School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Matrix Solvers for 2-D Heat Equation

Ibtisam Khalaf Alshammari

Submitted in accordance with the requirements for the degree of MSc Advanced Computer Science

2019/2020

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Beliverable 1	Report	SSO (28/08/20)
Deliverable 2	Source code https://github.com/Ibtisam-a/Project-Implementation.git	Supervisor, Assessor (28/08/20)

Type of project: Exploratory Software.

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

Summary

This thesis reports a variety of linear matrix methods for solving Sylvester Matrix that results from two-dimensional Heat Equation, and demonstrate the advantage of the structure of the problem. Such these linear matrix methods have a significant role in solving this matrix system. Moreover, these methods figure out the best performance and accurate results by observing the algorithm's behavior.

Acknowledgements

First of all, I would like to thank the great "God" Almighty, who gave me the ability to continue my studies and complete this project satisfactorily.

Second, I would like to express the enormous gratitude and appreciation go to my parents and my brother Adel for their continued support and encouragement for me throughout my studies. Also, I offer my gratitude to my dear son Rakan for his constant support.

In addition, I would like to offer my gratitude and sincere appreciation to my supervisor Dr. Thomas Ranner for his valuable guidance and continuous support throughout this project.

Finally, I would like to conclude with my heartfelt thanks to my country, the Kingdom of Saudi Arabia, which gave me this opportunity and offered me the sponsorship to continue my higher education.

Contents

List of Figures

List of Tables

1 Introduction

This project presents numerical solutions for solving Partial Differential Equations (PDEs) problems that rely on time and space such as two-dimensional Heat Equation. According to [11], [26], the 2-D Heat Equation will be discretised by Finite Difference Approximation in space and Backward Euler Method in time. The discussion about the numerical solution to the Heat Equation can be found in these sources [3], [22].The problem will be formulated for a matrix system taking into account of stacking all the unknowns of the problem into a single vector. This is attributed to the essential task of this project is to explore linear matrix methods for the purpose of solving matrix system that arise from Partial Differential Equations. The linear matrix methods will be applied by exploiting various solvers, which these solvers need to be tested, analysed and evaluated based on several criteria; determining mesh size and time step, identifying the maximum and average errors, finding the Experimental Order of Convergence (EOC) and the Experimental Order of Growth (EOG) and the last one is iterations number.As a result, exploring the best performance among these solvers, nevertheless, focusing on the effect of the different time steps and various mesh sizes on the EOG.

Primarily, Heat Equation may be used extensively to derive the model problem of this project, leading to the formulation of the problem by Sylvester Matrix to stack the unknowns and then the system will be ready for linear matrix solvers to be applied and compared.

1.1 Heat Equation

The spatial domain of the Heat equation is: $\mathcal{D} = \{(x, y) : 0 \leq x \leq 1, 0 \leq y \leq 1 \text{ on } [0, T]\}$ and let $u : \mathcal{D} \to \mathbb{R}$ is the solution of that PDE, so $u = 0$ on the boundary, $u = u_0$ at $t = 0$. Our two-dimensional Heat equation taking into account the Laplacian equation will be:

$$\begin{aligned} u_t - u_{xx} - u_{yy} &= f \text{on } \mathcal{D} \\ u &= 0 \text{ on } \partial\mathcal{D} \end{aligned} \tag{1.1}$$

we can write equation (1.1) as:

$$\frac{\partial}{\partial t} u_{(x,y,t)} - \frac{\partial^2}{\partial x^2} u_{(x,y,t)} - \frac{\partial^2}{\partial y^2} u_{(x,y,t)} = 0 \tag{1.2}$$

as shown in the next Figure 1:

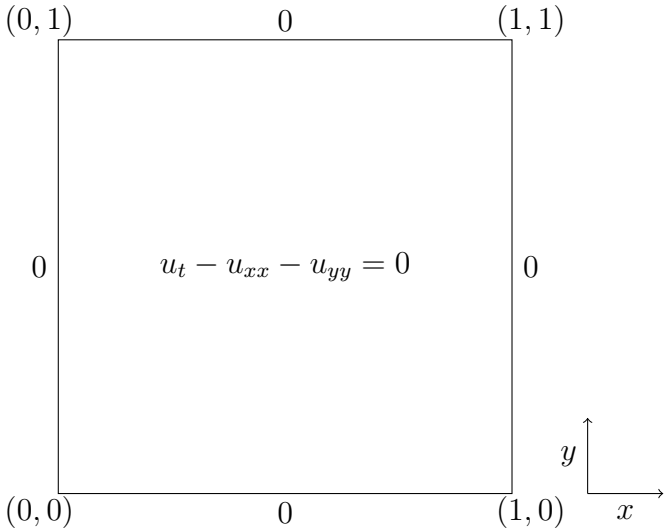


Figure 1: Domain for $u_t - u_{xx} - u_{yy} = 0$.

Let $u_{ij}^n = u_{(x,y,t)}$ and then apply the centred finite difference approximation in space and Implicit Euler method in time for each term of equation (1.2):

$$u_t \approx \frac{\left(u_{ij}^{n+1} - u_{ij}^n\right)}{\Delta t} \tag{1.3}$$

$$u_{xx} \approx \frac{\left(u_{i+1j}^{n+1} - 2u_{ij}^{n+1} + u_{i-1j}^{n+1}\right)}{\Delta x^2} \tag{1.4}$$

$$u_{yy} \approx \frac{\left(u_{ij+1}^{n+1} - 2u_{ij}^{n+1} + u_{ij-1}^{n+1}\right)}{\Delta y^2} \tag{1.5}$$

In discretised form the two-dimensional Heat equation (1.2) can be written as:

$$\frac{\left(u_{ij}^{n+1} - u_{ij}^n\right)}{\Delta t} - \frac{\left(u_{i+1j}^{n+1} - 2u_{ij}^{n+1} + u_{i-1j}^{n+1}\right)}{\Delta x^2} - \frac{\left(u_{ij+1}^{n+1} - 2u_{ij}^{n+1} + u_{ij-1}^{n+1}\right)}{\Delta y^2} = 0 \tag{1.6}$$

Figure 2 below shows the mesh with uniform spacing h:

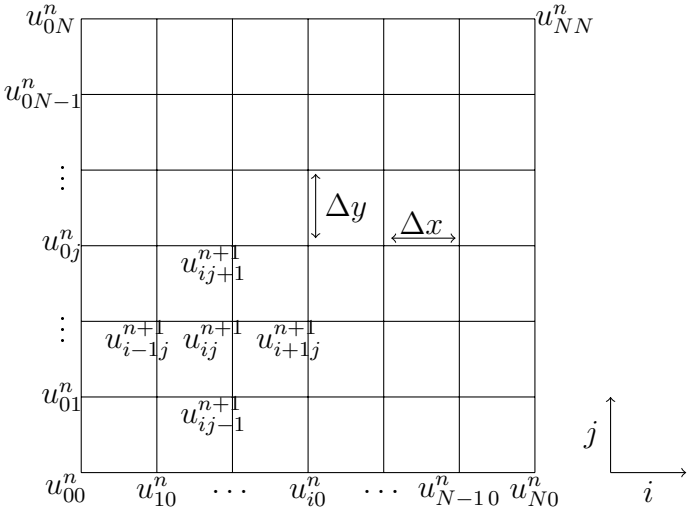


Figure 2: Discretised domain for $u_t - u_{xx} - u_{yy} = 0$.

after multiplying Δt on both sides and rearranging terms of equation (1.6), it becomes:

$$-\frac{\Delta t}{\Delta x^2}\left(u_{i+1j}^{n+1}+u_{i-1j}^{n+1}\right)+\left(1+\frac{2\Delta t}{\Delta x^2}+\frac{2\Delta t}{\Delta y^2}\right)u_{ij}^{n+1}-\frac{\Delta t}{\Delta y^2}\left(u_{ij+1}^{n+1}+u_{ij-1}^{n+1}\right)=u_{ij}^{n}$$

also, it could be written as:

$$-\frac{\Delta t}{\Delta x^2}\left(u_{i+1j}^{n+1}+u_{i-1j}^{n+1}\right)+\left(\frac{1}{2}+\frac{1}{2}+\frac{2\Delta t}{\Delta x^2}+\frac{2\Delta t}{\Delta y^2}\right)u_{ij}^{n+1}-\frac{\Delta t}{\Delta y^2}\left(u_{ij+1}^{n+1}+u_{ij-1}^{n+1}\right)=u_{ij}^{n}$$

then,

$$-\frac{\Delta t}{\Delta x^2}\left(u_{i+1j}^{n+1}+u_{i-1j}^{n+1}\right)+\left(\frac{1}{2}+\frac{2\Delta t}{\Delta x^2}+\right)u_{ij}^{n+1}+\left(\frac{1}{2}+\frac{2\Delta t}{\Delta y^2}\right)u_{ij}^{n+1}-\frac{\Delta t}{\Delta y^2}\left(u_{ij+1}^{n+1}+u_{ij-1}^{n+1}\right)$$
$$=u_{ij}^{n}$$

Let

$$\alpha=\frac{\Delta t}{\Delta x^2}\qquad\text{and}\qquad\beta=\frac{\Delta t}{\Delta y^2}$$

so we can obtain the final form:

$$-\alpha\left(u_{i+1j}^{n+1}+u_{i-1j}^{n+1}\right)+\left(\frac{1}{2}+2\alpha\right)u_{ij}^{n+1}+\left(\frac{1}{2}+2\beta\right)u_{ij}^{n+1}-\beta\left(u_{ij+1}^{n+1}+u_{ij-1}^{n+1}\right)\qquad(1.7)$$

Let

$$\Delta x\;=\;\Delta y\;=\;0.2\quad\text{and}\quad\alpha\;=\;\beta\;=\;0.5$$

When computing the previous values, we can find the final equation:

$$-0.5\left(u_{i+1j}^{n+1}+u_{i-1j}^{n+1}\right)+(1.5)\,u_{ij}^{n+1}-0.5\left(u_{ij+1}^{n+1}+u_{ij-1}^{n+1}\right)=u_{ij}^{n} \tag{1.8}$$

Therefore, we can stack the unknowns $u_{(ij)}^{n+1}$ into a single vector u as a result of the linear system $Au = b$ [28]. However, the next part demonstrates how to stack the unknowns $u_{(ij)}^{n+1}$ into a matrix by using the Sylvester equation.

1.2 Sylvester Equation

The Sylvester equation has a key role in control theory of the structured matrices [25], and it is a matrix equation of the form:

$$AX + XB = C \tag{1.9}$$

where $A \in \mathbb{R}^{(n\times n)}, B \in \mathbb{R}^{(m\times m)}, C \in \mathbb{R}^{(n\times m)}, and X \in \mathbb{R}^{(n\times m)}$. We can write (1.9) as the Sylvester form:

$$TU + UT = F \tag{1.10}$$

where the size of T, U and F is $n \times n$.

Subsequently, when applying the tridiagonal matrix equation for (1.9), T will be $= triding(-0.5, (1.5), -0.5)$, and $u_{ij}^n = u(x_i, y_j, t_n)$ where t_n can be determined as a specific time step. Also, (x_i, y_j) are the internal grid points for $i, j = 1, 2, \ldots, n$. The system has n unknowns in each direction which means the total of n unknowns is n^2. The matrix equation could be written as following:

$$\begin{pmatrix} 1.5 & -0.5 & 0 & \dots & 0 & 0 \\ -0.5 & 1.5 & -0.5 & \dots & 0 & 0 \\ 0 & -0.5 & 1.5 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1.5 & -0.5 \\ 0 & 0 & 0 & \dots & -0.5 & 1.5 \end{pmatrix} \begin{pmatrix} u_{11}^{n+1} & u_{12}^{n+1} & \dots & u_{1j}^{n+1} & \dots & u_{1n}^{n+1} \\ u_{21}^{n+1} & u_{22}^{n+1} & \dots & u_{2j}^{n+1} & \dots & u_{2n}^{n+1} \\ \vdots & \vdots & \ddots & \vdots & \dots & \vdots \\ u_{i1}^{n+1} & u_{i2}^{n+1} & \dots & u_{ij}^{n+1} & \dots & u_{in}^{n+1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{n1}^{n+1} & u_{n2}^{n+1} & \dots & u_{nj}^{n+1} & \dots & u_{nn}^{n+1} \end{pmatrix}$$

$$+ \begin{pmatrix} u_{11}^{n+1} & u_{12}^{n+1} & \dots & u_{1j}^{n+1} & \dots & u_{1n}^{n+1} \\ u_{21}^{n+1} & u_{22}^{n+1} & \dots & u_{2j}^{n+1} & \dots & u_{2n}^{n+1} \\ \vdots & \vdots & \ddots & \vdots & \dots & \vdots \\ u_{i1}^{n+1} & u_{i2}^{n+1} & \dots & u_{ij}^{n+1} & \dots & u_{in}^{n+1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{n1}^{n+1} & u_{n2}^{n+1} & \dots & u_{nj}^{n+1} & \dots & u_{nn}^{n+1} \end{pmatrix} \begin{pmatrix} 1.5 & -0.5 & 0 & \dots & 0 & 0 \\ -0.5 & 1.5 & -0.5 & \dots & 0 & 0 \\ 0 & -0.5 & 1.5 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1.5 & -0.5 \\ 0 & 0 & 0 & \dots & -0.5 & 1.5 \end{pmatrix} = F_{ij}^n$$

In this section, the derivation of the model problem is explained by using 2-d Heat equation. Additionally, it was built the Sylvester equation in order to stack all unknowns n in each direction. In the next part the reader will come across the project's structure and categories in order to have a clear idea of each section.

1.3 Project Structure

The structure of the project is as follows:

- **Section 2** is a scope and schedule that demonstrates the desired principal aims of this project, a number of key objectives in order to achieve our goals. Also, set the mandatory deliverables, show a detailed plan, the methodology upon which the project is based and the attendant risks.
- **Section 3** includes the essential content in this project, illustrates a wide variety of methods for solving the Sylvester equation.
- **Section 4** provides an evaluation of the project aims and objectives.

2 Scope and Schedule

2.1 Aim

The aim of this study is to search, apply, and compare a verity of matrix equation methods extensively. The linear matrix methods need to be applied, analysed and evaluated to determine which solver has a good result according to specific measurements.

2.2 Objectives

The following objectives have been identified in order to achieve the aims of the project:

- To use centred finite difference approximation to discretise the Heat equation domain, and implicit Euler method to solve the time.
- To build a matrix of our equation has the Sylvester form.
- To identify and study specific methods to be implemented on the given problem.
- To conduct a comprehensive literature review on the chosen solvers from various sources.
- To expand and use my programming experience and skills to implement the chosen methods for solving the heat equation appropriately by using Python.
- To evaluate each method and compare the results to decide which solver has best performance on the equation.
- To write a clear report to present the work that is carried out in this study by using LaTeX.

2.3 Deliverables

Herein, the deliverables of this project include:

- The project report explaining all strategies that have been determined.
- Code implementation for the matrix solvers in this project.

2.4 Planning and Project Management

The Gantt Chart methodology was used to plan this project. The main process in throughout the project is the trial and error process which explains why the project is exploratory and rending the project a repetitive and monotonous process.

2.4.1 Initial Plan

An initial project schedule was created depending on the following tasks:

Task / Period	March	April	May	June	July	August
Planning						
Find the problem						
Research						
Find methods						
Studying and applying						
Coding						
Writing the final report						

Table 1: Initial project plan.

2.4.2 Revised Plan

Task / Period	March				April				May				June				July				August			
	Weeks																							
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Scoping and Planning																								
Find the Problem																								
Background Research																								
Finding Solvers																								
Related Work Analysis																								
Intensive Research																								
Studying and Solving																								
Implementation and Evaluation																								
Report Writing																								
Proofreading																								
Submission																								

Table 2: Revised project plan.

2.5 Methodology

The methodology has been used for this project was rely on studying the chosen numerical solvers from a large number of different scientific sources. The fundamental part is to perform a series of experiments to compute the exact and computed solutions in a specific time until the author obtain a good convergence. Moreover, The programming language was implemented by Python, and the final report was written by LaTeX.

2.6 Risk Assessment

The most serious risk of this project is we have not access to the University laboratories due to COVID-19 crisis. Consequently, this will probably affect the accuracy of the code implementation results because of limited properties of the personal computer used in the calculations.

3 Matrix Solvers

Various solvers will be explored in this section, which can significantly contribute to solve a specific matrix equation in Sylvester form. As mentioned earlier, the Sylvester equations have been a remarkable achievements in many application, for instance the control theory, interpolation, signal processing and model reduction problems [8], [14], [30]. Initially, we need to form our specific model problem.

We need now to discretise the domain of our equation is:
$\mathcal{D} = \{(x, y) : 0 \leq x \leq 1, 0 \leq y \leq 1\}$on $[0, T]$ so $u = u_0$ at $t = 0$, and $u : \mathcal{D} \to \mathbb{R}$, let this $u_{(x,y,t)} = g(t) \sin(2\pi x) \sin(2\pi y)$ be the exact solution of the equation:

$$\begin{aligned} u_t - u_{xx} - u_{yy} &= f \text{ on } \mathcal{D} \\ u &= 0 \text{ on } \partial\mathcal{D} \end{aligned} \tag{3.1}$$

which gives:

$$\begin{aligned} u_t(x, y, t) &= g'(t) \sin(2\pi x) \sin(2\pi y) \\ u_{xx}(x, y, t) &= -4g(t) \sin(2\pi x) \sin(2\pi y) \\ u_{yy}(x, y, t) &= -4g(t) \sin(2\pi x) \sin(2\pi y) \\ f = u_t - u_{xx} - u_{yy} &= \left[g'(t) + 8\pi^2 g(t)\right] \sin(2\pi x) \sin(2\pi y) \end{aligned} \tag{3.2}$$

We can then rewrite (3.2) as follows:

$$f(x, y, t) = \left[g'(t) + 8\pi^2 g(t)\right] \sin(2\pi x) \sin(2\pi y) \tag{3.3}$$

where,

$$\begin{aligned} g(t) &= \epsilon^{-8\pi^2 t} \\ \frac{\partial g}{\partial t} = g'(t) &= -8\pi^2 \epsilon^{-8\pi^2 t} \\ g'(t) &= -8\pi^2 g(t) \end{aligned}$$

From equation (3.3) we can proof:

$$f(x, y, t) = \left[-8\pi^2 g(t) + 8\pi^2 g(t)\right] \sin(2\pi x) \sin(2\pi y)$$

$$f(x, y, t) = 0 \quad \text{at} \quad g(t) = \epsilon^{-8\pi^2 t} \tag{3.4}$$

The next is the final equation which can be solved:

$$u_{(x,y,t)} = \epsilon^{-8\pi^2 t} \sin(2\pi x) \sin(2\pi y) \tag{3.5}$$

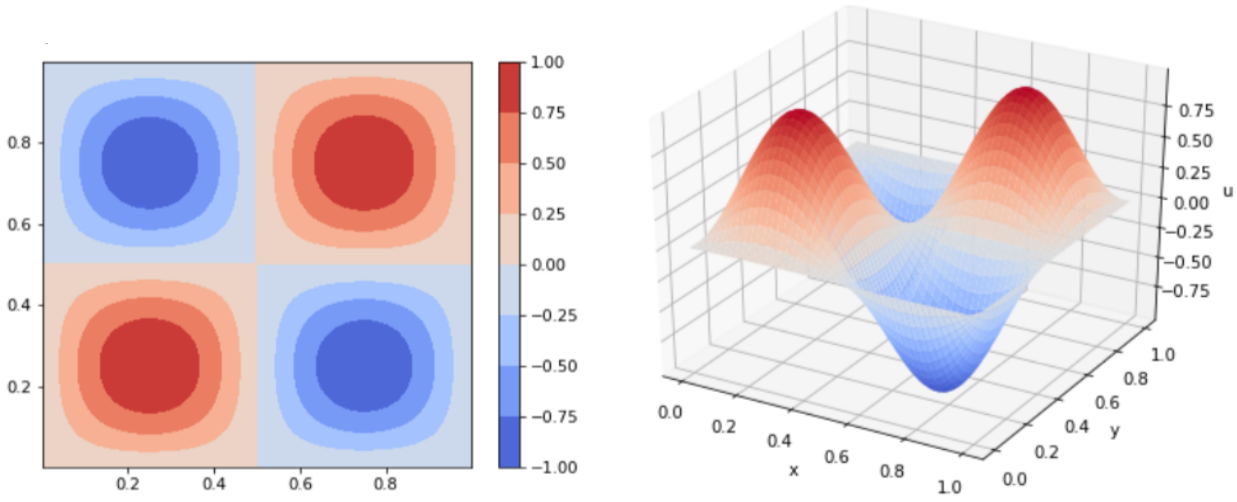
We need, now, to discretise the domain of our system by using the centred finite difference approximation with uniform grid in space, and Implicit Euler method in time:

$$-\alpha\left(u_{i+1j}^{n+1}+u_{i-1j}^{n+1}\right)+\left(\frac{1}{2}+2\alpha\right)u_{ij}^{n+1}+\left(\frac{1}{2}+2\beta\right)u_{ij}^{n+1}-\beta\left(u_{ij+1}^{n+1}+u_{ij-1}^{n+1}\right)=u_{ij}^{n} \quad (3.6)$$

with $u_{ij}^{n}=\epsilon^{-8\pi^2 t}\sin\left(2\pi x\right)\sin\left(2\pi y\right)$, to solve it at each internal grid point for $i,j=1,...,n$, n is the total number of unknowns in each direction. We can form the matrix of this equation as the next:

$$TU+UT=F \quad (3.7)$$

where $T=tridiag\left(-0.5,(1.5),-0.5\right)$ and $U_{ij}^{n}=u\left(x_i,y_j,t_n\right)$, and t_n can be determined as a specific time step. All $F_{ij}^{n}=\epsilon^{-8\pi^2 t}\sin\left(2\pi x\right)\sin\left(2\pi y\right)$ are matrices of size $n\times n$, so this form can be implemented by various solvers which will be explored in the third section, then compare the results to the exact solution $u_{(x,y,t)}=g(t)\sin\left(2\pi x\right)\sin\left(2\pi y\right)$. The following Figures are plots of the exact solution with $n=1519$ and $t=0.001$.



(a) 2-d plot of the solustion

(b) 3-d plot of the solution

Figure 3: Plots of the solution with $n=1519$ at $t=0.001$.

All the illustrated algorithms in the next section will be tested and analysed according to several criteria, which will be explain later. The initial aim of these criteria is to explore the performance of each solver, compare it with each other, and discover the best performance among them. Each of the implemented algorithm will be performed in a time not exceeding 5000 seconds in order to compute a solution. If an algorithm exceeds that time for a certain problem size, it will be mentioned in the results tables. Additionally, if an algorithm needs more memory, it will be mentioned in the related context. These algorithms will be applied and tested by a personal computer (PC) with `8GB of RAM and 8 cores`.

The following are our measurements to evaluate the performance of these algorithms:

- n: It is a problem size and denoting the in each direction for our system, where n^2 is the total number of unknowns. Also, $h = \frac{1}{n}$ is the step size and depends on the number of n, which the starting problem size will be $n = 189$.
- t: It is the time step, which will be different for each n value (problem size).
- Time(s): It shows the execution time for each time step in a specific problem size for each solver in seconds to compute a solution to a particular problem size.
- $||u - u_h||_{L^\infty}$: It means the maximum error difference between the exact and computed solution for u, it can be defined as:

$$||u - u_h||_{L^\infty} = \max_{ij} |u(x_i, y_i, t_n) - u_{ij}^n|$$

- $||u - u_h||_{L^2}$: It means the average error difference between the exact and computed solution for u, it can be defined as:

$$||u - u_h||_{L^2} = \sqrt{h^2 \sum |u(x_i, y_j, t_n) - u_{ij}^n|^2}$$

- Experimental order of convergence (EOC): It measures the rate of each method, since we use the centred finite approximation this should be close to 2. It can be defined as:

$$\text{eoc}(i) = \frac{\log(E_i/E_{i-1})}{\log(h_i/h_{i-1})}$$

  where E_i is represent the error (the chosen is the maximum error difference), and h_i is the step size.

- Experimental order of growth (EOG): It measures the order of the execution time growth of each method. It can be defined as:

$$\text{eog}(i) = \frac{\log(t_i/t_{i-1})}{\log(n_i/n_{i-1})}$$

  where t_i is the total time and n_i is the number of unknowns (the problem size).

- No_Iter (for the iterative algorithms): It is the number of the iterations taken for each method to converge to the given convergence tolerance.

3.1 Overview

Linear matrix methods for instance: direct methods (Kronecker product, Bartels-Stewart algorithm, Similarity Transformation and Shifted System); and iterative methods (Gradient-Based Iterative method, Modified Conjugate Gradient Method and Preconditioned MCG) have had significant attention from the early days, and are still being studied by researchers who are studying the use of deep tools of functional analysis as well as matrix theory. The area has considerably progressed in developing and expanding encompassed by the research and analysis of new and challenging functions more often encompassed by solving application problems. Findings provided by the researches mostly use equations and numerical strategies in their studies to explain their mathematical concepts [17]. It worth mentioning that this study focuses on finding out how these linear techniques have an impact on solving the Sylvester system.

3.2 Direct Methods

Theoretically, direct methods can compute the exact solution to a problem in a finite number of steps. Neri [24] indicates the purpose of the direct methods, which in turn used to rewrite the linear equation systems in a new form to be solved easily by performing a set of transformations in the matrix.

3.2.1 Kronecker Product

Kronecker Product is important for solving signal processing problems, the theory of linear matrix equation, and linear algebra equations. The Kronecker product which takes the name after the famous German mathematician, Leopold Kronecker is a highly investigated field in algebraic topics. Previous research has shown that in 1858, Johann Georg Zehfuss in [32] presented a published paper containing the widely known determinant conclusion

$$|A \otimes B| = |A|^n \ |B|^m$$

with square matrices A and B having the order n and m.

There are different applications of the Kronecker product to the Sylvester system including the system theory, usage of matrix equations, system identification, and matrix calculus application. Ali et al. [2] expounded the formulas for exponential functions associated with Kronecker products. Studies by [7] led to the development of generalizations of the Kronecker product related to the vector operator.

There has been a number of research work around the topic of Kronecker Product and

the provided solutions to the Sylvester matrix. Recently, it was proposed innovation of numerical algorithms by [23], which are computationally efficient and based on the hierarchical principle of identification for the equations of the Sylvester matrix. The coupled matrix equations are also included. Additionally, the study also contained iterative algorithms that were applicable in the Sylvester-conjugate matrix equations. The Kronecker product is beneficial in the Sylvester matrix to step-wise show the accuracy of the solution proposed in the matrix equation. The study came up with a new finding concerning the singular value normally identified in the Kronecker product that defined the matrix of the Vec-permutation in the Sylvester matrix. The findings were able to prove that the Kronecker product is important in solving the Sylvester matrix via enabling easy identification of mixed products in the matrix equations hence mathematicians and researchers can make their conclusions concerning the vector operator efficiently. The method also provides easy computational steps for the equations in the linear matrices as well as facilitating detailed accountability for available literature with uniformity and orderliness. Equations are used in the Kronecker product method to find solutions to the Sylvester matrix, which facilitates the identification of positive properties that are evident in the Sylvester matrix [9]. Consequently, The matrix could be arbitrary providing all available degrees of freedom in the matrix equation.

Accordingly, the Kronecker Product will be applied to the matrix form of our equation, and rewrite $TU + UT = F$ as a vector linear system [18]. We can write the Sylvester equation $AX + XB = C$ as a linear system as following:

$$Ax = c \tag{3.8}$$

with $\mathcal{A} = I \otimes A + B^* \otimes I$, I is the identity matrix and B^* is the conjugate transpose of B, $x = \text{vec}(X)$ and $c = \text{vec}(C)$, and both vectors were reshaped by using `vec`, also x and c have size n^2. This symbol $\otimes$ is denoted to the Kronecker Product, and $\mathcal{A}$ has dimension $n^2 \times n^2$.

then, the linear system for the matrix equation $TU + UT = F$ is:

$$Tu = f \tag{3.9}$$

with $\mathcal{T} = I \otimes T + T \otimes I$, has dimension n^2, and $u = vec(U)$ and $f = vec(F)$ are vectors of size n^2.

Therefore, this is the exact linear system which obtained from equation (1.8). All

unknowns $u_{(ij)}^{n+1}$ stacked into a single vector. As a result of the matrix $\mathcal{T}$ is sparse, we can solve the equation by using a `sparse.linalg.spsolve` from the SciPy library, which is a direct sparse solver. direct sparse solver.

Sparse.linalg.spsolve

Sparse solver aims to solve linear systems that have time complexity $O(n^3)$ in size n by using LU decompositions. However, our linear system has size n^2 which might be expected to run on more than $O(n^3)$, but not greater than $O(n^6)$.

To analyse the numerical solution of Sylvester equation, we firstly need to apply the Kronecker Product in order to form a linear system. The system can therefore be solved by `sparse.linalg.spsolve` to obtain some computational results at different node points with step size h in many different time steps. For example, Table 3 illustrates that the first problem size is $n = 189$, and the time step is $t = 1e^{-07}$. The idea is to observe the algorithm behaviour in several small time steps with large problem sizes for the purpose of comparing the results to see the effect of these two parameters as shown in the table below.

As can be observed by the results, the errors decreased by increasing the number of nodes. As we increased n the Experimental Order of Convergence (EOC) approaches to 2 for all n and the value of Experimental Order of Growth (EOG) grows beyond 3, which illustrates that this algorithm has awful and then cubic time complications exist. One of the observation noticed from table 3 is for $n = 3039$ our present method generates a memory error result in memory errors. This is because of the occurrence of memory error because of `sparse.linalg.spsolve` try to enumerate the LU decomposition of matrix $\mathcal{T}$ which breaks up the sparsity of the matrix and therefore more memory to store it.

The numerical results,in different time steps, which we have found by using the Kronecker Product method are almost similar to the exact solution which we can easily observe from the table because the difference between computational and exact solution is negligible and we obtained our desired results.

n	t	Time(s)	$\|\|u - u_h\|\|_{L^\infty}$	$\|\|u - u_h\|\|_{L^2}$	EOC	EOG
189	$1e^{-07}$	0.44900	0.00109	0.00054	-	-
379	$2.5e^{-08}$	2.76840	0.00027	0.00013	1.97533	2.61430
759	$6.25e^{-09}$	24.14004	$6.80304e^{-05}$	$3.40153e^{-05}$	1.99974	3.11837
1519	$1.5625e^{-09}$	211.42661	$1.69860e^{-05}$	$8.49304e^{-06}$	2.02403	3.12768
3039	Memory Error					

Table 3: Results obtained from solving the linear system $\mathcal{T}u = f$ using the direct solver `sparse.linalg.spsolve`.

3.2.2 Bartels–Stewart Algorithm

Over the recent past, the Bartels–Stewart method has been widely researched. Previous research have shown that this algorithm is applicable in the area of linear algebra, and is used in finding solutions to the Sylvester matrix equation which appears in the form of

$$AX - XB = C$$

Studies in [25] have shown that the method is suitable for small to medium-sized Sylvester matrix equations as well as Lyapunov equations. In more than two decades, Dinčić's paper is the only reference material that provides evidence for the direct computation of the Cholesky factor for finding X in the equations for small to medium solution n. Asari and Amirfakhrian in [5], and [31] indicated that this algorithm is ranked as the first approved method of systematically and accurately applying to equations. Researchers identified the technique as one that used the concept of the decomposition of real Schur of unknowns A and B to transform the equation $AX - XB = C$ to form a triangular applicable system that would allow the solving of equations using both forward and backward substitution [12].

The latest research [11] carried out resulted in the introduction of a revised version that is simpler to understand and apply. The newest algorithm was introduced by Dehghan and Shirilord which is family identified as Hessenberg–Schur. The version has provided a solution on how to solve the Sylvester matrix equations efficiently and the advantage is that the usage of the standard approach when the value of X is from a small to a moderate size. There are various applications of this method when solving the Sylvester matrix which include: the ability to control, observability which is used for controller design and study's stability that help in solving PDEs found in domains. Additionally, the Bartels–Stewart method is used in designing an optimal control as well as the computation of an order model (reduced) that contains a balanced truncation. The technique makes use of equations since they were found to be more efficient and accurate than Matlab (lyap.m) function as evidenced by the numerical figure of the previous research done.

Therefore, the Sylvester equation $AX - XB = C$ can be solved via the Bartels–Stewart algorithm [6], [12] by computing the schur decompositions of the coefficient matrices A and B. They also used to transform $AX - XB = C$ into an equivalent system that has upper and lower triangular matrices. Then, we can solve the triangular system through using one element at a particular time, and using its result to be able to find the original equation solution.

Generally, the algorithm is as follows:

- Decompose and compute Schur for matrix A and B : $A = PRP^*$ and $B = QSQ^*$.
- Solve $R^*Y + YS = \hat{C}$ for Y, where $\hat{C} = P^*CQ$.
- Compute $X = PYQ^*$.

Where P and Q are unitary matrices, and R and S are upper/lower triangular matrices, respectively, including all eigenvalues of A and B along their diagonal [4].

The cost for each part in the algorithm can be measured by dividing its parts. Firstly, QR algorithm will be presented from the SciPy library. It aims to compute a Schur factorization because of its ability to solve the eigenvalue problems of our square matrices with considering to its computationally expensive that requires $O(n^3)$ [16]. Secondly, compute each entry of Y as:

$$Y_{ij} = \hat{C}_{ij} - \sum_{p=1}^{i-1} R_{ip} Y_{pj} - \sum_{q=1}^{j-1} Y_{iq} S_{qj} \tag{3.10}$$

for $i, j = 1, ..., n$. The QR algorithm will have a for loop running from $i = 1, ..., n$, and a nested for loop running from $j = 1, ..., n$ then two nested for loops run from $p = 1$ to $i - 1$ and $q = 1$ to $j - 1$, due to its time complexity $O(n^3)$. A matrix multiplication will be the final step in consideration of our expectation this algorithm can be run in approximately $O(n^3)$.

So, for our equation $TU + UT = F$ the algorithm is simplified. It can be written as follows:

- Decompose and compute Schur for matrix T: $T = PRP^*$.
- Solve $R^*V + VR = \hat{F}$ for V, where $\hat{F} = P^*FP$.
- Compute $U = PVP^*$.

The eigenvectors of matrix T are orthogonal as a result of being symmetric, and R is a diagonal matrix so it has the eigenvalues of matrix T on the diagonal. Let us find the Schur decomposition for the general matrix $A = PRP^*$ by applying its 4 steps:

1. Find the eigenvalues of matrix A.
2. Find the eigenvectors of matrix A.
3. Compute the orthonormal set of the eigenvectors by using the eigenvectors of A.
4. The columns of P is formed by the eigenvectors which found from the previous step 3, so $R = P^*AP$.

Then, to make R be a diagonal matrix with $T's$ eigenvalues on the diagonal, we need to normalise the eigenvectors of T in step number 3 of the decomposition process as a result of the eigenvectors are orthogonal already.

The purpose of making matrix R as a diagonal is to try to reduce the complexity to $O(n^2)$ in the second step. This means that only the diagonal elements will be calculated. Therefore, our expectation is this algorithm probably runs in the worst case in $O(n^3)$. Also, we can deduce from the first step, it might take more time than the other steps. However, the algorithm can run faster than the general case. Next part illustrates the results of using two solvers of applying this method.

linalg.solve_sylvester

The Scipy solver has an inherent solver which is used for solving Sylvester equations, `linalg.solve_sylvester` which manipulates the Bartels-Stewart algorithm for generalized testing of this algorithm results by the usage of this solver observations are given in table 4. It was observed from table 4 that these results obtained by this solver are much vigorous than using the Kronecker product and `spsolve`. The errors are almost the same as errors in table 3 but this method consuming less time as compared to the previous one. For the large $n = 759$ and small time step $t = 6.25e^{-09}$ the value of EOC is very close to 2. In other best observation for all time steps t, the experimental order of converges round about 2. In the same parallel way effect of this method on EOG appears to change depending on the size of the problem and the step of the time, but this method has obtained result gretear then cubic time complexity as in this problem size $n = 3039$. After particular n, it is difficult for this method to analyse the steps of this solver directly, `Solve_sylvester` generates the use of `LAPACK` library for solving linear algebraic problems to find the Schur decomposition in the initial step. The basic function of `LAPACK` is to solve many different methods for evaluating eigenpairs of symmetric tridiagonal matrices. It is the same as method changes the results of any

problem depending on the problem size and time step, which causes the change in the result of EOC and EOG for $n = 6079$.

n	t	Time(s)	$\|\|u - u_h\|\|_{L^\infty}$	$\|\|u - u_h\|\|_{L^2}$	EOC	EOG
189	$1e^{-07}$	0.12292	0.00109	0.00054	-	-
379	$2.5e^{-08}$	0.64528	0.00027	0.00013	1.97533	2.38315
759	$6.25e^{-09}$	3.60728	$6.80304e^{-05}$	$3.40153e^{-05}$	1.99974	2.47833
1519	$1.5625e^{-09}$	46.55207	$1.69860e^{-05}$	$8.49304e^{-06}$	2.02403	3.68630
3039	$3.90625e^{-010}$	581.70890	$4.24376e^{-06}$	$2.12188e^{-06}$	2.00086	3.64163
6079	$9.765625e^{-011}$	2463.19220	$1.06059e^{-06}$	$5.30296e^{-07}$	1.91512	2.08166
12159	Memory Error					

Table 4: Results obtained from solving the `linalg.solve_sylvester`.

Simplified Bartels-Stewart Implementation

The results given in the following table, Table 5, are obtained using the simplified version of the Bartels-Stewart algorithm. As can be seen from the results, this solver is much faster than the previous solver, both of the errors are very similar to the errors in the `linalg.solve_sylvester` method. It is notable that the EOC results for all n are nearly the same for both of the previous two solvers. Another positive point for this method is that the EOG results are better than cubic time complexity, but began to rise very slightly when $n = 3039$ and 6079. Consequently, based on the results in Table 5 the simplified Bartels–Stewart is better than the `linalg.solve_sylvester` and `sparse.linalg.spsolve` solvers. This is due to the second step which led to reduce the time cost to $O(n^2)$. In addition, to comparison of the abovementioned methods, we have noticed that Bartels-Stewart method is more efficient than Kronecker Product method because when $n = 3039$ we obtained memory error while in Bartels-Stewart method we faced the memory error at $n = 12159$. The results of the this algorithm more efficient and accurate because we calculate the Schur decomposition in the first step which improves the results and gives us better desirable accuracy.

n	t	Time(s)	$\|\|u - u_h\|\|_{L^\infty}$	$\|\|u - u_h\|\|_{L^2}$	EOC	EOG
189	$1e^{-07}$	0.16015	0.00109	0.00054	-	-
379	$2.5e^{-08}$	0.59504	0.00027	0.00013	1.97533	1.88640
759	$6.25e^{-09}$	2.21739	$6.80304e^{-05}$	$3.40153e^{-05}$	1.99974	1.89421
1519	$1.5625e^{-09}$	9.98802	$1.69860e^{-05}$	$8.49304e^{-06}$	2.02403	2.76938
3039	$3.90625e^{-010}$	60.41356	$4.24376e^{-06}$	$2.12188e^{-06}$	2.00086	2.59543
6079	$9.765625e^{-011}$	404.40613	$1.06059e^{-06}$	$5.30296e^{-07}$	1.91512	2.74224
12159	Memory Error					

Table 5: Results obtained from solving the Bartels–Stewart Algorithm.

Table 6 lists the timing results of each step used in the simplified Bartels-Stewart algorithm. The headings of this table represent the problem size n, time step t and the steps for this algorithm. As it is clear in the table, the first step $T = PSP^*$, which is to compute the Schur decompositions, cost the most time to implement this algorithm, and this is because it has four steps. Also, we can see from Table 6, when the time step is decreased and the problem size is increased, the total time increases which affects the EOG.

n	t	Step 1	Step 2	Step 3	Total time	EOG
189	$1e^{-07}$	0.05788	0.10072	0.00155	0.16015	-
379	$2.5e^{-08}$	0.31777	0.26938	0.00787	0.59504	1.88640
759	$6.25e^{-09}$	1.22446	0.93776	0.05516	2.21739	1.89421
1519	$1.5625e^{-09}$	5.98040	3.57829	0.42933	9.98802	2.76938
3039	$3.90625e^{-010}$	40.99865	16.08612	3.32878	60.41356	2.59543
6079	$9.765625e^{-011}$	301.03569	76.93473	26.43571	404.40613	2.74224

Table 6: The time for each step of the Schur decompositions in different time steps using Bartels–Stewart Algorithm.

3.2.3 Similarity Transformation

Similarity Transformation is a conformal way of mapping and whose transformation matrix A is written as $A^{'} = BAB^{-1}$ whereby both A and $A^{'}$ are referred to as similar matrices [29]. Previous research in the last decade shows that the determinant in a similarity transformation that appears in a matrix is equally the same as that of the original matrix.

There are various applications of the similarity transformation technique to the Sylvester matrix including the computation of commutative algebra when testing the presence of a common factor between two polynomials that are non-constant. In such situation, the determinant is equal to zero. Additionally, Similarity transformation facilitate easy identification of the greatest common divisor using the Sylvester matrix method [20].

Based on the above contribution of this method to solving the Sylvester equation $AX + XB = C$, we will explain it also computationally. We assume that the coefficient matrices A and B can be diagonalised, this algorithm works by computing the eigende composition in order to reform the equation to obtain the solution.

In general case the method can be as follows:

1. Computing the eigende compositions $P^{-1}AP = \text{diag}(\lambda_1, ..., \lambda_n)$ and $Q^{-1}BQ = \text{diag}(\mu_1, \ldots, \mu_m)$:
   (a) Computing the eigenpairs of A and B.
   (b) Computing the inverses of P and Q.
2. Computing the solution of $X = P\widetilde{X}Q^{-1}$, where $\widetilde{X}_{ij} = \frac{\widetilde{C}_{ij}}{\lambda_i + \mu_j}$ and $\widetilde{C} = P^{-1}CQ$.

The eigenvalues of A are $(\lambda_1, ..., \lambda_n)$, while the eigenvalues of B are $(\mu_1, \ldots, \mu_m)$, and the eigenvectors of A and B are the columns of P and Q, respectively. With regard to finding the time complexity, we can break down and analyse the algorithm into its components. Step one has two parts; first computing the eigenpairs of A and B which runs generally in $O(n^3)$, second part is computing the inverse of P and Q which is at most in $O(n^3)$. Then, the last step is a simple matrix multiplication, which is no greater than $O(n^3)$. Our expectation therefore is this method runs in, the worst case, complexity $O(n^3)$, with the dominating of the second step b since it is solved in a specific time.

The algorithm is simplified for $TU + UT = F$. The steps as follows:

1. Computing the eigende compositions $P^{-1}TP = \text{diag}(\lambda_1, ..., \lambda_n)$:
   (a) Computing the eigenpairs of T.

(b) Computing the inverses of P.

2. Computing the solution of $U = P\widetilde{U}P^{-1}$, where $\widetilde{U}^n_{ij} = \frac{\widetilde{F}^n_{ij}}{\lambda_i+\mu_j}$ and $\widetilde{F} = P^{-1}FP$ in a specific time n.

As mentioned before that the eigenvalues of T are $(\lambda_1, ..., \lambda_n)$, and the eigenvectors of T are the columns of P. This significantly reduces cost of time complexity in the first step. Then all the eigenpairs of T will be computed and stored and the inverse of P will be calculated in order to reduce the time complexity to $O(n^2)$, due to the fact that P is a unitary matrix.

Numpy's linalg.eig

Similarity Transformation is another method that can be used to solve the Sylvester equations. The list of results in Table 7 is obtained by applying `Numpy's linalg.eig`. To the best our knowledge that the results of the errors are the same as for the previous algorithms. For the results of the experimental order of convergence EOC are closest to 2, and are completely identical to the outcomes of the `linalg.solve_sylvester`. The experimental order of growth (EOG) demonstrates that the algorithm has better than the cubic time complexity, except for the large n as 1519 and 3039 because they are worst than cubic time complexity, and for 6079 is close to 3. This means that when we increased the number of nodes and reduce the time steps, it is strongly affected on the EOG value. One of the disadvantage of this method is that it is not robust method as compare to previous Bartels-Steward method because of computational cost due to increase in the run time.

n	t	Time(s)	$\|\|u-u_h\|\|_{L^\infty}$	$\|\|u-u_h\|\|_{L^2}$	EOC	EOG
189	$1e^{-07}$	0.23627	0.00109	0.00054	-	-
379	$2.5e^{-08}$	0.77460	0.00027	0.00013	1.97533	1.71525
759	$6.25e^{-09}$	3.98970	$6.80304e^{-05}$	$3.40153e^{-05}$	1.99974	2.30772
1519	$1.5625e^{-09}$	45.04944	$1.69860e^{-05}$	$8.49304e^{-06}$	2.02403	3.43491
3039	$3.90625e^{-010}$	454.04319	$4.24376e^{-06}$	$2.12188e^{-06}$	2.00086	3.27690
6079	$9.765625e^{-011}$	2704.02770	$1.06059e^{-06}$	$5.30296e^{-07}$	1.91512	2.77466
12159	Memory Error					

Table 7: Results computing the eigenpairs by `Numpy.linalg.eig`.

Table 8 shows the timing results of each step of this method `Numpy's linalg.eig`. As can be seen from the following results, step 2 which is computing U is considered the most dominate step since it is calculated in different small time steps. Also, with respect to computing the eigenpairs of T took more time compared to computing the inverse of P.

n	t	Step 1a	Step 1b	Step 2	Total time	EOG
189	$1e^{-07}$	0.10566	0.00328	0.12732	0.23627	-
379	$2.5e^{-08}$	0.32177	0.01147	0.44134	0.77460	1.71525
759	$6.25e^{-09}$	1.35670	0.06231	2.57068	3.98970	2.30772
1519	$1.5625e^{-09}$	6.60790	0.38269	38.05883	45.04944	3.43491
3039	$3.90625e^{-010}$	47.31120	2.59359	404.13839	454.04319	3.27690
6079	$9.765625e^{-011}$	349.99879	21.9915	2332.03737	2704.02770	2.77466

Table 8: The time of each step, computing the eigenvalues and eigenvectors by `numpy.linalg.eig`.

3.2.4 Shifted System

The shifted system was an area of study that had great concentration through many researchers in the early decades. Simoncini [28] indicates that in Sylvester equation if A is large and B is small then the shifted linear system method can be used because Schur factorization requires a large amount of space which means that dense nature of a corresponding matrix creates an issue. With the help of shifted linear system the loss of generality can be minimized in which the equation will be visualized as follows:

$$AX + XB = C$$

This method is defined also as a projection method as explained in [28]. It has an important role in finding a solution to the Sylvester equation $AX + XB = C$. This method works on the base of calculating the eigende composition of the matrix of Sylvestre equation to reform the problems rather than solving n independent linear systems. To find the solution of X, it is essential to solve each of those linear systems together.

The following are the general steps of this method:

1. Compute the eigendecomposition $B = WSW^{-1}$:
   (a) Calculate the eigenpairs of B.
   (b) Calculate the inverse of W.
2. Solve the system $(A + s_iI)(\hat{X})_i = (\hat{C})_i$, where $(\hat{C})_i = CW$, for $i = 1, ..., n$.
3. Compute solution $X = \hat{X}W^{-1}$.

where $(\hat{X})_i$ is denoted to the i^{th} columns of $\hat{X}$, the columns of W are the eigenvectors of B, and $S = dig(s_1, ..., s_n)$ are the eigenvalues of B.

In general, the first step calculates the eigenpairs which runs in $O(n^3)$, computes therefore the inverse in roughly $O(n^2)$. Solving n linear systems is the second step. The time complexity could not be greater than $O(n^3)$ if a linear system solved directly such as the LU decomposition, in contracts, the time complexity of solving n of those linear systems may be more than $O(n^3)$ but no more than $O(n^4)$ if a direct solver is used, for example `solve_sylvester` and `sppsolve`. A matrix multiplication is the last step, and is no greater than $O(n^3)$. Consequently, we can expect this method runs in time complexity $O(n^4)$.

The following steps will be applied on our case $TU + UT = F$:

1. Compute the eigende composition $T = WSW^{-1}$:
   (a) Calculate the eigenpairs of T.

(b) Calculate the inverse of W.

2. Solve the system $(T + s_i I)(\hat{U})_i = (\hat{F})_i$, where $(\hat{F})_i = FW$, for $i = 1, ..., n$.
3. Compute solution $U = \hat{U}W^{-1}$.

where $(\hat{U})_i$ is denoted to the i^{th} columns of $\hat{U}$, the columns of W are the eigenvectors of T, and $S = dig(s_1, ..., s_n)$ are the eigenvalues of T.

The eigenpairs can be possibly calculated via using `Numpy.linalg.eig` in order to reduce the time complexity to $O(n)$ as used in the Similarity Transformation. Also, we can calculate W transpose instead of its inverse to reduce this to $O(n^2)$, and this is because W is unitary matrix. Hence, the expected point for this algorithm is that the second step of solving n linear systems will take the larger time.

The demonstrations of these results by using this method are given in Table 9. It is observed the outcomes of this method are not fast enough but efficient as compared to Bartels-Stewart in regards to the time factor. It consumes less time than other implemented methods. However, it increased the accuracy of the computed solution. The EOG illustrates that this algorithm has a cubic time complexity for large n which the value of EOG give us round about 3, except $n = 6079$ is a little more than 3. For $n = 12159$, this algorithm results in a memory error. Regarding the errors of this method there is no difference with the previous methods, and the results are the same.

n	t	Time(s)	$\|\|u - u_h\|\|_{L^\infty}$	$\|\|u - u_h\|\|_{L^2}$	EOC	EOG
189	$1e^{-07}$	0.32518	0.00109	0.00054	-	-
379	$2.5e^{-08}$	1.42463	0.00027	0.00013	1.97533	2.12313
759	$6.25e^{-09}$	10.26005	$6.80304e^{-05}$	$3.40153e^{-05}$	1.99974	2.84307
1519	$1.5625e^{-09}$	75.27210	$1.69860e^{-05}$	$8.49304e^{-06}$	2.02403	2.87235
3039	$3.90625e^{-010}$	576.86291	$4.24376e^{-06}$	$2.12188e^{-06}$	2.00086	2.93660
6079	$9.765625e^{-011}$	4858.94267	$1.06059e^{-06}$	$5.30296e^{-07}$	1.91512	3.07361
12159	Memory Error					

Table 9: Results solving n linear systems by `sparse.linalg.spsolve`.

The results are given in Table 10 are the outcomes of each step of the previous algorithm Shifted System. The results are confirmed our expectation which is the most expensive step is solving the n linear systems (the second step). What we can be clearly seen is for a large value of n as $n = 6079$ with small time step $9.765625e^{-011}$, the time taken in the second step is very large which is 4047.7 and this affects on the EOG value. Therefore, we can summarise from the table below that the total time is affected (increased) as the time step is reduced and the problem size is increased.

n	t	Step 1a	Step 1b	Step 2	Step 3	Total time	EOG
189	$1e^{-07}$	0.06621	0.00065	0.24857	0.00974	0.32518	-
379	$2.5e^{-08}$	0.32256	0.00087	1.02076	0.08042	1.42463	2.12313
759	$6.25e^{-09}$	1.40156	0.001254	8.13555	0.72167	10.26005	2.84307
1519	$1.5625e^{-09}$	6.53072	0.00335	61.64886	7.08915	75.27210	2.87235
3039	$3.90625e^{-010}$	48.46714	0.01125	471.86297	56.52153	576.86291	2.93660
6079	$9.765625e^{-011}$	357.02340	0.06922	4047.76757	454.08246	4858.94267	3.07361

Table 10: The time of each step to solve n linear systems.

3.3 Iterative Methods

Iterative methods are normally mathematical procedures that use a guess in the initial stage with the ultimate goal of generating a sequence that improves problems identified by providing approximate solutions [16]. Computational mathematics expresses the nth approximation which is usually derived from previous equations. Iterative methods can be convergent or heuristic-based methods. The convergent method occurs where convergence occurs on the corresponding sequence found in the initial approximations. On the other hand, heuristics methods use shortcuts to provide solutions within a set deadline where problems identified. Iterative methods are more preferred for solving nonlinear equations, unlike direct methods that solve small-medium scale mathematical equations using a series of many operations [13]. The privilege of iterative methods is that they are relatively cheaper compared to direct methods that are expensive.

3.3.1 Gradient Based Method

The gradient-based iterative methods make use of the principle of hierarchical identification for solving the Sylvester equations. Research shows that gradient-based iterative methods converge in various circumstances although at a very slow rate. Considering the possibility of preconditioning schemes to solve Sylvester equations, preconditioned gradient-based methods are normally derived through the selection of auxiliary equation matrices. Preconditioners are useful when the splitting of iterations linear system equations is necessary. Several studies by researchers have shown that the use of preconditioners is better and more efficient using the gradient-based iterative method, during the gradient's convergence [33]. It was noted that the convergence of PGBI is more linear and more efficient compared to the GBI method. For example, when comparing the convergence of GBI to PGBI, it was noted that the convergence of GBI was possible in 300 iterations while the GBI method required more than 1000 iterations for convergence to take place. The PGBI method is therefore the effective method in convergence as well as in solving the coupled Sylvester matrix equations.

Based on the above information, we can use this algorithm to solve our Sylvester equation $TU + UT = F$ by using the recursive sequences as:

$$U_k^{(1)} = U_{k-1}^{(1)} + \kappa T(F - TU_{k-1}^{(1)} - U_{k-1}^{(1)}T) \tag{3.11}$$

$$U_k^{(2)} = U_{k-1}^{(2)} + \kappa(F - TU_{k-1}^{(2)} - U_{k-1}^{(2)}T)T \tag{3.12}$$

where κ points out the step size, and the average of the sequences mentioned above can be resulted the approximation solution of U_k by the next equation:

$$U_k = \frac{U_k^{(1)} + U_k^{(2)}}{2} \tag{3.13}$$

The convergence of this solution will be occur if:

$$0 < \kappa < \frac{1}{\lambda_{\max}(T^2)} \tag{3.14}$$

the maximum eigenvalue of T^2 is represented by $\lambda_{\max}(T^2)$, and the eigenvalues squared are the eigenvalues of the square of a matrix. This is because an eigenvalue λ of A, which is the general matrix, and its eigenvector x. Hence, the value of κ should be very small as n is increased in order to converge.

The results in Table 11 obtained via `Numpy` which is using `linalg.eig`. As can be seen from these results that both of the errors are similar to the direct methods and the EOC, too. However, this method is slower than all the direct methods, as in this problem size $n = 1519$ takes more iteration 508 and costs 911.5 second to find the solution.
Regarding the EOG, this solver is demonstrated as better than cubic time complexity.

n	t	No.Iter	Time(s)	$\|\|u - u_h\|\|_{L^\infty}$	$\|\|u - u_h\|\|_{L^2}$	EOC	EOG
189	$1e^{-07}$	507	2.68500	0.00109	0.00054	-	-
379	$2.5e^{-08}$	508	17.66011	0.00027	0.00013	1.97533	2.70720
759	$6.25e^{-09}$	508	124.16595	$6.80304e^{-05}$	$3.40153e^{-05}$	1.99974	2.80849
1519	$1.5625e^{-09}$	508	911.51008	$1.69860e^{-05}$	$8.49304e^{-06}$	2.02403	2.87335
3039	Exceeded the specified time (5000)						

Table 11: Results obtained from solving Gradient-based method.

3.3.2 Modified Conjugate Gradient

The method is widely known for solving large complex matrix linear equations. The Conjugate gradient method (CG method) and the Modified Conjugate Gradient Method (MCG method) are identified to have the same level of convergence, and consequently their convergence speed is the same. The MCG method is an important one in engineering and field of science in dealing with solutions to linear matrix equations and with a high level of efficiency [1]. Researchers explain that large equation matrices are still being solved by iteration techniques or the transformation of real equations from the original one.
Sylvester equations are applicable in various fields including the composition of the matrix eigende, processing of images, the control theory, and reduction of models. The problem of how to solve large linear problems in the Sylvester matrix is widespread especially in elliptic PDEs and parabolic methods. Studies have therefore come up with two methods of solving the matrix equations including the standard direct method that was first introduced by Moghrabi [21]. The limitation of the method is that it cannot be applicable on large scale problems. The second method was the iterative technique that was first proposed by Zhang and Zhou [33] and was appropriate for solving large linear problems. The speed of convergence for MCG is directly related to the values available in the coefficient matrix. A greater concentration of singular values results in a higher speed of convergence. Notably, the speed of convergence is facilitated by the introduction of preconditioning principles which will result in a higher concentration of singular values identified in the coefficient matrix [27]
The preconditional needs resulted in the need for having an efficient algorithm that is parallel hence the introduction of the parallel preconditioned algorithm that was necessary to solve the large and sparse Sylvester linear matrix equation. The method enhanced the validity and increased the convergent rate with a greater number of applications compared to the parameter method.

The modified conjugate gradient MCG is suggested also via [15] to solve the Sylvester equation. As mentioned earlier, in the general algorithm $AX + XB = C$ the matrices A, B, C and$X$ are all of size $n \times n$, so the general algorithm as:

1. Choose initial guess $X^{(0)}$ and calculate:
   - $R^{(0)} = C - AX^{(0)} - X^{(0)}B$
   - $Q^{(0)} = A^T R^{(0)} + R^{(0)} + R^{(0)}B^T$
   - $Z^{(0)} = Q^{(0)}$
2. If $R^{(k)} <$ tol or $Z^{(k)} <$ tol, stop. Otherwise, return to step 3.
3. Calculate:

- $\gamma_k = \frac{[R^{(k)}, R^{(k)}]}{[Z^{(k)}, Z^{(k)}]}$, where $[R, R] = \text{tr}(R^T R)$ is the trace of the matrix $R^T R$
- $X^{(k+1)} = X^{(k)} + \gamma_k Z^{(k)}$
- $R^{(k+1)} = C - AX^{(k+1)} - X^{(k+1)}B$
- $Q^{(k+1)} = A^T R^{(k+1)} + R^{(k+1)} + R^{(k+1)} B^T$

and go to Step 2.

For our equation $TU + UT = F$, the algorithm will be as:

1. Choose initial guess $U^{(0)}$ and calculate:
   - $R^{(0)} = F - TU^{(0)} - U^{(0)}T$
   - $Q^{(0)} = TR^{(0)} + R^{(0)} + R^{(0)}T$
   - $Z^{(0)} = Q^{(0)}$
2. If $R^{(k)} < \text{tol}$ or $Z^{(k)} < \text{tol}$, stop. Otherwise, return to step 3.
3. Calculate:
   - $\gamma_k = \frac{[R^{(k)}, R^{(k)}]}{[Z^{(k)}, Z^{(k)}]}$
   - $U^{(k+1)} = U^{(k)} + \gamma_k Z^{(k)}$
   - $R^{(k+1)} = F - TU^{(k+1)} - U^{(k+1)}T$
   - $Q^{(k+1)} = TR^{(k+1)} + R^{(k+1)} + R^{(k+1)}T$

   and go to Step 2.

Nevertheless, this algorithm is not appropriate for solving our equation since the matrix multiplication is cost in each iteration. This is because it can be expected that this solver probably runs in worse than cubic time complexity, also take more memory size. This is conformed when it was implementing this method to solve this problem size $n = 189$, it generated memory error.

3.3.3 Preconditioned MCG

The conjugate gradient method is widely known by mathematicians as well as researchers as a technique that is useful in optimising linear and non-linear equations. The method was developed by Magnus and is mostly applied as an iterative method, although it is sometimes used as a direct method to provide a numerical solution [21]. Additionally, it is applicable for extremely huge systems which may be difficult to solve using a direct method. The conjugate gradient method arises from various perspectives which include Arnoldi variation including eigenvalue problems and the specialization of the technique method for optimization.

The conjugate gradient method is divided into two categories. Various studies identify the linear and non-linear conjugate gradient methods. The linear gradient method solves linear equations by finding the value of n conjugate vectors and eventually computing the coefficients α_k. The non-linear method is important in identifying the nature of iteration in the method [10].

Research shows that there is a pre-conditioned form that results in an iterative method that applies to sparse systems with different equations that arise from problems identified in the boundary. These include the two and three dimensional problems. Various advantages of the conjugate method are identified in solving the Sylvester matrix equations.

$$AX + XA = C$$

These include being the only technique that involves A multiplication with a vector A, where A is sparse and huge, it is an advantage since it may be unnecessary to store A in an $m \times m$ matrix. The only action required is to note down a special function for multiplying A and a vector. The method enhances easy solving of the Sylvester matrix due to a standard equation that facilitates forward and backward substitution [33]. The other advantage is that it provides a conducive space for solving the least square method encompassed by sparse and large matrix design. Finally, the technique is advantageous for large technical problems and provides a smooth solving method with easy steps hence efficiency and accuracy are appraised. Researchers identify the method as one that is included in the generalization in the minimization technique appropriate for non-quadratic functions.

The preconditioned MCG algorithm has a remarkable role in accelerating the convergence speed [15]. It can solve the Sylvester equation $AX + XB = C$ by solving the following equation:

$$X^{k+1} = \widetilde{A}X^{(k)}\widetilde{B} + \widetilde{C}$$

where:

$$\widetilde{A} = I + 2(\alpha A - I)^{-1}$$

$$\widetilde{B} = I + 2(\alpha B - I)^{-1}$$

$$\widetilde{C} = -2\alpha(\alpha A - I)^{-1}C(\alpha B - I)^{-1}$$

we can chose a value for parameter α, and the identity of the matrix is I. The value of α can be chosen as $\alpha = \pm\sqrt{n}/||A||_F$ or $\alpha = \pm\sqrt{n}/||B||_F$, where $||A||_F = \sqrt{[A, A]} = \sqrt{\text{tr}(A^T A)}$ as proposed in [15], in order to obtain a good convergence. After choosing α and calculating $\widetilde{A}, \widetilde{B}$ and $\widetilde{C}$, the algorithm will be as:

1. Choose initial point $X^{(0)}$ and calculate:
   - $R^{(0)} = -\widetilde{C} + X^{(0)} - \widetilde{A}X^{(0)}\widetilde{B}$
   - $Q^{(0)} = \widetilde{A}^T R^{(0)}\widetilde{B}^T + R^{(0)}$
   - $Z^{(0)} = Q^{(0)}$
2. If $R^{(k)} <$ tol or $Z^{(k)} <$ tol, stop. Otherwise, return to step 3.
3. Calculate:
   - $\gamma_k = \frac{[R^{(k)}, R^{(k)}]}{[Z^{(k)}, Z^{(k)}]}$
   - $X^{(k+1)} = X^{(k)} + \gamma_k Z^{(k)}$
   - $R^{(k+1)} = -\widetilde{C} + X^{(k+1)} - \widetilde{A}X^{(k+1)}\widetilde{B}$
   - $Q^{(k+1)} = \widetilde{A}^T R^{(k+1)}\widetilde{B}^T - R^{(k+1)}$
   - $\eta_k = \frac{[R^{(k+1)}, R^{(k+1)}]}{[R^{(k)}, R^{(k)}]}$
   - $Z^{(k+1)} = Q^{(k+1)} + \eta_k Z^{(k)}$

   and go to 2.

Then, to find a solution for our system, we will solve this equation $U^{(k+1)} = \widetilde{T}U^{(k)}\widetilde{T} + \widetilde{F}$ instead of $TU + UT = F$, where:

$$\widetilde{T} = I + 2(\alpha T - I)^{-1}$$

$$\widetilde{F} = -2\alpha(\alpha A - I)^{-1}C(\alpha B - I)^{-1}$$

and will choose $\alpha = \pm\sqrt{n}/||T||_F$, calculate $\widetilde{T}$ and $\widetilde{F}$, and the algorithm will be as follows:

1. Choose initial guess $U^{(0)}$ and calculate:

   - $R^{(0)} = -\widetilde{F} + U^{(0)} - \widetilde{T}U^{(0)}\widetilde{T}$
   - $Q^{(0)} = \widetilde{T}R^{(0)}\widetilde{T} + R^{(0)}$
   - $Z^{(0)} = Q^{(0)}$

2. If $R^{(k)} < \text{tol}$ or $Z^{(k)} < \text{tol}$, stop. Otherwise, return to step 3.

3. Calculate:

   - $\gamma_k = \frac{[R^{(k)}, R^{(k)}]}{[Z^{(k)}, Z^{(k)}]}$
   - $U^{(k+1)} = U^{(k)} + \gamma_k Z^{(k)}$
   - $R^{(k+1)} = -\widetilde{F} + U^{(k+1)} - \widetilde{T}U^{(k+1)}\widetilde{T}$
   - $Q^{(k+1)} = \widetilde{T}R^{(k+1)}\widetilde{T} - R^{(k+1)}$
   - $\eta_k = \frac{[R^{(k+1)}, R^{(k+1)}]}{[R^{(k)}, R^{(k)}]}$
   - $Z^{(k+1)} = Q^{(k+1)} + \eta_k Z^{(k)}$

   and go to 2.

Therefore, the matrix multiplication and matrix inversion are considered the dominating of this algorithm since they are the preconditioning step, and have time complexity $O(n^3)$. Also, each iteration can have time complexity $O(n^3)$ as a result the matrix multiplication is the last step. As this algorithm based on the number of iterations so the time complexity may be hard to be predicted. However, the number of iteration can be reduced by the given preconditioning step.

Table 12 lists the results that obtained from using the preconditioning MCG method via `Numpy` with choosing $\alpha = \pm\sqrt{n}/||T||_F$, and a tolerance of $1e^{-09}$. We can observe the positive impact on the MCG algorithm due to using the preconditioning step. According to the results in the table below, this method achieved better performance although the total time is increased when we reduced the time step. Also, it uses only one iteration for each problem size and time step to find the solution. The EOC value is similar to all the direct methods which round to the second-order convergence of the finite difference method, while the EOG value is better than cubic time complexity. With respect to the errors, this method gives result similar to the previous methods. We can therefore state that this algorithm is faster than all the direct methods.

n	t	No.Iter	Time(s)	$\|\|u-u_h\|\|_{L^\infty}$	$\|\|u-u_h\|\|_{L^2}$	EOC	EOG
189	$1e^{-07}$	1	0.02221	0.00109	0.00054	-	-
379	$2.5e^{-08}$	1	0.07544	0.00027	0.00013	1.97533	1.75742
759	$6.25e^{-09}$	1	0.44497	$6.80304e^{-05}$	$3.40153e^{-05}$	1.99974	2.55544
1519	$1.5625e^{-09}$	1	3.17645	$1.69860e^{-05}$	$8.49304e^{-06}$	2.02403	2.83294
3039	$3.90625e^{-010}$	1	23.71677	$4.24376e^{-06}$	$2.12188e^{-06}$	2.00086	2.89904
6079	$9.765625e^{-011}$	1	186.26687	$1.06059e^{-06}$	$5.30296e^{-07}$	1.91512	2.97268
12159	Memory Error						

Table 12: Results obtained from solving Preconditioned MCG.

3.4 Conclusion

We have already discussed the explanation of methods and closely examine the numerical results of the method, compared the efficacies obtained in tables. Correspondingly, we have investigated the procedure of method how they solve the Sylvester equation. Herein, we are going to inspect graphical comportment of each method, check their computational solution and effect on the Experimental Order of Growth for a large number of nodes. As can be seen by the results in this section, all the direct and iterative methods that compute a solution, have very similar errors. Based on the results, the Kronecker Product, as a direct method, is the slower method, while all the other direct techniques applied can accomplish linear equation faster than emendating the system by applying the Kronecker Product and implementing sparse direct solver. For much larger problem size these are applicable to accumulate to a numerical solution. From the method implemented and the EOG Figure 4, it was noticed that the advantage of the problem procedure which is obtained from the Sylvester form. This was manifested Bartels-Stewart and Shifted System procedure. From the plot below, we can conclude that the Bartels-Stewart Algorithm is the fastest direct method with less time complexity as compared to others.

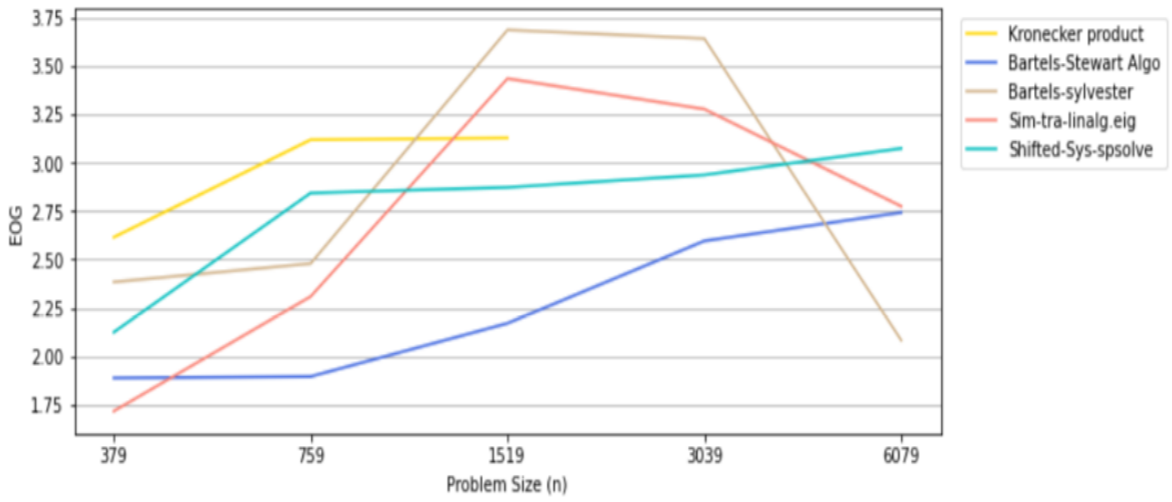


Figure 4: The experimental order of growth of the direct methods.

Regarding the iterative methods, Figure 5 illustrates that using the Preconditioned MCG was faster than the Gradient Based Method and computed a solution in less iterations, in contrast the other algorithm was used a large number of iterations to find a solution with expensive time complexity. Moreover, the Preconditioned MCG was the best performance because of its feature in using less execution time as compared to the Gradient Based Method.

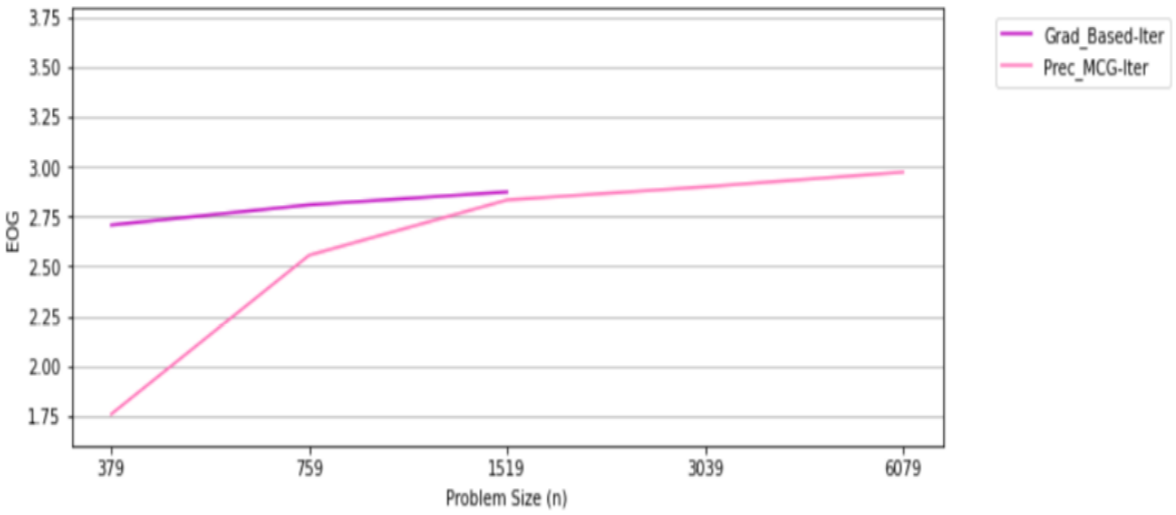


Figure 5: The experimental order of growth of the iterative methods.

It can be clearly noted from Figure 6, the Bartels-Stewart Algorithm was exploited the advantage of R as a diagonal matrix and computed only the diagonal elements of matrix R which affected on reducing the time complexity. This is therefore led to run this algorithm faster than the others. In contrast, Preconditioned MCG is considered good algorithm because of its EOG's behavior is similar to the Bartels-Stewart Algorithm, as shown in the following plot.

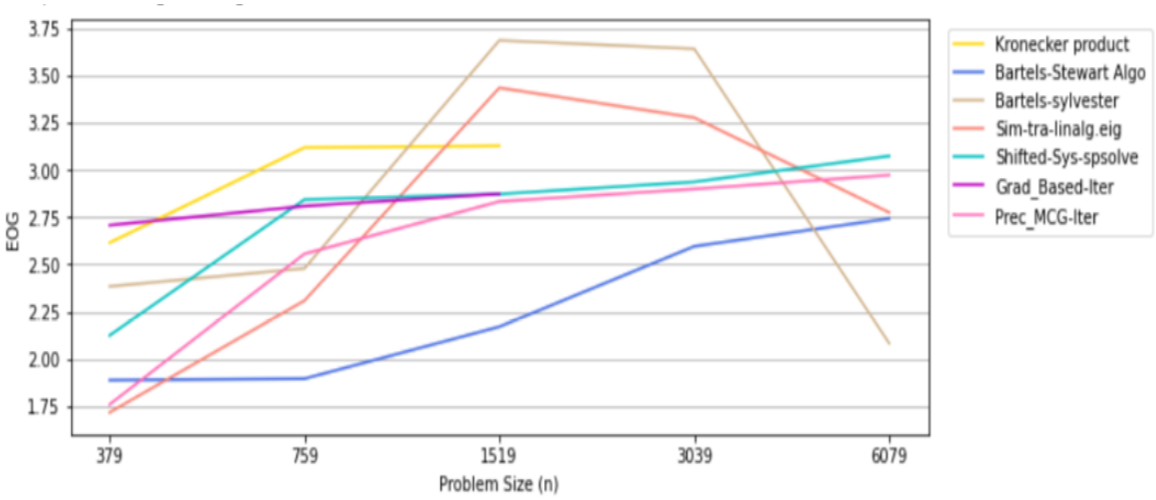


Figure 6: The experimental order of growth of both direct and iterative methods.

4 Evaluation

This section discusses the evaluation of the aims and objectives, proposes a possible future extension of the current work. It also gives a personal reflection on what was achieved on the project.

4.1 Aims and Objectives

The purpose of this project was to achieve two major aims, namely, first was to research, study, implement, and compare a number of various linear matrix methods and solvers for solving the Sylvester equation. This aim has been completed within this project; section 3 illustrated a comprehensive literature review on a variety of different direct and iterative methods, were studied, analysed, implemented. The second aim was to compare all the algorithms to each other to determine which solver (software) had an accurate results and convergence. This comparison was based fundamentally on the impact of a very small time step with a large problem size on the time complexity and convergence order.

The objectives also were met for this project by firstly solving our problem, which is the two-dimensional Heat Equation, by using centred Finite Difference Approximation in space and the Backward Euler Method in time. It was then formulated the problem to the Sylvester form to be implemented and compared by the chosen methods and solvers. Finally, this report was written by using Latex language, and the GitHub link was mentioned in the deliverables table.

4.2 Future Extension

The future work that may be conducted on this project is to implement some additional iterative linear matrix methods, for example, the Kronecker Product, Shifted Systems, Alternative Direct Implicit (ADI) [19]. These methods might be contributed to solving our problem in better performance and with less time complexity as compared to the chosen methods.

Moreover, make a comparison between a fixed time step with different large problem sizes and different time steps with large problem sizes in order to test and evaluate the behavior of each algorithm by observing the convergence, time complexity, and errors.

4.3 Personal Reflection

Despite the challenges while working on this project, several lessons have been learned. Furthermore, knowledge in the filed in particular was expanded, and the acquired skills were also developed. Fortunately, these lessons I learned were valuable guidance from my supervisor.

First, I realised the importance of time management and sticking to a plan that must be well thought out. Accordingly, when I follow the revised plan, I made a remarkable progress. Second, I have also extended my knowledge in the Scientific Computation by reading a variety of different sources, which made me learn basic concepts in Mathematics, for instance, numerical methods and linear matrix equations. In addition, throughout applying and implementing several experiments, my programming skills have been developed, and sufficient ability to analyse the outcomes of these experiences. Finally, in the final phase of this project was writing this report by LaTeX, which in turn considerably increased my productivity and progression.

References

[1] A. B. Abubakar, P. Kumam, A. M. Awwal, and P. Thounthong. A modified self-adaptive conjugate gradient method for solving convex constrained monotone nonlinear equations for signal recovery problems. 7:693, 2019.

[2] F. P. Ali Beik, F. S. Movahed, and S. Ahmadi-Asl. On the krylov subspace methods based on tensor format for positive definite sylvester tensor equations. numerical linear algebra with applications. 23:444–466, 2016.

[3] W. F. Ames. *Numerical Methods for Partial Differential Equations.* Third edition, 1992.

[4] P. Arbenz. Numerical methods for solving large scale eigenvalue problems, 2014.

[5] S. S. Asari and M. Amirfakhrian. Numerical solution of sylvester matrix equations: application to dynamical systems. *Journal of Interpolation and Approximation in Scientific Computing*, pages 1–13, 2016.

[6] R. H. Bartels and G. W. Stewart. Solution of the Matrix Equation AX + XB = C. *Commun. ACM*, 15:820–826, 1972.

[7] A. H. Bentbib, S. El-Halouy, and E. M. Sadek. Krylov subspace projection method for sylvester tensor equation with low rank right-hand side. numerical algorithms. pages 1–20, 2020.

[8] E. B. Castelan and V. G. da Silva. On the solution of a Sylvester equation appearing in descriptor systems control theory. *Systems & Control Letters*, 54(2):109 – 117, 2005.

[9] C. Y. Chiang. On the sylvester-like matrix equation ax+ f (x) b= c. *Journal of the Franklin Institute*, 353:1061–1074, 2016.

[10] S. Cools, J. Cornelis, P. Ghysels, and W. Vanroose. Improving the strong scaling of the conjugate gradient method for solving large linear systems using global reduction pipelining. 2019.

[11] M. Dehghan. Fully implicit finite differences methods for two-dimensional diffusion with a non-local boundary condition. *Journal of Computational and Applied Mathematics*, 106:255–269, 1999.

[12] N. Din˘ci´c. Solving the sylvester equation ax-xb= c when σ (a)$\cap\sigma$ (b)$\neq \emptyset$. *Electronic Journal of Linear Algebra.*, 35:1–23, 2019.

[13] Z. Ge, F. Ding, L. Xu, A. Alsaedi, and T. Hayat. Gradient-based iterative identification method for multivariate equation-error autoregressive moving average

systems using the decomposition technique. *Journal of the Franklin Institute*, 356:1658–1676, 2019.

[14] A. S. Hodel and P. Misra. Solution of underdetermined Sylvester equations in sensor array signal processing. *Linear Algebra and its Applications*, 249(1):1 – 14, 1996.

[15] J. Hou, Q. Lv, and M. Xiao. A parallel preconditioned modified conjugate gradient method for large sylvester matrix equation. *Mathematical Problems in Engineering*, 2014:1–7, 03 2014.

[16] E. Jarlebring. Qr algorithm., 2014.

[17] N. Kalantarova. Spectral properties of structured kronecker products and their applications. 2019.

[18] A. J. Laub. *Matrix Analysis For Scientists And Engineers.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004.

[19] L. Li and D. Xu. Alternating direction implicit-euler method for the two-dimensional fractional evolution equation. *Journal of Computational Physics*, 236:157–168, 2013.

[20] Y. Li, J. Zhu, S. C. Hoi, W. Song, Z. Wang, and H. Liu. Robust estimation of similarity transformation for visual object tracking. 33:8666–8673, 2019.

[21] I. A. Moghrabi. A new preconditioned conjugate gradient method for optimization. *IAENG International Journal of Applied Mathematic*, 49:1–8, 2019.

[22] K. Morton and D. Mayers. *Numerical Solution of Partial Differential Equations: An Introduction.* 1994.

[23] M. Najafi-Kalyani, F. P. A. Beik, and K. Jbilou. On global iterative schemes based on hessenberg process for (ill-posed) sylvester tensor equations. *Journal of Computational and Applied Mathematics*, 373, 2020.

[24] F. Neri. Linear algebra for computational sciences and engineering. pages 1–567, 2019.

[25] V. Prokip. On solvability of the matrix equation ax – xb = c over integer rings. *Modeling, Control and Information Technologies: Proceedings of International scientific and practical conference*, (3):55–58, Nov. 2019.

[26] G. A. Recktenwald. Finite-difference approximations to the heat equation. 2011.

[27] L. Shao, F. Zhao, and G. Hu. A numerical method for the approximation of reachable sets of linear control systems. *IMA Journal of Mathematical Control and Information*, 36:423–441, 2019.

[28] V. Simoncini. Computational methods for linear matrix equations. *SIAM Review*, 58:377–441, 01 2016.

[29] E. Weisstein. Similarity transformation. 2020.

[30] J. Willems, S. Hara, Y. Ohta, and H. Fujioka. Perspectives in mathematical system theory, control, and signal processing. *Journal of Computational and Applied Mathematics*, 2010.

[31] A. G. Wu, G. Feng, G. R. Duan, and W. J. Wu. Closed-form solutions to sylvester-conjugate matrix equations. *Computers and Mathematics with Applications*, 60:95–111, 2010.

[32] H. Zhang and F. Ding. Research article on the kronecker products and their applications. *Journal ofApplied Mathematics*, pages 1–8, 2013.

[33] W. Zhang and D. Zhou. Coupled iterative algorithms based on optimization for solving sylvester matrix equations. *IET Control Theory and Applications*, 13:584–593.

Appendices

A External Materials

B Ethical Issuses

There were no ethical issues involved in this project.