

$$[C_1, A^{-1}C_1] = V_1\Lambda_1 \quad \text{and} \quad [C_2, B^{-T}C_2] = V_2\Lambda_2. \tag{6.22}$$

The matrices $\Lambda_1, \Lambda_2 \in \mathbb{R}^{2s\times 2s}$ are upper triangular, such that

$$\Lambda_1 = \begin{bmatrix} \lambda_{11}^1 & \lambda_{12}^1 \\ & \lambda_{22}^1 \end{bmatrix} \quad \text{and} \quad \Lambda_2 = \begin{bmatrix} \lambda_{11}^2 & \lambda_{12}^2 \\ & \lambda_{22}^2 \end{bmatrix}, \tag{6.23}$$

where $\lambda_{i,j}^{\ell} \in \mathbb{R}^{s\times s}$ $(i, j, \ell = 1, 2)$. These QR decompositions also result in the simplifications

$$\mathbb{V}_k^T C_1 = E_1\lambda_{11}^1 \quad \text{and} \quad \mathbb{W}_k^T C_2 = E_1\lambda_{11}^2, \tag{6.24}$$

due to the orthonormality of $\mathbb{V}_k$ and $\mathbb{W}_k$. Here E_1 represents the first s columns of I_{2ks}. This information will be of importance when simplifying expressions in the derivation of the extended Krylov subspace method.

If the residual after k iterations is

$$R_k = AX_k + X_kB + C_1C_2^T, \tag{6.25}$$

then a Galerkin orthogonality condition can be imposed, such that

$$\mathbb{V}_k^T R_k \mathbb{W}_k = 0. \tag{6.26}$$

This changes (6.25) to

$$\mathbb{V}_k^T AX_k\mathbb{W}_k + \mathbb{V}_k^T X_kB\mathbb{W}_k + \mathbb{V}_k^T C_1C_2^T\mathbb{W}_k = 0, \tag{6.27}$$

which, by imposing (6.24), is equivalent to

$$\mathbb{V}_k^T AX_k\mathbb{W}_k + \mathbb{V}_k^T X_kB\mathbb{W}_k + E_1\lambda_{11}^1(\lambda_{11}^2)^T E_1^T = 0. \tag{6.28}$$

The fact that an approximation of the form $X_k = \mathbb{V}_kY_k\mathbb{W}_k^T$ is required is imposed such that

$$(\mathbb{V}_k^T AV_k)Y_k(W_k^T\mathbb{W}_k) + (\mathbb{V}_k^T V_k)Y_k(W_k^T B\mathbb{W}_k) + E_1\lambda_{11}^1(\lambda_{11}^2)^T E_1^T = 0, \tag{6.29}$$

which simplifies to

$$\mathbb{T}_k^A Y_k + Y_k(\mathbb{T}_k^B)^T + E_1\lambda_{11}^1(\lambda_{11}^2)^T E_1^T = 0, \tag{6.30}$$

due to (6.20). This system is now of smaller dimension than the original and can be solved for Y_m using an appropriate direct method. Finally, Y_k can once again be factored as $Y_k = \widehat{Y}_1\widehat{Y}_2^T$, such that $X = Z_1Z_2^T$, with $Z_1 = \mathbb{V}_k\hat{Y}_1$ and $Z_2 = \mathbb{W}_k\hat{Y}_2$, given that X has rapidly decaying singular values.

Once again, with large matrices, it will be inefficient to calculate the residual, using (6.25), at every iteration. This computation can be done using matrices of smaller dimension, by taking advantage of the shape of the matrix $\mathbb{T}_k$.

Proposition 6.3 ([29], Proposition 2). *Let Y_k be the exact solution of (6.30) and let $X_k = \mathbb{V}_kY_k\mathbb{W}_k^T$ be an approximation to X after k iterations, then the residual R_k associated to X_k satisfies*

$$||R_k||_F = \sqrt{\alpha^2 + \beta^2}, \tag{6.31}$$

where

$$\alpha = ||(\mathbb{T}_k^A)_{a,b}(Y_k)_{b,\,c}||_F$$

and

$$\beta = ||(\mathbb{T}_k^B)_{a,b}(Y_k^T)_{c,\,b}||_F,$$

and a represents $2ks+1$ to $2ks+s$, b represents $2(k-1)s+1$ to $2ks$ and c represents 1 to $2ks$.[2]

Proof. Consider the matrices $\mathbb{V}_{k+1} = [\mathbb{V}_k, V_{k+1}]$, $\mathbb{W}_{k+1} = [\mathbb{W}_k, W_{k+1}]$, $\widetilde{\mathbb{T}}_k^A = [(\mathbb{T}_k^A)^T, E_k(T_{k+1,k}^A)^T]^T$ and $\widetilde{\mathbb{T}}_k^B = [(\mathbb{T}_k^B)^T, E_k(T_{k+1,k}^B)^T]^T$. Here E_k contains the last $2s$ columns of I_{2ks}. Commencing with the residual expressed as

$$R_k = AX_k + X_kB + C_1C_2^T,$$

and imposing arguments similar to those in the proof of Proposition 6.1, this can be simplified to

$$||R_k||_F^2 = ||T_{k+1,k}^A E_k^T Y_k||_F^2 + ||Y_k E_k (T_{k+1,k}^B)^T||.$$

Recall from Figure 6.5 that the last s rows of $T_{k+1,k}^A$ and $T_{k+1,k}^B$ contain only zeros. Considering this and keeping in mind that $E_k = [\mathbf{0}_{2s\times 2(k-1)s}, I_{2s}]$, the desired result is obtained. This completes the proof. □

This result can be utilised in order to compute the residual inexpensively at each iteration. The summarised extended Krylov subspace method (SYLVEXT) appears in Algorithm 6.3. A visualisation of one iteration of SYLVEXT appears in Appendix A.

Algorithm 6.3: SYLVEXT

input : $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$, $C_1 \in \mathbb{R}^{n\times s}$, $C_2 \in \mathbb{R}^{m\times s}$, ϵ
output: $Z_1 \in \mathbb{R}^{n\times \ell}$, $Z_2 \in \mathbb{R}^{m\times \ell}$

Let $\beta_1 = ||C_1||_F$ and $\beta_2 = ||C_2||_F$;
Perform economy-size QR such that $[C_1, A^{-1}C_1] = V_1\Lambda_1$ and $[C_2, B^{-T}C_2] = W_1\Lambda_2$;
Set $\mathbb{V}_1 \equiv V_1$, $\mathbb{W}_1 \equiv W_1$;
for $k = 2, 3\ldots$ **do**
    Compute the next basis blocks V_k, W_k using Algorithm 6.2;
    Let $\mathbb{V}_k = [\mathbb{V}_{k-1}, V_k]$ and $\mathbb{W}_k = [\mathbb{W}_{k-1}, W_k]$;
    Update $\mathbb{T}_k^A$ and $\mathbb{T}_k^B$ using Proposition 6.2;
    Solve $\mathbb{T}_k^A Y_k + Y_k(\mathbb{T}_k^B)^T + E_1\lambda_{11}^1(\lambda_{11}^2)^T E_1^T = 0$ for Y_k;
    Compute $||R_k||_F$ using (6.31);
    if $||R_k||_F/(\beta_1\beta_2) < \epsilon$ **then**
        Stop
    end
end
Compute $Y_k = \widehat{Y}_1\widehat{Y}_2^T$ using the truncated SVD;
Set $Z_1 = \mathbb{V}_k\widehat{Y}_1$ and $Z_2 = \mathbb{W}_k\widehat{Y}_2$;

6.5 Comparison of methods

The aim of this section is to compare the the three projection methods described in this chapter, using some well-known model problems. The methods are compared in two ways. Firstly, by

[2] This essentially means that the first s rows of the block in the bottom right hand corner of $\mathbb{T}_k$ is being multiplied with the last s rows of Y_k. See Figure A.2.

the computational time needed to reach a selected residual tolerance[3], and secondly, by the size of the basis needed to reach this tolerance. The top performing projection method is then compared to the top performing direct method, for a specific model problem, in order to confirm why projection methods are used when the matrices become too large.

Example 6.1 ([32], example 4.1). Consider the one dimensional heat flow problem, on a rod of length one,

$$\frac{\partial T(x,t)}{\partial t} = \frac{\partial^2 T(x,t)}{\partial x^2}, \qquad x \in [0,1], \qquad T(x,0) = 0,$$

subject to $T(0,t) = 0$ and $T(1,t) = u(t)$, as seen in Figure 6.6.

FIGURE 6.6: *Visualisation of the boundary conditions for the one dimensional heat flow problem.*

The solution $T(x,t)$ represents the temperature of the rod at position x after t seconds. The finite difference discretisation results in a dynamical system, with a single input, of the form

$$\frac{dT(t)}{dt} = AT(t) + bu(t),$$

where

$$A = \frac{-1}{h^2}\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{n\times n} \quad \text{and} \quad b = \frac{1}{h^2}\begin{bmatrix} 0 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^{n\times 1},$$

with $h = 1/(n+1)$. As discussed in Chapter 3, a Lyapunov equation of the form

$$AX + XA^T + bb^T = 0 \tag{6.32}$$

has to be solved in order to calculate the controllability Gramian of the system. Notice that the right hand side has rank one, due to the single input. A comparison of the discussed projection methods for this model problem appears in Figure 6.7.

As conjectured, the application of the Kronecker formulation and FOM results in slow convergence. Even when using the built-in MATLAB function *pcg*, preconditioned by an incomplete Cholesky preconditioner, convergence takes 219 seconds. This method will therefore not be taken into account for further comparisons. SFOM shows rapid convergence at the beginning, after which it stagnates, moving almost asymptotically towards the chosen tolerance of 10^{-7}. This is not desirable. Finally, SYLVEXT converges rapidly, showing that it is the most well-suited method for this specific example.

The efficiency of the methods is also compared by the size of the respective bases needed for convergence. Only the two projection methods acting directly on the Lyapunov equation were taken into consideration for this comparison.

[3]This was chosen as 10^{-7} for all examples.

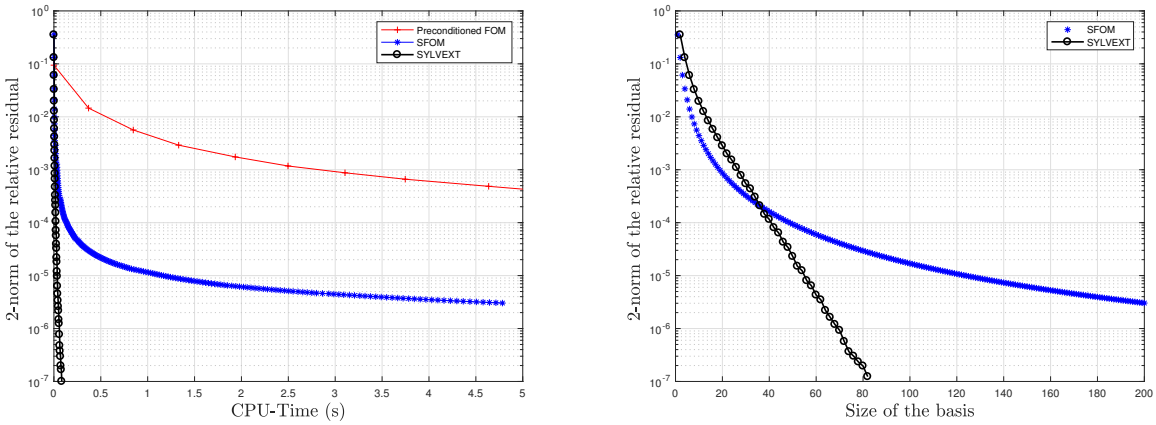


FIGURE 6.7: *Computational results depicting the performance of the discussed projection methods acting on Example 6.1. The figure on the left shows the computational time needed to reach the desired tolerance of* 10^{-7}*. The figure on the right depicts the size of the basis required for convergence for the two methods acting directly on the equation in Lyapunov form. The computations are done for* $n = 2000$*.*

Example 6.2. In this example we consider the two dimensional extension of Example 6.1. Consider the two dimensional heat equation,

$$\frac{\partial T(x,y,t)}{\partial t} = \frac{\partial^2 T(x,y,t)}{\partial x^2} + \frac{\partial^2 T(x,y,t)}{\partial y^2}, \qquad x, y \in [0,1], \qquad T(x,y,0) = 0$$

subject to $T(x,0,t) = u(T)$ and $T = 0$ on the other boundaries, as seen in Figure 6.8.

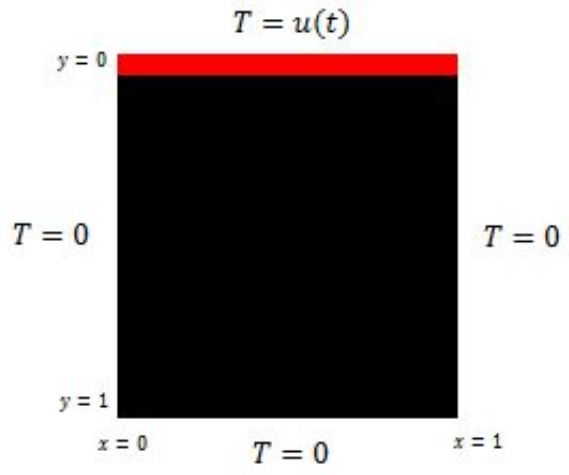


FIGURE 6.8: *Visualisation of the boundary conditions for the two dimensional heat flow problem.*

The discretisation results in a large dynamical system of the form

$$\frac{dT(t)}{dt} = AT(t) + bu(t),$$

with $A \in \mathbb{R}^{n^2 \times n^2}$ and $b \in \mathbb{R}^{n^2 \times 1}$. In this setting,

$$A = \frac{-1}{h^2} \begin{bmatrix} D & -I_n & & \\ -I_n & D & \ddots & \\ & \ddots & \ddots & -I_n \\ & & -I_n & D \end{bmatrix}, \text{ and } b = \frac{1}{h^2}\text{vec}(C),$$

where $h = 1/(n+1)$,

$$D = \begin{bmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{bmatrix} \in \mathbb{R}^{n \times n} \quad \text{and} \quad C = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

Once again, a Lyapunov equation needs to be solved to calculate the controllability Gramian of the system. For the experiments, a discretisation grid of size 501×501 was used, resulting in a Lyapunov system of dimension 250000. SFOM and SYLVEXT were, once again, compared by the computational time needed for convergence, as well as the size of the bases. The results appear in Figure 6.9.

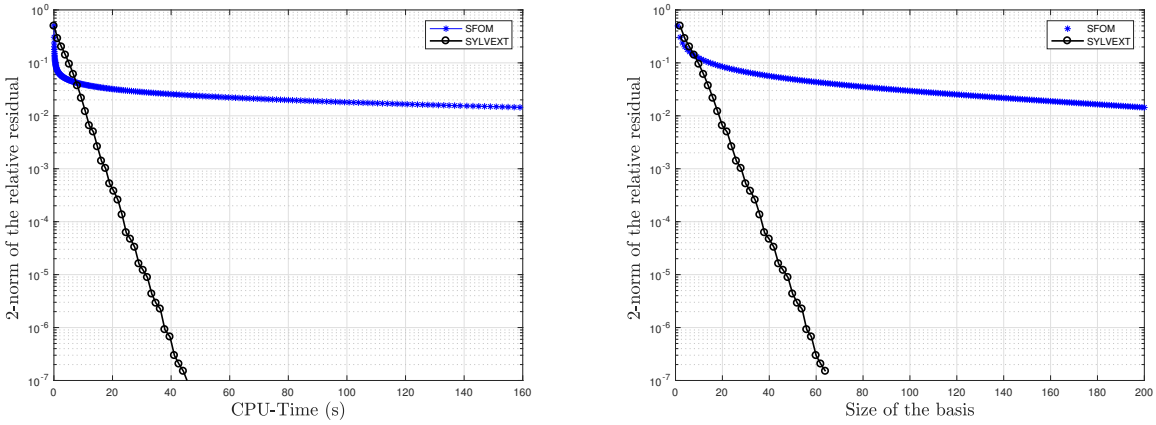


FIGURE 6.9: *Computational results depicting the performance of the discussed projection methods acting on Example 6.2. The figure on the left shows the computational time needed to reach the desired tolerance of* 10^{-7}*. The figure on the right depicts the size of the bases required for convergence. The discretisation is performed on a grid of size* 501×501*, therefore the coefficient matrices have dimension* 250000*.*

The results emphasise the suitability of SYLVEXT to this model problem. Despite the matrices being of dimension 250000, convergence occurs within 43 seconds. A small basis of size 64 is needed for convergence, which is also advantageous, since this reduces the storage requirements. SFOM shows slow convergence, once again.

Example 6.3. Consider once again the first model problem from Section 4.5, where the finite difference discretisation of the Poisson equation results in the sparse Lyapunov equation described by (4.26). This example is considered in order to motivate the use of projection methods. It was concluded in Chapter 4 that for this specific model problem, the eigenvalue method (described in Section 4.4) is dominant in speed. It is also just as stable as the other direct methods, due to the symmetry of the coefficient matrix A. We therefore compare this method to SYLVEXT in order to explain why projection methods are considered. SYLVEXT is first plotted against

SFOM, after which SYLVEXT is compared to the eigenvalue method for different sizes of the coefficient matrix A. For both plots the right-hand side is selected as bb^T, where b is a vector of ones, such that we have a rank one right-hand side. The results appear in Figure 6.10.

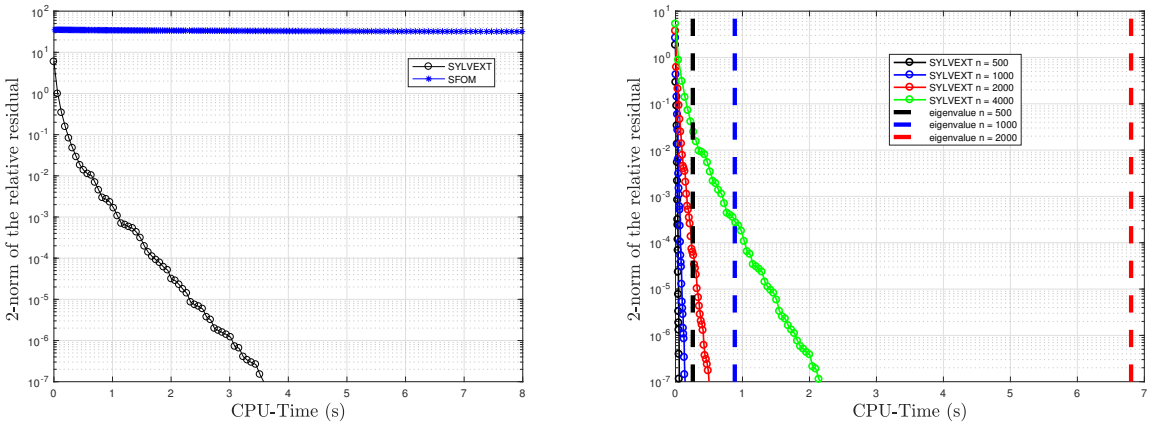


FIGURE 6.10: *Computational results for Example 6.3. On the left, SFOM is compared to SYLVEXT for $n = 5000$, with particularly slow convergence from SFOM. On the right SYLVEXT is compared to the best performing direct method for this example, for $n = 500, 1000, 2000, 4000$. The convergence time of the eigenvalue method for $n = 4000$ is not shown since $t = 56$s puts the other curves out of proportion.*

On the left of Figure 6.10, SFOM is compared to SYLVEXT for Example 6.3. For this specific model problem, SFOM shows particularly slow convergence, with the desired tolerance only reached when the computational time is greater than 10^3 seconds. That being said, SYLVEXT shows rapid convergence once again. On the right of Figure 6.10, SYLVEXT is compared to the most efficient direct method for this example (i.e, the eigenvalue method), for different sizes of A. The time required for SYLVEXT to reach the desired tolerance appears to be predominantly less than the direct solver needs to run. This ratio of required time between SYLVEXT and the direct method becomes greater as n increases, with the direct method requiring 56 seconds when $n = 4000$ and SYLVEXT requiring merely 2.2 seconds. This emphasises the importance of projection methods for solving large, sparse systems.

We make use of the same model problem in order to describe a shortcoming of projection methods, in particular, SYLVEXT. In Section 6.1 it is mentioned that the efficiency of the projection methods described is dependent on the rank of the right-hand side. This is, unfortunately, a shortcoming of these methods, since cases are present when one will not have control over the rank of the right-hand side (e.g., multiple-input-multiple-output (MIMO) dynamical systems [13]). The graphs appearing in Figure 6.11 show, respectively, the effect of the right-hand side has on the computational time as well as the size of the basis.

First, consider the graph on the left of Figure 6.11. We notice that the size of the bases required in order to reach the desired tolerance become large rather quickly as s increases. The storage of these bases requires a large amount of computing memory, which ends up slowing down the computation, as can be seen on the right of Figure 6.11. The efficient performance of the SYLVEXT still results in this method topping the direct methods and all other projection methods, but the storage of these bases can become a bigger problem when n and m are larger.

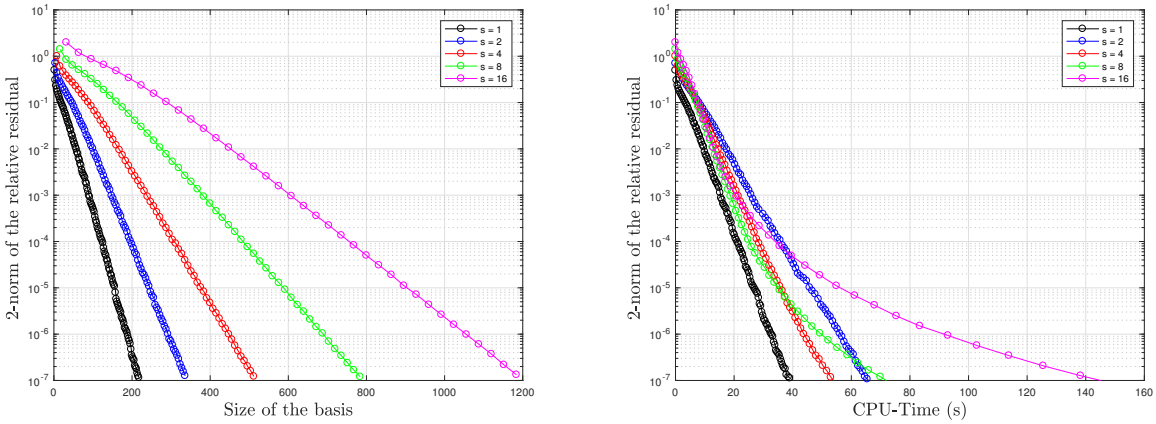


FIGURE 6.11: *Computational results depicting a shortcoming of SYLVEXT. The figure on the left shows the effect of the rank of the right-hand side, s, on the size of the basis needed for convergence. The figure on the right shows the effect of the right-hand side on the computational time needed for convergence. In both cases, we see a loss of efficiency as the rank of the right-hand side increases. For these computations, $n = 12000$ and the right-hand side is chosen as a matrix with rank s, with normally distributed random entries, where $s = 1, 2, 4, 8, 16$.*

6.6 Concluding remarks

In conclusion, the aim of this chapter was to survey the development of projection methods for solving the Sylvester and Lyapunov equations, when the coefficient matrices are large and sparse. The first method discussed was based on restating the equations in Kronecker form, and solving by an appropriate preconditioned Krylov subspace method, as discussed in Chapter 5. As conjectured, this method exhibits slow convergence, even when preconditioned, due to the extensive sizes that the coefficient matrix of the linear system can reach. Further, two Krylov subspace methods, applied directly to the Sylvester equation and based on a Galerkin orthogonality condition was discussed. The first method projects the coefficient matrices onto the standard Krylov subspace, using the block Arnoldi algorithm (Algorithm 5.2), where the second projects the coefficient matrices onto the extended Krylov subspace, using the extended Arnoldi algorithm (Algorithm 6.2). The possibility of forming a low-rank approximation of the solution X was also discussed for these methods.

The methods were compared to each other using well-known model problems, introduced in Chapter 3, arising in the fields of control theory, digital image processing and finite difference discretisation. The methods were compared by the computational time needed to reach the desired tolerance of 10^{-7}, as well as the size of the bases required for convergence. For all the examples, the extended Krylov subspace method (SYLVEXT) exhibits rapid convergence, with the nature of the problem not affecting this behaviour excessively. The standard Krylov subspace method (SFOM) exhibits slow convergence in comparison to SYLVEXT. The best performing iterative method (SYLVEXT) was then compared to the best performing direct method for a specific model problem from Chapter 4. The efficiency of the projection method against the direct method emphasises the importance of these methods when the coefficient matrices are large and sparse.

As a final addition to the chapter, a hurdle of projection methods was depicted, by using the same model problem from Chapter 4. It was shown that the computational efficiency of SYLVEXT

decreases when the rank of the right hand side increases, due to the size of the bases that are required to reach convergence when s increases. This problem is addressed in [13], where the direction of expansion of the approximation space is dynamically selected, based on adaptive tangential interpolation. Results show how the size of the required bases can be decreased by more than 50%.

CHAPTER 7

Conclusions and Future Work

The main aim of this thesis was to review the development of direct and projection methods for solving the Sylvester and Lyapunov equations and to compare these methods for applicable model problems.

The possibility of restating the system in Kronecker form and solving by LU factorisation based methods was considered, as well as three other transformation methods acting on the system in Sylvester form. The four methods were first compared algebraically by operation counts, after which they were compared numerically using applicable model problems. Four different examples were considered, arising from the finite difference and spectral discretisation of the standard Poisson equation and the Poisson equation with non-constant coefficients. The transformation method based on the eigenvalue decomposition proved to be the fastest method in all four settings. This is unfortunately to the expense of stability when the coefficient matrices are nonsymmetric, due to the lack of orthogonality of the matrices of eigenvectors. It was therefore concluded that this method is most efficient, but only when the coefficient matrices are symmetric. When the matrices are nonsymmetric, the efficiency of the methods is dependent on the sparsity of the coefficient matrices. If the coefficient matrices are nonsymmetric, but sparse, the solution in Kronecker form, using *sparse backslash* appears to be most efficient when the size of the matrices increase. Finally, for dense, nonsymmetric coefficient matrices, the built-in MATLAB solver *lyap*, based on the Bartels–Stewart algorithm, appears to perform best. Another model problem, arising from image processing, was considered in order to test the direct methods in a setting where one matrix is dense, the other sparse and both relatively large. Results showed that Hessenberg–Schur is faster than Bartels–Stewart in this setting and that the eigenvalue method is the most efficient. The comparisons for the direct methods were only done for coefficient matrices up to dimension 2686. The projection methods were considered for larger systems.

In the discussion of projection methods, the possibility of restating the system in Kronecker form and solving with standard Krylov subspace methods discussed in Chapter 5, was discussed, but as anticipated this method is highly inefficient due to the extensive sizes of the coefficient matrix in Kronecker form. The two main projection methods that were discussed are based on using a Galerkin orthogonality condition to project the coefficient matrices of the Sylvester or Lyapunov equation onto a smaller subspace containing enough spectral information of the matrix in order to form an accurate approximation to the solution. The first subspace considered was the standard Krylov subspace, resulting in the algorithm SFOM. The option of preconditioning in Sylvester form was discussed, but this proved to be impractical. A richer approximation space, known as the extended Krylov subspace was then rather considered, resulting in the algorithm

SYLVEXT. These two algorithms were then compared using some more well-known model problems discussed in Chapter 3. The methods were compared by the computational time needed to reach a desired tolerance, as well as the size of the bases required for convergence. In all settings, SYLVEXT exhibited rapid convergence, where the convergence history of SFOM appeared to be rather slow and highly dependent on the nature of the problem. A shortcoming of SYLVEXT was also discussed, namely that an increase in the rank of the right hand side can slow down the computational performance, due to large storage requirements.

The top-performing projection method (SYLVEXT) was then compared to the top performing direct method (the eigenvalue method) for the finite difference discretisation of the Poisson equation. This comparison depicted the importance of projection methods, showing that direct methods become highly inefficient when the coefficient matrices are too large, but that SYLVEXT also loses efficiency as the rank of the right hand side increases. A final contribution that was made to the thesis is a visualisation of the existent algorithms in order to understand the transformation and reduction of the Sylvester systems.

Future work will entail the extension of the comparisons to include other iterative methods such as the alternating direction implicit (ADI) iteration. See [6, 41, 43, 67] and the references therein. Equation (1.1) stems from a more general multi-term linear matrix equation given by

$$A_1XB_1 + A_2XB_2 + \cdots + A_\ell XB_\ell = C, \tag{7.1}$$

where A_j, B_j, $j = 1, 2, \ldots, \ell$, are square matrices of size $n \times n$ and $m \times m$, respectively, and C has dimension $n \times m$. Future work can also include a comparison of methods for solving this generalised Sylvester equation. Further research can then also be done into the two main open questions in the field. Firstly, the question of solving large Sylvester systems that do not have a low rank right hand side or a specific banded structure. An initial idea for approaching this would be to use Gaussian elimination with full pivoting to form a low rank approximation of the right hand side [64]. Finally, the idea of preconditioning in Sylvester form is still seen as impractical, but further research into this question could potentially prove otherwise.

References

[1] Antoulas AC, 2005, *Approximation of large-scale dynamical systems*, volume 6 of *Advances in Design and Control*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, with a foreword by Jan C. Willems.

[2] Arnoldi WE, 1951, *The principle of minimized iteration in the solution of the matrix eigenvalue problem*, Quart. Appl. Math., **9**, pp. 17–29.

[3] Barrett J, 1992, *Bibliography of A.M. Lyapunov's work*, Internat. J. Control, **55(3)**, pp. 785–790.

[4] Bartels R & Stewart G, 1972, *Algorithm 432: Solution of the matrix equation* $AX + XB = C$, Comm. ACM, **15(2)**, pp. 820–826.

[5] Benner P, 2004, *Factorized solution of Sylvester equations with applications in control*, Proceedings of the Sixteenth International Symposium on: Mathematical Theory of Network and Systems (MTNS), Leuven, Belgium, July 5-9, 2004.

[6] Benner P, Li R & Truhar N, 2009, *On the ADI method for Sylvester equations*, J. Comput. Appl. Math., **233(4)**, pp. 1035–1045.

[7] Bini D, Iannazzo B & Meini B, 2012, *Numerical solution of algebraic Riccati equations*, volume 9 of *Fundamentals of Algorithms*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[8] Calvetti D & Reichel L, 1996, *Application of ADI iterative methods to the restoration of noisy images*, SIAM J. Matrix Anal. Appl., **17(1)**, pp. 165–186.

[9] Datta BN & Datta K, 1986, *Theoretical and computational aspects of some linear algebra problems in control theory*, pp. 201–212 in *Computational and combinatorial methods in systems theory (Stockholm, 1985)*, pp. 201–212. North-Holland, Amsterdam.

[10] Demmel JW, Marques OA, Parlett BN & Vömel C, 2008, *Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers*, SIAM J. Sci. Comput., **30(3)**, pp. 1508–1526.

[11] Dongarra J, Pozo R & Walker D, 1993, *LAPACK++: A design overview of object-oriented extensions for high performance linear algebra*, Proceedings of the Supercomputing'93. Proceedings, pp. 162–171.

[12] Druskin V & Knizhnerman L, 1998, *Extended Krylov subspaces: approximation of the matrix square root and related functions*, SIAM J. Matrix Anal. Appl., **19(3)**, pp. 755–771.

[13] Druskin V, Simoncini V & Zaslavsky M, 2014, *Adaptive tangential interpolation in rational Krylov subspaces for MIMO dynamical systems*, SIAM J. Matrix Anal. Appl., **35(2)**, pp. 476–498.

[14] El Guennouni A, Jbilou K & Riquet AJ, 2002, *Block Krylov subspace methods for solving large Sylvester equations*, Numer. Algorithms, **29(1-3)**, pp. 75–96, matrix iterative analysis and biorthogonality (Luminy, 2000).

[15] Ellner N & Wachspress E, 1986, *New ADI model problem applications*, Proceedings of the Fall Joint Computer Conference, November 2-6, 1986, Dallas, Texas, USA, IEEE Computer Society, pp. 528–534.

[16] Epton MA, 1980, *Methods for the solution of $AXD - BXC = E$ and its application in the numerical solution of implicit ordinary differential equations*, BIT, **20(3)**, pp. 341–345.

[17] Fernando K & Nicholson H, 1984, *On a fundamental property of the cross-Gramian matrix*, IEEE Trans. Circuits Syst., **31(5)**, pp. 504–505.

[18] Fernando KV & Nicholson Ha, 1983, *On the structure of balanced and other principal representations of SISO systems*, IEEE Trans. Automat. Control, **28(2)**, pp. 228–231.

[19] Gajic Z & Qureshi MTJ, 1995, *Lyapunov matrix equation in system stability and control*, volume 195 of *Mathematics in Science and Engineering*, Academic Press, Inc., San Diego, CA.

[20] Gallivan K, Vandendorpe A & Van Dooren P, 2004, *Sylvester equations and projection-based model reduction*, Proceedings of the 1st International Conference on Linear Algebra and Arithmetic (Rabat, 2001), volume 162, pp. 213–229.

[21] Gardiner J, Laub A, Amato J & Moler C, 1992, *Solution of the Sylvester matrix equation $AXB^T + CXD^T = E$*, ACM Trans. Math. Softw. (TOMS), **18(2)**, pp. 223–231.

[22] George A & Liu J, 1981, *Computer solution of large sparse positive definite systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J., prentice-Hall Series in Computational Mathematics.

[23] Gilbert JR, Moler C & Schreiber R, 1992, *Sparse matrices in MATLAB: design and implementation*, SIAM J. Matrix Anal. Appl., **13(1)**, pp. 333–356.

[24] Golub GH, Nash S & Van Loan C, 1979, *A Hessenberg-Schur method for the problem $AX + XB = C$*, IEEE Trans. Automat. Control, **24(6)**, pp. 909–913.

[25] Golub GH & Van Loan CF, 1996, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, 3rd Edition, Johns Hopkins University Press, Baltimore, MD.

[26] Gonzalez R & Woods R, 2006, *Digital Image Processing (3rd Edition)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[27] Heath M, 2005, *Scientific Computing: An Introductory Survey*, McGraw-Hill Education.

[28] Henrici P, 1979, *Fast Fourier methods in computational complex analysis*, SIAM Rev., **21(4)**, pp. 481–527.

[29] Heyouni M, 2010, *Extended Arnoldi methods for large low-rank Sylvester matrix equations*, Appl. Numer. Math., **60(11)**, pp. 1171–1182.

[30] Hochbruck M & Starke G, 1995, *Preconditioned Krylov subspace methods for Lyapunov matrix equations*, SIAM J. Matrix Anal. Appl., **16(1)**, pp. 156–171.

[31] Hodel AS & Misra P, 1997, *Least-squares approximate solution of overdetermined Sylvester equations*, SIAM J. Matrix Anal. Appl., **18(2)**, pp. 279–290.

[32] Hodel AS, Tenison B & Poolla K, 1996, *Numerical solution of the Lyapunov equation by approximate power iteration*, Linear Algebra Appl., **236**, pp. 205–230.

[33] Horn RA & Johnson CR, 1994, *Topics in matrix analysis*, Cambridge University Press, Cambridge, corrected reprint of the 1991 original.

[34] Hu DY & Reichel L, 1992, *Krylov-subspace methods for the Sylvester equation*, Linear Algebra Appl., **172**, pp. 283–313, second NIU Conference on Linear Algebra, Numerical Linear Algebra and Applications (DeKalb, IL, 1991).

[35] Jbilou K, 2006, *Low rank approximate solutions to large Sylvester matrix equations*, Appl. Math. Comput., **177(1)**, pp. 365–376.

[36] Jonsson I & Kågström B, 2002, *Recursive blocked algorithm for solving triangular systems. I. One-sided and coupled Sylvester-type matrix equations*, ACM Trans. Math. Software, **28(4)**, pp. 392–415.

[37] Jonsson I & Kågström B, 2002, *Recursive blocked algorithm for solving triangular systems. II. Two-sided and generalized Sylvester and Lyapunov matrix equations*, ACM Trans. Math. Software, **28(4)**, pp. 416–435.

[38] Konidaris G, Patera A, Penn J & Yano M, 2012, *Draft v1.2, math, numerics and programming (for mechanical engineers)*, Available from `https://ocw.mit.edu/help/faq-cite-ocw-content/`.

[39] Lancaster P, 1970, *Explicit solutions of linear matrix equations*, SIAM Rev., **12**, pp. 544–566.

[40] LeVeque R, 2007, *Finite difference methods for ordinary and partial differential equations*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, steady-state and time-dependent problems.

[41] Li J & White J, 2004, *Low-rank solution of Lyapunov equations*, SIAM Rev., **46(4)**, pp. 693–713.

[42] Lin Y & Simoncini V, 2013, *Minimal residual methods for large scale Lyapunov equations*, Appl. Numer. Math., **72**, pp. 52–71.

[43] Lu A & Wachspress EL, 1991, *Solution of Lyapunov equations by alternating direction implicit iteration*, Comput. Math. Appl., **21(9)**, pp. 43–58.

[44] Palitta D & Simoncini V, 2017, *Numerical methods for large-scale Lyapunov equations with symmetric banded data*, arXiv preprint arXiv:1711.04187.

[45] Pan VY & Chen ZQ, 1999, *The complexity of the matrix eigenproblem*, pp. 507–516 in *Annual ACM Symposium on Theory of Computing (Atlanta, GA, 1999)*, pp. 507–516. ACM, New York.

[46] Penzl T, 1999/00, *A cyclic low-rank Smith method for large sparse Lyapunov equations*, SIAM J. Sci. Comput., **21(4)**, pp. 1401–1418.

[47] Robbé M & Sadkane M, 2002, *A convergence analysis of GMRES and FOM methods for Sylvester equations*, Numer Algorithms, **30(1)**, pp. 71–89.

[48] Roth W, 1952, *The equations AX-YB = C and AX-XB = C in matrices*, Proc. Amer. Math. Soc., **3(3)**, pp. 392–396.

[49] Ruhe A, 1984, *Rational Krylov sequence methods for eigenvalue computation*, Linear Algebra Appl., **58**, pp. 391–405.

[50] Rutherford D, 1932, *On the solution of the matrix equation AX + XB = C*, Proc. Kon. Nederl. Akad. Weten.(Amsterdam), **35**, pp. 54–9.

[51] Saad Y, 1981, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp., **37(155)**, pp. 105–126.

[52] Saad Y, 1989, *Numerical solution of large Lyapunov equations*, pp. 503–511 in *Signal processing, scattering and operator theory, and numerical methods (Amsterdam, 1989)*, volume 5 of *Progr. Systems Control Theory*, pp. 503–511. Birkhäuser Boston, Boston, MA.

[53] Saad Y, 2003, *Iterative methods for sparse linear systems*, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA.

[54] Saad Y & Schultz MH, 1986, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., **7(3)**, pp. 856–869.

[55] Sabino J, 2007, *Solution of large-scale Lyapunov equations via the block modified Smith method*, ProQuest LLC, Ann Arbor, MI, thesis (Ph.D.)–Rice University.

[56] Sadkane M, 1993, *Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems*, Numer. Math., **64(2)**, pp. 195–211.

[57] Schendel U, 1989, *Sparse matrices*, Ellis Horwood Series: Mathematics and its Applications, Ellis Horwood Ltd., Chichester; Halsted Press [John Wiley & Sons, Inc.], New York, numerical aspects with applications for scientists and engineers, Translated from the German.

[58] Simoncini V, 2007, *A new iterative method for solving large-scale Lyapunov matrix equations*, SIAM J. Sci. Comput., **29(3)**, pp. 1268–1288.

[59] Simoncini V, 2010, *Extended Krylov subspace for parameter dependent systems*, Appl. Numer. Math., **60(5)**, pp. 550–560.

[60] Simoncini V, 2016, *Computational methods for linear matrix equations*, SIAM Rev., **58(3)**, pp. 377–441.

[61] Sorensen DC & Antoulas AC, 2002, *The Sylvester equation and approximate balanced reduction*, Linear Algebra Appl., **352**, pp. 671–700, fourth special issue on linear systems and control.

[62] Sorensen DC & Zhou Y, 2003, *Direct methods for matrix Sylvester and Lyapunov equations*, J. Appl. Math., pp. 277–303.

[63] Sylvester J, 1884, *Sur l'equation en matrices px = xq*, CR Acad. Sci. Paris, **99(2)**, pp. 67–71.

[64] Townsend A, 2016, *Gaussian elimination corrects pivoting mistakes*, arXiv preprint arXiv:1602.06602.

[65] Trefethen LN, 2000, *Spectral methods in MATLAB*, volume 10 of *Software, Environments, and Tools*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[66] Van den Boom A, Brown A, Dumortier F, Geurts A, Hammarling S, Kool R, Vanbegin M, Van Dooren P & Van Huffel S, 1999, *SLICOT—a subroutine library in systems and control theory*, pp. 499–539 in *Applied and computational control, signals, and circuits, Vol. 1*, volume 1 of *Appl. Comput. Control Signals Circuits*, pp. 499–539. Birkhäuser Boston, Boston, MA.

[67] Wachspress E, 1988, *Iterative solution of the Lyapunov matrix equation*, Appl. Math. Lett., **1(1)**, pp. 87–90.

[68] Wilkinson JH, 1988, *The algebraic eigenvalue problem*, Monographs on Numerical Analysis, The Clarendon Press, Oxford University Press, New York, Oxford Science Publications.

APPENDIX A

Visualisation of projection algorithms

Figures A.1 and A.2 respectively show one iteration of Algorithms 6.1 and 6.3 on a small scale. First consider Figure A.1, representing one iteration of Algorithm 6.1.

For this visualisation, $n = m = 10$, $s = 1$ and $k = 4$. The first image in Figure A.1, labelled (a), represents the original Sylvester system, with all matrices of size 10×10. Steps (b) and (c) show, respectively, how the matrices A and B are projected onto the standard Krylov subspace, after applying four steps of Algorithm 5.2. Step (d) is a visualisation of the projected Sylvester equation (6.12), which can now be solved by using an appropriate direct method, in order to recover Y_k. Once Y_k has been recovered, step (e) depicts how (6.14) can be used to calculate the residual norm on small scale. Supposing that the residual norm at step (e) is smaller than a certain chosen tolerance, steps (f) and (g) depict how the information from Section 6.3.1 can be used to calculate a low-rank approximation of the solution X.

Next, consider Figure A.2, representing one iteration of Algorithm 6.3. For this visualisation, $n = m = 10$, $s = 1$ and $k = 4$. The first image in Figure A.1, labelled (a), represents the original Sylvester system, with all matrices of size 10×10. Steps (b) and (c) respectively show how the matrices A and B are projected onto the extended Krylov subspace, after applying four steps of Algorithm 6.2. The respective pink and orange columns represent columns formed by A and B, where the light blue and light green columns represent columns formed by A^{-1} and B^{-1}. Step (d) is a visualisation of the projected Sylvester equation (6.30), which can now be solved by using an appropriate direct method, in order to recover Y_k. Once Y_k has been recovered, step (e) depicts how (6.31) can be used to calculate the residual norm on small scale. Supposing that the residual norm at step (e) is smaller than a certain chosen tolerance, steps (f) and (g) depict how the information from Section 6.3.1 can be used to calculate a low-rank approximation of the solution X.

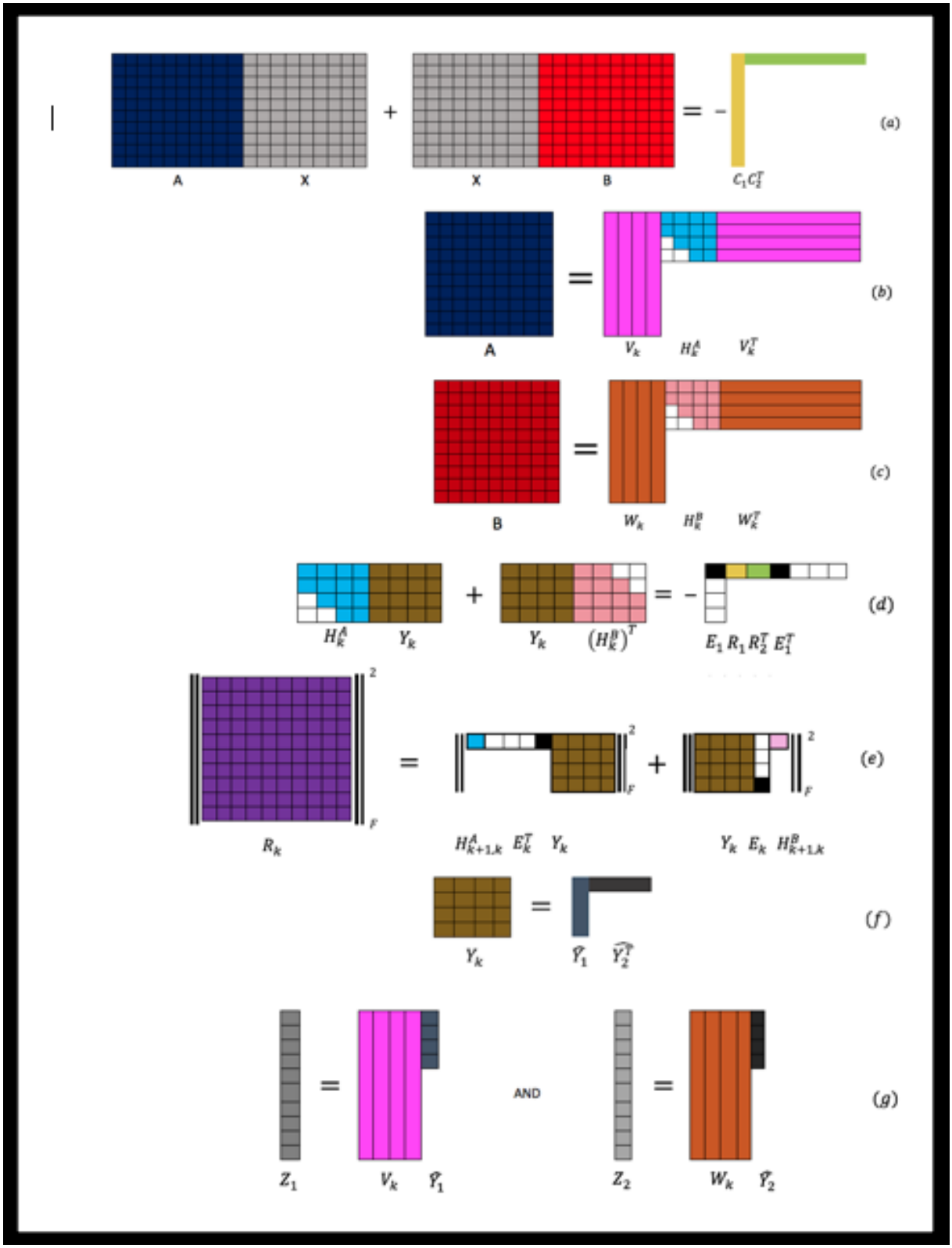


FIGURE A.1: *Visualisation of one iteration of Sylvester FOM*

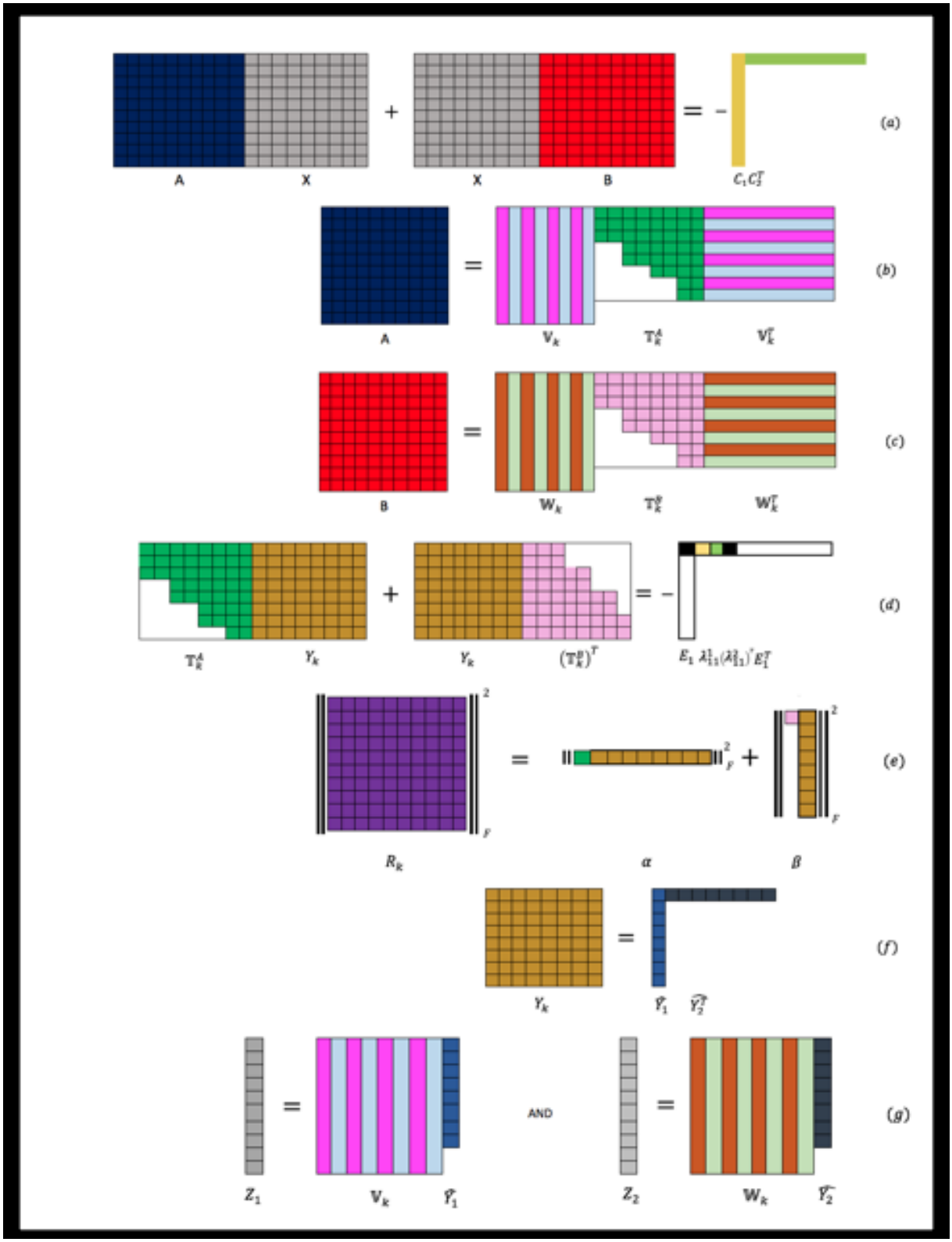


FIGURE A.2: *Visualisation of one iteration of the extended Krylov subspace method*