Applied Mathematics and Computational Intelligence
Volume 6, 2017 [41-52]

Jacobi-Davidson, Gauss-Seidel and Successive Over-Relaxation for Solving Systems of Linear Equations

Fatini Dalili Mohammed[1,a] and Mohd Rivaie[b]

[1]*Department of Computer Sciences and Mathematics, Universiti Teknologi MARA (UiTM) Terengganu, Campus Kuala Terengganu, Malaysia*

ABSTRACT

Linear systems are applied in many applications such as calculating variables, rates, budgets, making a prediction and others. Generally, there are two techniques of solving system of linear equation including direct methods and iterative methods. Some basic solution methods known as direct methods are ineffective in solving many equations in large systems due to slower computation. Due to inability of direct methods, iterative methods are practical to be used in large systems of linear equations as they do not need much storage. In this project, three indirect methods are used to solve large system of linear equations. The methods are Jacobi Davidson, Gauss-Seidel and Successive Over-Relaxation (SOR) which are well known in the field of numerical analysis. The comparative results analysis of the three methods is considered. These three methods are compared based on number of iterations, CPU time and error. The numerical results show that Gauss-Seidel method and SOR method with ω=1.25 are more efficient than others. This research allows researcher to appreciate the use of iterative techniques for solving systems of linear equations that is widely used in industrial applications.

Keywords: system of linear equation, iterative method, Jacobi-Davidson, Gauss-Seidel, Successive Over-Relaxation

1. INTRODUCTION

Linear systems are important in our real life. They are applied in various applications such as in calculating variables, rates, budgets, making a prediction and others. A system of linear equation is defined as collections of two or more linear equations with the same variables (Jamil, 2012). In general, a linear equation is given as follows:

$$a_1x_1 + \ \ldots \ + \ a_nx_n = b_m \tag{1}$$

Due to many equations and variables, the systems of linear equations must be represented by matrices equation since they are impossible to be solved (Gerald & Wheatley, 2003). The following form is considered in the systems of linear equations.

$$Ax = b \tag{2}$$

where,

A : m x n matrix
x : column vector with n entries
b : column vector with m entries

[a] fatinidalili@gmail.com, [b] rivaie75@tganu.uitm.edu.my

A rectangular array with m row and n columns is called a matrix. Each number in the matrix is known as an entry. A size of the matrix can be identified by checking on the number of rows and columns. Usually, the matrix is denoted as $m \times n$ which the notations refer to a row and a column respectively. A general $m \times n$ matrix is:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = [a_{ij}] \tag{3}$$

Based on Eq. (3), a capital letter A represents the notation for a matrix while a lowercase letter a refers to a matrix entry. Both notation commonly used the same letter (Robinson, 2011). In identifying an entry of a matrix, double subscripts of entry's row and column is written as a_{ij} where $i = 1,2, \dots, m$ and $j = 1,2, \dots, n$ represent the row and column respectively (Hogben, 2013). From the previous equation, matrices that refer to system of linear equation would be written as:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{4}$$

Mathematically, small systems of linear equations are generally solved by using direct methods which are Gauss Jordan elimination, Cramer's Rule and Gauss elimination. Direct methods can be solved manually or by using computer. However, since many applications take the form of large systems of linear equations, direct methods are not applicable because the computation becomes cumbersome as they consists large number of variables. Hence, iterative method is used as an effective method. An iterative method is usually started by an initial guess. This method is practical to be used in large systems of linear equations because it does not require large storage. There are many examples of iterative method, however in this paper, only three iterative methods will be discussed which are Jacobi Davidson (JD), Gauss-Seidel (GS) and Successive Over-Relaxation (SOR). Therefore, the main purpose of this research is to determine the best indirect method to be used to solve large systems of linear equations.

2. LITERATURE REVIEW

According to Burden & Faires (2011), linear system is in form of $Ax = b$. Saha (2017) examined the solutions of non-square systems of linear equations by generalizing Jacobi-Davidson and Gauss-Seidel. She found that the methods are applied to reduced row echelon system until the solution is obtained. Meanwhile, Wheaton & Awoniyi (2017) found a new iterative method for solving the non-square which is Fourier-Motzkin.

Jacobi-Davidson (JD) is used in solving large, sparse symmetric matrices (Bergamaschi et al., 2003). The Jacobi name was picked by Carl Gustav Jakob Jacobi (Emmanuel, 2015). According to Gutknecht (2007), JD is the one of the simplest iterative methods and it is defined on matrices with non-zero diagonals (HarpinderKaur, 2012). JD is a basic method to solve system of linear equations especially when the system is huge. Referring to Akhtar & Alzghoul (2009), there are two types of storage array required in this JD including old approximation and the new one. The updated values can be done in the same time. For this reason, JD is also known as simultaneous displacement.

Next, GS is defined as the most familiar method used to solve system of linear equation as suggested by Jamil et al. (2013). Actually, this 'Gauss Seidel' method is named by Carl Friedrich

and Philip Ludwig Von Seidel Gauss. In previous research by Hong (2012), GS is better than JD in term of efficiency. This is because GS computes the iteration of $x^{(k+1)}$ while JD only iterates vector of $x^{(k)}$.

SOR method is a method that applies an extrapolation which is known as ω to the GS method. This method extrapolates a weighted average between the previous iteration and the computed GS iteration successively (Mittal, 2014). David M. Young was the first to introduced SOR method around 1950 (Leveque & Trefethen, 1986). SOR is also known as an improvement of the GS. It is used to speed up the convergence of the GS method by introducing a parameter, ω. Regarding to Kumar (2015), in any iterative numerical method, convergence means each successive iteration results in a solution that moves progressively closer to the true solution. There are some theories from researches that discussed about the rate of convergence of these three iterative methods. Referring to a research done by HarpinderKaur (2012), GS method converges faster than Jacobi method. Despite JD slow convergence and not really used for application in our life, it can be parallelized easily (Mittal, 2014). Based on this literature reviews, it can be seen that most researchers found that SOR performs much better methods than JD and GS method because SOR converges quickly.

3. FUNDAMENTAL OF ITERATIVE METHODS

For solving sparse large systems that mostly contain elements of zero, direct method is not appropriate to solve. Iterative methods are used for the solution of large systems of linear equations. Iterative methods start with an initial approximate solution which is vector $x^{(0)}$. The approximation is keep improved until achieved an absolute error that is less than the tolerance value. In this research, the three main iterative methods are presented: JD, GS and SOR.

3.1 Jacobi-Davidson

In numerical linear algebra, JD is an algorithm for determining the solutions of a diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in. The process is iterated until it converges. The formula can be written as:

$$x_i{}^{(k)} = \left(\frac{1}{a_{ii}}\right)\left(\sum_{j=1}^{n}\left(-a_{ij}x_j{}^{(k-1)}\right) + b_i\right) \tag{5}$$

where,

x: values of x that obtained in k^{th} iteration
a: values of element in matrix A in i row and column j
b: values of element in matrix b
$i = 1,2, \dots, m$ that refer to row
$j = 1,2, \dots, n$ that refer to column

Algorithm 1. Steps in Jacobi-Davidson method

Step 1 Choose an initial guess $x^{(0)} = 0$ for $k = 1,2,\cdots, N$ and for $i = 1, 2, 3, \cdots N$
Step 2 While $(k \leq N)$ do steps 3-6
Step 3 For $i = 1, \cdots, n$. Then, set $x_i = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{n}\left(a_{ij}XO_j\right) + b_i\right]$
Step 4 If $\|x - XO\| < TOL$. Then, output $(x_1, \cdots, x_n)$;
$(The\ procedure\ was\ sucessful)$. STOP
Step 5 Set $k = k + 1$
Step 6 For $i = 1, \cdots, n$. Then set $XO_i = x_i$

Step 7 OUTPUT ($'Maximum\ number\ of\ iterations\ exceeded'$);
($The\ \ procedure\ was\ succesful.$) . STOP

3.2 Gauss-Seidel

GS method is a variant of the Jacobi method that usually improves the rate of convergence by using the new values of $x_i^{(k+1)}$. As a simplification, the formula used for Gauss Seidel method is written as:

$$x_i{}^{(k)} = \left(\frac{1}{a_{ii}}\right)\left(-\sum_{j=1}^{n}\left(a_{ij}x_j{}^{(k)}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j{}^{(k-1)}\right) + b_i \right) \tag{6}$$

where,
x: values of x that obtained in kth iteration
a: values of element in matrix A in i row and column j
b: values of element in matrix b
$i = 1,2, \dots, m$ that refer to row
$j = 1,2, \dots, n$ that refer to column

ALGORITHM 2. Steps in Gauss-Seidel method

Step 1 Choose an initial guess $x^{(0)} = 0$ for $k = 1,2, \cdots, N$ and for $i = 1, 2, 3, \cdots N$
Step 2 Let $k = 1$
Step 3 While $(k \leq N)$ do steps 3-6
Step 4 For $i = 1, \cdots, n$. Then, set $x_i = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n}\left(a_{ij}XO_j\right) + b_i\right]$
Step 5 If $\|x - XO\| < TOL$. Then, output $(x_1, \cdots, x_n)$;
($The\ procedure\ was\ sucessful$). STOP
Step 6 Set $k = k + 1$
Step 7 For $i = 1, \cdots, n$. Then set $XO_i = x_i$
Step 8 OUTPUT ($'Maximum\ number\ of\ iterations\ exceeded'$);
($The\ \ procedure\ was\ succesful.$) . STOP

3.3 Successive Over-Relaxation

SOR method is a variant of the Gauss–Seidel method for solving a linear system of equations, resulting in faster convergence. A similar method can be used for any slowly converging iterative process. The SOR method can be derived by multiplying the decomposed system obtained from the Gauss-Seidel method by the relaxation parameter, ω. The iterative parameter ω should always be chosen such that 0 < ω < 2. Besides that, extrapolation factors, ω that used in this project are 0.5, 1.25 and 1.75. The SOR method can be written as:

$$x_i{}^{(k)} = (1-\omega)x_i{}^{(k-1)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1}\left(a_{ij}x_j{}^{k}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j{}^{k-1}\right) \right) \tag{7}$$

where:
x: values of x that obtained in kth iteration
a: values of element in matrix A in i row and column j
b: values of element in matrix b
ω: extrapolation factor

$i = 1,2, \dots, m$ that refer to row
$j = 1,2, \dots, n$ that refer to column

Algorithm 3. Steps in Successive Over-Relaxation method

Step 1 Choose an initial guess $x^{(0)} = 0$ for $k = 1,2, \cdots, k_{max}$ and for $i = 1, 2, 3, \cdots N$

Step 2
$$x_1{}^{(1)} = (1-\omega)x_1{}^{(0)} + \frac{\omega}{a_{11}}\left(b_1 - \left(a_{12}x_2{}^{(0)}\right) - \left(a_{13}x_3{}^{(0)}\right)\right)$$
$$x_2{}^{(1)} = (1-\omega)x_2{}^{(0)} + \frac{\omega}{a_{22}}\left(b_2 - \left(a_{21}x_1{}^{(1)}\right) - \left(a_{23}x_3{}^{(0)}\right)\right)$$
$$x_3{}^{(1)} = (1-\omega)x_3{}^{(0)} + \frac{\omega}{a_{33}}\left(b_i - \left(a_{31}x_1{}^{(2)}\right) - \left(a_{32}x_2{}^{(1)}\right)\right)$$
$$x_i{}^{(k)} = (1-\omega)x_i{}^{(0)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1}\left(a_{ij}x_j{}^{k}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j{}^{k-1}\right)\right)$$

Step 3 Until $(k \leq N)$ repeat steps 1-2

Step 4 Repeat the whole calculation until achieved the stopping criteria.

4. THE CONVERGENCE OF ITERATIVE METHODS

The three methods which are JD, GS and SOR are iterated until the stopping criteria are achieved. According to Gismalla (2014), the stopping criteria are calculated by:

$$\frac{\|x^{(k)} - x^{(k-1)}\|}{\|x^{(k)}\|} < \in \qquad ; \text{where } \in > 0$$

The symbol $\in$ represents a tolerance factor. The values of tolerance factor in this project are $10^{-3}, 10^{-6}$ and 10^{-9}. Generally, a tolerance is used to know the distance the last iteration in a method to the exact solution. Instead, various tolerance factors are used to compare which stopping criterion gives the best performance.

5. ANALYSIS OF RESULTS

In this world, there are many applications involving thousands equations and variables. The square systems of linear equations are widely used in our real life. Related to that, the large systems of linear equations are used in this project. To solve the large system of linear equation, the solutions can be obtained by three iterative methods which are Jacobi-Davidson (JD) method, Gauss-Seidel (GS) method and Successive Over-Relaxation (SOR) method since they are well-known basic iterative methods in numerical analysis (Salkuyeh, 2007).

In the beginning, the datasets are generated using Maple 16. Then, all analyses are performed in MATLAB since it is one of the best software for numerical computation. This software is more efficient than another program such as Python. Furthermore, it is faster to execute the large datasets when using this software. MATLAB is selected since the programming code had been provided and guided by Gismalla (2014). The approximated values from MATLAB then compared with exact values that had been executed in Maple by using inverse methods.

This section introduced some problems that will be tested on each method which is JD, GS and SOR. As aforementioned, the systems of linear equations are better simplified in matrices form. There are 5 practical examples applied in this paper. Symmetric matrices are considered in this study. The symmetric matrices on $2 \times 2, 3 \times 3, 5 \times 5, 10 \times 10$ and 20×20 systems of linear equations are illustrated in following Table 1.

The following five problems are tested using the same processor performance which is Intel® Core™ i5-3210M CPU @ 2.50 GHz on Asus Series A45V. It is important to perform the tests using the same processor in order to get a uniform and accurate CPU time. This research involved symmetric and non-symmetric systems of linear equations which are positive and negative definite. However, the data tabulation only showed symmetric positive definite (SPD) applied on an initial point of (0,0,….0). For the detail of the remaining results, research by Mohammed (2017) can be referred.

Table 1 List of linear equations

Size	$Ax = b$
2×2	$\begin{bmatrix} 3 & -2 \\ -2 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}=\begin{bmatrix} 7 \\ 3 \end{bmatrix}$
3×3	$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 7 \\ 3 \\ 2 \end{bmatrix}$
5×5	$\begin{bmatrix} 5 & -2 & 3 & -2 & -2 \\ -2 & 5 & -2 & 3 & -2 \\ 3 & -2 & 5 & -2 & -2 \\ -2 & 3 & -2 & 5 & -2 \\ -2 & -2 & -2 & -2 & 5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}=\begin{bmatrix} 7 \\ 3 \\ 2 \\ 2 \\ 2 \end{bmatrix}$
10×10	$\begin{bmatrix} 7 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 \\ -1 & 7 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & -1 & 7 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ -1 & 0 & -1 & 7 & -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 & 7 & -1 & 0 & -1 & 0 & -1 \\ -1 & 0 & -1 & 0 & -1 & 7 & -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 & 7 & -1 & 0 & -1 \\ -1 & 0 & -1 & 0 & -1 & 0 & -1 & 7 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 7 & -1 \\ 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix}=\begin{bmatrix} 7 \\ 3 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$

Table 1 (CONT) List of linear equations

Size	$Ax = b$
20×20	$\begin{bmatrix} 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 12 \end{bmatrix} = \begin{bmatrix} 7 \\ 3 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$

The result that produced by the four equations are given in Table 2, 3, 4, 5 and 6.

Table 2 Linear equation of 2×2

Methods	Number of Iteration	CPU Time	Error
Jacobi-Davidson	55	1.2413E-03	1.4651E-09
Gauss-Seidel	29	1.0210E-03	5.0747E-10
Sucessive Over-Relaxation with $\omega = 0.5$	95	2.0173E-03	3.3331E-09
Sucessive Over-Relaxation with $\omega = 1.25$	17	9.0233E-04	6.3046E-10
Sucessive Over-Relaxation with $\omega = 1.75$	80	1.7767E-03	4.3829E-10

Table 3 Linear equation of 3×3 3×3

Methods	Number of Iteration	CPU Time	Error
Jacobi-Davidson	65	1.3957E-03	1.8645E-09
Gauss-Seidel	34	1.0570E-03	7.4216E-10
Sucessive Over-Relaxation with $\omega = 0.5$	110	2.1387E-03	3.7693E-09
Sucessive Over-Relaxation with $\omega = 1.25$	19	9.7267E-04	2.7376E-11
Sucessive Over-Relaxation with $\omega = 1.75$	81	1.8273E-03	6.0185E-10

Table 4 Linear equation of 5×5

Methods	Number of Iteration	CPU Time	Error
Jacobi-Davidson	Fail	Fail	Fail
Gauss-Seidel	111	1.9220E-03	4.3016E-09
Sucessive Over-Relaxation with $\omega = 0.5$	329	4.6047E-03	1.4742E-08
Sucessive Over-Relaxation with $\omega = 1.25$	69	1.7200E-03	2.1552E-09
Sucessive Over-Relaxation with $\omega = 1.75$	98	1.8670E-03	2.8764E-09

Table 5 Linear equation of 10×10

Methods	Number of Iteration	CPU Time	Error
Jacobi-Davidson	58	1.4397E-03	1.6250E-09
Gauss-Seidel	32	1.2100E-03	6.3162E-10
Sucessive Over-Relaxation with $\omega = 0.5$	99	2.0163E-03	3.6451E-09
Sucessive Over-Relaxation with $\omega = 1.25$	21	1.1007E-03	2.5873E-10
Sucessive Over-Relaxation with $\omega = 1.75$	93	2.0893E-03	4.4658E-10

Table 6 Linear equation of 10×10

Methods	Number of Iteration	CPU Time	Error
Jacobi-Davidson	Fail	Fail	Fail
Gauss-Seidel	20	1.1880E-03	1.2948E-10
Sucessive Over-Relaxation with $\omega = 0.5$	31	1.6427E-03	9.1859E-10
Sucessive Over-Relaxation with $\omega = 1.25$	31	1.9150E-03	3.5445E-10
Sucessive Over-Relaxation with $\omega = 1.75$	129	2.8523E-03	4.8811E-10

6. DISCUSSION

An interpretation is the process of organizing data that is analyzed and collected. A common method of interpreting numerical result graphically is known as a performance profile that is introduced by Dolan and More (2001). The performance profile is one of the ways to demonstrate trend line in data and make comparisons between JD, GS, and SOR.

This section provides further explanation for numerical results that focuses on three aspects which are number of iterations, CPU time and error for the tolerance factor of 10^{-9}. The

performance profiles that combined both positive definite and negative definite are shown below based on tolerance 10^{-9} only. The performance profile of all methods based on the number of iterations is illustrated in Figure 1 and the performance profile based on the CPU time is shown in Figure 2. For error, the performance profile is drawn in Figure 3.

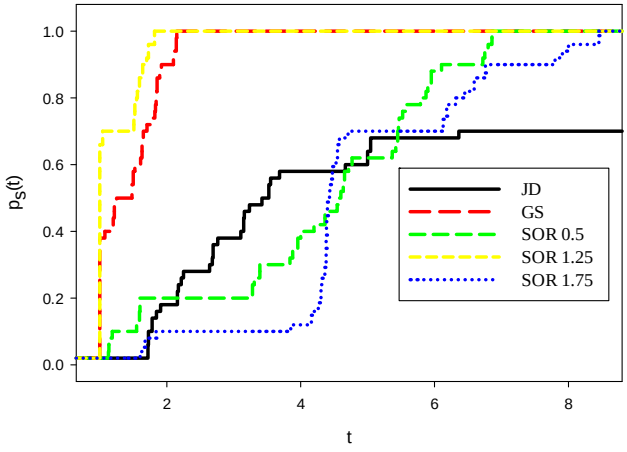

Figure 1. Performance profile based on number of iterations

Figure 1, shows the performance profile based on the number of iterations that combine symmetric positive definite and symmetric negative definite based on tolerance 10^{-9}. It can be seen that, GS, SOR with extrapolation factor, ω=0.5, SOR with extrapolation factor, ω=1.25 and SOR with extrapolation factor, ω=1.75 are able to reach value $p_s(t)$=1. But, there is a line that does not achieve $p_s(t)$=1 which is black line that represents JD. This line only reach $p_s(t)$=0.7 since some of the selected problems failed to obtain the solution. For the remaining methods, they are able to solve the problems. From the left side of the performance profile, it can be used to determine the efficiency of a method. SOR with extrapolation factor, ω=1.25 is declared as the best method based on the number of iterations.

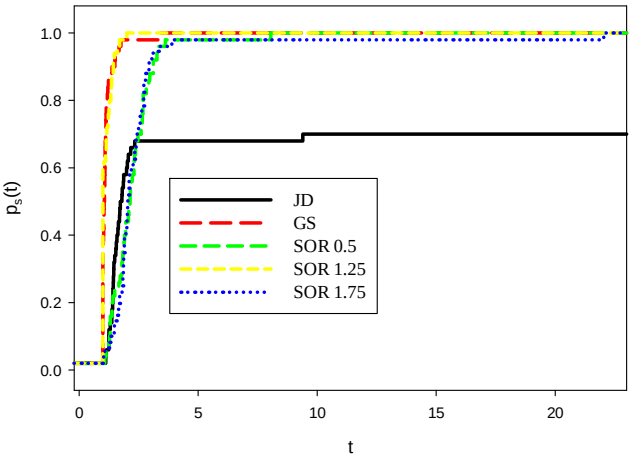

Figure 2. Performance profile based on CPU time

In Figure 2, the performance profile based on CPU time that combines symmetric positive definite and symmetric negative definite based on tolerance 10^{-9} is plotted. The figure shows that GS, SOR with extrapolation factor, ω=0.5, SOR with extrapolation factor, ω=1.25 and SOR with extrapolation factor, ω=1.75 are able to reach value $p_s(t)$=1. But, there is a line that does not achieve $p_s(t)$=1 which is black line that represents JD. This line only reach $p_s(t)$=0.7 since

some of selected problems failed to obtain the solution. The remaining methods able to solve the problems. From the left side of the performance profile, it can help determine the efficiency of a method. Thus, SOR with extrapolation factor, ω=1.25 is declared as the best method based on the CPU time.

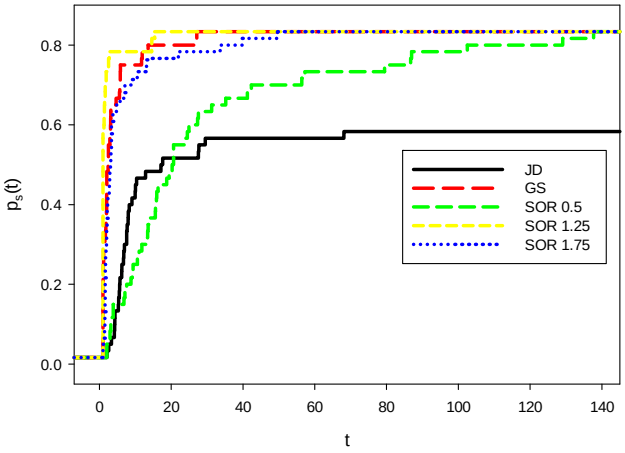

Figure 3. Performance profile based on CPU time

In Figure 3, performance profile based on error that put together symmetric positive definite and symmetric negative definite based on tolerance 10^{-9} is illustrated. It can be seen that, GS, SOR with extrapolation factor, ω=0.5, SOR with extrapolation factor, ω=1.25 and SOR with extrapolation factor, ω=1.75 are able to reach value $p_s(t)$=1. But, there is a line that does not achieve $p_s(t)$=1 which is black line that represents JD. This line only reach $p_s(t)$=0.833 since some of selected problems failed to obtain the solution. For the remaining methods, they are able to solve the problems. From the left side of the performance profile, it can determine the efficiency of a method. As shown in Figure 3, SOR is declared as the best method based on the error.

7. CONCLUSION

The numerical results are presented in each aspect namely number of iterations, CPU time and error. The three methods are tested using five problems of symmetric positive definite systems of linear equations with initial point of (0;0;...;0). The approximate values obtained mostly able to perform until closer to the exact solutions. Results shows that SOR with extrapolation factor, ω=1.25 has less number of iteration, faster in term CPU time and closer to the exact values as compared to other methods. The SOR with extrapolation factor, ω=1.25 possessed the minimum number of iteration. For the 3×3, 5×5 and 10×10 linear equations, it has 19, 69 and 21 iterations. It also takes shorter CPU time for the linear equations of 3×3, 5×5 and 10×10 which are 0.00097 seconds, 0.00172 seconds and 0.0011 seconds. In terms of error, the SOR with extrapolation factor, ω=1.25 got the least error. Thus, it can be concluded that the SOR with extrapolation factor, ω=1.25 is more efficient and has good performance among the three methods.

8. FUTURE WORK

In future work, there are some improvements can be done. In this project, the problems of linear equations are selected to be tested with iterative methods: JD, GS and SOR. It would be more interesting if non-linear equation can be tested to compare the speed of executable

computation and accuracy. Next, it is recommended to compare the systems of linear equations using more problems in higher dimensional of the matrices.

ACKNOWLEDGEMENTS

First, the authors would like to thank lecturers in Department of Computer Sciences and Mathematics, Universiti Teknologi MARA (UiTM) Terengganu, Campus Kuala Terengganu, Malaysia for any help and endless support when finishing this paper. For your information, this paper was presented in Final Year Project Viva-Voce on 15th June 2017 in Universiti Teknologi MARA (UiTM) Terengganu, Campus Kuala Terengganu, Malaysia. Hence, the authors would like to take this opportunity to thank the panel and examiner during the presentation namely Madam Zanariah binti Mohd Yusof and Madam Ruhana binti Jaafar for their comments and advices. The authors are also grateful to the editors for their valuable suggestions.

REFERENCES

[1] N. Jamil, International Journal of Emerging Sciences **2**, 310 (2012)
[2] C.F. Gerald and P.O. Wheatley, *Applied numerical analysis* (Pearson, Boston, 2003).
[3] D. J. S. Robinson, *A course in linear algebra with applications*. (*World Scientific*, 2011).
[4] L. Hogben, Handbook of Linear Algebra (CRC Press, 2013).
[5] R.L. Burden and J.D. Faires, Numerical analysis (Brooks Cole Pub, 2011).
[6] M. Saha, Generalized Jacobi and Gauss-Seidel Method for Solving Non-Square Linear Systems, (2017).
[7] I. Wheaton and S. Awoniyi, Journal of Computational and Applied Mathematics **322**, 1 (2017).
[8] L. Bergamaschi, G. Pini, and F. Sartoretto, Journal of Computational Physics **188**, 318 (2003).
[9] M.H. Gutknecht, Frontiers of Computational Science 1 (2007).
[10] N. Akhtar and A. Alzghoul, School of Mathematics and Systems Engineering Matematiska. (2009)
[11] F.S. Emmanuel, Computational nad Applied Mathematics **1**, 21 (2015).
[12] H. Kaur and K. Kaur, IOSR Journal of Mathematics (IOSRJM) 2, 20 (2012).
[13] N. Jamil, J. Muller, C. Lutteroth, and G. Weber, "Speeding up SOR solvers for constraint-based GUIs with a warm-start strategy," in *Eighth International Conference on Digital Information Management (ICDIM),* (Department of Computer Science, New Zealand, 2013), pp. 268-273.
[14] D. Hong, D-Iteration method or how to improve Gauss-Seidel method (2012). Retrieved from http://arxiv.org/abs/1202.1163
[15] S. Mittal, International Journal of High Performance Computing and Networking **7**, 292 (2014).
[16] R.J. Leveque and L.N. Trefethen, Institute for Computer Applications in Science and Engineering 86 (1986)
[17] R. Kumar, Journal of Computer and Mathematical Sciences **6**, 290 (2010).
[18] P. D. Gismalla, International Journal of Engineering and Technical Research (IJETR) **2**, 7 (2014)
[19] D.K. Salkuyeh, Numerical Mathematics, A Journal of Chinese Universities **16**, 164 (2007).
[20] F.D. Mohammed, "Solving large systems of linear equations using iterative methods (Jacobi-Davidson, Gauss-Seidel and Successive-Over Relaxation)," B.Sc. thesis, MARA University of Technology, 2017.
[21] E. Dolan and J.J. More, Mathematics Programming, **91**, 201 (2001).

APPENDIX

If any, the appendix should appear directly after the references without numbering, and on a new page.