DOCTORAL THESIS

Enhanced Lanczos Algorithms for Solving Systems of Linear Equations with Embedding Interpolation and Extrapolation

Maharani Maharani

A thesis submitted in fulfilment of the requirements

for the degree of Doctor of Philosophy

at the

Department of Mathematical Sciences

University of Essex

July 2015

Dedicated to

I dedicate my thesis work to my family and my parents. A special feeling of gratitude to my loving husband, Muhammad Munawar Yusro, who consistently supports and encourages me during my study. My children, Maharaj Fawwaz Almuqaddim Yusran, Kallista Adelin Musyaffa Yusran and Muhammad Abrizam Rizqullah Yusran, have never left my side and are very special. I also dedicate this thesis to my mother in the heaven who sacrificed for me and provided unconditional love and care. My respect also to my mother-in-law and my father who always support me through his prayers.

Abstract

Lanczos-type algorithms are prone to breaking down before convergence to an acceptable solution is achieved. This study investigates a number of ways to deal with this issue. In the first instance, we investigate the quality of three types of restarting points in the restarting strategy when applied to a particular Lanczos-type algorithm namely Orthodir. The main contribution of the thesis, however, is concerned with using regression as an alternative way to deal with breakdown. A Lanczos-type algorithm is run for a number of iterations and then stopped, ideally, just before breakdown occurs. The sequence of generated iterates is used to build up a regression model that captures the characteristic of this sequence. The model is then used to generate new iterates that belong to that sequence. Since the iterative process of Lanczos is circumvented, or ignored, while using the model to find new points, the breakdown issue is resolved, at least temporarily, unless convergence is achieved. This new approach, called EIEMLA, is shown formally, through extrapolation, that it generates a new point which is at least as good as the last point generated by the Lanczos-type algorithm prior to stoppage.

The remaining part of the thesis reports on the implementation of EIEMLA sequentially and in parallel on a standard parallel machine provided locally and on a Cloud Computing platform, namely Domino Data Lab. Through these implementations, we have shown that problems with up to 10^6 variables and equations can be solved with the new approach. Extensive numerical results are included in this thesis. Moreover, we point out some important issues for further investigation.

Declaration

The work in this thesis is based on research carried out at the Numerical Analysis Group, the Department of Mathematical Sciences, University of Essex, UK. No part of this thesis has been submitted elsewhere for any other degree or qualification and it all my own work unless referenced to the contrary in the text. **Copyright © 2015 by Maharani, Maharani**.

Acknowledgements

All praise is due to Allah the Almighty and Most Merciful , the sustainer of all the worlds and peace and blessings be upon his last messenger Prophet Muhammad who taught us that whosoever does not thank people is not thankful Allah.

I am heartily thankful to my supervisor, Prof. Dr. Abdellah Salhi, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. Abdel has been disciplined and very strict of all the things related to my study. One example, we should come every day, from Monday to Friday and from 09.00 to 17.00. He would check our desk every day. However, he has been very kind and responsible of my life in the UK, specially when I lost my mother and other struggle of life. I am also very grateful to Dr. John Ford for his scientific advice and acknowledge. He has been a supervisory member for almost three years who has given brilliants comments and suggestions during the supervisory meetings. He also helped me out understanding Matlab and Latex and numerical methods at my first year of my study. My grateful is also to the internal and external examiners for many hours of reading and checking my thesis. I also have to thank Dr. Susanto Hadi for many hours reviewing my paper and transferring knowledge to me. I also thank the graduate administrator, McNally Shauna for her supports and helps during this study. I hope that I could be lively and energetic as Shauna.

I especially thank my family. No words can illustrate how grateful I am to my mother-in law, my mother, my father, my sisters-in-law, my brothers and my sister. My hard-working mother has sacrificed her lives for me and provided unconditional love and care. My thought and prayer are always to her in the heaven. My mother-in-law has consistently supported and prayed for me to finish this study. A special thank to my big brother, Muhamad Akbar, who has supported in financial when I really need his help. I would also like to thank my beloved husband, Muhamad Yusro, who has sacrificed his life in my country to stay here in the UK. He never stopped supporting and encouraging me throughout living in the UK. My children. Fawwaz Yusran, Adelin Yusran, and Abrizam Yusran, who have been cheering me up.

I will forever be thankful to my former research colleagues , Muhammad Farooq and Nudrat Ameer. Farooq has been helpful in providing many journals and ebooks and also some discussions regarding Lanczos. Nudrat has been inspired me to be a student as well as a mother. She has been supportive and helpful in advising many things. I also thank my office mates, Al-Karkhi Tahani. Tahani has very much supported and helped out on any life problems. Many thanks to my colleague, Sulaiman for discussing and transferring knowledge to me. I also thank my former office mates, Asma Gul, Miftahudin, Sri Utami Zuliana. We have spent many quality times together during few years. Many thanks for my office mates, Tahani, Marios, Amal, Junaed, and Guffran Syed, for discussing many things with me. Many thanks to other colleagues, Zardad Khan, Osama, Baba Bakar, Irem, etc. I would also thank all Indonesian students in Colchester, Hakimul Ikhwan, Idham Ananta, Monika Hidayanti, Arief Setyanto, Yuslenita Muda, and Rudy Kusdiantara for gathering and sharing and discussing many things during my study.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Literature Review

1.1 Introduction

Systems of Linear Equations (SLE's) are ubiquitous in science and engineering applications. There are several approaches to solving them broadly classed as direct methods and iterative methods.

There are two general classes of iterative methods to solve SLEs: stationary methods and non-stationary methods. Stationary methods which include Richardson [38], Jacobi, Gauss-Seidel [4], and Successive Over-relaxation (SOR) [19] among others, and Non-stationary methods which include Conjugate Gradient [54], BCG, BICG [41], GMRES, Arnoldi type [59], and Lanczos type [51, 52]. Non-stationary methods are generally more efficient and suitable for a large systems of linear equations. They are based on orthogonal polynomials. They are often referred to as Krylov subspace methods [66].

Among the iterative methods, for solving large linear systems with a sparse non-symmetric matrix, those based on the Lanczos process are the most effective.

This is because they feature short recurrence relations for the generation of the Krylov subspace, which means low cost and low memory requirement, [41]. They are, however, prone to breakdown, [7]. For these reasons, ways to avoid breakdown in Krylov subspace algorithms have been researched extensively in the last decades, resulting in new more robust and efficient algorithm.

In this thesis, we will briefly review the relevant literature, illustrate how these algorithms can be derived, and provide an example of a Lanczos-type algorithm namely Orthores, also referred to as A_4, [6]. We will also recall a Lanczos-type algorithm, namely the Method of Recursive Zoom (MRZ), [11, 12], which is designed to avoid breakdown by jumping over the non-existent orthogonal polynomials.

This thesis is organized as follow. Chapter 1, as said above, introduces the background theory of Lanczos-type algorithms for solving SLE's. It also explains the breakdown issue and recalls two of the main algorithmic approaches to deal with it. Chapter 2 discusses restarting from three different points and how to deal with the breakdown in Lanczos-type algorithms, using restarting. Chapter 3 investigates the use of interpolation and extrapolation as a means to overcome the breakdown problem. Algorithm EIEMLA, or Embedded Interpolation and Extrapolation in Lanczos-type Algorithm, is then presented. Chapter 4 exploits restarting based on an output of EIEMLA; this restarting algorithm is referred to as REIEMLA. Chapter 5 investigates the use of parallel processing as well as cloud computing to handle large scale problems (upto 10^6 variables). Extensive experimental results are included where necessary. Chapter 6 is the conclusion and future work.

1.2 Lanczos-type algorithms

In 1950, Lanczos proposed an iterative method for solving the Eigenvalue problem [51]. In this method, an $n \times n$ matrix can be transformed into a tridiagonal one in order to simplify the problem. The key feature of the method is that no matrix to matrix multiplication or matrix inversion is ever required. The algorithm uses at most matrix to vector multiplication. That is where its efficiency resides. This approach was then adapted to solve systems of linear equations, [52]. The derivation of these algorithms is commonly done using Formal Orthogonal Polynomials (FOP's), [6, 7]. There are also alternative ways to derive them such as matrix algebra, [13].

1.2.1 Krylov Subspace Methods

This section begins with the derivation of Krylov Subspace (KS) methods by orthogonalizing the natural basis of KS.

Consider the system of linear equations

$$A\mathbf{x} = \mathbf{b} \tag{1.1}$$

where $A \in R^{n\times n}$ and vectors $\mathbf{x}$ and $\mathbf{b} \in \mathbf{R}^n$. Let $\mathcal{K}_k$ and $\mathcal{L}_k$ be two subspaces of dimensions k. The projection method, [67], for solving the system (A.1.1) is derived by choosing an initial approximate solution $\mathbf{x}_0$ and defining the sequence of vectors $\{\mathbf{x}_k\}$ by two conditions :

$$\mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k \tag{1.2}$$

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k \perp \mathcal{L}_k. \tag{1.3}$$

If $\mathcal{K}_k = K_k(A, \mathbf{r}_0)$ is a KS of dimension k defined by

$$K_k(A, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \dots, A^{k-1}\mathbf{r}_0\} \tag{1.4}$$

where $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, then the projection method is called the Krylov subspace methods (KSM), [5]. Moreover, the choice of $\mathcal{L}_k$ leads to several KSMs [14]. For instance, if $\mathcal{L}_k = K_k(A^T, \mathbf{y})$, for an arbitrary nonzero vector $\mathbf{y}$, then the KSM method is known as the Lanczos method.

1.2.2 Lanczos Methods

Let us compute the condition (1.2) with the KS as given in (1.4). From (1.2) , $\mathbf{x}_k - \mathbf{x}_0$ can be written as :

$$\mathbf{x}_k - \mathbf{x}_0 = -\alpha_1\mathbf{r}_0 - \alpha_2 A\mathbf{r}_0 - \cdots - \alpha_k A^{k-1}\mathbf{r}_0, \tag{1.5}$$

and thus, multiplying both sides by A, adding and subtracting $\mathbf{b}$, we obtain :

$$\begin{aligned}
A(\mathbf{x}_k - \mathbf{x}_0) &= A(-\alpha_1\mathbf{r}_0 - \alpha_2 A\mathbf{r}_0 - \cdots - \alpha_k A^{k-1}\mathbf{r}_0),\\
A\mathbf{x}_k - A\mathbf{x}_0 &= -\alpha_1 A\mathbf{r}_0 - \alpha_2 A^2\mathbf{r}_0 - \cdots - \alpha_k A^k\mathbf{r}_0,\\
\mathbf{b} - (A\mathbf{x}_k - A\mathbf{x}_0) &= \mathbf{b} + \alpha_1 A\mathbf{r}_0 + \alpha_2 A^2\mathbf{r}_0 + \cdots + \alpha_k A^k\mathbf{r}_0,\\
\mathbf{b} - A\mathbf{x}_k &= (\mathbf{b} - A\mathbf{x}_0) + \alpha_1 A\mathbf{r}_0 + \alpha_2 A^2\mathbf{r}_0 + \cdots + \alpha_k A^k\mathbf{r}_0,\\
\mathbf{r}_k &= \mathbf{r}_0 + \alpha_1 A\mathbf{r}_0 + \alpha_2 A^2\mathbf{r}_0 + \cdots + \alpha_k A^k\mathbf{r}_0. \qquad (1.6)
\end{aligned}$$

From the last relation, we can write $\mathbf{r}_k$ as :

$$\mathbf{r}_k = P_k(A)\mathbf{r}_0, \qquad (1.7)$$

where $P_k(t) = 1 + \alpha_1 t + \cdots + \alpha_k t^k$, [12]. Applying the orthogonality condition in relation (1.3), we obtain

$$\langle (A^T)^i\mathbf{y}, \mathbf{r}_k\rangle = \langle \mathbf{y}, A^i\mathbf{r}_k\rangle = \langle \mathbf{y}, A^i P_k(A)\mathbf{r}_0\rangle = 0, \quad \text{for } i = 0, 1, \cdots, k-1, \qquad (1.8)$$

which leads to a system of linear equations in coefficients $\alpha_1, \alpha_2, \ldots, \alpha_k$, as follows

$$\begin{aligned}
\alpha_1\langle \mathbf{y}, A\mathbf{r}_0\rangle + \alpha_2\langle \mathbf{y}, A^2\mathbf{r}_0\rangle + \cdots + \alpha_k\langle \mathbf{y}, A^k\mathbf{r}_0\rangle &= -\langle \mathbf{y}, \mathbf{r}_0\rangle,\\
\alpha_1\langle A^T\mathbf{y}, A\mathbf{r}_0\rangle + \alpha_2\langle A^T\mathbf{y}, A^2\mathbf{r}_0\rangle + \cdots + \alpha_k\langle A^T\mathbf{y}, A^k\mathbf{r}_0\rangle &= -\langle A^T\mathbf{y}, \mathbf{r}_0\rangle,\\
&\ \vdots\\
\alpha_1\langle (A^T)^{k-1}\mathbf{y}, A\mathbf{r}_0\rangle + \alpha_2\langle (A^T)^{k-1}\mathbf{y}, A^2\mathbf{r}_0\rangle + \cdots + \alpha_k\langle (A^T)^{k-1}\mathbf{y}, A^k\mathbf{r}_0\rangle &= -\langle (A^T)^{k-1}\mathbf{y}, \mathbf{r}_0\rangle. \qquad (1.9)
\end{aligned}$$

The above system is consistent if and only if the determinant

$$\begin{vmatrix} \langle \mathbf{y}, A\mathbf{r}_0 \rangle & \langle \mathbf{y}, A^2\mathbf{r}_0 \rangle & \cdots & \langle \mathbf{y}, A^k\mathbf{r}_0 \rangle \\ \langle A^T\mathbf{y}, A\mathbf{r}_0 \rangle & \langle A^T\mathbf{y}, A^2\mathbf{r}_0 \rangle & \cdots & \langle A^T\mathbf{y}, A^k\mathbf{r}_0 \rangle \\ \vdots & \vdots & \cdots & \vdots \\ \langle (A^T)^{k-1}\mathbf{y}, A\mathbf{r}_0 \rangle & \langle (A^T)^{k-1}\mathbf{y}, A^2\mathbf{r}_0 \rangle & \cdots & \langle (A^T)^{k-1}\mathbf{y}, A^k\mathbf{r}_0 \rangle \end{vmatrix} \tag{1.10}$$

is different from zero, [29]. Obviously, solving the systems (1.9) seems impractical. The easiest way to determine the solution of the system is by computing recursively the polynomial P_k with considering the degrees of the polynomials in the right- and the left-hand sides, and making them balanced. This is discussed in the following section.

1.3 Formal Orthogonal Polynomials and their Use in Lanczos Methods

Let c be a linear functional on the vector space of complex polynomials and let it be defined by :

$$c(t^i) = c_i = \left\langle \mathbf{y}, A^i\mathbf{r}_0 \right\rangle, \quad \text{for } i = 0, 1, \ldots \tag{1.11}$$

For any polynomial $P(t) = \alpha_0 + \alpha_1 t + \cdots + \alpha_k t^k$ and by linear combination, we have

$$\langle \mathbf{y}, P(A)\mathbf{r}_0 \rangle = c(P(A)). \tag{1.12}$$

So, relation (1.8) becomes

$$c\left(A^i P_k(A)\right) = 0, \quad \text{for } i = 0, 1, \ldots, k-1. \tag{1.13}$$

The family of polynomials satisfying these conditions for all k is called the family of orthogonal polynomials (FOP) with respect to c. We now have definition of a family of orthogonal polynomials (FOP) as follows.

Definition 1 ([29]). *The family of polynomials $\{P_k\}$ are said to form the family of FOP with respect to c if they satisfy*

1. *the degree of P_k is at most k, for all k*
2. $c(t^i P_k(t)) = 0$, $i = 0, 1, \dots, k-1$.

If we substitute $P_k(t) = \alpha_0 + \alpha_1 tA + \cdots + \alpha_k t^k$ above into (1.11), we obtain a system of linear equations

$$\alpha_0 c_i + \alpha_1 c_{i+1} + \cdots + \alpha_k c_{i+k} = 0, \tag{1.14}$$

for $i = 0, 1, \dots, k-1$. Adding an equation $-P_k(t) + \alpha_0 + \alpha_1 t + \cdots + \alpha_k t^k = 0$ to the system, we obtain a $(k+1)\times(k+1)$ system of linear equations in α_i for $i = 0, 1, \dots, k$ with the determinant of the matrix coefficient given by

$$D_k = \begin{vmatrix} c_0 & c_1 & \dots & c_k \\ \vdots & & & \\ c_{k-1} & c_k & \cdots & c_{2k-1} \\ 1 & t & \cdots & t^k \end{vmatrix}. \tag{1.15}$$

In fact, if we set a Hankel determinant matrix

$$H_k^{(0)} = \begin{vmatrix} c_0 & c_1 & \dots & c_{k-1} \\ \vdots & & & \\ c_{k-1} & c_k & \cdots & c_{2k-2} \end{vmatrix}, \tag{1.16}$$

then, $P_k(t)$ can be expressed by the determinantal formula as follows :

$$P_k(t) = \frac{D_k}{H_k^{(0)}}, \tag{1.17}$$

[6]. It is clear that P_k exists and is unique if and only if the Hankel determinant is different from zero. The normalization of P_k is obtained by the fact that $P_k(0) = 1$. The detail of the computation of the recurrence relationships form of P_k can be seen in [8] and is discussed as below.

Interestingly, it is possible to construct another formula of the family of FOP by taking advantage of the orthogonality condition, [8]. Let us now consider the linear functional $c^{(1)}$ which is defined by

$$c^{(1)}(t^i) = c(t^{i+1}) = c_{i+1}, \quad \text{for } i = 0, 1, \ldots \tag{1.18}$$

A family of polynomials P_k of FOP, ${P_k}^{(1)}(t)$, can also be written in the determinantal formula as follows :

$${P_k}^{(1)}(t) = \frac{\begin{vmatrix} c_1 & c_2 & \ldots & c_{k+1} \\ \vdots & & & \\ c_k & c_{k+1} & \cdots & c_{2k-1} \\ 1 & t & \cdots & t^k \end{vmatrix}}{\begin{vmatrix} c_1 & c_2 & \ldots & c_{k+1} \\ \vdots & & & \\ c_k & c_{k+1} & \cdots & c_{2k-1} \\ b_0 & b_1 & \cdots & b_k \end{vmatrix}} \tag{1.19}$$

where the b_i's are numbers which depend on k, [8]. It also satisfies the orthogonality condition :

$$c^{(1)}(t^i {P_k}^{(1)}) = 0, \quad \text{for } i = 0, 1, \ldots, k-1. \tag{1.20}$$

Both families of $\{P_k\}$ and $\{{P_k}^{(1)}\}$ are called adjacent families of FOPs, [8].

1.3.1 The Computational Recurrence Relationships of FOP in Performing the Lanczos-type Algorithms

The FOP approach relies on a set of recurrence relations which express iterate $\mathbf{x}_k$ in terms of $\mathbf{x}_{k-1}$, $\mathbf{x}_{k-2}$ or even earlier iterates. This means that FOP's of different degrees are involved from the family of orthogonal polynomials, P_k, or adjacent families of orthogonal polynomials, $P_k^{(1)}$, [3, 6, 9]. In other words, FOP's of a certain degree can be written in terms of other FOP's. Below is one example of the derivation of Orthores, which is a Lanczos-type algorithm and based on formula A_4, [6], by using the theory FOPs.

The Formula $\mathbf{A}_4$

Consider for instance, the three term recurrence relationships below

$$P_k(t) = (A_k t^2 + B_k t + C_k)P_{k-2}(t) + (D_k t + E_k)P_{k-1}(t), \tag{1.21}$$

with $P_k(0) = 1$. Multiplying both sides of (1.21) by t^i for $i = 0, 1, \ldots, k-1$ and imposing the orthogonality condition with respect to the linear function c, we obtain

$$\begin{aligned} c(t^i P_k(t)) = {} & A_k c(t^{i+2} P_{k-2}(t)) + B_k c(t^{i+1} P_{k-2}(t)) + C_k c(t^i P_{k-2}(t)) + \\ & D_k c(t^{i+1} P_{k-1}(t)) + E_k c(t^i P_{k-1}(t)), \end{aligned} \tag{1.22}$$

for $i = 0, 1, \ldots, k-1$. For $i = k-4$, we get :

$$\begin{aligned} c(t^{k-4} P_k) = {} & A_k c(t^{k-2} P_{k-2}) + B_k c(t^{k-3} P_{k-2}) + C_k c(t^{k-4} P_{k-2}) + D_k c(t^{k-3} P_{k-1}) + \\ & E_k c(t^{k-4} P_{k-1}) \end{aligned}$$

The only non-zero term of the relation is $A_kc(t^{k-2}P_{k-2})$. Since the left hand side of this relation is also zero, while $c(t^{k-2}P_{k-2})$ is never zero, we obtain

$$A_k = 0. \tag{1.23}$$

For $i = k-3$, we obtain :

$$c(t^{k-3}P_k) = A_kc(t^{k-1}P_{k-2}) + B_kc(t^{k-2}P_{k-2}) + C_kc(t^{k-3}P_{k-2}) + D_kc(t^{k-2}P_{k-1}) + \\ E_kc(t^{k-3}P_{k-1}).$$

The remaining terms which are not zero are $A_kc(t^{k-1}P_{k-2}) + B_kc(t^{k-2}P_{k-2})$. Since we have (1.23) and since $c(t^{(k-2)}P_{k-2}) \neq 0$ then we get

$$B_k = 0. \tag{1.24}$$

For $i = k-2$, we get

$$c(t^{k-2}P_k) = A_kc(t^{k}P_{k-2}) + B_kc(t^{k-1}P_{k-2}) + C_kc(t^{k-2}P_{k-2}) + D_kc(t^{k-1}P_{k-1}) + \\ E_kc(t^{k-2}P_{k-1}).$$

All of the terms on both the right and the left sides are different from zero, but $E_kc(t^{k-2}P_{k-1})$. Consequently, we obtain

$$C_kc(t^{k-2}P_{k-2}) + D_kc(t^{k-1}P_{k-1}) = 0, \tag{1.25}$$

by applying (1.23) and (1.24). For $i = k-1$, we obtain :

$$c(t^{k-1}P_k) = A_kc(t^{k+1}P_{k-2}) + B_kc(t^{k}P_{k-2}) + C_kc(t^{k-1}P_{k-2}) + D_kc(t^{k}P_{k-1}) + \\ E_kc(t^{k-1}P_{k-1})$$

which implies

$$C_k c(t^{k-1}P_{k-2}) + D_k c(t^k P_{k-1}) + E_k c(t^{k-1}P_{k-1}) = 0. \tag{1.26}$$

If we apply the condition $P_k(0) = 1$ to (1.21), then we have relation of

$$C_k + E_k = 1. \tag{1.27}$$

We now have a 3×3 system of linear equations in C_k, D_k and E_k of (1.25),(1.26), and (1.27). This system has determinant and the right-hand side respectively as follows

$$\begin{aligned}\Delta_k &= \begin{vmatrix} c(t^{k-2}P_{k-2}) & c(t^{k-1}P_{k-1}) & 0 \\ c(t^{k-1}P_{k-2}) & c(t^k P_{k-1}) & c(t^{k-1}P_{k-1}) \\ 1 & 0 & 1 \end{vmatrix} \\ &= c(t^{k-1}P_{k-1})^2 + c(t^{k-2}P_{k-2})c(t^k P_{k-1}) - c(t^{k-1}P_{k-1})c(t^{k-1}P_{k-2}). \end{aligned} \tag{1.28}$$

$$b = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Solving the system above gives

$$C_k = \frac{c(t^{k-1}P_{k-1})^2}{\Delta_k} \tag{1.29}$$

$$D_k = \frac{c(t^{k-2}P_{k-2})c(t^{k-1}P_{k-1})}{\Delta_k} \tag{1.30}$$

$$E_k = \frac{c(t^{k-2}P_{k-2})c(t^k P_{k-1}) - c(t^{k-1}P_{k-1})c(t^{k-1}P_{k-2})}{\Delta_k}. \tag{1.31}$$

Now we substitute all of the coefficients into P_k in (1.21) to obtain

$$P_k(t) = C_k P_{k-2}(t) + (D_k t + E_k)P_{k-1}(t), \tag{1.32}$$

with C_k, D_k, and E_k being given by (1.29),(1.30), and (1.31) respectively.

As an illustration of the use of the theory of FOPs, let us explain how $\mathbf{r}_k$ can be calculated recursively through the relation (1.32). Since $\mathbf{r}_k = P_k(A)\mathbf{r}_0$, we have

$$\begin{aligned}\mathbf{r}_k &= P_k(A)\mathbf{r}_0\\ &= C_k P_{k-2}(A)\mathbf{r}_0 + D_k A P_{k-1}(A)\mathbf{r}_0 + E_k P_{k-1}(A)\mathbf{r}_0\\ &= C_k\mathbf{r}_{k-2} + D_k A\mathbf{r}_{k-1} + E_k\mathbf{r}_{k-1}. \end{aligned}\tag{1.33}$$

On the other hand, we have

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k,$$

and by re-ordering it, we obtain $\mathbf{x}_k$ as follows.

$$\begin{aligned}A\mathbf{x}_k &= \mathbf{b} - \mathbf{r}_k\\ &= \mathbf{b} - (D_k A\mathbf{r}_{k-1} + E_k\mathbf{r}_{k-1} + C_k\mathbf{r}_{k-2})\\ \mathbf{x}_k &= A^{-1}\mathbf{b} - D_k\mathbf{r}_{k-1} - E_k A^{-1}\mathbf{r}_{k-1} - C_k A^{-1}\mathbf{r}_{k-2}\\ &= \mathbf{x}_{k-1} - D_k\mathbf{r}_{k-1} - C_k(\mathbf{x}_{k-1} - \mathbf{x}_{k-2})\\ &= (1 - C_k)\mathbf{x}_{k-1} + C_k\mathbf{x}_{k-2} - D_k\mathbf{r}_{k-1}\\ &= E_k\mathbf{x}_{k-1} + C_k\mathbf{x}_{k-2} - D_k\mathbf{r}_{k-1},\\ &= D_k(\frac{E_k}{D_k}\mathbf{x}_{k-1} + \frac{C_k}{D_k}\mathbf{x}_{k-2} - \mathbf{r}_{k-1})\\ &= D_k(\gamma_k\mathbf{x}_{k-1} + \delta_k\mathbf{x}_{k-2} - \mathbf{r}_{k-1}), \end{aligned}\tag{1.34}$$

where the scalars γ_k and δ_k are computed as following

$$\begin{aligned}\delta_k &= \frac{C_k}{D_k} \\ &= \frac{c(t^{k-1}P_{k-1})^2}{\Delta_k}\frac{\Delta_k}{c(t^{k-2}P_{k-2})c(t^{k-1}P_{k-1})} \\ &= \frac{c(t^{k-1}P_{k-1})}{c(t^{k-2}P_{k-2})} \end{aligned} \tag{1.35}$$

and

$$\begin{aligned}\gamma_k &= \frac{E_k}{D_k} \\ &= \frac{c(t^{k-2}P_{k-2})c(t^kP_{k-1}) - c(t^{k-1}P_{k-1})c(t^{k-1}P_{k-2})}{\Delta_k}\frac{\Delta_k}{c(t^{k-2}P_{k-2})c(t^{k-1}P_{k-1})} \\ &= \frac{c(t^{k-2}P_{k-2})c(t^kP_{k-1}) - c(t^{k-1}P_{k-1})c(t^{k-1}P_{k-2})}{c(t^{k-2}P_{k-2})c(t^{k-1}P_{k-1})} \\ &= \frac{c(t^kP_{k-1}) - \delta_k c(t^{k-1}P_{k-2})}{c(t^{k-1}P_{k-1})}. \end{aligned} \tag{1.36}$$

On the other hand, from (1.35) and (1.36), we have information that

$$\delta_k + \gamma_k = \frac{1}{D_k},$$

or

$$D_k = \frac{1}{\delta_k + \gamma_k}. \tag{1.37}$$

Similarly, relations (1.33) can be written as

$$\begin{aligned}\mathbf{r}_{k+1} &= D_{k+1}(A\mathbf{r}_k + \frac{E_{k+1}}{D_{k+1}}\mathbf{r}_k + \frac{C_{k+1}}{D_{k+1}}\mathbf{r}_{k-1}) \\ &= D_{k+1}(A\mathbf{r}_k + \gamma_{k+1}\mathbf{r}_k + \delta_{k+1}\mathbf{r}_{k-1}), \end{aligned} \tag{1.38}$$

where

$$\begin{aligned}\delta_{k+1} &= \frac{c(t^k P_k)}{c(t^{k-1}P_{k-1})}\\ &= \frac{\langle \mathbf{y}, A^k \mathbf{r}_k\rangle}{\langle \mathbf{y}, A^{k-1}\mathbf{r}_{k-1}\rangle}\\ &= \frac{\langle (A^T)^k \mathbf{y}, \mathbf{r}_k\rangle}{\langle (A^T)^{k-1}\mathbf{y}, \mathbf{r}_{k-1}\rangle}\\ &= \frac{\langle \mathbf{y}_k, \mathbf{r}_k\rangle}{\langle \mathbf{y}_{k-1}, \mathbf{r}_{k-1}\rangle},\end{aligned}$$

$$\begin{aligned}\gamma_{k+1} &= \frac{c(t^{k+1}P_k) - \delta_{k+1}c(t^k P_{k-1})}{c(t^k P_k)}\\ &= \frac{\langle \mathbf{y}, A^{k+1}\mathbf{r}_k\rangle - \delta_{k+1}\langle \mathbf{y}, A^k \mathbf{r}_{k-1}\rangle}{\langle \mathbf{y}, A^k \mathbf{r}_k\rangle}\\ &= \frac{\langle (A^T)^k \mathbf{y}, A\mathbf{r}_k\rangle - \delta_{k+1}\langle (A^T)^k \mathbf{y}, \mathbf{r}_{k-1}\rangle}{\langle (A^T)^k \mathbf{y}, \mathbf{r}_k\rangle}\\ &= \frac{\langle \mathbf{y}_k, A\mathbf{r}_k\rangle - \delta_{k+1}\langle \mathbf{y}_k, \mathbf{r}_{k-1}\rangle}{\langle \mathbf{y}_k, \mathbf{r}_k\rangle}\end{aligned}$$

and

$$D_{k+1} = \frac{1}{\delta_{k+1} + \gamma_{k+1}}.$$

This algorithm for the computation of the vectors $\mathbf{x}_k$ is known as Orthores, [83]. Its implementation can be seen in the Algorithm 1.

Formula A$_8$

Formula A$_8$ is obtained by calculating recursively the family of orthogonal polynomial $\{P_k\}$ from the $P^{(1)}_{k-1}$ and P_{k-1}. Now we follow the computation of the formula.

Algorithm 1 Implementation of A_4 (Orthores), [6]

```
1: Input :
       • $A$, an $n \times n$ matrix,
       • $\mathbf{b}$, an $n$ -vector,
       • $k$, a positive integer less than or equal to $n$.
2: Output :
       • the approximations solution, $\mathbf{x}_{k+1}$
       • the norm of residuals , $||\mathbf{r}_{k+1}||$
3: Fix the number of iterations to $k$ and the tolerance $\epsilon$ to $1E - 13$.
4: Initialization.    Choose $\mathbf{x}_0$ and $\mathbf{y}$. Set
       $$\begin{aligned} \mathbf{r}_0 &= \mathbf{b} - A\mathbf{x}_0, \\ \mathbf{y}_0 &= \mathbf{y}, \\ \delta_1 &= 0, \\ \gamma_1 &= \frac{\langle \mathbf{y}_0, A\mathbf{r}_0 \rangle}{\langle \mathbf{y}_0, \mathbf{r}_0 \rangle}, \\ D_1 &= \frac{1}{\gamma_1}, \\ \mathbf{x}_1 &= D_1(\gamma_1 \mathbf{x}_0 - \mathbf{r}_0), \\ \mathbf{r}_1 &= D_1(A\mathbf{r}_0 + \gamma_1 \mathbf{r}_0). \end{aligned}$$
5: for $k = 1, 2, \ldots$ do
6:     $\mathbf{y}_k = A^T \mathbf{y}_{k-1}$
7:     $\delta_{k+1} = \frac{\langle \mathbf{y}_k, \mathbf{r}_k \rangle}{\langle \mathbf{y}_{k-1}, \mathbf{r}_{k-1} \rangle}$
8:     $\gamma_{k+1} = \frac{\langle \mathbf{y}_k, A\mathbf{r}_k \rangle - \delta_{k+1} \langle \mathbf{y}_k, \mathbf{r}_{k-1} \rangle}{\langle \mathbf{y}_k, \mathbf{r}_k \rangle}$
9:     $D_{k+1} = \frac{1}{\delta_{k+1} + \gamma_{k+1}}$
10:    $\mathbf{x}_{k+1} = D_{k+1}(\gamma_{k+1} \mathbf{x}_k + \delta_{k+1} \mathbf{x}_{k-1} - \mathbf{r}_k)$
11:    $\mathbf{r}_{k+1} = D_{k+1}(A\mathbf{r}_k + \gamma_{k+1} \mathbf{r}_k + \delta_{k+1} \mathbf{r}_{k-1})$
12: end for
```

Consider following relations

$$P_k(t) = A_k t P^{(1)}_{k-1}(t) + P_{k-1}(t), \tag{1.39}$$

with $P_k(0) = 1$. Multiply both sides of (1.39) by t^i for $i = 0, 1, \ldots, k-1$ and impose orthogonality condition with respect to the linear function c to obtain :

$$c(t^i P_k) = A_k c(t^{i+1} P^{(1)}_{k-1}) + c(t^i P_{k-1}) \tag{1.40}$$

for $i = 0, 1, \ldots, k-1$. In fact, for $i = 0, 1, \ldots, k-2$, both sides agree each other. Now shall we look at for $i = k-1$. We have

$$c(t^{k-1} P_k) = A_k c(t^k P^{(1)}_{k-1}) + c(t^{k-1} P_{k-1}). \tag{1.41}$$

Since the left hand side of this relation is zero, we obtain the solution for A_k as follows

$$A_k = \frac{-c(t^{k-1} P_{k-1})}{c(t^k P^{(1)}_{k-1})}. \tag{1.42}$$

The computation of the residuals $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k = P_k(A)\mathbf{r}_0$ and the corresponding $\mathbf{x}_k$ are given below.

$$\begin{aligned} \mathbf{r}_{k+1} &= P_{k+1}(A)\mathbf{r}_0 \\ &= A_{k+1} A P^{(1)}_k(A)\mathbf{r}_0 + P_k(A)\mathbf{r}_0 \\ &= A_{k+1} A \mathbf{z}_k + \mathbf{r}_k. \end{aligned} \tag{1.43}$$

On the other hand, we have

$$\mathbf{r}_{k+1} = \mathbf{b} - A\mathbf{x}_{k+1},$$

and by re-order it, we obtain the form of the $\mathbf{x}_k$ as follows.

$$\begin{aligned} A\mathbf{x}_{k+1} &= \mathbf{b} - \mathbf{r}_{k+1} \\ &= \mathbf{b} - A_{k+1}A\mathbf{z}_k - \mathbf{r}_k \\ \mathbf{x}_{k+1} &= A^{-1}\mathbf{b} - \mathbf{z}_k - A^{-1}\mathbf{r}_k \\ &= \mathbf{x}_k - \mathbf{z}_k, \end{aligned} \tag{1.44}$$

where

$$\begin{aligned} A_{k+1} &= \frac{-c(t^k P_k)}{c(t^{k+1}P_k^{(1)})} \\ &= \frac{-\langle \mathbf{y}, A^k \mathbf{r}_k\rangle}{\langle \mathbf{y}, A^{k+1}\mathbf{z}_k\rangle} \\ &= \frac{-\langle (A^T)^k \mathbf{y}, \mathbf{r}_k\rangle}{\langle (A^T)^k \mathbf{y}, A\mathbf{z}_k\rangle} \\ &= \frac{-\langle \mathbf{y}_k, \mathbf{r}_k\rangle}{\langle \mathbf{y}_k, A\mathbf{z}_k\rangle}. \end{aligned} \tag{1.45}$$

Formula B$_6$

It is obtained by calculating recursively the family of orthogonal polynomial $P_k^{(1)}$ from the $P_{k-1}^{(1)}$ and $P_{k-2}^{(1)}$. Now we follow the computation of the formula.

Consider relations below :

$$P_k^{(1)}(t) = B_k P_{k-2}^{(1)}(t) + (t + C_k)P_{k-1}^{(1)}(t). \tag{1.46}$$

Multiply both sides of (1.46) by t^i for $i = 0, 1, \ldots, k-1$ and impose the orthogonality condition with respect to the linear function $c^{(1)}$, we obtain

$$c^{(1)}(t^i P_k^{(1)}) = B_k c^{(1)}(t^i P_{k-2}^{(1)}) + c^{(1)}(t^{i+1}P_{k-1}^{(1)}) + C_k c^{(1)}(t^i P_{k-1}^{(1)}). \tag{1.47}$$

Shall we apply some i's into this relation to obtain the solutions. For $i = 0, 1, \ldots, k-$

3, both sides of (1.47) meet each other. For $i = k - 2$, we have

$$\begin{aligned}
c^{(1)}(t^{k-2}P_k^{(1)}) &= B_k c^{(1)}(t^{k-2}P_{k-2}^{(1)}) + c^{(1)}(t^{k-1}P_{k-1}^{(1)}) + C_k c^{(1)}(t^{k-2}P_{k-1}^{(1)}) \\
0 &= B_k c^{(1)}(t^{k-2}P_{k-2}^{(1)}) + c^{(1)}(t^{k-1}P_{k-1}^{(1)}) \\
B_k &= \frac{-c^{(1)}(t^{k-1}P_{k-1}^{(1)})}{c^{(1)}(t^{k-2}P_{k-2}^{(1)})} \\
&= \frac{-c(t^k P_k^{(1)})}{c(t^{k-1}P_{k-2}^{(1)})}.
\end{aligned} \tag{1.48}$$

For $i = k - 1$, we have

$$\begin{aligned}
c^{(1)}(t^{k-1}P_k^{(1)}) &= B_k c^{(1)}(t^{k-1}P_{k-2}^{(1)}) + c^{(1)}(t^k P_{k-1}^{(1)}) + C_k c^{(1)}(t^{k-1}P_{k-1}^{(1)}) \\
0 &= B_k c^{(1)}(t^{k-1}P_{k-2}^{(1)}) + c^{(1)}(t^k P_{k-1}^{(1)}) + C_k c^{(1)}(t^k P_{k-1}^{(1)}) \\
C_k &= \frac{-B_k c^{(1)}(t^{k-1}P_{k-2}^{(1)}) - c^{(1)}(t^k P_{k-1}^{(1)})}{c^{(1)}(t^k P_{k-1}^{(1)})} \\
&= \frac{-B_k c(t^k P_k^{(1)}) - c(t^{k+1}P_{k-2}^{(1)})}{c(t^{k+1}P_{k-1}^{(1)})}.
\end{aligned} \tag{1.49}$$

Shall we now calculate formula $\mathbf{z}_k$ by considering the form of $\mathbf{z}_k = P_k^{(1)}(A)\mathbf{z}_0$. We obtain

$$\begin{aligned}
\mathbf{z}_{k+1} &= P_{k+1}^{(1)}(A)\mathbf{z}_0 \\
&= B_{k+1}P_{k-1}^{(1)}(A) + (A + C_{k+1})P_k^{(1)}(A) \\
&= B_{k+1}\mathbf{z}_{k-2} + A\mathbf{z}_{k-1} + C_{k+1}\mathbf{z}_{k-1},
\end{aligned} \tag{1.50}$$

where :

$$\begin{aligned} B_{k+1} &= \frac{-c(t^{k+1}P_{k+1}^{(1)})}{c(t^k P_{k-1}^{(1)})} \\ &= \frac{-\langle \mathbf{y}, A^{k+1}\mathbf{z}_{k+1}\rangle}{\langle \mathbf{y}, A^k \mathbf{z}_{k-1}\rangle} \\ &= \frac{-\langle (A^T)^k \mathbf{y}, A\mathbf{z}_{k+1}\rangle}{\langle (A^T)^k \mathbf{y}, \mathbf{z}_{k-1}\rangle} \\ &= \frac{-\langle \mathbf{y}_k, A\mathbf{z}_{k+1}\rangle}{\langle \mathbf{y}_k, \mathbf{z}_{k-1}\rangle}. \end{aligned} \tag{1.51}$$

and

$$\begin{aligned} C_{k+1} &= \frac{-B_{k+1}c(t^{k+1}P_{k+1}^{(1)}) - c(t^{k+2}P_{k-1}^{(1)})}{c(t^{k+2}P_k^{(1)})} \\ &= \frac{-B_{k+1}\langle \mathbf{y}, A^{k+1}\mathbf{z}_{k+1}\rangle - \langle A\mathbf{y}, A^{k+1}\mathbf{z}_{k-1}\rangle}{\langle A\mathbf{y}, A^{k+1}\mathbf{z}_k\rangle} \\ &= \frac{-B_{k+1}\langle (A^T)^{k+1}\mathbf{y}, \mathbf{z}_{k+1}\rangle - \langle (A^T)^{k+1}\mathbf{y}, A\mathbf{z}_{k-1}\rangle}{\langle (A^T)^{k+1}\mathbf{y}, A\mathbf{z}_k\rangle} \\ &= \frac{-B_{k+1}\langle \mathbf{y}_{k+1}, \mathbf{z}_{k+1}\rangle - \langle \mathbf{y}_{k+1}, \mathbf{z}_{k-1}\rangle}{\langle \mathbf{y}_{k+1}, A\mathbf{z}_k\rangle} \end{aligned} \tag{1.52}$$

We will apply the form this $\mathbf{z}_k$ when the combination with the formula B_i is needed.

Formula A$_8$/B$_6$

The implementation of this combination is well known as the ORTHODIR algorithm [6]. Let us now design an algorithm which combines A$_8$ and B$_6$for computing the approximate solutions x_{k+1} and the residuals r_{k+1}. As we have (1.43) and (1.44) in the formula A$_8$, then we can apply B$_6$ for the purpose of calculating z_k which is the result can be seen in (1.50). Thus together with all of the scalars are put in the following algorithm.

Any number of algorithms can be established by following the steps explained

Algorithm 2 Algorithm Orthodir, [6]

```
 1: Initialization.    Choose $\mathbf{x}_0$ and $\mathbf{y}$. Set
        • $\mathbf{x} = \mathbf{x}_0$
        • $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
        • $\mathbf{y}_0 = \mathbf{y}$
        • $\mathbf{z}_0 = \mathbf{r}_0$
 2: Fix the number of iterations to $k$ and the tolerance $\epsilon$ to $1E - 13$.
 3: for $k = 0, 1, 2, \dots$ do
 4:     $\mathbf{y}_k = A^T \mathbf{y}_{k-1}$
 5:     $A_{k+1} = \frac{-\langle \mathbf{y}_k, \mathbf{r}_k \rangle}{\langle \mathbf{y}_k, A\mathbf{z}_k \rangle}$
 6:     $\mathbf{x}_{k+1} = \mathbf{x}_k - A_{k+1}\mathbf{z}_k$
 7:     $\mathbf{r}_{k+1} = \mathbf{r}_k + A_{k+1}A\mathbf{z}_k$
 8:     $B_{k+1} = \frac{-\langle \mathbf{y}_k, A\mathbf{z}_{k+1} \rangle}{\langle \mathbf{y}_k, \mathbf{z}_{k-1} \rangle}$
 9:     $C_{k+1} = \frac{-B_{k+1}\langle \mathbf{y}_{k+1}, \mathbf{z}_{k+1} \rangle - \langle \mathbf{y}_{k+1}, \mathbf{z}_{k-1} \rangle}{\langle \mathbf{y}_{k+1}, A\mathbf{z}_k \rangle}$
10:     $\mathbf{z}_{k+1} = B_{k+1}\mathbf{z}_{k-2} + A\mathbf{z}_{k-1} + C_{k+1}\mathbf{z}_{k-1}$.
11: end for
12: $sol_{last} = \mathbf{x}_k$
13: $norm_{last} = ||\mathbf{r}_k||$
14: Stop.
```

above. They differ by the degree of the orthogonal polynomials used in the recurrence relationships. However, there is common sense advantage in using low degree polynomials since only few coefficients will have to be identified. That is why, as appears in [3, 29, 31], common Lanczos-type algorithms are mostly obtained by forming the three-term recurrence relationship with the degree difference between the right and left hand-side being at most three.

1.4 The Breakdown Issue in Lanczos-type Algorithms

Although, in theory, Lanczos-type algorithms in exact arithmetic produce an exact solution in at most n steps, where n is the dimension of the system, a satisfactory approximate solution is often achieved in more than n steps. Breakdown in Lanczos-type algorithms occurs in the computation of orthogonal polynomials often, [8, 10], due to division by zero. However, it can also due to the non-existence of some orthogonal polynomials, which is known as *true breakdown* when computing the recurrence relationships, or because of the accumulation of computation errors (*ghost breakdown*) when running these algorithms. When breakdown happens in Lanczos-type algorithms, they stop. Therefore, strategies which deal with this situation are continually being developed.

There are many such strategies; the look-ahead strategy, [7], also called the Method of Recursive Zoom (MRZ), the look-around strategy, [39, 40], typically try to get over and/or around the non-existing orthogonal polynomials. Other strategies such as switching between Lanczos-type algorithms and restarting them have also been considered, [29, 30, 32]. These latter ones have been shown to perform better than MRZ in terms of robustness, [30, 32].

1.4.1 Breakdown in Computing Orthogonal Polynomials when Solving SLEs

Breakdown in Lanczos-type algorithms can be caused by either the non-existence of some orthogonal polynomials or an inappropriate recurrence relationship is used. Let us have a look at how the breakdown occurs when computing orthogonal polynomials, for more detail should we refer to [8],

Consider the three recurrence relationship of the monic polynomial P_{k+1} as follows :

$$P_{k+1}(t) = (\alpha t + \beta)P_k(t) + \gamma P_{k-1}(t), k = 0, 1, \ldots. \tag{1.53}$$

This polynomial satisfies the conditions :

1. $P_{-1}(t) = 0$,
2. $P_0(t) = 1$,
3. $c(t^i P_k) = 0$, for $i = 0, 1, \ldots, k-1$ and c is a linear function.

Condition 2 leads to the equation,

$$1 = \beta + \gamma. \tag{1.54}$$

Condition 3, which is also known as the orthogonality condition, gives :

$$c(t^i P_{k+1}) = \alpha c(t^{i+1}P_k(t)) + \beta c(t^i P_k(t)) + \gamma c(t^i P_{k-1}(t)), \tag{1.55}$$

for $i = 0, 1, \ldots, k$. In particular, for $i = k-1$ and $i = k$, we get relations

$$0 = \alpha c(t^k P_k(t)) + \gamma c(t^{k-1} P_{k-1}(t)), \tag{1.56}$$

$$0 = \alpha c(t^{k+1} P_k(t)) + \beta c(t^k P_k(t)) + \gamma c(t^k P_{k-1}(t)), \tag{1.57}$$

respectively. As we can see, we now have a system of linear equations in α, β, γ, the solutions of which is

$$\begin{aligned}
\alpha &= 1 \\
\gamma &= \frac{-c(t^k P_k(t))}{c(t^{k-1} P_{k-1}(t))} \\
\beta &= \frac{\gamma c(t^k P_{k-1}(t)) - c t^{k+1} P_k(t)}{c(t^k P_k(t))}.
\end{aligned} \tag{1.58}$$

We can see here that in order for the relation (1.53) to be finite, the scalars $c(t^{k-1}P_{k-1}(t))$ and $c(t^kP_k(t))$ must not be zero, or sufficiently close to zero, i.e. small enough to cause a NaN output in the computation. Otherwise, a breakdown occurs.

1.4.2 The Method of Recursive Zoom

MRZ, also known as the look-ahead algorithm detects the non-existence orthogonal polynomials and allows the process to jump over these and find existing ones to continue with the solution,[11, 12].

Consider the family of orthogonal polynomials

$$P_k(t) = a_0{}^{(k)} + a_1{}^{(k)} t + \cdots + a_k{}^{(k)} t^k. \tag{1.59}$$

The normal condition is satisfied when $a_0{}^{(k)} = 1$. In addition, the orthogonality condition ($c(t^k P_k(t) = 0)$) gives a systems of linear equations as following :

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ c_0 & c_1 & \cdots & c_k \\ c_1 & c_2 & \cdots & c_{k+1} \\ \vdots & & & \\ c_{k-1} & c_k & \cdots & c_{2k-1} \end{bmatrix} \begin{bmatrix} a_0^{(k)} \\ a_0^{(k)} \\ \vdots \\ a_0^{(k)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.60}$$

Let N_k be the coefficient matrix of the system, $\mathbf{d}_k$ be the right hand-side vector, and $\mathbf{z}_k$ the solution. Then we have

$$N_k \ \mathbf{z}_k = \mathbf{d}_k. \tag{1.61}$$

Before going further, let us have a look at the block bordering method which plays an important role in the derivation of the MRZ algorithm. Let N_k be an $n_k \times n_k$ matrix. Consider the N_{k+1} matrix

$$N_{k+1} = \begin{bmatrix} N_k & U_k \\ V_k & M_k \end{bmatrix} \tag{1.62}$$

where U_k, V_k, and M_k are matrices of dimension $n_k \times m_k$, $m_k \times n_k$, and $m_k \times m_k$, respectively. If B_k is the Schur Complement, [22], of N_k in N_{k+1}, then

$$B_k = M_k - V_k N_K^{-1} U_k. \tag{1.63}$$

In fact, from the Schur Complement, we have

$$\det N_{k+1} = \det N_k \det B_k. \tag{1.64}$$

If B_k is non-singular then we obtain

$$N_{k+1}^{-1} = \begin{bmatrix} N_k^{-1} + N_k^{-1}U_kB_K^{-1}V_kN_k^{-1} & -N_k^{-1}U_kB_k^{-1} \\ -B_k^{-1}V_kN_k^{-1} & B_k^{-1} \end{bmatrix} \tag{1.65}$$

As we have (1.61), it is easy to follow that

$$N_{k+1}\ \mathbf{z}_{k+1} = \mathbf{d}_{k+1} = \begin{bmatrix} \mathbf{d}_k \\ \mathbf{f}_k \end{bmatrix} \tag{1.66}$$

where $\mathbf{f}_k$ is a vector of dimension m. Thus we get

$$\begin{aligned} \mathbf{z}_{k+1} &= N_{k+1}^{-1} \begin{bmatrix} \mathbf{d}_k \\ \mathbf{f}_k \end{bmatrix} \\ &= \begin{bmatrix} N_k^{-1} + N_k^{-1}U_kB_K^{-1}V_kN_k^{-1} & -N_k^{-1}U_kB_k^{-1} \\ -B_k^{-1}V_kN_k^{-1} & B_k^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{d}_k \\ \mathbf{f}_k \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{z}_k + N_k^{-1}U_kB_K^{-1}V_k\mathbf{z}_k - N_k^{-1}U_kB_k^{-1}\mathbf{f}_k \\ -B_k^{-1}V_k\mathbf{z}_k + B_k^{-1}\mathbf{f}_k \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{z}_k \\ 0 \end{bmatrix} - \begin{bmatrix} -N_k^{-1}U_k \\ I \end{bmatrix} B_k^{-1}V_k\mathbf{z}_k \end{aligned} \tag{1.67}$$

If we multiply both sides of (1.83) by vector $\begin{bmatrix} 1 & t & \cdots & t^{n_k+1} \end{bmatrix}^T$, then we obtain

$$\begin{aligned} \mathbf{z}_{k+1} &= \begin{bmatrix} \mathbf{z}_k \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ \vdots \\ t^{n_k+1} \end{bmatrix} - \begin{bmatrix} -N_k^{-1}U_k \\ I \end{bmatrix} B_k^{-1}V_k\mathbf{z}_k \begin{bmatrix} 1 \\ t \\ \vdots \\ t^{n_k+1} \end{bmatrix} & (1.68) \\ P_{k+1} &= P_k(t) - Q_{k+1}(t). & (1.69) \end{aligned}$$

The problem is how to determine the polynomial Q_k. However, from (1.68) we know that the Q_{k+1} has degree at most $(n_k + 1)$ and satisfies

$$c(t^iQ_{k+1}(t)) = c(t^iP_k(t)) - c(t^iP_{k+1})(t)) = \begin{cases} 0, & \text{for } i = 0, 1, \cdots, n_{k-1} \\ c(t^iP_k(t)), & \text{for } i = n_k, n_{k+1}, \cdots, n_k + m_k - 1 \end{cases} \tag{1.70}$$

Let $P_k^{(1)}$ be the monic polynomial of degree n_k belonging to the family of formal orthogonal polynomial with respect to the linear functional $c^{(1)}$ which satisfies :

$$c^{(1)}(t^iP_k^{(1)}(t)) = c(t^{i+1}P_k^{(1)}(t)) = 0, \tag{1.71}$$

for $i = 0, 1, \cdots, n_k - 1$. Consider the form of Q_{k+1} as following

$$\begin{aligned} Q_{k+1} &= t(\beta_0 + \beta_1 t + \cdots + \beta_{m_k-1}t^{m_k-1})P_k^{(1)}(t) \\ &= tw_k(t)P_k^{(1)}(t), \end{aligned} \tag{1.72}$$

where w_k is the polynomial of degree $(m_k - 1)$. According to the Draux's theorem in [24], the *jumping* m_k is defined by the conditions

$$c^{(1)}(t^iP_k^{(1)}(t)) = \begin{cases} 0, & \text{for } i = 0, 1, \cdots, n_k + m_k - 2 \\ \neq 0, & \text{for } i = n_k + m_k - 1 \end{cases} \tag{1.73}$$

Thus this gives us

$$c(t^iQ_{k+1}(t)) = c(t^{i+1}w_kP_k^{(1)}(t)) = 0, \tag{1.74}$$

for $i = 0, 1, \cdots, n_k - 1$. Thus we obtain the systems of linear equations by applying the values of i's into (1.70). For $i = n_k$

$$\begin{aligned}
c^{(1)}(t^{n_k}(\beta_0 + \beta_1 t + \cdots + \beta_{m_k-1}t^{m_k-1})P_k^{(1)}) &= c(t^{n_k}P_k(t)) \\
\beta_0 c^{(1)}(t^{n_k}P_k^{(1)}) + \beta_1 c^{(1)}(t^{n_k+1}P_k^{(1)}) + \cdots + \beta_{m_k-1}c^{(1)}(t^{m_k+n_k-1}P_k^{(1)}) &= c(t^{n_k}P_k(t)) \\
\beta_{m_k-1}c^{(1)}(t^{m_k+n_k-1}P_k^{(1)}) &= c(t^{n_k}P_k(t)).
\end{aligned}$$

For $i = n_k + 1$,

$$\begin{aligned}
c^{(1)}(t^{n_k+1}(\beta_0 + \beta_1 t + \cdots + \beta_{m_k-1}t^{m_k-1})P_k^{(1)}) &= c(t^{n_k+1}P_k(t)) \\
\beta_0 c^{(1)}(t^{n_k+1}P_k^{(1)}) + \beta_1 c^{(1)}(t^{n_k+2}P_k^{(1)}) + \cdots + \beta_{m_k-1}c^{(1)}(t^{m_k+n_k}P_k^{(1)}) &= c(t^{n_k+1}P_k(t)) \\
\beta_{m_k-2}c^{(1)}(t^{m_k+n_k-1}P_k^{(1)}) + \beta_{m_k-1}c^{(1)}(t^{m_k+n_k}P_k^{(1)}) &= c(t^{n_k+1}P_k(t)).
\end{aligned}$$

$$\vdots$$

For $i = n_k + m_k - 1$

$$\begin{aligned}
c^{(1)}(t^{n_k+m_k-1}(\beta_0 + \beta_1 t + \cdots + \beta_{m_k-1}t^{m_k-1})P_k^{(1)}) &= c(t^{n_k+m_k-1}P_k(t)) \\
\beta_0 c^{(1)}(t^{n_k+m_k-1}P_k^{(1)}) + \beta_1 c^{(1)}(t^{n_k+m_k}P_k^{(1)}) + \cdots + \beta_{m_k-1}c^{(1)}(t^{n_k+2m_k-2}P_k^{(1)}) &= c(t^{n_k+m_k-1}P_k(t)).
\end{aligned}$$

Thus we have the following triangular system of linear equations.

$$\begin{bmatrix} c^{(1)}(t^{n_k+m_k-1}P_k^{(1)}) & c^{(1)}(t^{n_k+m_k}P_k^{(1)}) & \cdots & c^{(1)}(t^{n_k+2m_k-2}P_k^{(1)}) \\ \vdots & & & \\ 0 & 0 & \cdots & c^{(1)}(t^{n_k+m_k}P_k^{(1)}) \\ 0 & 0 & \cdots & c^{(1)}(t^{n_k+m_k-1}P_k^{(1)} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{m_k-1} \end{bmatrix} = \begin{bmatrix} c(t^{n_k}P_k(t)) \\ c(t^{n_k+1}P_k(t)) \\ \vdots \\ c(t^{n_k+m_k-1}P_k(t)) \end{bmatrix}. \tag{1.75}$$

The solutions of this system must be consistent since the coefficient matrix is a non-singular. From (1.68), eventually we have formula for the P_{k+1}

$$P_{k+1} = P_k - tw_k(t)P_k^{(1)}(t), \tag{1.76}$$

where

$$w_k = \beta_0 + \beta_1 t + \cdots + \beta_{m_k-1} t^{m_k-1}, \tag{1.77}$$

and the scalars $\beta_0, \beta_1, \ldots, \beta_{m_k-1}$ can be calculated by solving the system (1.75). We still need to compute the formula for $P^{(1)}_{k+1}$ to complete the whole of the algorithm. For that, let us consider the form of the monic polynomial q_k of degree m_k below

$$q_k(t) = \alpha_0 + \alpha_1 t + \cdots + \alpha_{m_k-1} t^{m_k-1} + t^{m_k}, \tag{1.78}$$

and also consider the recurrence relationship of the form

$$P^{(1)}_{k+1}(t) = q_k(t) P^{(1)}_k(t) - C_{k+1} P^{(1)}_{k-1}(t), \tag{1.79}$$

for $k = 0, 1, \ldots$. Similarly, we multiply both sides of (1.79), applying the orthogonality condition with respect to the linear function $c^{(1)}$ we obtain a triangular system of linear equations which means the solutions must exist.

$$c^{(1)}(t^i P^{(1)}_{k+1}(t)) = c^{(1)}(t^i q_k(t) P^{(1)}_k(t)) - C_{k+1} c^{(1)}(t^i P^{(1)}_{k-1}(t)). \tag{1.80}$$

Finally, the recursive computation of the residual $\mathbf{r}_{k+1}$ and the approximate solutions $\mathbf{x}_{k+1}$ are considered by applying both formulae (1.76) and (1.79). The resulting method is called the MRZ algorithm (or the Method of Recursive Zoom) which guarantees that we will not face breakdown. The only breakdown possibly faced is the incurable hard breakdown. After applying $\mathbf{r}_k = P_k(A)\mathbf{r}_0$ and $\mathbf{z}_k =$

$P_k^{(1)}(A)\mathbf{r}_0$, we have the relations as following :

$$\begin{aligned}\mathbf{r}_{k+1} &= P_{k+1}(A)\mathbf{r}_0\\ &= P_k(A)\mathbf{r}_0 - Aw_k(A)P_k^{(1)}(A)\mathbf{r}_0\\ &= \mathbf{r}_k - Aw_k(A)\mathbf{z}_k, \end{aligned}\tag{1.81}$$

which is used to compute the residuals. To compute the approximate solutions, we use the fact of $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ and thus we obtain

$$\begin{aligned}A\mathbf{x}_{k+1} &= \mathbf{b} - \mathbf{r}_{k+1}\\ &= \mathbf{b} - (\mathbf{r}_k - Aw_k(A)\mathbf{z}_k)\\ \mathbf{x}_{k+1} &= A^{-1}\mathbf{b} - A^{-1}\mathbf{r}_k + w_k(A)\mathbf{z}_k\\ &= \mathbf{x}_k + w_k(A)\mathbf{z}_k. \end{aligned}\tag{1.82}$$

For calculating z_{k+1}, we do as following

$$\begin{aligned}\mathbf{z}_{k+1} &= P_{k+1}^{(1)}(A)\mathbf{z}_0\\ &= q_k(A)P_k^{(1)}(A)\mathbf{z}_0 - C_{k+1}P_{k-1}^{(1)}(A)\mathbf{z}_0\\ &= q_k(A)\mathbf{z}_k - C_{k+1}\mathbf{z}_{k-1}. \end{aligned}\tag{1.83}$$

Altogether we have the following algorithm :

Algorithm 3 Implementation of MRZ, [12]

```
1: Initialization.  Choose $\mathbf{x}_0$ and $\mathbf{y}$. Set $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{y}_0 = \mathbf{y}$, $\mathbf{z}_0 = \mathbf{r}_0$, $n_0 = 0$.
2: for $k = 1, 2, \ldots$ do
3:     if $n_k = n$ then
4:         The solutions are obtained after $n$ iterations.
5:         STOP.
6:         $\mathbf{z}_1 = A\mathbf{z}_0$.
7:     end if
8:     if $\langle y, z_1 \rangle = 0$ and $n_k = n - 1$ then
9:         Incurable Breakdown.
10:        STOP.
11:        $m_k = 1$.
12:    end if
13:    while $\langle \mathbf{y}, \mathbf{z}_{m_k} \rangle \neq 0$ and $m_k < n - n_k$ do
14:        $m_k = m_k + 1$.
15:        $\mathbf{z}_{m_k} = A\mathbf{z}_{m_k - 1}$.
16:    end while
17:    if $m_k = n - n_k$ and $\langle \mathbf{y}, \mathbf{z}_{m_k} \rangle = 0$ then
18:        INCURABLE BREAKDOWN.
19:        STOP.
20:        $c_0 = \langle \mathbf{y}, \mathbf{r}_k \rangle$,
21:        $d_0 = \langle \mathbf{y}, \mathbf{z}_{m_k} \rangle$,
22:        $\beta_{m_k - 1} = \frac{c_0}{d_0}$.
23:    end if
24:    for $i = 1, 2, \ldots, m_k$ do
25:        $\mathbf{y} = A^T\mathbf{y}$,
26:        $d_i = \langle \mathbf{y}, \mathbf{z}_{m_k} \rangle$,
27:        $c_i = \langle \mathbf{y}, \mathbf{r}_k \rangle$.
28:        if $i \neq m_k$ , compute then
29:            $\beta_{m_k - i - 1}$
30:        end if
31:        if $k \neq 0$ and $i > m_{k-1}$ then
32:            $p_i = \langle \mathbf{y}, \mathbf{z}_{k-1} \rangle$
33:        end if
34:        $\mathbf{r}_{k+1} = \mathbf{r}_k - Aw_k(A)\mathbf{z}_k$
35:        $\mathbf{x}_{k+1} = x_k + w_k(A)\mathbf{z}_k)$
36:        if $\mathbf{r}_{k+1} = 0$ then
37:            SOLUTION REACHED.
38:            STOP.
39:            $n_{k+1} = n_k + m_k$
40:        end if
41:        if $k = 0$ then
42:            $C_1 = 0$
43:            $p_0 = 0$
44:        else
45:            $C_{k+1} = d_0/p_0$
46:        end if
47:        $\alpha_{m_k - i}$
48:        if $k = 0$ then
49:            $p_i = 0$
50:        end if
51:    end for
52:    for $i = 0, 1, \ldots, m_k$ do
53:        $\mathbf{z}_{k+1} = q_k(A)\mathbf{z}_k - C_{k+1}\mathbf{z}_{k-1}$.
54:        STOP.
55:    end for
56: end for
```

1.4.3 The Restarting Approach

Restarting of iterative methods to improve convergence and avoid breakdown is not new. The restarting of an Arnoldi-type algorithm, namely GMRES, [81], has been considered in, [76]. Restarting Lanczos-type algorithms is more recent; it has been investigated in [29, 30] where it has been found to be effective. The idea is either to stop the Lanczos-type algorithm pre-emptively and restart it with some iterate or wait until breakdown occurs and then restart from some iterate. This simple approach proves to be very effective although the pre-emptive variant is very difficult to implement since it is not easy to detect breakdown prior to its occurence. One important question on restarting is the quality of the restarting point. This issue will be the subject of Chapter 2 where restarting will be treated more comprehensively. The specific algorithm A_8B_6, also referred to the literature as Orthodir, [6], will be considered.

1.5 The Aims of the project

The aim of this project is to investigate a number of strategies to deal with breakdown in Lanczos-type algorithms. This aim contains several individual objectives as follows.

- Revisiting Lanczos-type algorithms and their derivation, reviewing the most important work on the issue of breakdown in particular.
- Investigating the suitability of certain points as starting ones for a Lanczos-type algorithm. These are the last iterate preceding breakdown, the iterate with the lowest residual norm in the sequence of iterates generated so far,

and the iterate whose entries are the median values of all entries of the iterates generated so far.

- Introducing an alternative approach to dealing with breakdown by embedding a regression model in the Lanczos-type algorithm. Provide a theoretical justification for it and test it empirically within the framework of restarting.
- Solving large scale problems (upto 10^6 variables) with the suggested approach using local parallel computing facilities and those provided by Cloud computing organisations.
- Comparing different approaches as appropriate on non trivial problems and indicate, where possible, the superior approach on the residual norm criterion.

1.6 Summary

In this chapter we have discussed the literature of Lanczos-type algorithms for solving systems of linear equations, the theory of FOPs on which they are based and how to derive them. An example of a Lanczos-type algorithm derived in this way, namely Orthores (or A_4), has been given. The aims of the project are also set.

Chapter 2

Restarting Lanczos-type Algorithms from Specific Points

In this chapter, we focus on restarting from three specific points which are attractive as starting points. These are the last iterate preceding breakdown, the iterate with the lowest residual norm, and the iterate whose entries are the median values of all entries of the iterates generated so far. We will also look at the theoretical aspect of restarting by following the results of [47]. Some numerical results will be provided from solving large scale problems, ranging from 1000 to 70,000 dimensions.

2.1 Restarting Lanczos-type Algorithms

2.1.1 The Restarting Strategy

As an illustration of the restarting approach, [29, 30], to avoid breakdown in a Lanczos-type algorithm, we consider the specific case of Orthodir algorithm, [6]. Note, Orthodir algorithm is based on combination of formulae A_8 and B_6. Formula A_8 is obtained by calculating recursively the orthogonal polynomial $\{P_k\}$ from $P_{k-1}^{(1)}$ and P_{k-1}, whereas formula B_6 is obtained by calculating recursively the orthogonal polynomial $P_k^{(1)}$ from $P_{k-1}^{(1)}$ and $P_{k-2}^{(1)}$.

We run this algorithm on SLE's $A\mathbf{x} = \mathbf{b}$, where matrix A is given in (2.29), [3], and $\mathbf{x} = (1, 2, \ldots, n)^T$ is the solution of the system. For $n = 200$, $\mathbf{x}_0 = \mathbf{0}$, and $\mathbf{y} = (1, \ldots, 1)^T$, the problematic coefficient, i.e. the coefficient causing breakdown, is $A_{k+1} = \frac{-\langle \mathbf{y}_k, \mathbf{r}_k \rangle}{\langle \mathbf{y}_k, A\mathbf{z}_k \rangle} = NaN$, for $k = 133$. This coefficient is used in relations :

$$\mathbf{x}_{k+1} = \mathbf{x}_k - A_{k+1}\mathbf{z}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k + A_{k+1}A\mathbf{z}_k.$$

Putting $\mathbf{x}_k$ ($k = 133$) above as an initial guess of Orthodir algorithm, does not seem to create a breakdown immediately. In fact, it is reasonable to assume it would not, since the process starts working in different Krylov space base, [29].

2.1.2 The Quality of the Restarting Point

Here, we consider different implementations of restarting from three different points. RLLastIt restarts from the last iterate; RLMinRes restarts from the iterate with the minimum residual norm; RLMedVal restarts from a point made up of the median values of all the entries of previous iterates in a sequence generated by the Lanczos-type algorithm under consideration. Restarting in all cases is of the pre-emptive type as described in [30]. The aim is to see whether different starting points lead to different performances.

2.1.3 Restarting from the Last Iterate

Suppose we generate the approximate solutions of a SLE using a Lanczos-type algorithm. Assume that we stop the algorithm after k iterations. Note that a high k may mean the algorithm may breakdown. After a pre-set number of iterations, we obtain the iterate $\mathbf{x}_k$, which is the last iterate, with residual norm $\|\mathbf{r}_k\|$. Since we use this iterate as a starting point, we initialize the algorithm with

$$\mathbf{x}_0 = \mathbf{x}_k, \tag{2.1}$$

$$\mathbf{y} = \mathbf{b} - A\mathbf{x}_0. \tag{2.2}$$

Restarting Lanczos-type algorithms from the last iterate, or the RLLastIt algorithm, is described in Algorithm 4. Here, firstly, we run a Lanczos-type algorithm over a certain number of iterations, say k. We then check the residual norm of the last iterate, $norm_{last}$; if it is less than the tolerance, ϵ, then we have a good approximate solution. Otherwise, we initialize the Lanczos-type algorithm with (2.1) and (2.2).

Algorithm 4 The RLLastIt algorithm

```
1: Fix the number of iterations to, say $k$, and the tolerance, $\epsilon$, to $1E-13$.
2: Run a Lanczos-type algorithm for $k$ iterations and get the approximate solution
   $sol_{last} = \mathbf{x}_k$, as well as the residual norm $norm_{last} = \|\mathbf{r}_k\|$.
3: **while** $norm_{last} \geq \epsilon$ **do**
4:     Initialize the algorithm with
           $\mathbf{x} = sol_{last}$,
           $\mathbf{y} = \mathbf{b} - A\mathbf{x}$.
5:     Run the Lanczos-type algorithm for $k$ iterations.
6: **end while**
7: Take $sol_{last}$ as the approximate solution.
8: Stop.
```

2.1.4 Restarting from the Iterate with Minimum Residual Norm

Because of the potential accumulation of errors, the last iterate after k iteration is not necessarily the one with smallest residual norm. It is therefore reasonable to consider restarting from the point among the k iterates with the lowest residual norm. Figure 2.1 illustrates the residual norm behaviour of the iterates after running Orthodir algorithm, [6], when solving problems in 300 dimensions using 150 iterations. This figure describes the situation mentioned earlier that the last iterate is not always the better one. In this case, the iterate with the lowest residual norm is the 29^{th}.

The procedures of finding the iterate with the minimum residual norm (LMinRes) can be seen in Algorithm 5. We collect all of the iterates generated in k iterations and their residual norms.

$$data_{sol} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}, \tag{2.3}$$

$$data_{resnorm} = \{||\mathbf{r}_1||, ||\mathbf{r}_2||, \ldots, ||\mathbf{r}_k||\}. \tag{2.4}$$

We then find the minimum value of (2.4) as

$$norm_{min} = min(data_{resnorm}), \tag{2.5}$$

and its index m. Our new solution is the iterate in (2.3) with index m.

Considering the iterate with the lowest residual norm as a restarting point leads to a better solution. The idea is that the better the restarting point, the better the solution after restarting. As in RLLastIt, restarting from the iterate with the minimum residual norm (RLMinRes) is initialized as follows

$$\mathbf{x}_0 = \mathbf{x}_m, \tag{2.6}$$

and $\mathbf{y}$ as in (2.2). If we repeat this step, we will find a good approximate solution as explained in the previous subsection. These procedures are presented in Algorithm 6.

Algorithm 5 The LMinRes algorithm

```
1: Initialize a Lanczos-type algorithm with $\mathbf{x}_0$ and $\mathbf{y}$. Run it for $k$ iterations.
2: Collect all $k$ iterates as in (2.3).
3: Collect all of the residual norms as in (2.4).
4: Compute the minimum value as in (2.5) and specify the index of the minimum
   value as $m$.
5: Obtain the approximate solution as well as the residual norm as follows
       $sol_{min} = data_{sol}(m)$        (2.7)
       $norm_{min} = \|\mathbf{r}_m\|$.    (2.8)
6: Take $sol_{min}$ as the approximate solution.
7: Stop.
```

Algorithm 6 The RLMinRes algorithm

```
1: Fix the number of iterations to, say $k$, and the tolerance, $\epsilon$, to $1E-13$.
2: Run the LMinRes algorithm for $k$ iterations and obtain $sol_{min}$ and $norm_{min}$.
3: while $norm_{min} \geq \epsilon$ do
4:     Initialize the algorithm with
           $\mathbf{x} = sol_{min}$,
           $\mathbf{y} = \mathbf{b} - A\mathbf{x}$.
5:     Run the LMinRes algorithm for $k$ iterations.
6: end while
7: Take $sol_{min}$ as the approximate solution.
8: Stop.
```

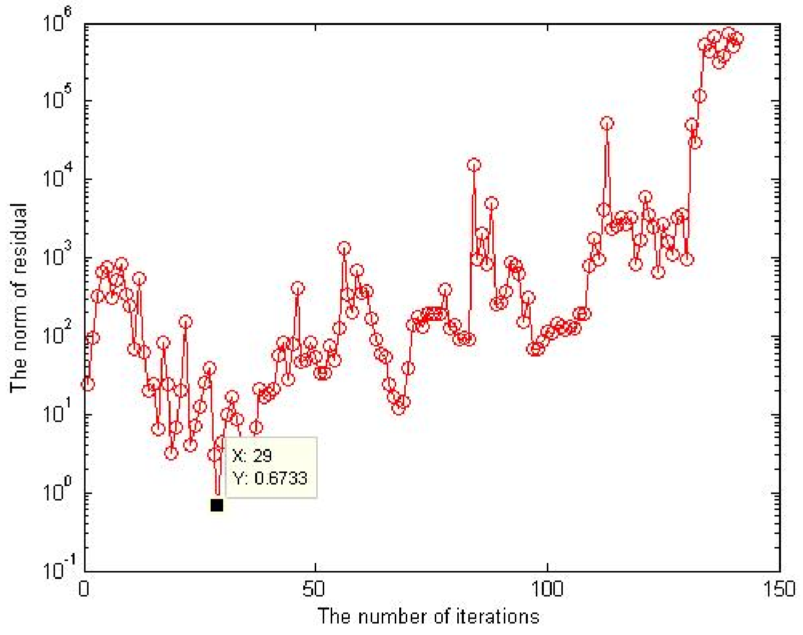


Figure 2.1: *A typical behaviour of the residual norms of SLE's in dim = 300.*

2.1.5 Restarting from the Vector of Median Value Entries

Our investigation of the existence of a good iterate with a small residual norm generated by Lanczos-type algorithms leads to the idea of using a statistical description to find it. Therefore, we calculated the mean, the median, and the mode of all the iterates. Let us have a look at this in detail.

Consider again relations (2.3) and (2.4). For each $\mathbf{x}_i$, $i = 1, 2, \ldots, n$, we set

$$\begin{aligned} w_1 &= \left\{\mathbf{x}_1{}^{(1)}, \mathbf{x}_2{}^{(1)}, \cdots, \mathbf{x}_k{}^{(1)}\right\} \\ w_2 &= \left\{\mathbf{x}_1{}^{(2)}, \mathbf{x}_2{}^{(2)}, \cdots, \mathbf{x}_k{}^{(2)}\right\} \\ &\vdots \\ w_n &= \left\{\mathbf{x}_1{}^{(n)}, \mathbf{x}_2{}^{(n)}, \cdots, \mathbf{x}_k{}^{(n)}\right\} \end{aligned} \tag{2.9}$$

where w_i is arranged such that it consists of the i^{th} entry of each vector $\mathbf{x}_i$, for

$i = 1, 2, \dots, n$. For instance, w_1 is set of all of the first entries of the iterates, w_2 is set of all of the second entries of the iterates, etc. If we calculate mean, median, and mode values of each w_i, and we use them as the first entry, the second entry, etc. of a vector, it leads to a new vector solution. In other words, the new vector contains the values of either mean, median, or mode of all the first entries, the second entries, etc. of all the iterates. Our investigation of these entries showed that the best results are obtained with the median values. The approximate solution based on the median values of entries is as follows.

$$\mathbf{x}_{med} = \begin{bmatrix} {x_1}^{med} \\ {x_2}^{med} \\ \vdots \\ {x_n}^{med} \end{bmatrix}, \tag{2.10}$$

and ${x_i}^{med} = med\,\{w_i\}$, for $i = 1, 2, \dots, n$.

The procedure described above for creating an approximate solution from the median values of entries of iterates generated by the Lanczos-type algorithm used is presented in algorithmic form as Algorithm 7 or LMedVal. Firstly, we run a Lanczos-type algorithm with a fixed number of iterations, say k. Secondly, we collect all of k iterates and save them in $data_{sol}$. Next, for $i = 1, 2, \dots, n$, we set w_i as in (2.9), we then compute the median value of each w_i. Hence we set all of the median values to form a vector, called sol_{med}, which is our new solution of the system. We compute the residual vector, res_{med}, as well as the residual norm, $norm_{med}$, to check the accuracy of the approximate solution.

The approximate solution resulted in LMedVal, i.e. sol_{med}, is then used as a restarting point in RLMedVal. It is described in Algorithm 8. Its initialization is as follows.

$$\mathbf{x}_0 = sol_{med}, \tag{2.11}$$

and vector $\mathbf{y}$ as in (2.2), where $\mathbf{x}_{med}$ as in (2.10). The RLMedVal algorithm can be seen in Algorithm 8.

Algorithm 7 The LMedVal algorithm

```
1: Initialize a Lanczos-type algorithm with guess solutions $\mathbf{x}_0$ and $\mathbf{y}$. Run it for
   $k$ iterations.
2: Collect all $k$ iterates as in (2.3).
3: for $i = 1, 2, \ldots, n$ do
4:     Set $w_i$ as in (2.9)
5:     Compute the median value of the entries of each $w_i$ as follows
           $med(i) = median(w_i).$
6: end for
7: Compute the approximate solution as follows
           $sol_{med} = med(1:n)^T.$
8: Compute the residual norm of the solution as follows
           $res_{med} = \mathbf{b} - Asol_{med},$
           $norm_{med} = norm(res_{med}).$
9: Take $sol_{med}$ as the approximate solution.
10: Stop.
```

2.1.6 The Theoretical Point of View

Suppose we solve the SLE's using RLLastIt algorithm. We denote $\mathbf{x}_k{}^{(i)}$, for $i = 1, 2, \ldots, s$, and for any integer $s > 0$, as the k^{th} iterate of the i^{th} run of a Lanczos-type algorithm. We assume that the residual norm of the first cycle is greater than

Algorithm 8 The RLMedVal algorithm

```
1: Fix the number of iterations to $k$ and the tolerance $\epsilon$ to $1E-13$.
2: Run LMedVal for $k$ iterations and obtain $sol_{med}$ as well as $norm_{med}$.
3: while $norm_{med} \geq \epsilon$ do
4:     Initialize the algorithm with
           $\mathbf{x} = sol_{med},$
           $\mathbf{y} = \mathbf{b} - A\mathbf{x}.$
5:     Run LMedVal for $k$ iterations.
6: end while
7: Take $sol_{med}$ as the approximate solution.
8: Stop.
```

the convergence tolerance, i.e.

$$\left\|\mathbf{r}_k{}^{(1)}\right\| > \epsilon_1, \tag{2.12}$$

for any $\epsilon_1 > 0$. Considering the results in Subsection 2.1.3 and by the definition of the Krylov subspace given in Chapter 1, particularly in Subsection **??**, we now have two conditions for the second cycle of restarting the Lanczos-type algorithm

$$\mathbf{x}_k{}^{(2)} - \mathbf{x}_0 \in K_k(A, \mathbf{r}_0) \quad \text{and} \tag{2.13}$$

$$\mathbf{r}_k{}^{(2)} = \mathbf{b} - A\mathbf{x}_k{}^{(2)} \perp K_k(A^T, y), \tag{2.14}$$

where $\mathbf{r}_0 = \mathbf{r}_k{}^{(1)}$. Substituting (2.1) into (2.13), we get :

$$\mathbf{x}_k{}^{(2)} - \mathbf{x}_k{}^{(1)} \in K_k(A, \mathbf{r}_k{}^{(1)}). \tag{2.15}$$

Following the results in (1.6), we have :

$$\begin{aligned}\mathbf{r}_k{}^{(2)} &= \mathbf{r}_k{}^{(1)} + \alpha_1 A\mathbf{r}_k{}^{(1)} + \alpha_2 A^2\mathbf{r}_k{}^{(1)} + \cdots + \alpha_k A^k\mathbf{r}_k{}^{(1)} \\ &= P_k(A)\mathbf{r}_k{}^{(1)},\end{aligned} \tag{2.16}$$

where $P_k(A) = 1 + \alpha_1 A + \cdots + \alpha_k A^k$. Calculating the norm of $\mathbf{r}_k{}^{(2)}$ in (2.16) yields

$$\begin{aligned}\left\|\mathbf{r}_k{}^{(2)}\right\| &= \left\|P_k(A)\mathbf{r}_k{}^{(1)}\right\| \\ &\leq \|P_k(A)\| \left\|\mathbf{r}_k{}^{(1)}\right\|. \end{aligned} \tag{2.17}$$

If this residual norm is still bigger than the tolerance, we apply restarting again. In this case, we will have the third cycle giving

$$\mathbf{r}_k{}^{(3)} = P_k(A)\mathbf{r}_k{}^{(2)}, \tag{2.18}$$

and calculating the norm of $\mathbf{r}_k{}^{(3)}$ as

$$\begin{aligned}\left\|\mathbf{r}_k{}^{(3)}\right\| &= \left\|P_k(A)\mathbf{r}_k{}^{(2)}\right\| \\ &\leq \|P_k(A)\| \left\|\mathbf{r}_k{}^{(2)}\right\| \\ &\leq \|P_k(A)\|^2 \left\|\mathbf{r}_k{}^{(1)}\right\| \quad \text{since we have (2.17).}\end{aligned} \tag{2.19}$$

If this procedure is continued, then for the s^{th} cycle, we have :

$$\begin{aligned}\left\|\mathbf{r}_k{}^{(s)}\right\| &= \left\|P_k(A)\mathbf{r}_k{}^{(s-1)}\right\| \\ &\leq \|P_k(A)\|^s \left\|\mathbf{r}_k{}^{(1)}\right\|. \end{aligned} \tag{2.20}$$

From (2.17), (2.19), and (2.20), and since $\|P_k(A)\| > 0$, we conclude that

$$\left\|\mathbf{r}_k{}^{(s)}\right\| \leq \left\|\mathbf{r}_k{}^{(s-1)}\right\| \leq \ldots \leq \left\|\mathbf{r}_k{}^{(1)}\right\|, \tag{2.21}$$

where $s \geq 1$ is the number of cycles. This leads to the following result.

Theorem 2.1.1. *Suppose we solve a SLE using a Lanczos-type algorithm. Let $\mathbf{x}_k{}^{(1)}$ be the last iterate generated by this algorithm after k iterations. Given a tolerance $\epsilon > 0$, we assume that the associated residual norm $\left\|\mathbf{r}_k{}^{(1)}\right\| > \epsilon$. Restarting the algorithm with $\mathbf{x}_k{}^{(1)}$ allows it to generate an iterate with a better residual norm than those of the previous iterates.*

In practice, breakdown may still occur between cycles which will cause the algorithm to stop before reaching a good approximate solution. We cannot guarantee that a breakdown will not occur in the first cycle, or any cycle for that matter.

2.2 Numerical Results and Discussion

We solved different size problems ranging from dimension 1000 to 70000. The test problems are carried out in MatLab 2012b using two different systems, Windows with processor RAM 6GB for solving the medium scale problems, and Unix0 with 256 GB for solving the large scale problems. We implemented RLLastIt, RLMinRes, and RLMedVal algorithms as well as other supporting procedures such as LMinRes and LMedVal. The experiments have been carried out via Algorithm 9 which calls RLLastIt, RLMinRes, RLMedVal.

Algorithm 9 Restarting Lanczos-type algorithms from three different points

```
1: Fix the number of iterations to $k$ and the tolerance to $\epsilon$.
2: Run RLLastIt, RLMinRes, RLMedVal for $k$ iterations.
3: while $norm_{last} \geq \epsilon$ do
4:     Run RLLastIt for $k$ iterations.
5:     Run RLMinRes for $k$ iterations.
6:     Run RLMedVal for $k$ iterations.
7: end while
8: Take $sol_{last}$, $sol_{min}$, and $sol_{med}$ as the approximate solution of each algorithm, respectively.
9: Stop.
```

2.2.1 Test Problem

Before describing the test problem considered in [3], we briefly describe discretization process.

Discretization is the process of transferring continuous differential equations into discrete or coupled equations. There are basically two main ways to discretize differential equations, namely finite difference methods (FDMs) and finite element methods (FEMs), [58, 79]. In this study, we focus on FDMs. FDMs use finite differences to approximate differential operators. These methods are known as the simplest and extensively used for solving differential equations. Typically, FDMs are used on a regular mesh and yield matrices with a regular structure that are easy to store in the computer memory, [79]. The model problem that was widely used for testing algorithms is Poisson equation on the open unit square $\Omega = (0, 1) \times (0, 1)$, [58].

$$\begin{aligned} -\Delta u = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f \quad \text{in}\, \Omega, \\ u|_{\partial\Omega} = 0, \end{aligned} \tag{2.22}$$

where $\partial\Omega$ is the boundary of Ω and f is given in an appropriate functional space. To get an approximation on this problem, we cover Ω with a regular mesh Ω_h having m points in each direction. This gives a step size $h = \frac{1}{m+1}$.

Let $u_{i,j}$ be the approximation of u at $\left(x_i, y_j\right)$, where $\left(x_i, y_j\right) \in \Omega$. Then we approximate $\frac{\partial u}{\partial x}$ at $\left(i + \frac{1}{2}h, jh\right)$ by the finite difference

$$\left(\frac{\partial u}{\partial x}\right)_{i+\frac{1}{2},j} \approx \frac{u_{i+1,j} - u_{i,j}}{h}. \tag{2.23}$$

Hence,

$$\begin{aligned} \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} &\approx \frac{1}{h}\left(\left(\frac{\partial u}{\partial x}\right)_{1+\frac{1}{2},j} - \left(\frac{\partial u}{\partial x}\right)_{1-\frac{1}{2},j}\right), \\ &= \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}. \end{aligned} \tag{2.24}$$

Doing this for both directions, we get an approximation of minus the Laplacian Δ, that we denote (after multiplication by h^2) by $-\Delta_5$.

$$\begin{aligned}(-\Delta_5)_{i,j} &= -u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i,j-1} - u_{i,j+1} \\ &= h^2 f(x_i, y_j), \quad i = 1, \ldots, m,\ j = 1, \ldots, m\ . \end{aligned} \tag{2.25}$$

The solution is already known for points on $\partial\Omega$, that is for $i = 0$ or $m + 1$, $j = 0$ or $m + 1$, so we have m^2 equations with m^2 unknowns and a linear systems to solve. If we numbered the points from left to right and from bottom to top, and renamed the unknowns u into $\mathbf{x}$, the systems can be written as

$$A\mathbf{x} = \mathbf{b}, \tag{2.26}$$

where

$$\begin{aligned}\mathbf{x} &= \{u_{1,1}, u_{1,2}, \ldots, u_{m,m}\}^T, \\ \mathbf{b} &= h^2\{f_{1,1}, f_{1,2}, \ldots, f_{m,m}\}^T, \quad \text{where } f_{i,j} \text{ being the value of } f \text{ at point } (i, j), \\ A &= \begin{pmatrix} T & -I & \cdots & \cdots & 0 \\ -I & T & -I & & \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & -I & T & -I \\ 0 & \cdots & \cdots & -I & T \end{pmatrix}, \end{aligned} \tag{2.27}$$

where I is the $m \times m$ identity matrix and T is a tridiagonal matrix of order m,

$$T = \begin{pmatrix} 4 & -1 & \cdots & \cdots & 0 \\ -1 & 4 & -1 & & \ldots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & -1 & 4 & -1 \\ 0 & \cdots & \cdots & -1 & 4 \end{pmatrix}. \tag{2.28}$$

Our test problems used in this study arise in the 5-point discretisation of the operator $-\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2} + \gamma \frac{\partial}{\partial x}$ on a rectangular region, [3]. This yields the systems (2.26) with

$$A = \begin{pmatrix} B & -I & \cdots & \cdots & 0 \\ -I & B & -I & & \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & -I & B & -I \\ 0 & \cdots & \cdots & -I & B \end{pmatrix}, \tag{2.29}$$

and

$$B = \begin{pmatrix} 4 & \alpha & \cdots & \cdots & 0 \\ \beta & 4 & \alpha & & \ldots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \beta & 4 & \alpha \\ 0 & \cdots & \cdots & \beta & 4 \end{pmatrix} \tag{2.30}$$

with $\alpha = -1 + \delta$, $\beta = -1 - \delta$, and δ in our work here takes values 0.0, 0.2, 0.5, 0.8, 5.0, and 8.0. The parameter of δ plays an important role on the singularity of matrix A. For instance, when $\delta = 0$, A is symmetric matrix. The condition number of matrix A for this particular value of delta is large enough, i.e. 119.9999. Any systems which are built by this matrix tend to be ill-conditioned, [17, 43]. When the δ large enough, the matrix A can become well-conditioned.

In addition, the right hand side $\mathbf{b}$ is taken to be $\mathbf{b} = A\mathbf{x}$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, is the solution of the systems, and x_i, $i = 1, 2, \ldots, n$ is a random value between 0 and 1. We use Orthodir algorithm, or Algorithm 2, in this experiment as a representative of Lanczos-type algorithms. We also use 100 iterations to make up a cycle, which means that the intermediate solution is found after 100 iterations.

2.2.2 Test Problem I : $\delta = 0.0$

According to the information available in Table 2.1, overall, RLMinRes has the best performance, compared with RLLastIt and RLMedVal. It consistently convergences in the smallest number of cycles. For instance, when solving the SLEs dimensions 1000, RLMinRes needed 11 cycles with total time of 1.1293 seconds, whereas RLLastIt and RLMedVal needed respectively 16 and 12 cycles with total times are about 2.0159 and 1.9818 seconds. A significant difference of using the time is made when solving a higher dimensions. For instance, when solving problems dimensions 9000, 10000, and 20000, RLMinRes needed respectively about 64, 80, and 318 seconds, RLMedVal took about 70, 94, and 1235 seconds, respectively, whereas RLLastIt needed respectively about 161, 166, and 1285 seconds.

In term of robustness, there is no doubt that RLMinRes is more robust than RLLastIt and RLMedVal, since it never experienced breakdown. The breakdown itself in this experiment is shown by ”NaN” in the residual norm column. Experimental results show that, in general, RLLastIt experienced breakdown when solving medium scale problems, such as dimensions 2000, 6000, 7000, and only one problem for a large dimension, namely 30000. In contrast, RLMedVal faced breakdown when solving large scale problems, such as in dimensions 30000, 40000, 50000, and 60000.

The behaviour of the restarting from three different points is represented in Figures 2.2 and 2.3. Breakdown in this algorithm is illustrated by the curves which either do not appear, or do not reach a small residual norm values. For instance, when solving a problem dimensions 2000, the red curve, which represents RLLastIt, appears on the top of the curve, and it runs for 3 cycles only. Similarly, for a problem dimensions 3000, the red curve never hit a small residual norm. It broke down before convergence. Furthermore, when solving problems dimensions 6000 and 7000, the red curve appears with one dot only which also indicates that breakdown occurred.

Similarly, the green curve, which represents RLMedVal, when solving a problem dimension 8000, was not visible. In other problems, the green curve seems to be long, which means that it runs over many cycles.

Table 2.1: *Numerical results of RLLastIt, RLMinRes, and RLMedVal on problems with* $\delta = 0.0$.

Dim	RLLastIt			RLMinRes			RLMedVal		
n	$\|\|r_k\|\|$	$T(s)$	cycles*	$\|\|r_{min}\|\|$	$T(s)$	cycles*	$\|\|r_{med}\|\|$	$T(s)$	cycles*
1000	$5.7290E-14$	2.0159	16	$7.4630E-14$	1.1293	11	$7.7686E-14$	1.9818	12
2000	NaN	NA	NA	$7.9179E-14$	3.5393	10	$7.2049E-14$	5.1083	8
3000	1608	NA	25	$7.1515E-14$	8.0831	10	$9.2942E-14$	10.2765	12
4000	$6.6150E-14$	34.3719	24	$7.1737E-14$	12.2651	9	$9.7700E-14$	17.8791	12
5000	$5.3701E-14$	79.1186	36	$6.8902E-14$	18.5584	9	$7.9847E-14$	24.7685	12
6000	NaN	NA	NA	$7.1850E-14$	25.8040	9	$7.4920E-14$	101.59	9
7000	NaN	NA	NA	$7.9874E-14$	37.3090	12	$7.4949E-14$	50.6792	12
8000	$9.0251E-14$	67.9472	14	$6.7684E-14$	45.5536	9	NaN	NA	NA
9000	$6.0401E-14$	161.2933	24	$7.0927E-14$	63.9086	10	$8.5236E-14$	70.1496	11
10000	$6.6456E-14$	165.7143	20	$8.1881E-14$	79.8814	10	$8.5545E-14$	93.5818	11
20000	$7.2688E-14$	1285.500	40	$7.4526E-14$	318.1595	10	$9.9168E-14$	1234.488	12
30000	NaN	NA	NA	$7.9381E-14$	22848.70	9	NaN	NA	NA
40000	$9.5703E-14$	19798.6225	47	$8.0789E-14$	9589.513	7	NaN	NA	NA
50000	202.3106	17797.106	10	$8.5386E-14$	15375.207	10	NaN	24780.773	4
60000	0.0014	26531.409	10	$7.5384E-14$	21178.311	10	NaN	NA	NA
70000	$3.8427E-08$	50351.809	15	$9.2520E-14$	30739.034	10	$8.4645E-14$	64255.166	15

NA = Non Applicable. The computation time and cycles are not calculated if breakdown occurs which is indicated by "NaN" in the residual norm column.

2.2.3 Test Problem II : $\delta = 0.2$

The condition number of matrix A of the problem instances for this particular value of delta is 98.6081 which is large enough too, which means that the matrix is categorized as ill-conditioned. Similar to the previous case, RLMinRes is still the best compared to RLLast and RLMedVal. It can be seen in Table 2.2, for instance, that for SLEs dimensions 1000, RLMinRes needed 7 cycles with total time of about 0.8079 seconds, whereas RLMedVal needed 8 cycles with total time of about 1.9484 seconds, and RLLastIt needed 16 cycles with total time of about 2.0260 seconds.

Furthermore, RLMinRes needed to restart 7 and 8 times when solving SLEs dimensions 9000 and 10000, with total times respectively 48.2827 and 61.9360 seconds. RLMedVal solved the same problems with 8 cycles with total times respectively 106.9637 and 118.8027 seconds. The worst one is RLLastIt which cycled 21 and 25 times with total times of 152.2841 and 205.5778 seconds, respectively. In addition, for solving large scale problems, i.e. dimensions 30000 to 70000, RLMinRes was consistently stable and needed only few cycles with lower total computation time than the other two. It can be seen from the table that breakdown still occurs in RLLastIt and RLMedVal when solving systems dimensions 3000, 5000, 8000, 20000, and 40000.

The performance of RLLast, RLMinRes, and RLMedVal, can be seen in Figures 2.4 and 2.5. The explanation of some curves in this figure can refer to the previous case.

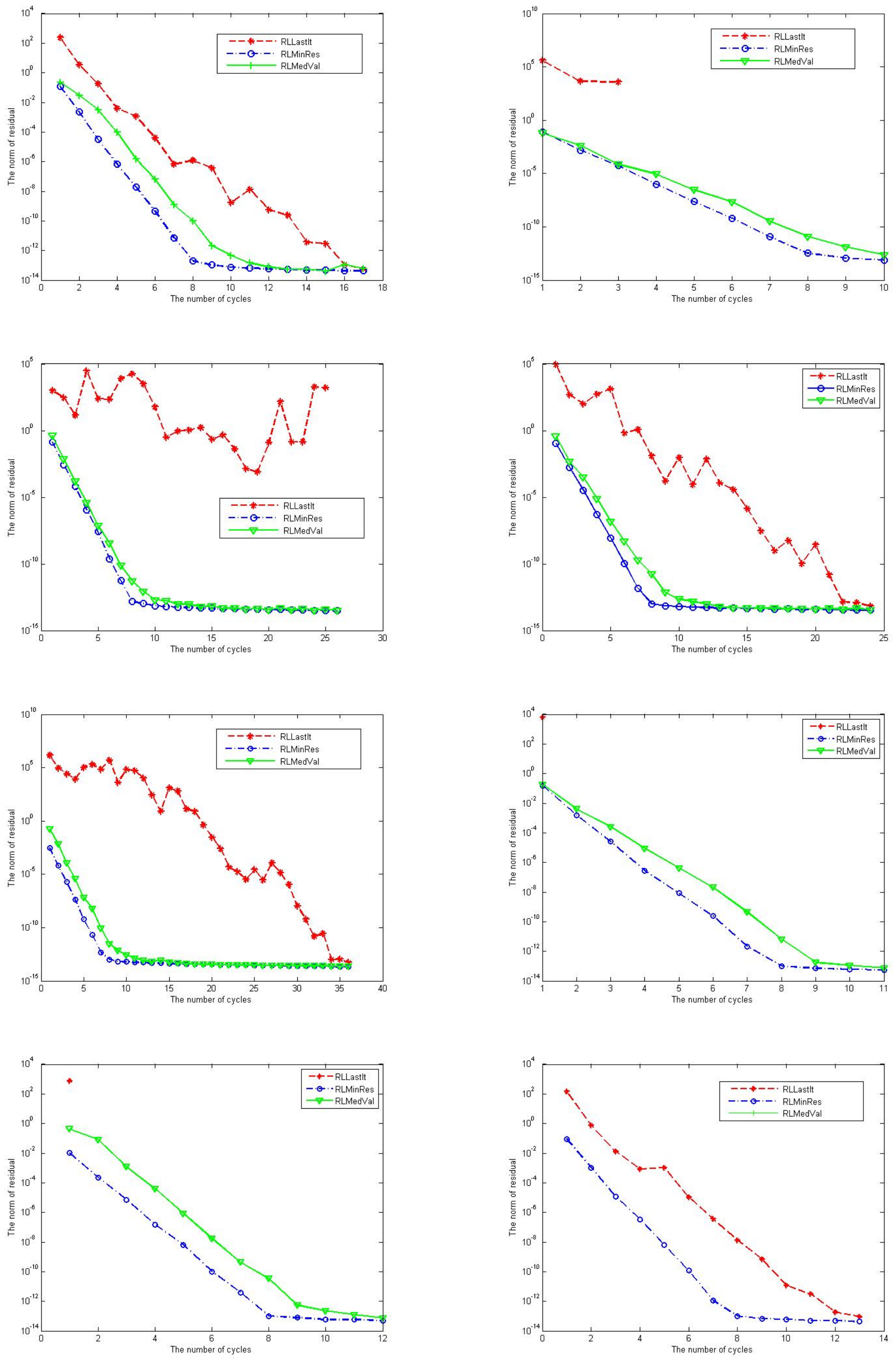


Figure 2.2: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 1000 to 8000, for* $\delta = 0.0$

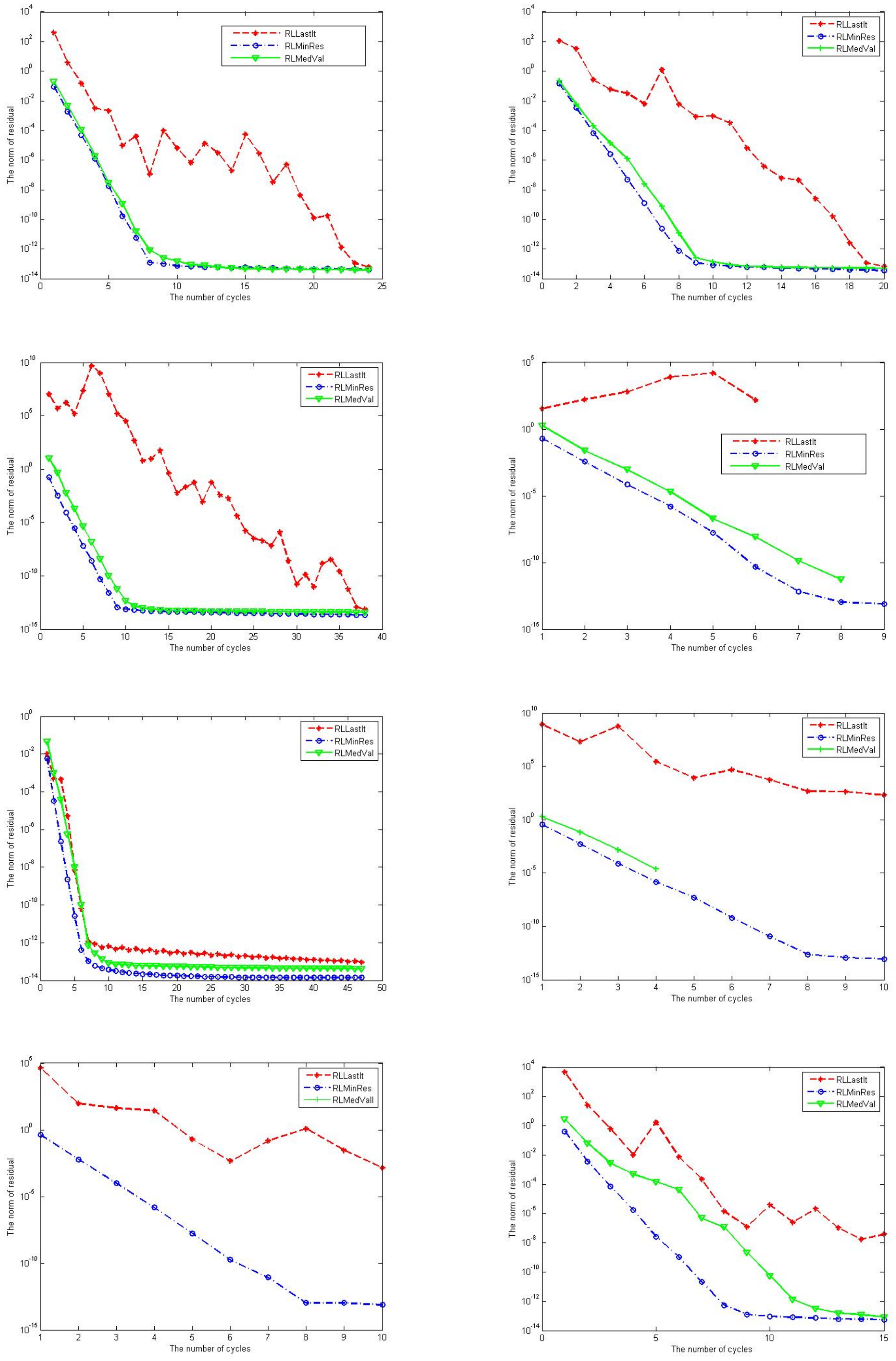


Figure 2.3: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 9000 to 70000, for* $\delta = 0.0$

Table 2.2: *Numerical results of RLLastIt, RLMinRes, and RLMedVal on problems with* $\delta = 0.2$.

Dim	RLLastIt			RLMinRes			RLMedVal		
n	$\lVert r_k \rVert$	$T(s)$	cycles*	$\lVert r_{min} \rVert$	$T(s)$	cycles*	$\lVert r_{med} \rVert$	$T(s)$	cycles*
1000	$6.5481E-14$	2.0260	16	$8.0389E-14$	0.8079	7	$7.7606E-14$	1.9484	8
2000	$3.3041E-14$	5.7515	14	$1.2429E-13$	7.6131	14	$7.9129E-14$	6.750	8
3000	NaN	NA	NA	$8.7363E-14$	6.9168	8	NaN	NA	NA
4000	$4.0211E-14$	23.8768	16	$7.9746E-14$	9.8956	7	NaN	NA	NA
5000	NaN	NA	NA	$5.1093E-14$	17.1567	9	NaN	NA	NA
6000	$3.3399E-14$	49.6966	15	$7.0909E-14$	21.6026	7	$6.4865E-14$	45.89899	9
7000	$5.4049E-14$	76.8363	17	$7.3018E-14$	34.2061	8	$8.6172E-14$	68.6571	10
8000	NaN	NA	NA	$8.0677E-14$	35.25096	7	$7.8672E-14$	71.3026	8
9000	$5.0507E-14$	152.2841	21	$6.6680E-14$	48.2827	7	$8.9150E-14$	106.9637	8
10000	$5.6855E-14$	205.5778	25	$6.3812E-14$	61.9360	8	$8.2954E-14$	118.8027	8
20000	NaN	NA	NA	$7.4526E-14$	318.1595	10	NaN	NA	NA
30000	$9.7800E-14$	$1.1635E+04$	50	$6.7992E-14$	$3.1132E+03$	9	$7.6804E-14$	$7.1155E+03$	10
40000	NaN	NA	NA	$5.9208E-14$	$6.1620E+03$	8	NaN	NA	NA
50000	$9.9120E-14$	$3.6563E+04$	75	$7.4136E-14$	$7.3843E+03$	8	$9.8225E-14$	$1.3690E+04$	10
60000	$9.9668E-14$	$2.5750E+04$	44	$8.3763E-14$	$9.6811E+03$	7	$8.5780E-14$	$2.3055E+04$	11
70000	$1.5299E+03$	$5.5568E+04$	10	$7.1895E-14$	$1.5601E+04$	8	$9.2065E-14$	$2.7346E+04$	10

NA = Non Applicable. The computation time and cycles are not calculated if breakdown occurs which is indicated by ″NaN″ in the residual norm column.

2.2.4 Test Problem III : $\delta = 0.5$

The condition number of matrix A for this particular value of delta is 62.2227. The SLEs with this matrix are categorized as ill-conditioned. Similar to the previous case, RLLastIt and RLMedVal still suffered from breakdown in some problems (such as dimensions 1000, 2000, 10000, and 20000), whereas RLMinRes remained robust. As can be seen in Table 2.3, RLMinRes needed fewer cycles, compared with the other two. When solving a problem dimensions 8000, for instance, it managed with total time of about 24.9942 seconds only, while RLLastIt and RLMedVal needed about 46.6633 and 32.3674 seconds, respectively.

Also, to solve the large scale problems which involve 40000, 50000, and 60000 variables, RLMinRes needed respectively 4959.626, 9216.815, and 9699.505 seconds after total cycles of 6, 5, and 6, respectively. Slightly different from this, RLMedVal converged after 7, 7, and 6 cycles, respectively, and with the total time of about 10546.413, 17177.462, and 15538.778 seconds, respectively. In contrast, these numbers rose significantly when RLLastIt was used; total times were about 13454.310, 13499.907, and 17342.312 seconds, respectively, and 18, 12, and 20 cycles needed, respectively.

The visualization of restarting Lanczos Orthodir from three different points can be seen in Figures 2.6 and 2.7. We only highlight here that for solving a problem dimensions 10000, the red and green curves have disappeared due to breakdown. Other trends are similar to these of the previous cases.

We can conclude here, based on the results above, that RLMinRes, again, has the best performance, RLMedVal be the second best performance, while RLLastIt

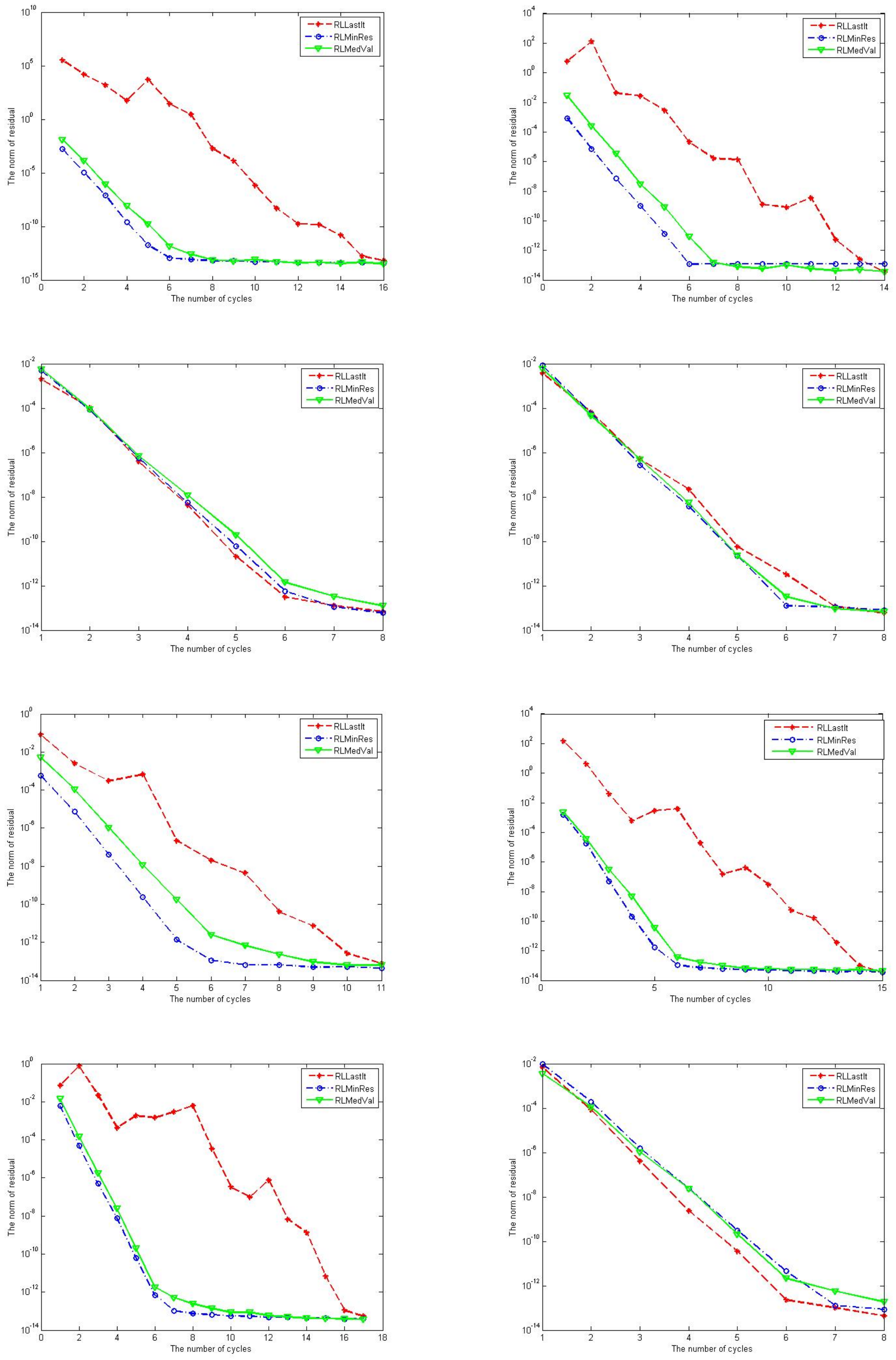


Figure 2.4: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 1000 to 8000, for* $\delta = 0.2$

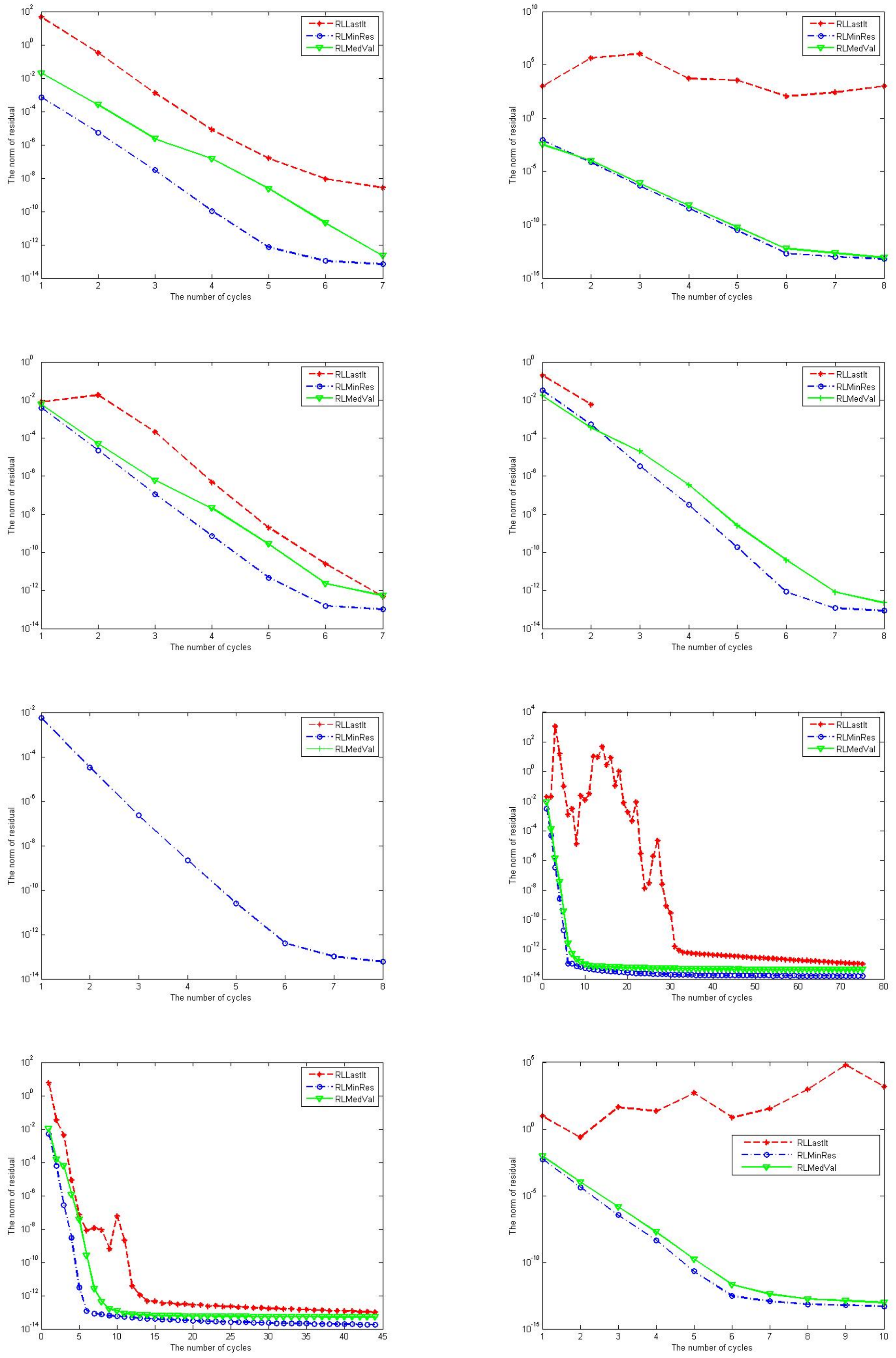


Figure 2.5: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 9000 to 70000, for* $\delta = 0.2$

has the worst performance. This is easy to explain; indeed of the three case, the last iterate is the worst in terms of residual norm. This is because the process is heading towards breakdown. That is why we stop it in the first place. RLMinRes and RLMedVal restart from better points, hence they superior performance.

Table 2.3: *Numerical results of RLLastIt, RLMinRes, and RLMedVal on problems with* $\delta = 0.5$.

Dim	RLLastIt			RLMinRes			RLMedVal		
n	$\|\|r_k\|\|$	$T(s)$	cycles*	$\|\|r_{min}\|\|$	$T(s)$	cycles*	$\|\|r_{med}\|\|$	$T(s)$	cycles*
1000	NaN	NA	NA	$6.7241E-14$	0.4985	6	$8.0317E-14$	0.9676	6
2000	$6.6028E-14$	3.0730	8	$7.1644E-14$	1.6560	5	NaN	NA	NA
3000	$2.0914E-14$	6.1322	8	$2.9349E-14$	3.7583	5	$8.2315E-14$	5.5742	7
4000	$4.0636E-14$	10.5073	8	$8.5704E-14$	6.8387	5	$7.3850E-14$	8.6492	6
5000	$5.3701E-14$	13.8288	7	$8.7432E-14$	9.4709	5	$6.6026E-14$	13.3348	7
6000	$4.0015E-14$	14.3912	8	$4.4088E-14$	13.5595	5	$7.9435E-14$	19.0317	7
7000	$1.9384E-14$	28.4045	7	$6.1367E-14$	19.8723	5	$9.3416E-14$	23.4865	7
8000	$7.6507E-14$	46.6633	9	$6.7684E-14$	24.9942	5	$7.3388E-14$	32.3674	6
9000	$2.0877E-14$	47.3239	10	$8.1464E-14$	32.4579	6	$7.5110E-14$	43.3506	6
10000	NaN	NA	NA	$7.6418E-14$	36.7934	5	NaN	NA	NA
20000	NaN	NA	NA	$5.5184E-14$	171.5886	6	$7.9658E-14$	164.7378	6
30000	$3.0472E-14$	456.2760	8	$8.3265E-14$	435.3542	6	$8.6487E-14$	438.9958	6
40000	$9.1562E-14$	13454.310	18	$7.6668E-14$	4959.626	6	$8.5023E-14$	10546.413	7
50000	$7.5725E-14$	13499.907	12	$8.0480E-14$	9216.815	5	$8.8901E-14$	17177.462	7
60000	$9.5620E-14$	17342.312	20	$7.8496E-14$	9699.505	6	$8.1470E-14$	15538.778	6
70000	$9.8198E-14$	33085.995	25	$6.9337E-14$	11972.368	7	$8.2264E-14$	23236.937	7

NA = Non Applicable. The computation time and cycles are not calculated if breakdown occurs which is indicated by "NaN" in the residual norm column.

2.2.5 Test Problem IV : $\delta = 0.8$

The condition number of matrix A for this particular value of delta is still a quite big which is 44.3210. The SLEs which have this matrix are categorized as ill-conditioned. As can be seen in Table 2.4, RLLastIt and RLMedVal experienced breakdown when solving 7000, 8000, and 30000 dimensional problems. As an illustration, we can see from Figures 2.8 and 2.9, that the red and green curves have disappeared or are invisible due to breakdown. In contrast, RLMinRes consistently converges without facing breakdown on all problems. In fact, it needed less number of cycles to converge. We also highlight here that on all problems, RLMinRes used the shortest time, followed by RLMedVal, and RLLastIt.

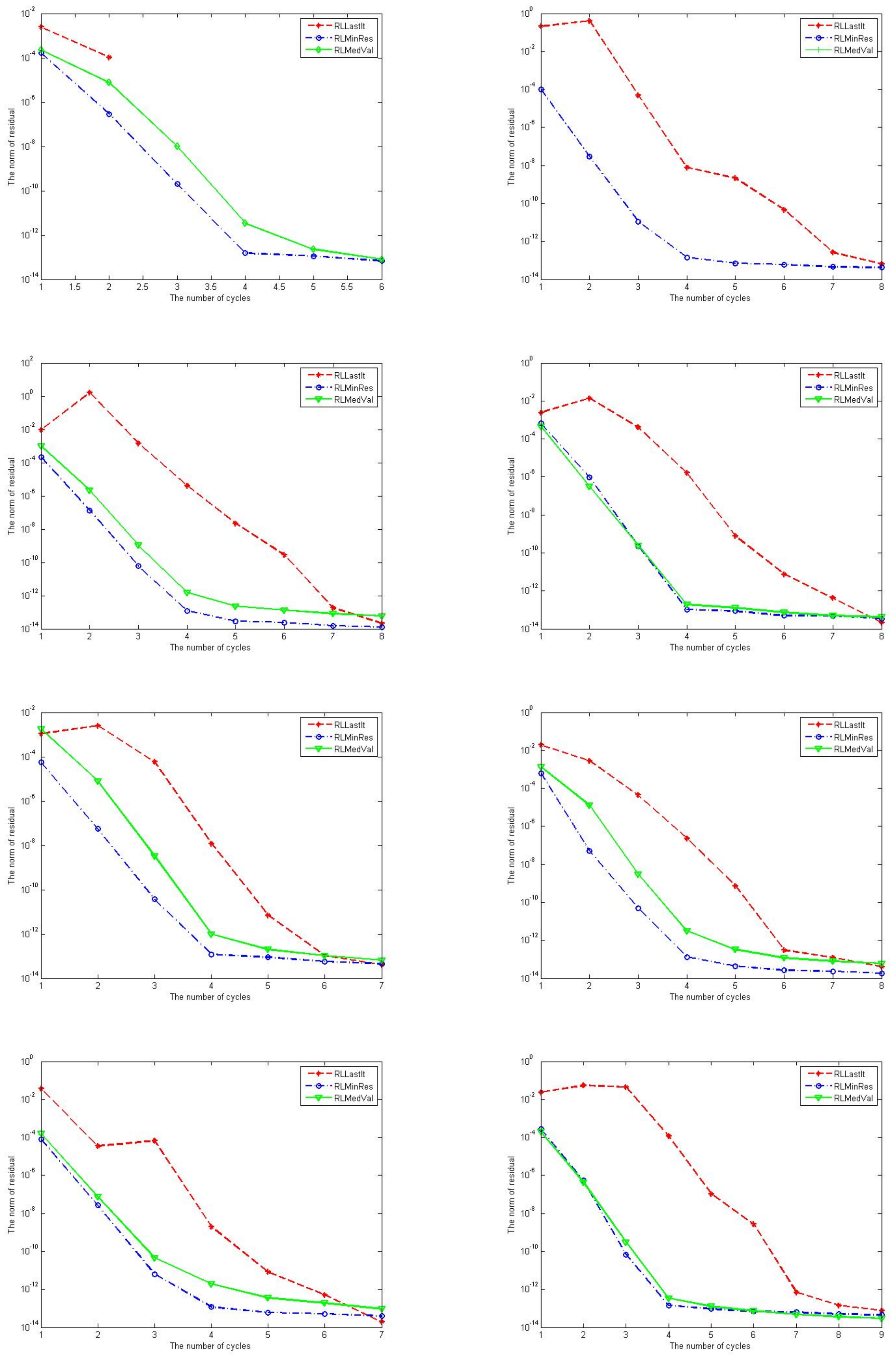


Figure 2.6: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 1000 to 8000, for* $\delta = 0.5$

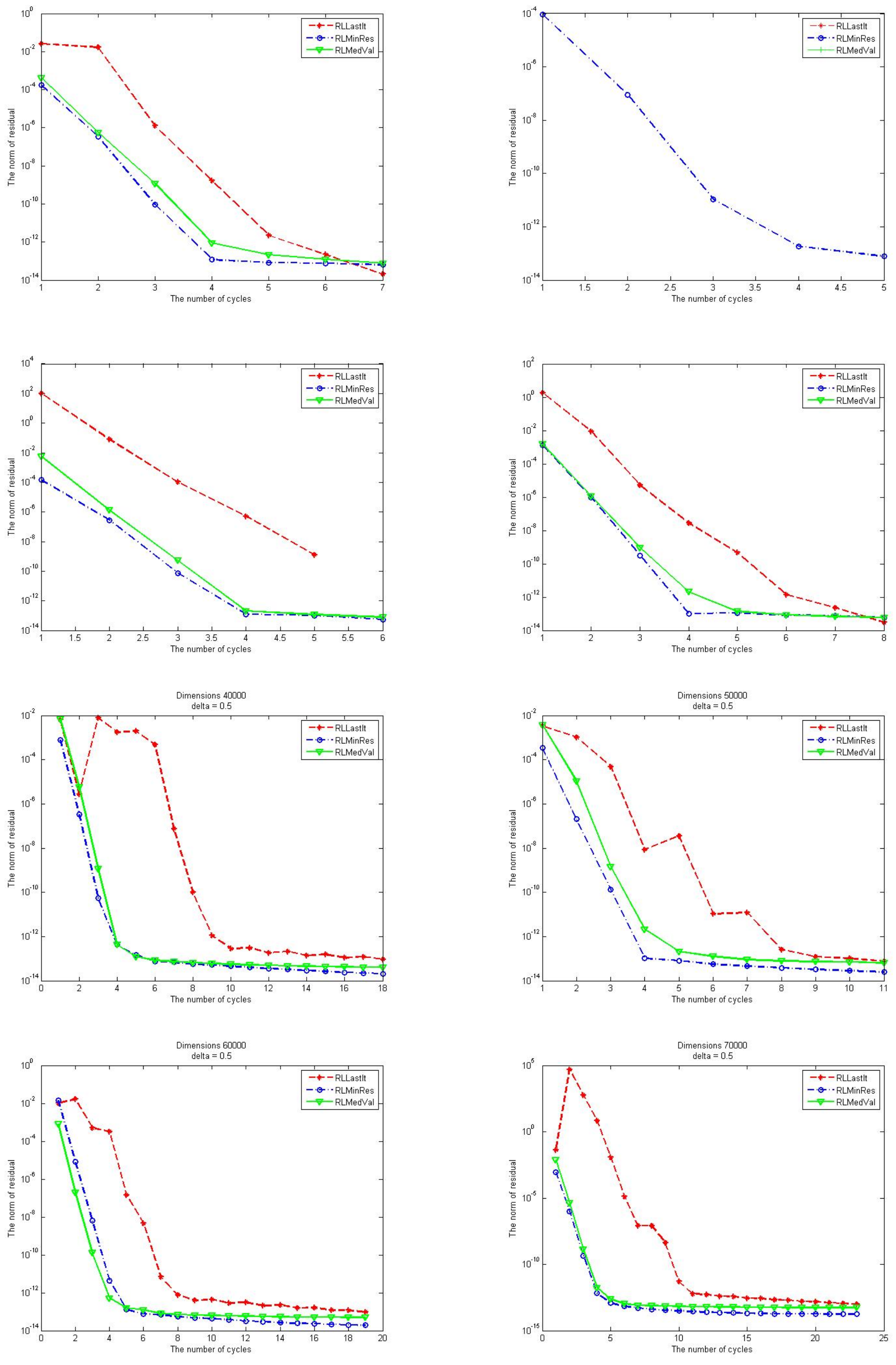


Figure 2.7: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 9000 to 70000, for* $\delta = 0.5$

Table 2.4: *Numerical results of RLLastIt, RLMinRes, and RLMedVal on problems with* $\delta = 0.8$.

Dim	RLLastIt			RLMinRes			RLMedVal		
n	$\|\|r_k\|\|$	$T(s)$	cycles*	$\|\|r_{min}\|\|$	$T(s)$	cycles*	$\|\|r_{med}\|\|$	$T(s)$	cycles*
1000	$4.6646E-14$	0.3.9578	4	$1.2362E-13$	0.4031	4	$1.9188E-13$	0.5585	4
2000	$9.0517E-14$	2.7572	6	$7.1644E-14$	1.6560	6	$6.2576E-14$	3.0358	6
3000	$5.2252E-14$	3.5538	5	$9.9039E-14$	2.7697	4	$9.1631E-14$	3.4874	5
4000	$5.1236E-14$	6.2464	5	$5.0994E-14$	4.8783	5	$8.4532E-14$	6.1081	6
5000	$2.6627E-14$	9.3331	5	$4.6503E-14$	6.4221	4	$7.5342E-14$	8.7590	6
6000	$4.0015E-14$	14.3912	5	$1.5693E-14$	13.8603	5	$8.5979E-14$	14.7563	5
7000	NaN	NA	NA	$6.4924E-14$	15.4388	5	NaN	NA	NA
8000	NaN	NA	NA	$4.8607E-14$	25.5804	5	$6.8697E-14$	24.2593	7
9000	$7.2386E-14$	32.2374	5	$6.9432E-14$	28.7406	4	$8.7765E-14$	27.7122	5
10000	$4.4203E-14$	40.7000	5	$5.8453E-14$	30.8759	5	$9.5466E-14$	35.9651	6
20000	$3.3223E-14$	159.2071	5	$6.9427E-14$	117.3181	4	$8.1042E-14$	133.1245	5
30000	NaN	NA	NA	$8.5189E-14$	408.2613	6	NaN	NA	NA
40000	$9.3120E-14$	3433.988	14	$9.9397E-14$	3840.327	5	$9.7811E-14$	6969.433	4
50000	$7.3387E-14$	3964.413	6	$8.0193E-14$	4340.405	4	$8.3000E-14$	7910.365	6
60000	$7.7921E-14$	12950.531	8	$9.4813E-14$	7505.311	4	$8.0392E-14$	7492.451	6
70000	$8.4024E-14$	19026.327	8	$7.8985E-14$	9986.536	4	$7.5368E-14$	13291.442	5

NA = Non Applicable. The computation time and cycles are not calculated if breakdown occurs which is indicated by "NaN" in the residual norm column.

2.2.6 Test Problem V : $\delta = 5$

The condition number of matrix A for this particular value of delta is 27.4932. The SLEs which have this matrix tend to be well-conditioned. The trend in this case is quite different from the previous cases where the 3 algorithms do not face breakdown in all problems. In fact, RLMinRes is not the best one in this case since it only achieved the approximate solution with a residual norm of $1E-12$. Interestingly, RLMedVal has the best performance with residual norm of $1E-13$, while the RLLastIt, was still the worst one.

As can be seen in Table 2.5, for solving dimension 10000, for instance, RLLastIt reached the approximate solution with the residual norm of $1.0468E-13$, needing 13 cycles with total time of about 81.2740 seconds; RLMinRes also needed 13 cycles with total time of 71.6121 seconds, and residual norm of $1.2264E-13$, while RLMedVal managed with 74.9466 seconds, 7 cycles, and a residual norm of $8.6507E-14$. For other problems, in dimensions 30000, 40000, 50000, 60000, and 70000, RLLastIt managed the solution with the residual norms of $1E-012$ only, with computation times below the others; RLMinREs found the approximate solution with residual norm of $1E-013$, whereas RLMedVal achieved the solution with residual norm of $1E-014$, with the longest computation time. To illustrate, the performance of this particular case can be seen in Figures 2.10 and 2.11.

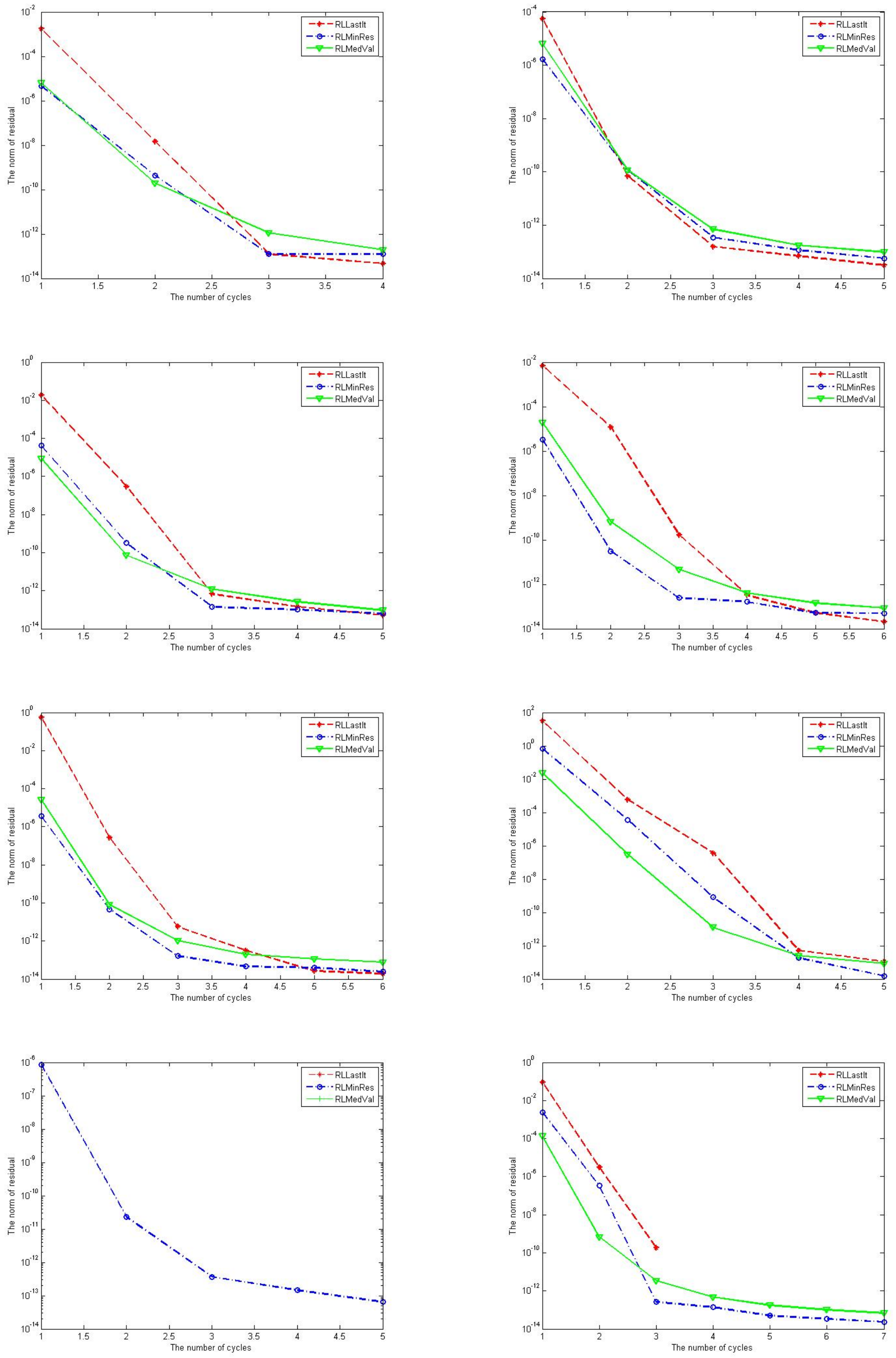


Figure 2.8: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 1000 to 8000, for* $\delta = 0.8$

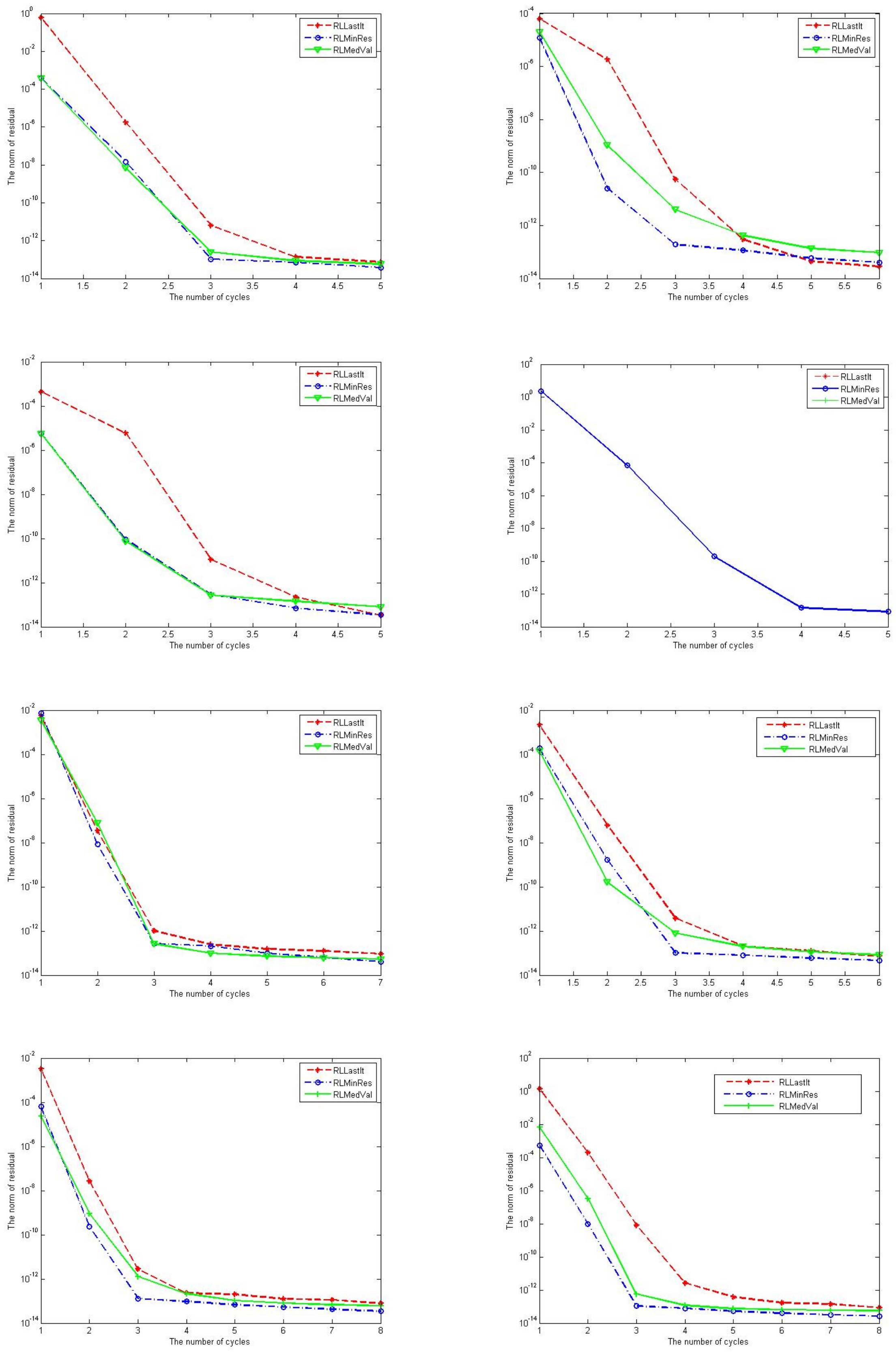


Figure 2.9: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 9000 to 70000, for* $\delta = 0.8$

Table 2.5: *Numerical results of RLLastIt, RLMinRes, and RLMedVal on problems with $\delta = 5$.*

Dim	RLLastIt			RLMinRes			RLMedVal		
n	$\|\|r_k\|\|$	$T(s)$	cycles*	$\|\|r_{min}\|\|$	$T(s)$	cycles*	$\|\|r_{med}\|\|$	$T(s)$	cycles*
1000	$6.6783E-13$	0.8566	9	$1.1544E-13$	0.7058	9	$7.5428E-14$	0.9535	9
2000	$9.0517E-14$	2.4700	9	$1.6451E-13$	2.1698	9	$4.3856E-14$	3.0832	9
3000	$8.9938E-14$	6.1439	8	$1.1733E-13$	5.2819	8	$5.1801E-14$	6.3681	8
4000	$2.0649E-13$	9.8154	6	$1.4507E-13$	8.6705	6	$9.6175E-14$	12.1345	6
5000	$2.0727E-13$	16.0040	8	$1.3336E-13$	12.3384	8	$8.6350E-14$	14.3220	7
6000	$9.1796E-14$	21.2654	10	$2.1285E-13$	19.3882	10	$7.6867E-14$	21.5071	7
7000	$9.2341E-14$	32.1371	13	$1.1998E-13$	29.0991	13	$9.6799E-14$	28.639	7
8000	$9.4310E-14$	43.2287	12	$1.1228E-13$	37.4504	12	$9.4843E-14$	41.8971	6
9000	$9.9740E-13$	45.5492	10	$1.1051E-13$	40.1467	10	$9.9463E-14$	45.5433	6
10000	$1.0468E-13$	81.2740	13	$1.2264E-13$	71.6121	13	$8.6507E-14$	74.9466	7
20000	$9.8099E-14$	785.5109	183	$1.8305E-13$	758.8923	183	$9.2670E-14$	222.3481	8
30000	$2.5585E-13$	594.4041	8	$1.3291E-13$	475.0696	8	$9.3863E-14$	532.8888	8
40000	$1.6819E-12$	5486.982	10	$1.3931E-13$	4419.003	10	$9.4113E-14$	8213.771	10
50000	$1.3379E-12$	11607.939	12	$1.5053E-13$	1.2285.904	12	$9.5873E-14$	20937.437	12
60000	$1.5239E-12$	14511.220	12	$1.5722E-13$	12381.548	12	$9.9868E-14$	23439.061	12
70000	$1.2905E-12$	21904.942	13	$1.5767E-13$	26015.439	13	$9.8246E-14$	44072.510	13

NA = Non Applicable. The computation time and cycles are not calculated if breakdown occurs which is indicated by "NaN" in the residual norm column.

2.2.7 Test Problem VI : $\delta = 8$

The condition number of matrix A for this particular value of delta is 24.4970. The SLEs having this matrix are categorized as well-conditioned. Similar to the case of $\delta = 5$, RLLastIt and RLMinRes are not as accurate as RLMedVal in solving most problems, since they could only achieve solutions with residual norms of $1E-12$. Therefore, RLMedVal performed best on medium and large scale problems.

As can be seen in Table (2.6) and Figures 2.12 and 2.13 for dimension 5000, for instance, RLLastIt, which reached the approximate solution with residual norm of $2.1612E-13$, needed 9 cycles with the total time of about 18.2973 seconds; the RLMinREs also needed 9 cycles with a total time of 15.7859 seconds, with the residual norm of $1.5348E-13$, while RLMedVal managed with 16.1618 seconds, 9 cycles, and achieved a residual norm of $8.2624E-14$. On other problems, for dimensions between 10000 and 70000, RLLastIt managed the approximate solutions with the residual norms of $3.3040E-13$ and $1.2417E-13$ respectively, whereas RLMinRes found a good approximate solution with the residual norms of $1.5087E-13$ and $3.9254E-13$, respectively. RLMedVal achieved the solution with $9.8224E-14$ and $5.5118E-13$, respectively.

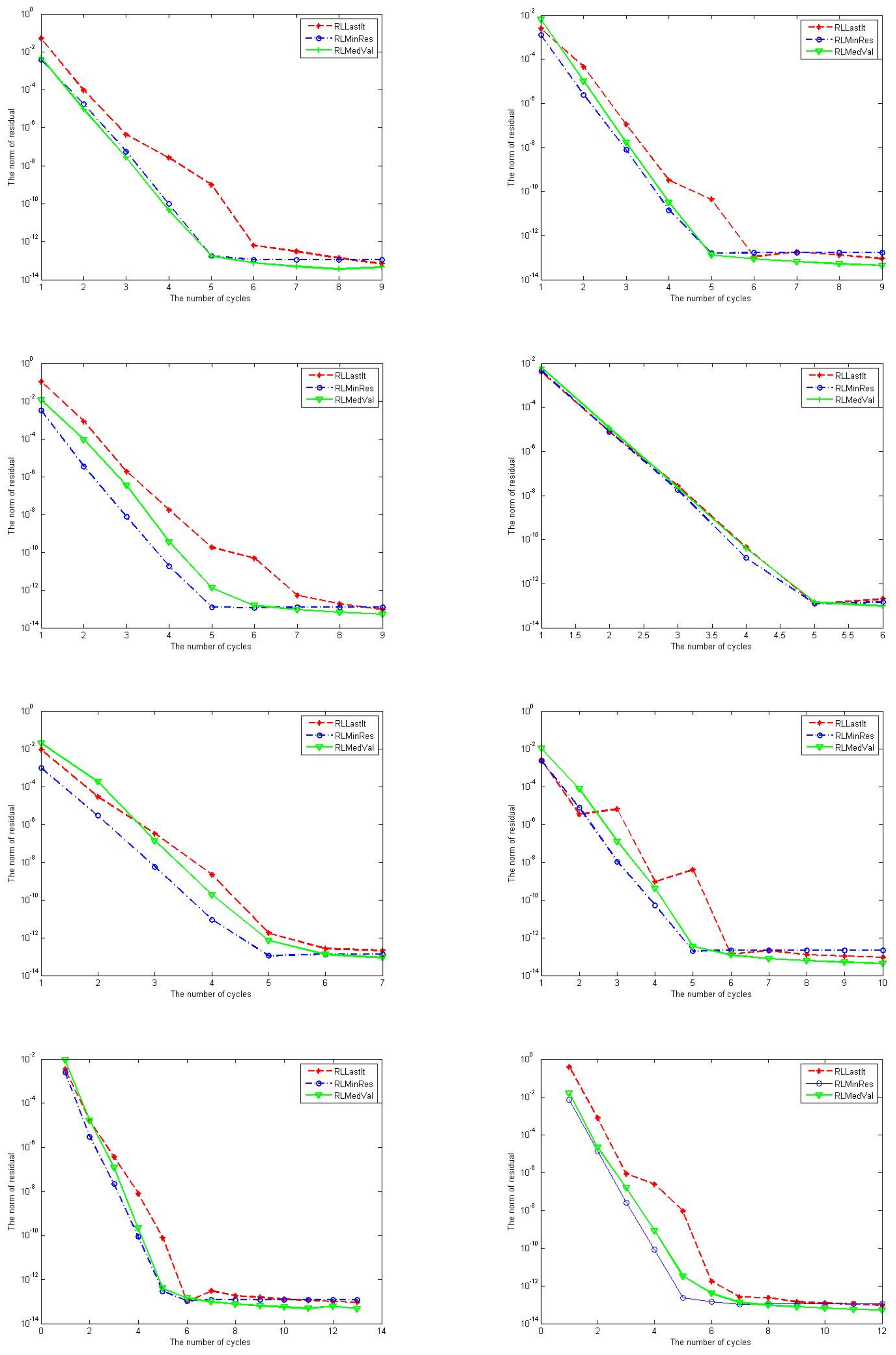


Figure 2.10: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 1000 to 8000, for* $\delta = 5$

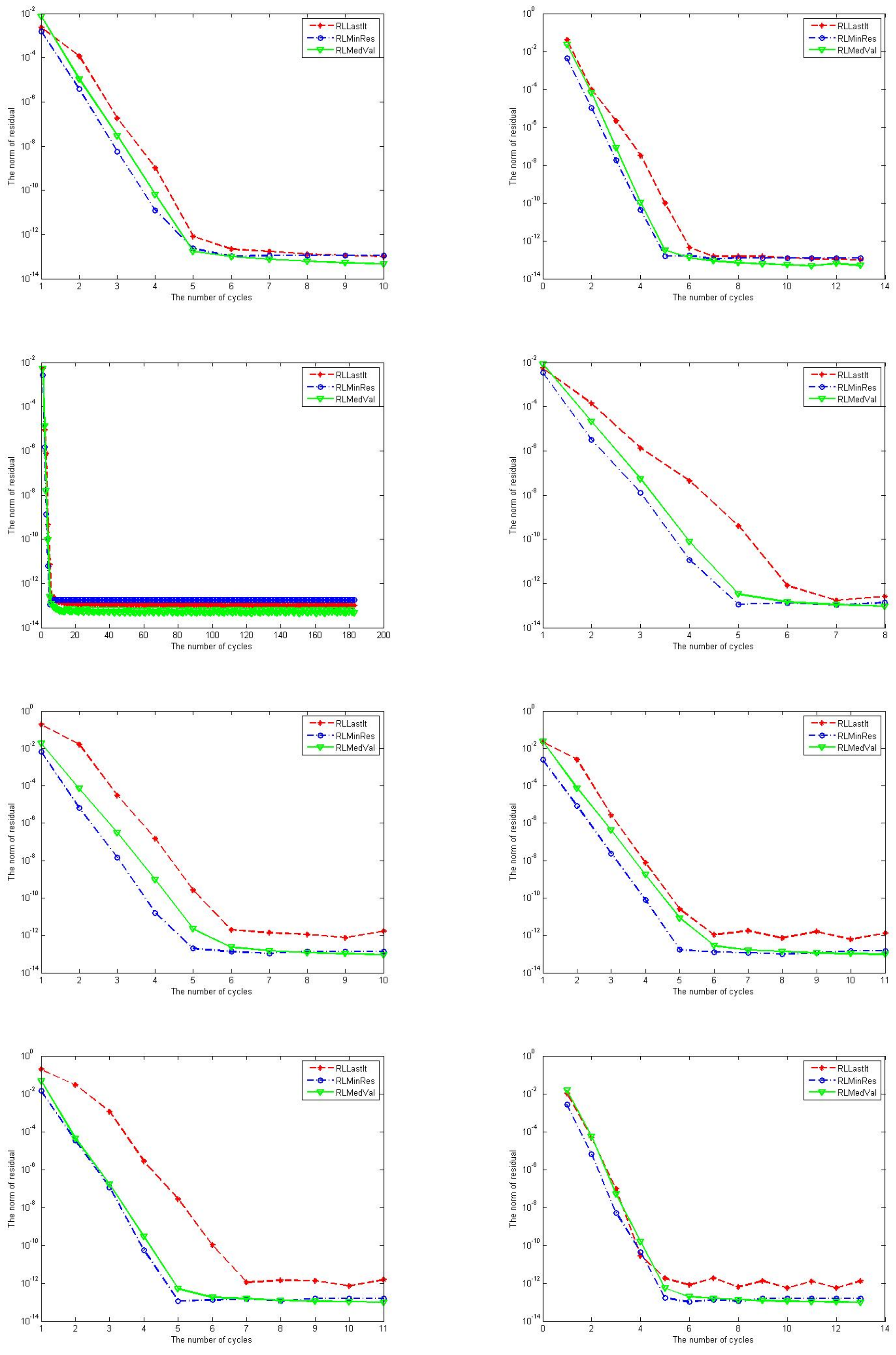


Figure 2.11: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 9000 to 70000, for* $\delta = 5$

Table 2.6: *Numerical results of RLLastIt, RLMinRes, and RLMedVal on problems with $\delta = 8$.*

Dim	RLLastIt			RLMinRes			RLMedVal		
n	$\|\|r_k\|\|$	$T(s)$	cycles*	$\|\|r_{min}\|\|$	$T(s)$	cycles*	$\|\|r_{med}\|\|$	$T(s)$	cycles*
1000	$9.4847E-14$	0.8644	12	$1.4204E-13$	0.7227	12	$7.5478E-14$	2.0025	9
2000	$9.7642E-14$	3.2058	18	$1.2579E-13$	2.8480	9	$6.2576E-14$	2.8289	9
3000	$1.8544E-13$	7.3663	9	$1.2535E-13$	5.7008	9	$7.6937E-14$	6.4005	9
4000	$2.6912E-13$	9.8464	9	$1.0890E-13$	9.8928	9	$9.6175E-14$	11.1164	9
5000	$2.1612E-13$	18.2973	9	$1.5348E-13$	15.7859	9	$8.2624E-14$	16.1618	9
6000	$2.4049E-13$	26.4696	9	$1.8742E-13$	20.8982	9	$8.7058E-14$	23.6583	9
7000	$3.8371E-13$	33.9227	9	$1.1998E-13$	30.0858	9	$9.4308E-14$	32.8377	9
8000	$1.0312E-13$	41.5876	8	$1.4894E-13$	39.6557	8	$9.8204E-14$	42.3492	8
9000	$2.9206E-13$	55.3891	10	$1.4201E-13$	50.4974	10	$8.5654E-14$	57.2675	10
10000	$3.3040E-13$	69.0426	9	$1.5087E-13$	62.4519	9	$9.8224E-14$	62.4430	9
20000	$3.9022E-13$	276.9322	11	$1.4875E-13$	259.4267	11	$9.3115E-14$	288.0455	11
30000	NaN	NA	NA	$8.5189E-14$	408.2613	6	NaN	NA	NA
40000	$1.1998E-12$	12629.417	20	$1.6024E-13$	6249.628	20	$9.9614E-14$	7235.611	20
50000	$8.5401E-13$	33035.391	26	$1.0820E-13$	1.6956.171	26	$9.9792E-14$	14038.619	26
60000	$1.3279E-12$	32253.732	7	$1.1333E-13$	17224.470	7	$5.4308E-13$	20694.178	7
70000	$1.2417E-13$	53545.694	10	$3.9254E-13$	26515.312	6	$5.5118E-13$	34015.217	7

NA = Non Applicable. The computation time and cycles are not calculated if breakdown occurs which is indicated by "NaN" in the residual norm column.

2.3 Summary

In this chapter, we have discussed restarting of Orthodir algorithm from certain points to combat breakdown when solving SLE's. Restarting strategy for treating breakdown has been previously investigated in [29, 30], where one starting point considered. Here, we investigated the quality of starting points, particularly, three different points were considered; each used in one of the implementations RLLastIt, RLMinRes, and RLMedVal. The problems sizes also increased from the previous work which solved up to 4000 dimensions only, to 70,000 dimensions. We conclude that for the case of the ill-conditioned systems, the best restarting point is the iterate with the minimum residual norm, or in this case it is the RLMinRes implementation which is best. However, for systems which are well-conditioned, restarting from the iterate with the median value, or RLMedVal, is slightly more robust than RLMinRes. Between RLMedVal and RLLastIt, the differences are too small to warrant any advantage in using one rather than the other. RLLastIt, is worst on both well-conditioned and ill-conditioned systems.

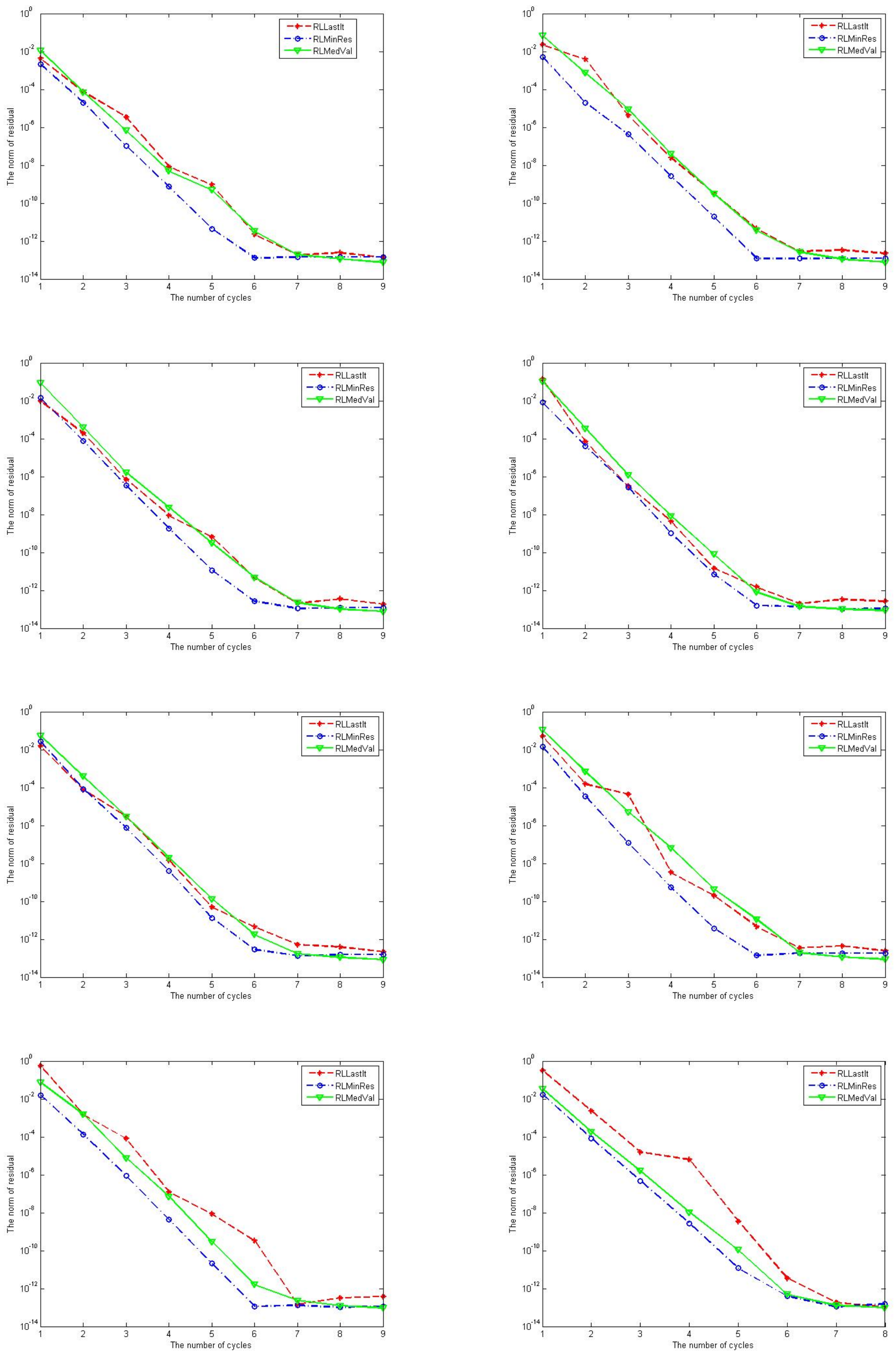


Figure 2.12: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 1000 to 8000, for* $\delta = 8$

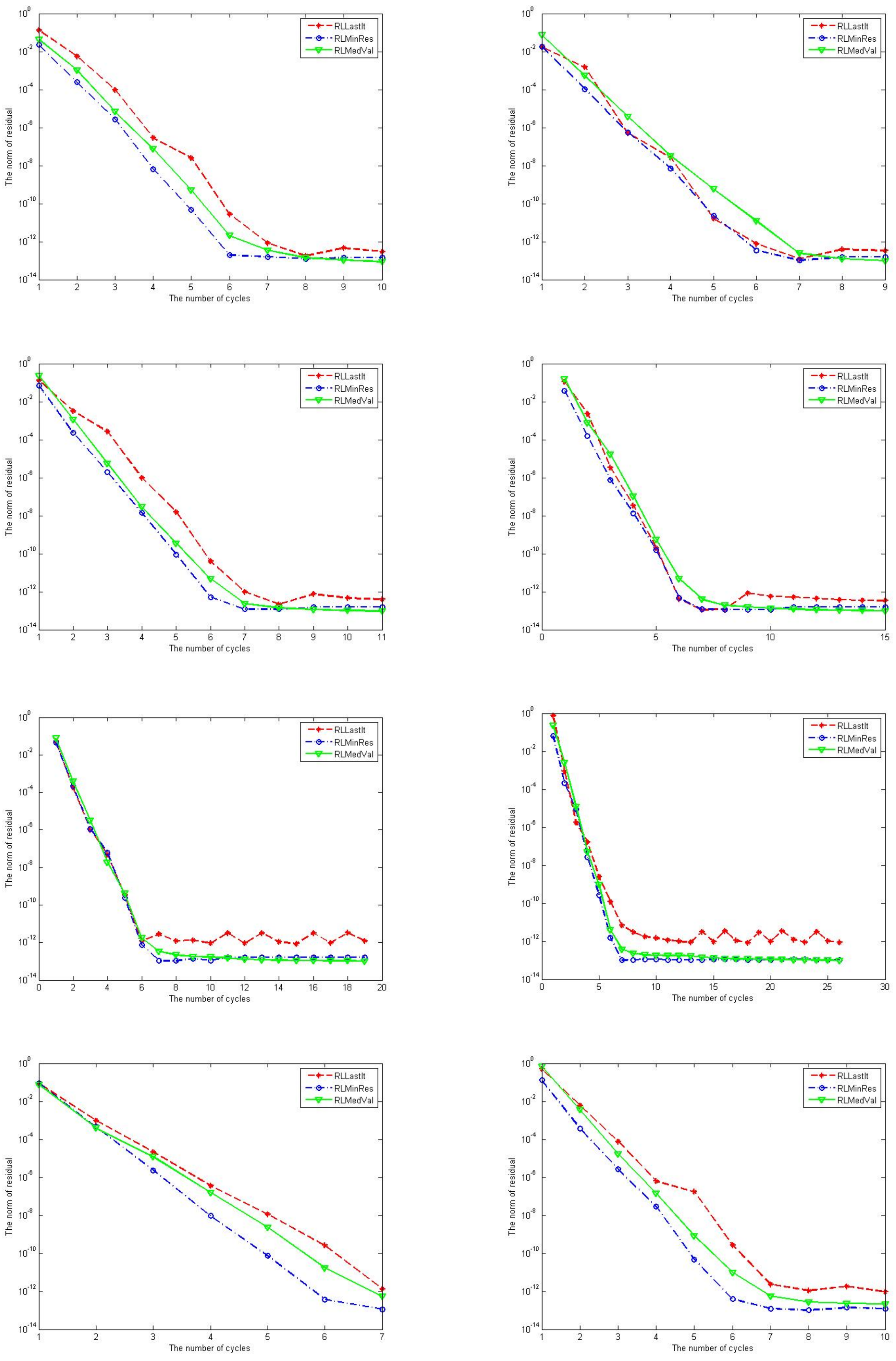


Figure 2.13: *Performance of RLLastIt, RLMinRes, and RLMedVal on SLE's of dimensions from 9000 to 70000, for* $\delta = 8$

Chapter 3

Enhancing the Stability in Lanczos-type Algorithms by Embedding Interpolation and Extrapolation

3.1 Motivation

Breakdown can be avoided in a variety of ways. Here, we suggest regression as a means for that purpose. When breakdown occurs, it is possible to remove the last iterate that caused breakdown from the sequence of Lanczos-type algorithm points, regress on this sequence (their entries) to find a suitable model and then use this model to generate new points which are in the sequence of points that would have been generated by the Lanczos-type algorithm if breakdown did not occur.

The idea is to exploit patterns that may exist in the sequences generated by Lanczos-type algorithms. After we have run such an algorithm for a certain number of iterations, it is intentionally and pre-emptively stopped before it breaks down. We then consider the generated iterates to see if any patterns exist. To illustrate what we mentioned above, we consider a Lanczos-type algorithm such as Orthodir, [6], run it for 30 iterations to solve an SLE in 50 dimensions. We collect all iterates and save them in (3.1) as follows.

$$data_{sol} = \begin{bmatrix} x_1{}^{(1)} & x_1{}^{(2)} & \dots & x_1{}^{(50)} \\ x_2{}^{(1)} & x_2{}^{(2)} & \dots & x_2{}^{(50)} \\ \dots & & & \\ x_{30}{}^{(1)} & x_{30}{}^{(2)} & \dots & x_{30}{}^{(50)}. \end{bmatrix} \tag{3.1}$$

We then plot all above iterates by using Parallel Coordinate System (PCS), [33, 49, 62, 73]. To visualize high-dimensional data, PCS gives an insight into the behaviour of high-dimensional iterates generated over a number of iterations. This is the main motivation behind our idea to exploit patterns that may exist in iterates generated by Lanczos-type algorithms to improve the robustness of these algorithms.

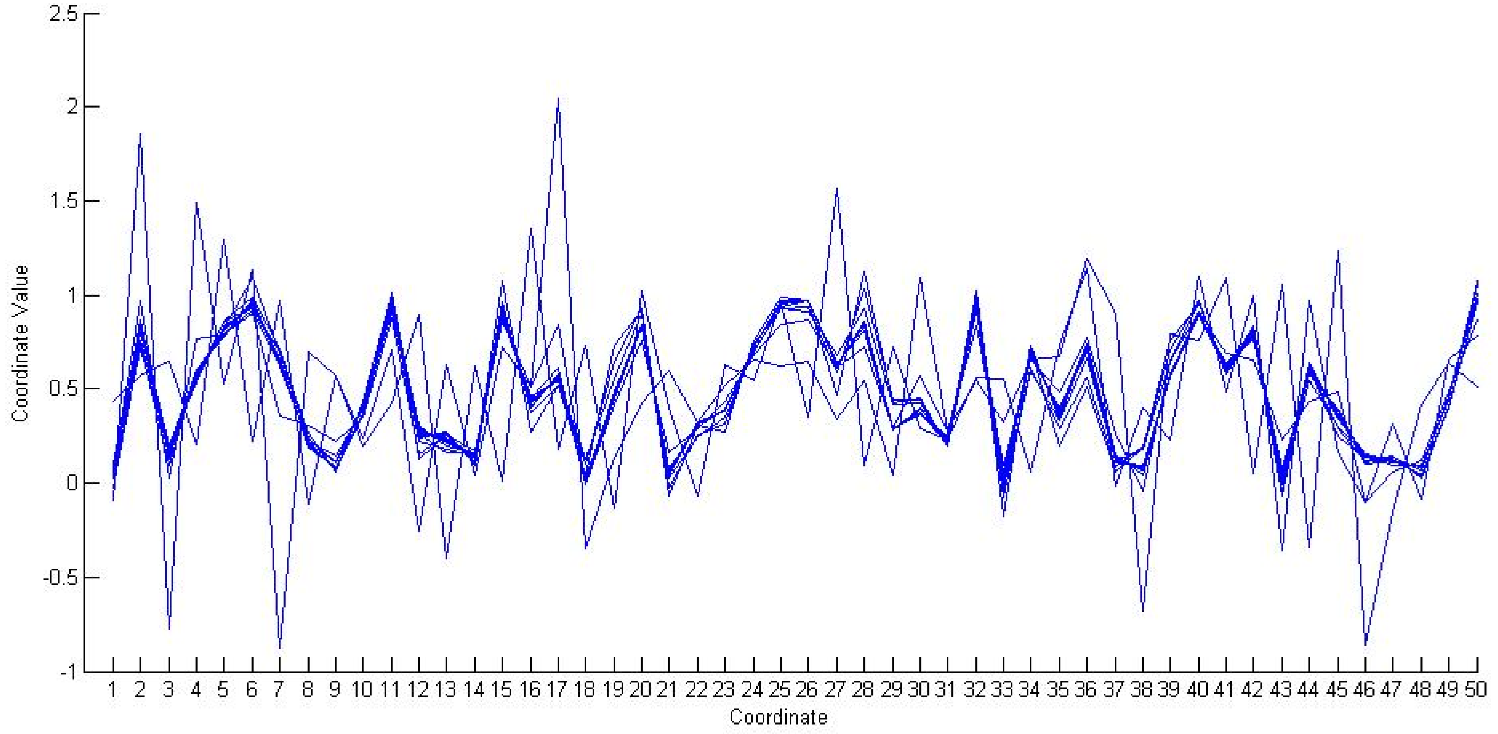
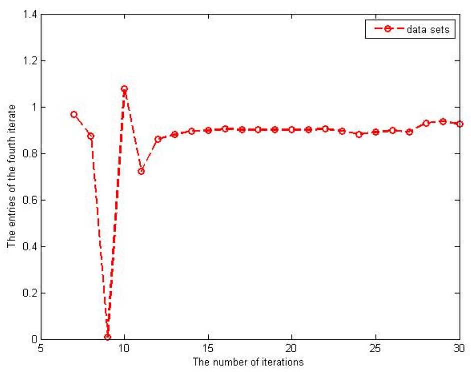


Figure 3.1: *PCS representation of 30 iterates in 50 dimensions.*

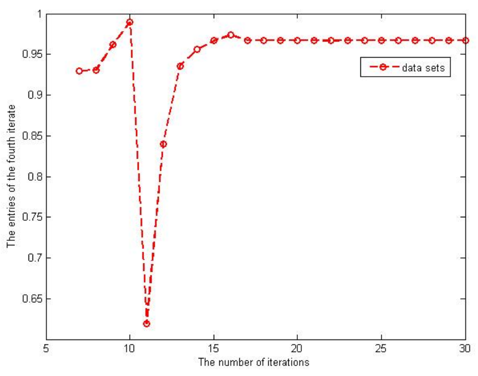
Figure 3.1 illustrates the PCS representation of all iterates generated by Orthodir. The entries of the iterates are represented on the x axis, whereas the values of those entries are represented on the y axis. As we can see here, after some times, several iterates form similar shape. Our assumption of this fact is that these iterates are close enough to the true solution. In other words, these iterates have a small residual norms. We then investigate it further by exploiting the patterns at the level of a single coordinate. More precisely, we are interested in the patterns of the entries of the iterates. This allows us to work in single dimension.

If we have k iterates of a n dimensions problem, the following sequence $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, holds them all. Now, consider the first entries of all iterates in S, the second entries of all iterates in S, etc. A regression model over all first entries will be used to generate the 1st entry of a new point; a regression model over all 2nd entries will be used to generate the second entry of the new point etc... In this fashion, we will generate a new point which, we will establish later, belongs to the sequence of iterates of the Lanczos-type algorithm that has been stopped prematurely, according to some norm. This idea should be applicable for instance when Lanczos process-based algorithms are used to generate eigenvalues and eigenvectors and in any iterative process where breakdown is an issue.

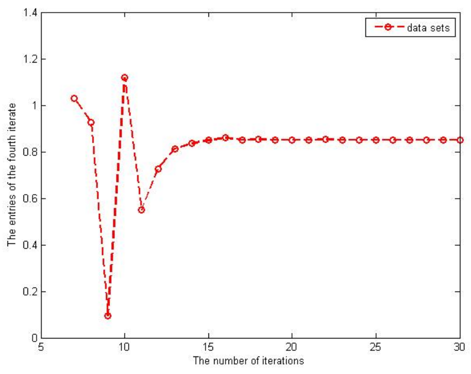
Figure 3.2 illustrates the PCS representation of some entries of the iterates for above problem. As can be seen in the panels of the figure, the coordinate values of the entries at, for examples, 15th, 25th, 28th and 30th, settle down close to some value which may be reasonable assumed to be the true value at convergence. Of course this is not necessarily true for all coordinates. For instance in the case of 3.2(a),



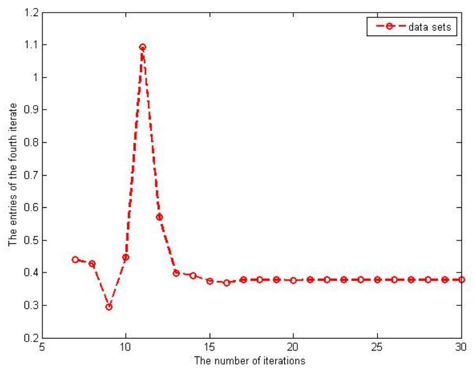
(a) The 15th entries



(b) The 25th entries



(c) The 28th entries



(d) The 30th entries

Figure 3.2: *Representations of some entries of S*

the entry values of coordinate 15 change in the last iterations, then settles down again. These patterns are the ones we are concerned with and not the ones seen in the PCS representation, although they are linked. We might use an interpolation tool to find a good model. The function as a result of the interpolation process, is used to predict a new point which is out of the sequence. In other words, we use extrapolation to generate a new approximate solution. This approximate solution is expected to be better than all of the sequences generated by the Lanczos-type algorithm.

The above observations raise some important questions:

1. Is it possible to build a model which represents the solutions generated by

Lanczos-type algorithms?

2. If so, after how many iterations does it settle and persist until convergence occurs?

3. If such a model does exist, how good an approximate solution can be generated using it instead of the iterative process itself?

If the answers to questions (1) and (3) are, "YES", then we can stipulate that it is possible to halt the iterative process to avoid breakdown and use the model to generate better solutions instead.

Generally, since the above process uses few iterations, breakdown can be avoided. Also, we avoid computing the approximate solution iteratively. Instead, we embed a model function into a Lanczos algorithm to get a new point. This new approach is called embedded interpolation and extrapolation model in Lanczos-type algorithms (EIEMLA).

This chapter is organized as follows. The first section discusses the motivation of the EIEMLA. The second section explains in detail the EIEMLA method including some theoretical results to justify that our approach works. The last section provides some numerical results involving the embedded interpolation and extrapolation model (EIEM) in various Lanczos-type algorithms, namely EIEM Orthodir, EIEM Orthomin, EIEM A_{12}.

3.2 Background Theories

We briefly gives some theories related to our new approach explained in the previous section.

3.2.1 Interpolation and Extrapolation

Numerical interpolation, [71], is a frequently used tool in numerical applications that arise in a variety of domains. A common problematic situation is when a function f of a set of points $t_1, t_2, \ldots, t_n$ can not be cast in an analytic form. For instance, f may represent some data points from experiments or the results of a large-scale problem computation, or some physical quantity. We then may need to evaluate the function f at some points t within the data set $t_1, t_2, \ldots, t_n$, but where t differs from the tabulated values. In this case we are dealing with interpolation, [61]. If t is outside then we are facing extrapolation, [65].

There are many interpolation methods. The most common is, possibly, the polynomial interpolation which simply connects each data point with polynomial functions. The other type is called piecewice-polynomial interpolation which is based on the *spline* functions [36]. Splines are polynomials on subintervals that are connected in a smooth way, [61]. A special case of spline functions is the *cubic spline* which involves a polynomial of degree ≤ 3 as follows

$$f(t) = \alpha_3 t^3 + \alpha_2 t^2 + \alpha_1 t + \alpha_0 \tag{3.2}$$

This polynomial is known as the *cubic spline* function and an interpolation which is based on it, is called the *cubic spline interpolation*, [20, 82]. Cubic splines which use low-order polynomials are especially attractive for curve fitting because they reduce the computational requirements and numerical instabilities that arise with higher degree curves, [23]. The other type of spline interpolation is the *cubic Hermite spline* interpolation which is based on the cubic Hermite function.

Some methods, [36, 50, 82], claim that the cubic spline is smoother than the cubic Hermite interpolation. It is reasonable since the spline has two continuous derivatives, while the cubic Hermite interpolation has only one. However, the cubic Hermite interpolation is guaranteed to preserve the shape, but the spline might not. This special property is the main reason we choose to use it here.

3.2.2 Piecewise Cubic Hermite Interpolating Polynomial

Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) has been proposed to preserve the monotonicity, positivity, and convexity of the data, [35]. As above mentioned, it is very important to preserve these properties of the data to get an accurate interpolant and to get a good prediction. PCHIP is designed with these purposes in mind.

Let $a = t_1 < t_2 < \cdots < t_n = b$ be a partition of the interval $[a, b]$, [50]. Let $\{(t_i, f_i) : i = 1, 2, \cdots, n\}$ be the given data, where $f_i \leq f_{i+1}$ (monotonic increasing) or $f_i \geq f_{i+1}$ (monotonic decreasing). For $t \in [t_i, t_{i+1}]$, $i = 1, 2, \cdots, n-1$, $p(x)$ is a cubic polynomial which can be represented as follows.

$$p(x) = f_i H_1(t) + f_{i+1} H_2(t) + d_i H_3(t) + d_{i+1} H_4(t), \tag{3.3}$$

where $d_j = p'(t_i)$. $j = i, i+1$ and $H_k(t)$ are the usual cubic Hermite basis functions given by

$$
\begin{aligned}
H_1(t) &= \phi\left(\frac{t_{i+1} - t_i}{h_i}\right) \\
H_2(t) &= \phi\left(\frac{t - t_i}{h_i}\right) \\
H_3(t) &= -h_i\psi\left(\frac{t_{i+1} - t_i}{h_i}\right) \\
H_4(t) &= -h_i\psi\left(\frac{t - t_i}{h_i}\right),
\end{aligned}
\tag{3.4}
$$

and $h_i = t_{i+1} - t_i$, $\phi(w) = 3w^2 - 2w^3$, $\psi(w) = w^3 - w^2$. Fritsch and Carlson [35] derived the necessary and sufficient conditions for the function (3.3) to be monotonic. They also produced an algorithm for determining the derivative values $d_1, d_2, \cdots, d_n$. Those values are used to make the piecewise cubic Hermite polynomial in (3.4) monotonic. For more detail about PCHIP and its applications refer to [2, 18, 20, 23, 34, 44, 46, 61, 82].

3.3 Embedded Interpolation and Extrapolation Model in Lanzos-type Algorithms

This approach involves designing a model function as a result of interpolation over a sequence of iterates generated by the algorithms. The sequence in question is generated by a Lanczos-type algorithm over a certain number of iterations, say k. We assume that the algorithm breaks down after k iterations. We then try to predict the next iterate, $\mathbf{x}_{k+1}$, that would be generated by the algorithm if it did not break (or has not been stopped). In other words, this iterate is linking up all the previous iterates, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$, to the next one without a further iteration of the process. How good is this new iterate, depends on the interpolation tool that we use to capture the existence of patterns in the sequence of the iterates generated by Lanczos-type algorithms.

3.3.1 Derivation of EIEMLA

Consider k iterates which form $data_{sol}$ as in (2.3). As explained earlier, among the $\mathbf{x}_k$, there are some iterates with small residual norms. These particular entries might have a special property, such as increasing/decreasing monotonically. We consider PCHIP to interpolate them, so that a model function as a result of this process can represent the points considered.

Let $\mathbf{x}_m$, be the iterate with the lowest residual norm, $||\mathbf{r}_m||$, where $m \leq k$. Assume that some good iterates, namely those with small residual norms, concentrate in interval $[m - j, k]$, for some integer j. Set

$$V_1 = \left\{\mathbf{x}_{m-j}, \mathbf{x}_{m-j+1}, \ldots, \mathbf{x}_k\right\}, \tag{3.5}$$

which is a subset of (2.3). Write the components of each iterate in (3.5) as

$$\begin{aligned} v_1 &= \left\{x_{m-j}{}^{(1)}, x_{m-j+1}{}^{(1)}, \ldots, x_k{}^{(1)}\right\} \\ v_2 &= \left\{x_{m-j}{}^{(2)}, x_{m-j+1}{}^{(2)}, \ldots, x_k{}^{(2)}\right\} \\ &\vdots \\ v_n &= \left\{x_{m-j}{}^{(n)}, x_{m-j+1}{}^{(n)}, \ldots, x_k{}^{(n)}\right\} \end{aligned} \tag{3.6}$$

namely, each v_i contains all of the i^{th} entries of iterates $\mathbf{x}_l$, for $l = m-j, m-j+1, \cdots, k$, and for $i = 1, 2, \cdots, n$. For instance, v_1 contains all of the first entries of iterates $\mathbf{x}_l$, v_2 contains all of the second entries of iterates $\mathbf{x}_l$, etc. Thus, we find a function which interpolates each set of v_i's using PCHIP interpolant. We assume that each sequence of ${x_{m-j}}^{(i)}, {x_{m-j+1}}^{(i)}, \ldots, {x_k}^{(i)}$ is monotonic and convergent for some j and $i = 1, 2, \ldots, n$, to its limit, [74], i.e.

$$\lim_{k\to\infty} {x_k}^{(i)} = {x_*}^{(i)}. \tag{3.7}$$

Let t be elements in R. Set

$$\begin{aligned} w_1 &= \left\{\left(t_{m-j}, {x_{m-j}}^{(1)}\right), \left(t_{m-j+1}, {x_{m-j+1}}^{(1)}\right), \ldots, \left(t_k, {x_k}^{(1)}\right)\right\} \\ w_2 &= \left\{\left(t_{m-j}, {x_{m-j}}^{(2)}\right), \left(t_{m-j+1}, {x_{m-j+1}}^{(2)}\right), \ldots, \left(t_k, {x_k}^{(2)}\right)\right\} \\ &\vdots \\ w_n &= \left\{\left(t_{m-j}, {x_{m-j}}^{(n)}\right), \left(t_{m-j+1}, {x_{m-j+1}}^{(n)}\right), \ldots, \left(t_k, {x_k}^{(n)}\right)\right\}. \end{aligned} \tag{3.8}$$

Using PCHIP to interpolate each w_i, for $i = 1, 2, \ldots, n$, yields functions f_i. As it is a regular interpolation process in R, then for some $t = m - j, m - j + 1, \ldots, k$, f_i satisfy

$$f_i(t) \approx x_t^{(i)} \qquad \text{for } i = 1, 2, \ldots, n. \tag{3.9}$$

For instance,

$$\begin{aligned} f_i(m-j) &\approx x_{m-j}^{(i)} \\ f_i(m-j+1) &\approx x_{m-j+1}^{(i)} \\ &\vdots \\ f_i(k) &\approx x_k^{(i)} \qquad \text{for } i = 1, 2, \ldots, n. \end{aligned} \tag{3.10}$$

Since we use an appropriate interpolant to interpolate the data, i.e. the one that preserves the monotonicity of the data, then the extrapolation based on this interpolation process enables us to get the next point outside of the range. It means that if we calculate $f_i(t^*)$ with $t^* \in [k+1, s] \subset R$, where $s \geq k+1$, then we obtain

$$f_i(t^*) \approx x_r^{(i)} \qquad \text{for } i = 1, 2, \ldots, n, \tag{3.11}$$

where each $x_r^{(i)}$ has a similar property as $x_t^{(i)}$ in (3.9). In other words, if the sequence of $x_t^{(i)}$ is monotonically increasing/decreasing, so is $x_r^{(i)}$. Thus arranging vector $\mathbf{x}_r$, with $x_r{}^{(i)}$ being the i^{th} entries of the vector, yields an approximate solution of the system.

Theoretically, since PCHIP captures the persistent pattern of the $i-th$ entry, $i = 1, ..., n$, of the iterates generated by a Lanczos-type algorithm, the entries of the new iterate, as a result of the model function, are likely to behave as those entries. Furthermore, we can produce as many vector solutions as we want by applying the functions f_i over $t^* \in R$, without running Lanczos-type algorithms again. However, we should be aware that the quality of these generated solutions may not be good enough, in which case we must either restart the iterative process from the best point or take the best solution found so far as the candidate solution. It is therefore, reasonable to choose the integer s such that the residual norms of the iterates generated by these functions, $\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, \ldots, \mathbf{x}_s$ are small enough. In this case, we obtain another sequence of the iterates generated by EIEMLA. It is expected that these iterates replace the ″missing″ iterates not generated by the Lanczos-type algorithm due to breakdown.

3.3.2 Implementation of the EIEMLA Method

The above results suggest a procedure for EIEMLA method (see Algorithm 10). As an illustration of the implementation of EIEMLA, consider (A.1.1) with $n = 50$.

Orthodir is used to solve it for 50 iterations. Following the above procedures, we first collect all of the 50 iterates as $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{50}\}$.

Algorithm 10 The EIEMLA method

```
1:  Initialization.   Choose $\mathbf{x}_0$ and $\mathbf{y}$. Set $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{y}_0 = \mathbf{y}$, and $\mathbf{z}_0 = \mathbf{r}_0$.
2:  Fix the number of iterations to, say, $k$, and the tolerance, $\epsilon$, to $E - 13$ and run a
    Lanczos-type algorithm.
3:  if $||\mathbf{r}_k|| \leq \epsilon$ then
4:      The solution is obtained
5:      Stop
6:  else
7:      Collect all $k$ vector solutions as in (2.3).
8:      Choose some $j$ such that $m - j \leq k$.
9:      Set $w_i$ as in (3.8), for $i = 1, 2, \ldots, n$.
10:     Interpolate $w_i$ using PCHIP to get $f_i$.
11:     Choose $t^* \in [m, s] \subset R$, where $s \geq m \geq k$ is an integer, and calculate $f_i(t^*)$.
12:     for $q = 1, 2, \ldots, l$ do
13:         Arrange vectors
```

$$\mathbf{x}_*{}^q = \begin{pmatrix} (f_1{}^q)(t^{*q}) \\ (f_2{}^q)(t^{*q}) \\ \vdots \\ (f_n{}^q)(t^{*q}) \end{pmatrix}, \tag{3.12}$$

```
            where $l = length([m, s])$.
14:         Calculate the residual norms of (3.12) as follows
```

$$||\mathbf{r}_*{}^q|| = ||\mathbf{b} - A\mathbf{x}_*{}^q|| \tag{3.13}$$

```
15:     end for
16: end if
17: The solutions of the systems are $\mathbf{x}_*{}^{(1)}, \mathbf{x}_*^{(2)}, \ldots, \mathbf{x}_*{}^{(l)}$.
18: Stop.
```

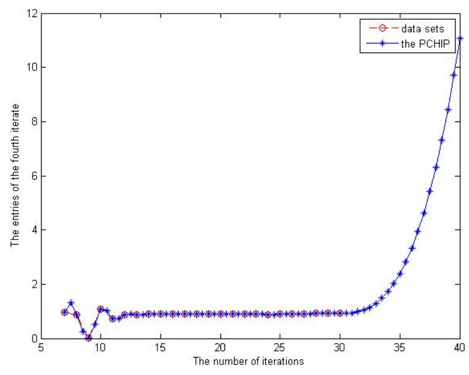
The visualization of several entries of S is as shown in Figure 3.2, in Section 1.1. Secondly, we calculate index m of the iterate $\mathbf{x}_m$ associated with the lowest residual norm $||\mathbf{r}_m||$. In this particular example, $||\mathbf{r}_m|| = 0.0067$ with $m = 17$. Thus re-arranging S as in (3.6), would yield the sequence as follows

$$\begin{aligned} w_1 &= \left\{\left(t_7, x_7{}^{(1)}\right), \left(t_8, x_8{}^{(1)}\right), \ldots, \left(t_{50}, x_{50}{}^{(1)}\right)\right\} \\ w_2 &= \left\{\left(t_7, x_7{}^{(2)}\right), \left(t_8, x_8{}^{(2)}\right), \ldots, \left(t_{50}, x_{50}{}^{(2)}\right)\right\} \\ &\vdots \\ w_{50} &= \left\{\left(t_7, x_7{}^{(50)}\right), \left(t_8, x_8{}^{(50)}\right), \ldots, \left(t_{50}, x_{50}{}^{(50)}\right)\right\}, \end{aligned} \tag{3.14}$$

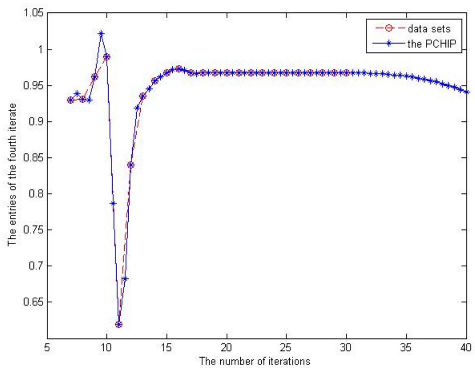
where t is an integer in $[m-10, 50]$. The choice of this range is based on our observation that some good iterates can be found in it. Also, based on Figure 3.2, the entries of the iterates are increasing or decreasing monotonically in that range.

The next step is to interpolate each w_i in (3.14) using PCHIP to get functions f_i. We use these functions to extrapolate and generate points which are out of the range. In this case, we choose $t^* \in [17, 70]$. Interpolation and extrapolation results are captured in Figure 3.3. It represents the interpolation and extrapolation model (EIM) of the data in Figures 3.2(a), 3.2(b), 3.2(c), and 3.2(d), respectively. As can be seen here, PCHIP interpolates the data accurately and smoothly (see the blue curves). However, extrapolation results in some points being of poor quality. In Figure 3.3(a), for instance, the blue curve which represents the PCHIP data, goes up after the 50^{th} iteration. PCHIP curve, on the other hand, seems monotone after the 50^{th} iteration in Figures 3.3(b) and 3.3(c), 3.3(d).

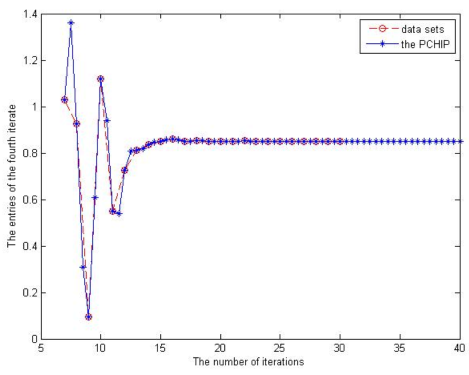
Amongst the iterates generated by Orthdir, some entries behave as Figure 3.3(a). This means their trend goes up or down after k iterations. However, there are many other entries which behave monotonically. Since we put all entries in one vector, overall, the effect of the fluctuating entries is less which means that a



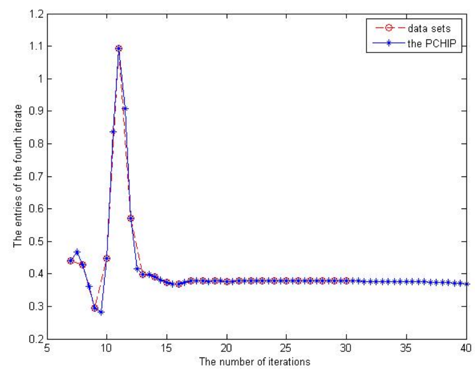
(a) The IEM of the 15th entries



(b) The IEM of the 25th entries



(c) The IEM of the 28th entries



(d) The IEM of the 30th entries

Figure 3.3: *The interpolation and extrapolation results of some entries of S*

good approximate solution may be generated.

After calculating functions f_i over t^*, we now have the approximate solutions as in (3.12). We compute the residual norms accordingly. The behaviour of EIEM Orthodir and Orthodir for this case, is represented in Figure 3.4. As can be seen here, generally the residual norms of iterates generated by EIEM Orthodir (the blue curve) are found below those generated by Orthodir (the red curve) from iteration 17 to 50. In fact, at iteration 31, the residual norm hits the value of 0.003. However, it goes up after iteration 71.

We can safely say that some iterates generated by EIEM Orthodir have smaller residual norm than all previous iterates generated by the Orthodir. On its own, it

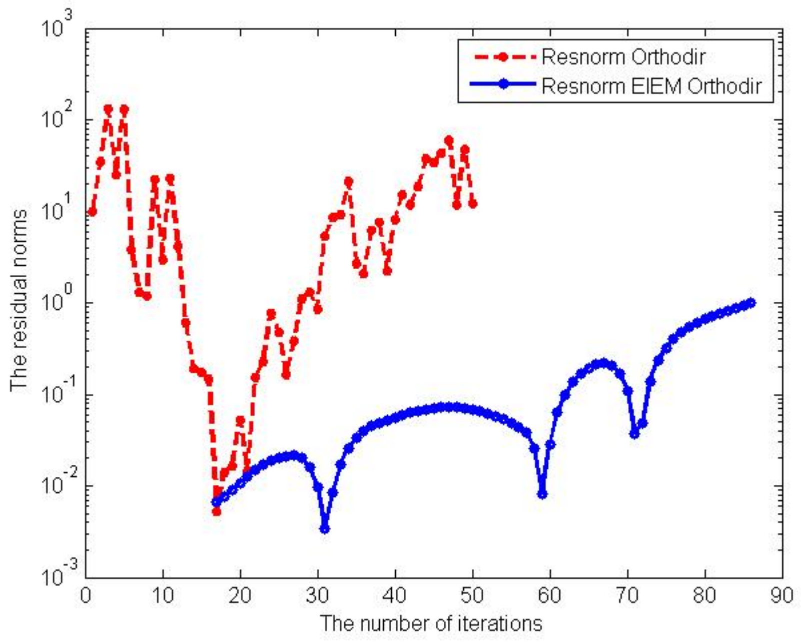


Figure 3.4: *Behaviour of EIEM Orthodir and Orthodir algorithms on SLE's in 50 dimensions. The blue line which corresponds to EIEM Orthodir has some very good points and grows much slower than that corresponding to Orthodir.*

doesn't find solutions with such low residual norms even with a high number of iterations.

3.3.3 Formal Basis of EIEMLA

As mentioned in Section 3.2.2, the sequences generated by the Lanczos-type algorithm have the property of monotonicity. Since we consider PCHIP which preserves monotonicity, [21], to interpolate the sequences, we can assume that points returned by the function, are also monotonic. This leads to the theorems below which guarantee the monotonicity property of sequences generated by Lanczos-type algorithms.

Theorem 3.3.1. *Given a sequence $\{\mathbf{x}_k\}$ of k iterates generated by a Lanczos-type algorithm, sequences of $x_k{}^{(i)}$, $i = 1, 2, \cdots, n$, and $k = 1, \ldots$ namely the entries of k iterates, are monotonic.*

Proof. Consider sequences generated by Orthodir as follows.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - a_{k+1}\mathbf{z}_k, \tag{3.15}$$

where a_{k+1} is a constant defined as follows.

$$a_{k+1} = \frac{-c(t^k P_k)}{c(t^{k+1} P_k^{(1)})}, \tag{3.16}$$

c is a linear function and P_k is an orthogonal polynomial, [29]. We write (3.15) in an extended form as follows

$$\begin{pmatrix} {x_{k+1}}^{(1)} \\ {x_{k+1}}^{(2)} \\ \vdots \\ {x_{k+1}}^{(n)} \end{pmatrix} = \begin{pmatrix} {x_k}^{(1)} \\ {x_k}^{(2)} \\ \vdots \\ {x_k}^{(n)} \end{pmatrix} - a_{k+1} \begin{pmatrix} {z_k}^{(1)} \\ {z_k}^{(2)} \\ \vdots \\ {z_k}^{(n)} \end{pmatrix}.$$

Therefore, we now have a sequence for every single variable , namely

$$\begin{aligned} {x_{k+1}}^{(1)} &= {x_k}^{(1)} - a_{k+1}{z_k}^{(1)} \\ {x_{k+1}}^{(2)} &= {x_k}^{(2)} - a_{k+1}{z_k}^{(2)} \\ &\vdots \\ {x_{k+1}}^{(n)} &= {x_k}^{(n)} - a_{k+1}{z_k}^{(n)}. \end{aligned}$$

Thus, we show that every single ${x_k}^{(i)}$, for $i = 1, 2, \cdots, n$, is monotonically increasing by assuming that variables other than ${x_k}^{(i)}$ are constant. A sequence ${x_k}^{(i)}$ is monotonically increasing if and only if

$$\begin{aligned} x_k{}^{(i)} &\leq x_{k+1}{}^{(i)} \\ &\leq x_k{}^{(i)} - a_{k+1} z_k{}^{(i)} \\ 0 &\leq -a_{k+1} z_k{}^{(i)} \\ a_{k+1} z_k{}^{(i)} &\leq 0. \end{aligned} \tag{3.17}$$

We claim that $a_{k+1} > 0$. Since we have (3.16), this last relation is valid. To prove the above claim, we look at again the relation between this coefficient and some variables involved in Orthodir algorithm. According to Algorithm 2 in Chapter 1, the residual vector is computed as follows

$$\mathbf{r}_{k+1} = \mathbf{r}_k + a_{k+1} A \mathbf{z}_k.$$

We want $||\mathbf{r}_{k+1}|| \leq ||\mathbf{r}_k||$, so in this case, we have to minimize $||a_{k+1} A \mathbf{z}_k||$. As (3.16), coefficient a_{k+1} must be a large positive number. So, the claim is true. $\diamond$

Suppose we have sequences as in (2.3), which are obtained by running Orthodir for k iterations. If we generate the next single point using the same algorithm, we have a sequence

$$V' = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \mathbf{x}_{k+1}\}. \tag{3.18}$$

Since EIEMLA enables us to predict some further points after k iterates, then we need to guarantee that point, $\mathbf{x}_{model}$, for instance, is in the sequence (3.18). The idea is that if it is in the sequence of V', then it is the approximate solution of our system. For that, we need to show that the distance between two vectors and $\mathbf{x}_{k+1}$

and $\mathbf{x}_{model}$ is sufficiently small.

Lemma 1. *[48] Let $A = VAV^{-1}$ be a nonsingular diagonalizable matrix, where V is a matrix which consists of the eigenvectors of A, S is a diagonal matrix with $a_{ii} \in \sigma(A)$ being the diagonal entries, and $\sigma(A)$ is the set of the eigenvalues of A. Then,*

$$||P_k(A)|| \leq \kappa_2(V) \max_{\lambda \in \sigma(A)} |\lambda|,$$

where $\kappa_2(V)$ is the condition of matrix V.

Proof. Let λ be any eigenvalue of matrix A. Then, we have

$$\begin{aligned} ||P_k(A)|| &= \left\| VP_k(S)V^{-1} \right\| \\ &\leq ||V||\, ||P_k(S)|| \left\| V^{-1} \right\| \\ &= \kappa_2(V)\, ||P_k(S)|| \\ &= \kappa_2(V) \max_{\lambda \in \sigma(A)} |\lambda|, \end{aligned} \tag{3.19}$$

where $\kappa_2(V)$ is the condition number of matrix V. $\diamond$.

Theorem 3.3.2. *Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \mathbf{x}_{k+1}$ be the iterates generated by Orthodir algorithm. Let $\mathbf{x}_{model}$ be a vector returned by EIEMLA as explained in the previous section. Then, for some $\epsilon > 0$,*

$$||\mathbf{x}_{k+1} - \mathbf{x}_{model}|| \leq \epsilon, \tag{3.20}$$

where $||\cdot||$ is the Euclidean norm.

Proof. Orthodir iterates are represented by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{z}_k, \tag{3.21}$$

where α is given in (3.16), and $\mathbf{z}_k = P_k(A)\mathbf{z}_0$. We write the vectors as their components as

$$\mathbf{x}_{k+1} - \mathbf{x}_{model} = \mathbf{x}_k - \alpha \mathbf{z}_k - \mathbf{x}_{model}. \tag{3.22}$$

Taking the norm of both sides of (3.22) we get

$$\begin{aligned} \|\mathbf{x}_{k+1} - \mathbf{x}_{model}\| &= \|\mathbf{x}_k - \alpha \mathbf{z}_k - \mathbf{x}_{model}\| \\ &\leq \|\mathbf{x}_k - \mathbf{x}_{model}\| + |-\alpha| \, \|\mathbf{z}_k\| \,. \end{aligned} \tag{3.23}$$

We split relation (3.23) into two terms, i.e. $\|\mathbf{x}_k - \mathbf{x}_{model}\|$ and $|-\alpha| \, \|\mathbf{z}_k\|$. First, we compute the term

$$\begin{aligned} \|\mathbf{x}_k - \mathbf{x}_{model}\| &= \left\| \begin{matrix} x_k{}^{(1)} - x_{model}{}^{(1)} \\ x_k{}^{(2)} - x_{model}{}^{(2)} \\ \vdots \\ x_k{}^{(n)} - x_{model}{}^{(n)} \end{matrix} \right\| \\ &= \left\| \begin{matrix} x_k{}^{(1)} - f_1(t) \\ x_k{}^{(2)} - f_2(t) \\ \vdots \\ x_k{}^{(n)} - f_n(t) \end{matrix} \right\|, \end{aligned} \tag{3.24}$$

for some $t \in [1, k]$. Since $f_i(t)$, for $i = 1, 2, \ldots, n$ are returned functions at some above t, then we have

$$f_i(t) \approx x_i^k.$$

Therefore, (3.24) is less than ϵ_1, for some $\epsilon_1 > 0$.

The second term of (3.23) is computed as follows. Using the result given

Lemma 1, [48], by putting (3.19) into the norm of $\mathbf{z}_k$, we obtain

$$\begin{aligned}
||-\alpha\mathbf{z}_k|| &= |-\alpha|\,||P_k(A)\mathbf{z}_0|| \\
&= \alpha\,||P_k(A)\mathbf{z}_0|| \\
&= \alpha\,||P_k(A)\mathbf{r}_0||\,, \quad \text{since } \mathbf{z}_0 = \mathbf{r}_0 \text{ at the beginning of the iteration,} \\
&\leq \alpha\,||P_k(A)||\,||\mathbf{r}_0||\,, \quad \text{since } ||.|| \text{ is the induced norm.} \\
&\leq \alpha\kappa_2(V)\max_{\lambda\in\sigma(A)}|\lambda|\,||\mathbf{r}_0||\,.
\end{aligned}$$

We assume that the initial guess is chosen such that it is close to the true solution so that $||\mathbf{r}_0|| \leq \epsilon_2$, for some $\epsilon_2 > 0$. So, we now have

$$||-\alpha\mathbf{z}_k|| \leq \alpha\epsilon_2\kappa_2(V)\max_{\lambda\in\sigma(A)}|\lambda|\,. \tag{3.25}$$

Hence we substitute (3.24) and (3.25) into (3.23) to get

$$\begin{aligned}
||\mathbf{x}_{k+1} - \mathbf{x}_{model}|| &\leq ||\mathbf{x}_k - \mathbf{x}_{model}|| + |-\alpha|\,||\mathbf{z}_k|| \\
&\leq \epsilon_1 + \alpha\epsilon_2\kappa_2(V)\max_{\lambda\in\sigma(A)}|\lambda| \\
&\leq \max\left\{\epsilon_1, \alpha\epsilon_2\kappa_2(V)\max_{\lambda\in\sigma(A)}|\lambda|\right\}. \qquad \diamond
\end{aligned}$$

An alternative way to show that the generated point via regression belongs to the sequence of points generated by the Lanczos-type algorithm is through an envelope. Such a device could be found by fitting a couple of curves one to the lower bounds on the the iterates in the Lanczos-type algorithm sequence, and the other to the upper bounds on these same points. The conjecture is that any point that falls within this envelope must belong to the sequence. Note, it is suggested for further work to prove Theorem 3.3.2 using this idea.

Based on the above theorem, we finally show that the residual norm of the iterate generated by EIEMLA is always smaller or equal to that of the iterate generated by the Lanczos-type algorithms considered. In other words, better solutions are generated through this process.

Theorem 3.3.3. *Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ be the iterates generated by Orthodir algorithm. Let $\mathbf{r}_{model}$ be a residual vector which corresponds to the iterate generated by EIEMLA. Then,*

$$||\mathbf{r}_{model}|| \leq (1 + |\alpha|)\, ||\mathbf{r}_k||\,. \tag{3.26}$$

Proof. Since we have proved that $\mathbf{x}_{model} \approx \mathbf{x}_{k+1}$, i.e. $\mathbf{x}_{model}$ is similar to the next iterate when Orthodir was run for k iterations, then we can now prove that

$$||\mathbf{r}_{k+1}|| \leq (1 + |\alpha|)\, ||\mathbf{r}_k||\,. \tag{3.27}$$

Using the expression of $\mathbf{x}_k$ given in (3.15), we have

$$\begin{aligned} ||\mathbf{r}_{k+1}|| &= ||\mathbf{b} - A\mathbf{x}_{k+1}|| \\ &= ||\mathbf{b} - A(\mathbf{x}_k - \alpha \mathbf{z}_k)|| \\ &= ||(\mathbf{b} - A\mathbf{x}_k) - A\alpha \mathbf{z}_k)|| \\ &= ||\mathbf{r}_k - A\alpha \mathbf{z}_k|| \\ &\leq ||\mathbf{r}_k|| + ||A\alpha \mathbf{z}_k|| \end{aligned} \tag{3.28}$$

So,

$$\begin{aligned}
\frac{||\mathbf{r}_{k+1}||}{||\mathbf{r}_k||} &\leq 1 + \frac{||A\alpha\mathbf{z}_k||}{||\mathbf{r}_k||} \\
&\leq 1 + \frac{\left\|A\alpha {P_k}^{(1)}(A)\mathbf{r}_0\right\|}{||P_k(A)\mathbf{r}_0||} \qquad (3.29) \\
&\leq 1 + ||\alpha A|| \frac{\left\|{P_k}^{(1)}(A)\mathbf{r}_0\right\|}{||P_k(A)\mathbf{r}_0||} \qquad (3.30) \\
&\leq 1 + ||\alpha A|| \frac{\max_{\lambda\in\sigma(A)} |\lambda|}{\max_{\lambda\in\sigma(A)} |\lambda|}, \\
&\leq 1 + ||\alpha A|| \\
&\leq 1 + |\alpha|\,\rho(A) \\
&\leq 1 + |\alpha| \quad \text{since } (\rho(A) < 1)
\end{aligned}$$

which proves the theorem. $\diamond$.

3.4 Numerical Results and Discussion

Systems of varying dimensions are solved starting from the initial guess vector $\mathbf{x}_0 = (0, 0, \ldots 0)^T$. We run a number of Lanczos-type algorithms for $k = 100$ and $k = 200$ iterations, with $k \leq n$. The choice of k is based on the observation that Lanczos-type algorithms generally fail after 200 iterations.

The general matrix used in Chapter 2, Section 2.2, is also used here for the case $\delta = 0.2$. EIEM in Orthodir, Orthomin, and A_{12} algorithms, is applied. The implementation of EIEM in A_8B_8 and Orthores algorithms can be seen in Appendix A. Note that Orthomin is a Lanczos-type algorithm which is based on A_5B_{10} formula. The derivation of formulae A_iB_j, for $i, j = 1, 2, \ldots, 10$, as well as their implementations can be seen in Chapter 1. These formulae were discovered by Baheux, [3]. Algorithm A_{12}, in addition, is a new implementation of the Lanczos process proposed by Farooq, [31]. This algorithm is based on formulae A_{12}.

The aim of this experiment is to examine the reduction in the residual norm of the iterate generated by EIEMLA, compared with the original Lanczos-type algorithms. The other aim is the possibility of optimizing approximate solution by increasing the number of iterations, from 100 to 200. All of the results are recorded and the residual norms are represented graphically.

3.4.1 The Embedded Interpolation and Extrapolation Model in Orthodir Algorithm

In this section, we will look at some results obtained with the implementation of embedding the interpolation and extrapolation model (EIEM) in Orthodir. They are recorded in Tables 3.1 and 3.2 for 100 and 200 iterations, respectively. Here, we compare the residual norm of the iterate $\mathbf{x}_k$, the minimum residual norm of iterate $\mathbf{x}_m$, where both are generated by Orthodir, and the minimum residual norm of the iterate generated by EIEM Orthodir. Note that we obtained a sequence of iterates after running EIEM Orthodir. The minimum residual norm is referred to as $||\mathbf{r}_{model}||$.

Overall, the approximate solutions generated by EIEM Orthodir are better than all of previous iterates generated by Orthodir alone. This is clearly visible in the decrease column of the table. For instance, when solving a problem of 1000 dimensions, the decrease is 8.3276; this means that the residual norm of the iterate generated by EIEM Orthodir algorithm is about 8 times smaller than the lowest residual norm of the iterates generated by Orthodir. Similarly, Orthodir with EIEM performs better than Orthodir when solving 7000 dimensional problems.

The residual norm of its solution is about 6 times smaller. The decrease factor remains stable at 3 when solving large scale problems i.e. between 20000 and 70000 dimensions. We notice here that breakdown still occurs in the Orthodir algorithm for some problems, such as in dimensions 20000 and 40000. This is indicated by *INF* in the residual norms column.

We captured the residual norms behaviour of EIEM in Orthodir algorithm for problems of dimensions ranging from 1000 to 70000 in Figure 3.5. These results are for a selection of problems only; others can be seen in Appendix A. Here, the red curve represents the residual norms of 100 iterates generated by Orthodir, whereas the blue curve represents the residual norms of the approximate solutions generated by EIEM Orthodir . Note, the blue curve does not start from iteration 1; instead, it begins from the iteration corresponding to the iterate with the lowest residual norm, as explained in Algorithm 10. In Figure 3.5(e), for instance, we only present few residual norms, whereas other figures have more than that. We can say here that, overall, the blue curve is consistently under the red curve, before it goes up considerably. It means that, EIEM Orthodir generates iterates which are than those generated by Orthodir.

Table 3.2 provides some results of the implementation of EIEM Orthodir for 200 iterations. Overall, Orthodir with EIEM improved the residual norms. In fact, for some problems, such as those in dimensions 5000, 7000, 8000, 9000, 10000, and 70000, the residual norms of the approximate solutions generated by EIEM Orthodir are even better than those in Table 3.1. The comparison between the residual norms of the iterates generated by EIEM Orthodir for 100 and 200

Table 3.1: *Comparison of the residual norms of the iterates generated by Orthodir and those generated by EIEM in Orthodir over 100 iterations*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$2.2003E+04$	7.0560	0.8473	8.3276	0.7566
2000	$4.7374E+02$	5.0727	1.8774	2.7019	1.3513
3000	$2.6217E+02$	17.0350	3.4173	4.9849	2.5198
4000	32.5071	7.3863	2.7530	2.6830	3.9224
5000	7.1466	7.1466	2.4345	2.9356	5.5196
6000	$6.8731E+02$	8.8374	2.2929	3.8543	7.3874
7000	2.0905	17.8229	6.7562	5.7562	7.8552
8000	$3.7821E+02$	9.0646	5.5186	1.6426	12.0198
9000	$2.2027E+03$	25.1725	18.8932	1.3324	14.8194
10000	4.8718	3.4108	1.4640	2.3298	18.1777
20000	*INF*	5.6270	1.8035	3.1200	63.9889
30000	$6.1941E+02$	34.1018	12.3647	2.7579	$3.5772E+02$
40000	*INF*	16.0039	5.8027	3.3323	$8.0827E+02$
50000	11.3670	8.2159	2.9005	2.8326	$1.765E+03$
60000	$1.0451E+02$	33.4570	10.6191	3.1506	$1.7263E+03$
70000	28.8041	13.1793	4.3908	3.0016	$2.179E+03$

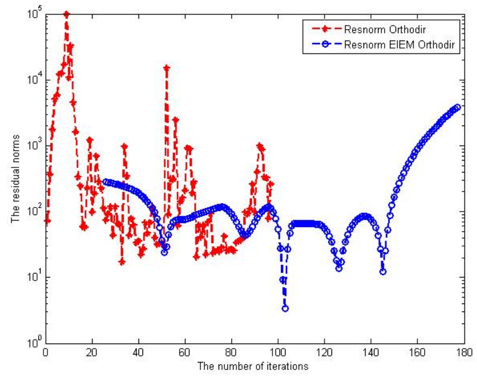
iterations is described in Table 3.3. Here, we computed the percentage decrease from 100 to 200 iterations. For instance, when solving a SLE of dimensions 7000, the percentage decrease is about 89.76% . It means that the residual norm of the iterate generated by EIEM Orthodir for 200 iterations, decreased by about 89.76% from the one generated by EIEM Orthodir for 100 iterations. The significant improvement also appears for dimensions 8000 and 9000, where their percentage decrease is about 50%. There are some problems for which there is no improvement in the residual norms for 100 to 200 iterations. A clear comparison is made in Figure 3.6.

Table 3.2: *Comparison of the residual norms of the iterates generated by Orthodir and those generated by EIEM in Orthodir over 200 iterations*

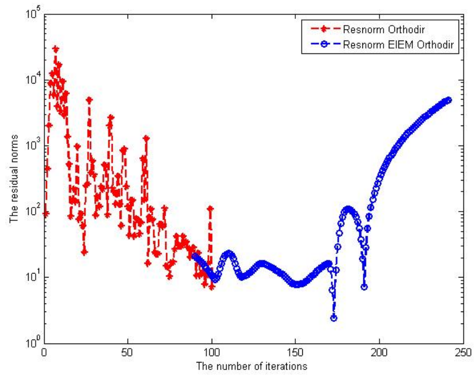
Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$4.6882E+09$	7.0560	0.8473	8.3276	0.5918
2000	$5.1039E+08$	5.0727	1.8774	2.7019	1.5781
3000	$2.6217E+02$	17.0350	3.4173	4.9849	2.4077
4000	$1.4087E+09$	7.3863	2.7530	2.683	4.8380
5000	88.3012	7.1466	1.9527	3.6598	6.08
6000	$5.3162E+08$	8.8374	2.2929	3.8543	9.2981
7000	*INF*	2.3138	0.6917	3.3451	10.7136
8000	$2.4089E+06$	9.0646	2.7315	3.3185	14.0900
9000	$1.5578E+10$	25.1725	9.3516	2.6918	19.871
10000	$8.6226E+04$	3.4108	1.2703	2.6850	25.0024
20000	*INF*	5.6270	1.8035	3.1200	64.4431
30000	$2.1509E+08$	34.1018	12.3647	2.7579	$4.8966E+02$
40000	*INF*	16.0039	5.8027	3.3323	$5.8288E+02$
50000	$1.5989E+05$	8.2159	2.9005	2.8326	$1.4261E+03$
60000	$5.0367E+02$	30.5078	10.7341	2.8421	$2.109E+03$
70000	$2.3107E+02$	12.9988	3.7122	3.5016	$4.7479E+03$

Table 3.3: *Comparison of the EIEM Orthodir performances over 100 and 200 iterations*

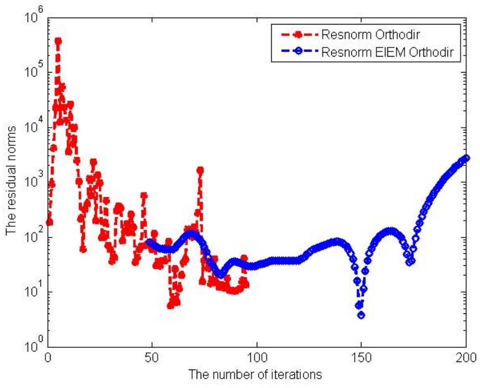
Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.8473	0.8473	0
2000	1.8774	1.8774	0
3000	3.4173	3.4173	0
4000	2.7530	2.7530	0
5000	2.4345	1.9527	19.8
6000	2.2929	2.2929	0
7000	6.7562	0.6917	89.76
8000	5.5186	2.7315	50.5
9000	18.8932	9.3516	50.5
10000	1.4640	1.2703	13.23
20000	1.8035	1.8035	0
30000	12.3647	12.3647	0
40000	5.8027	5.8027	0
50000	2.9005	2.9005	0
60000	10.7341	10.7341	0
70000	4.3908	3.712	15.46



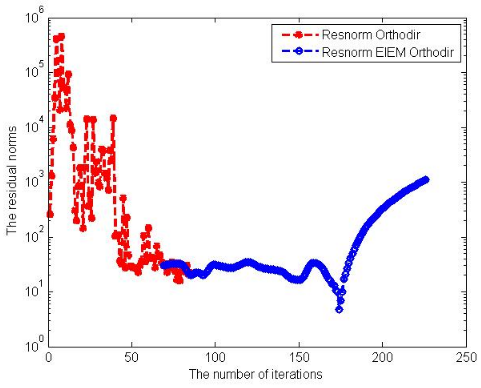
(a) Dim 3000



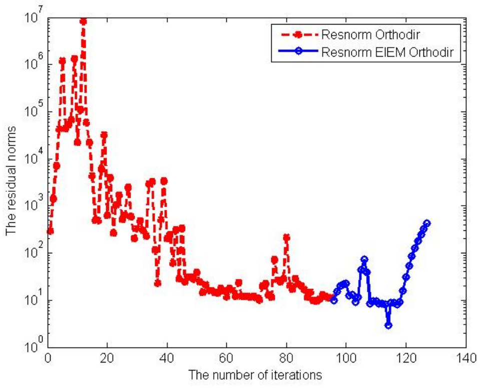
(b) Dim 5000



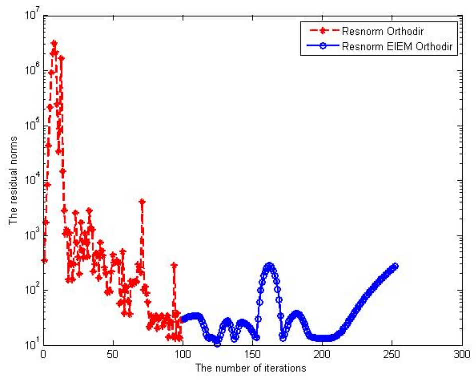
(c) Dim 20000



(d) Dim 40000



(e) Dim 50000



(f) Dim 70000

Figure 3.5: *The performance of EIEM in Orthodir on a variety of SLE's*

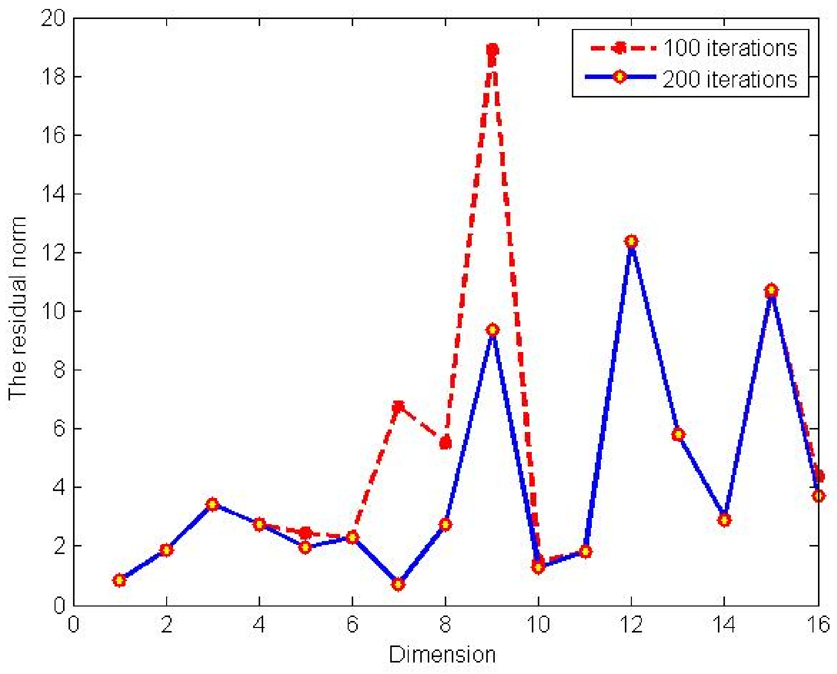


Figure 3.6: *The residual norms behaviour of the iterates generated by EIEM Orthodir for 100 and 200 iterations*

3.4.2 EIEM in Orthomin Algorithm

In this section, we will discuss some numerical results of running EIEM Orthomin for both 100 and 200 iterations. They are presented in Tables 3.4 and 3.5, respectively. Similar to the previous subsection, the residual norms of the iterates generated by EIEM Orthomin are also better than those of all the iterates generated by Orthodir. This improvement is measured as a decrease factor provided in both tables. It describes the ratio of the lowest residual norm of the iterate generated by Orthomin to the residual norm of the approximate solution generated by EIEM Orthomin. For instance, when solving problems of dimensions 5000 (see Table 3.4), the decrease factor shows 4.5668. It means that the residual norm of the iterate generated by EIEM Orthomin is about 5 times smaller than the lowest residual norm of the iterate generated by the original Orthomin. Furthermore, for

Table 3.4: *Comparison of the residual norms of the iterates generated by Orthomin and those generated by EIEM in Orthomin algorithm over 100 iterations*

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$4.2971E+02$	5.8802	2.5796	3.2801	1.1671
2000	8.9656	0.3085	0.1228	2.6380	2.5122
3000	53.6782	6.7353	1.9238	3.5010	5.0146
4000	27.4180	7.4770	3.2823	2.2780	8.4496
5000	13.8094	5.0350	1.1025	4.5668	12.7465
6000	$6.1534E+02$	6.8183	2.9931	2.2780	18.0645
7000	$1.4583E+02$	12.6500	5.2531	2.7783	15.9065
8000	59.8963	11.0986	4.5721	2.4274	31.5433
9000	11.0851	9.7388	1.5923	6.1162	39.7061
10000	56.4802	9.5349	1.8986	5.0221	42.6168
20000	$4.03780E+04$	16.6678	13.7344	2.6043	$1.7655E+02$
30000	$2.6099E+02$	74.4084	19.7072	3.7757	$2.7980E+03$
40000	$1.7379E+02$	57.4135	26.7537	2.1460	$2.4308E+03$
50000	$2.4956E+02$	2.4307	0.6062	4.0097	$1.1559E+04$
60000	$5.0489E+06$	102.3427	38.1449	2.6829	$7.5423E+03$
70000	64.8401	21.2823	7.9323	2.6829	$2.4591E+04$

problems of dimensions 9000, 10000, and 50000, the decrease factors are about 6, 5, and 4, respectively. Similarly, in Table 3.5, the decrease factors of problems of dimensions 10000 and 40000 are about 5 and 6, respectively, which indicate some improvements have been made by the EIEM Orthomin.

Enhanced EIEM Orthomin for solving SLEs is also made by increasing the iterations from 100 to 200. It is presented in Table 3.6 and is captured in Figure 3.7. As can be seen here, for a problem of dimensions 3000, for instance, the percentage decrease is about 67%. It means that increasing the iterations up to 200 leads an improvement of the residual norm. The significant improvement is visible when solving a problem of dimensions 8000, where the percentage decrease is about 99%. Other problems, in dimensions 30000, 40000, and 70000 dimensions, have the percentages decrease 74%, 53%, and 66%, respectively.

Table 3.5: *Comparison of the residual norms of the iterates generated by Orthomin and those generated by EIEM in Orthomin algorithm over 200 iterations*

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$4.2970E+02$	5.8802	1.7927	2.2795	1.2812
2000	92.1952	0.3085	0.1228	2.5287	3.997
3000	4.1797	1.2657	0.4446	2.8469	8.8176
4000	62.8293	7.4770	3.1823	2.3496	9.8983
5000	76.0691	2.4404	0.5713	4.2717	15.533
6000	65.6983	6.8183	2.5931	2.6294	31.1042
7000	$1.4583E+02$	12.6500	4.5531	2.4081	17.2935
8000	35.1760	0.0701	0.0254	2.7598	55.4165
9000	11.2883	3.8938	0.8661	4.4958	69.2796
10000	56.4801	9.5349	1.8986	5.0221	42.9135
20000	$4.0378E+04$	16.6678	6.4000	1.2136	$1.7589E+02$
30000	$5.4198E+02$	27.2705	5.1769	5.2677	$1.5216E+03$
40000	56.7181	77.1316	12.6267	6.1086	$2.5159E+03$
50000	59.1980	2.4307	0.6062	4.0097	$1.9875E+03$
60000	$2.2618E+09$	102.3427	38.1449	2.6829	$5.3279E+03$
70000	1.6062	1.5868	0.4914	3.2291	$8.5158E+03$

Table 3.6: *Comparison of EIEM Orthomin performances over 100 and 200 iterations*

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	2.579	1.7927	30.49
2000	0.1228	0.1228	0
3000	1.9238	0.4446	76.89
4000	3.2823	3.2823	0
5000	1.1025	0.5713	48.18
6000	2.9931	2.5931	13.36
7000	5.2531	4.5531	15.37
8000	4.5721	0.0254	99.06
9000	1.5923	0.8661	45.6
10000	1.8986	1.8986	0
20000	13.7344	6.4000	53.4
30000	19.7072	5.1769	73.73
40000	26.7537	12.6267	52.8
50000	0.6062	0.6062	0
60000	38.1449	38.14491	0
70000	7.9323	2.6831	66.17

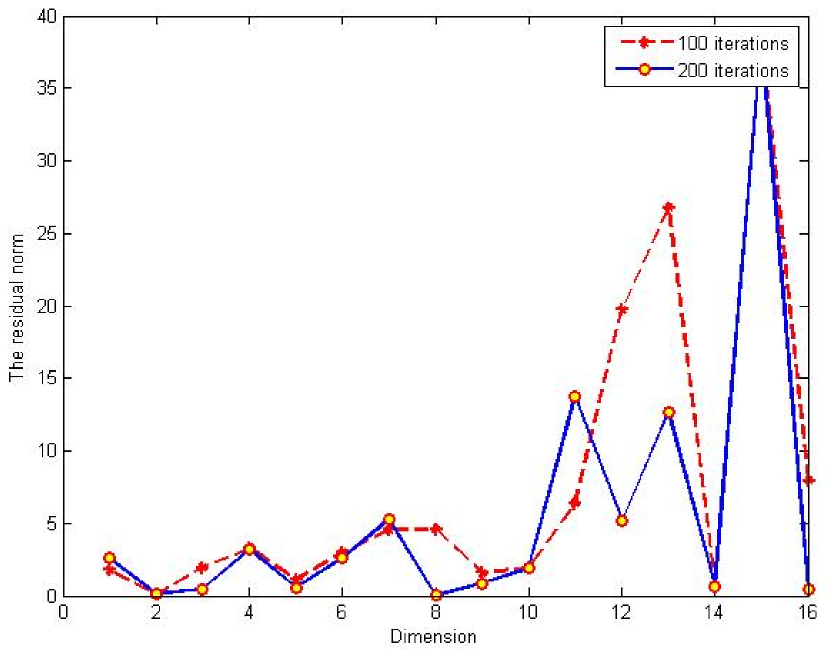


Figure 3.7: *The residual norms behaviour of the iterates generated by EIEM Orthomin over 100 and 200 iterations*

3.4.3 EIEM in A_{12} Algorithm

In this particular section, some numerical results obtained with EIEM A_{12} algorithm are provided in Tables 3.7 and 3.8. The first one presents some findings of the implementation of EIEM A_{12} for 100 iterations, whereas those for 200 iterations are described in the second table. Here, we will look at the decrease factor which represents the improvement factor made by EIEM A_{12}. We will also look at the percentage change of the algorithm from 100 to 200 iterations.

As can be seen in Table 3.7, overall, the residual norms of the iterates processed in EIEM A_{12} improved slightly from those generated by the original A_{12}. This is indicated by the decrease factors which show the value of three in most of problems. This trend also appears in Table 3.8. Our explanation of this fact is that, according to [31], A_{12} is a Lanczos-type algorithm which is more stable than

Table 3.7: *Comparison of the residual norms of the iterates generated by A_{12} and those generated by EIEM in A_{12} algorithm over 100 iterations*

Dim	A_{12}		A_{12} with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	1.0666	0.2077	0.0787	2.6391	0.6033
2000	0.1728	0.1500	0.0539	2.7829	2.0654
3000	0.4990	0.2386	0.0793	3.0088	3.5116
4000	1.9006	0.3920	0.1473	2.7434	5.394
5000	0.1894	0.1319	0.0488	3.2975	7.9718
6000	0.6113	0.3155	0.1128	2.7969	11.0885
7000	0.57347	0.3291	0.0872	3.7741	13.8402
8000	0.3459	0.2362	0.0786	3.0051	17.4355
9000	2.2946	0.4227	0.1466	2.8834	21.6374
10000	0.1613	0.1211	0.0359	3.3733	26.0479
20000	0.2352	0.2232	0.0856	2.6075	94.635
30000	1.1744	0.3562	0.1258	2.8315	$1.9610E+02$
40000	5.9105	0.1309	0.0596	2.1963	$1.8112E+03$
50000	0.5109	0.2461	0.0839	2.9333	$3.8659E+03$
60000	1.1011	0.7062	0.2652	2.6629	$7.0075E+03$
70000	1.6178	1.6178	0.6232	2.5959	$9.4073E+03$

Orthodir, Orthomin, and other Lanczos-type algorithms discussed in [3]. In other words, the approximate solutions generated by A_{12} are consistently closer to the true solution than those generated by Orthodir and Orthomin. This, therefore, makes EIEM A_{12} unable to improve on the residual norms significantly.

The percentage decrease of some residual norms of the approximate solutions generated by EIEM in A_{12} as a result of increasing iterations from 100 to 200 is described in Table 3.9. It is also illustrated in Figure 3.8. Although the percentage change is made in most of problems, it is not significant. For instance, the highest percentage decrease was made when solving 1000 dimensions, which was about 51%. In contrast, the lowest of the percentage decrease was made when solving 2000 dimension problems, which was about 2% only. Other problems have their percentage decrease between these values.

Table 3.8: *Comparison of the residual norms of the iterates generated by A_{12} and those generated by EIEM in A_{12} algorithm over 200 iterations*

Dim	A_{12}		A_{12} with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	0.1617	0.1027	0.0389	2.64010	0.775
2000	0.1497	0.1472	0.0535	2.7514	2.1049
3000	0.2259	0.1575	0.0590	2.6695	3.9526
4000	0.3745	0.2420	0.1420	1.5475	6.4392
5000	0.9537	0.1022	0.0400	2.0943	9.24
6000	0.39619	0.3155	0.1128	2.7969	12.296
7000	0.2134	0.1589	0.0502	3.1653	16.4137
8000	0.1866	0.1582	0.0574	2.7561	21.3164
9000	0.6881	0.4227	0.1466	2.8834	26.1331
10000	0.1731	0.1211	0.0359	3.3733	31.1463
20000	0.3791	0.1808	0.0579	3.1226	$1.1765E+02$
30000	6.7972	0.3008	0.1108	2.7148	$2.4268E+02$
40000	0.5451	0.1309	0.0396	3.3056	$1.7166E+03$
50000	0.3696	0.2293	0.0779	2.9435	$6.2089E+03$
60000	0.4791	0.4329	0.1466	2.9529	$5.2633E+03$
70000	2.9535	1.5841	0.5487	2.8870	$9.237E+03$

Table 3.9: *Comparison of the EIEM A_{12} performances over 100 and 200 iterations*

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.0787	0.0389	50.57
2000	0.0539	0.0535	0.74
3000	0.0793	0.0590	25.6
4000	0.1473	0.1420	3.6
5000	0.0488	0.0400	22
6000	0.1128	0.1128	0
7000	0.0872	0.0502	42.43
8000	0.0786	0.0574	26.97
9000	0.1466	0.1466	0
10000	0.0359	0.0359	0
20000	0.0856	0.0579	32.36
30000	0.1258	0.1108	11.92
40000	0.0596	0.0596	0
50000	0.0839	0.0779	7.15
60000	0.2652	0.1466	44.72
70000	0.6232	0.5487	11.95

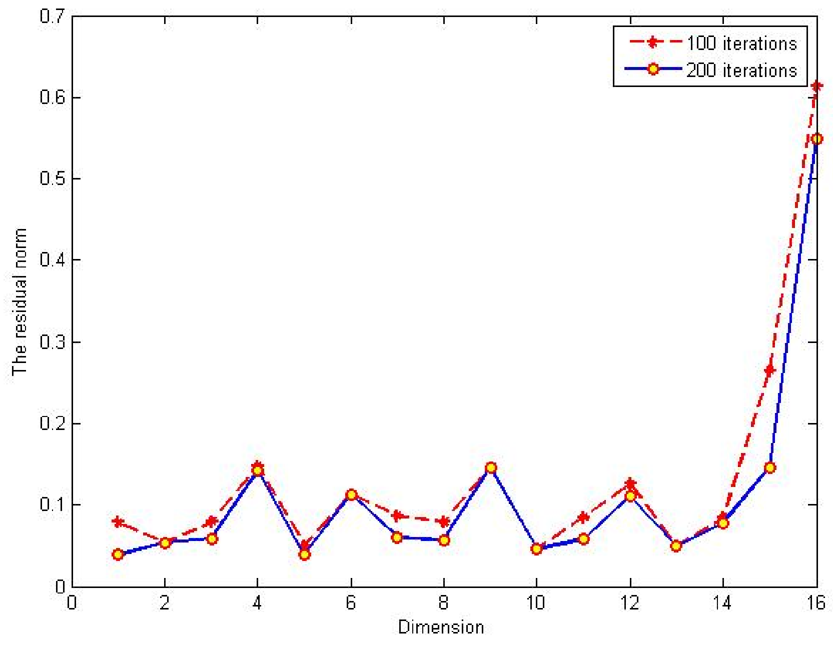


Figure 3.8: *The behaviour of residual norms of the iterates generated by EIEM A_{12} over 100 and 200 iterations*

3.4.4 The Minimal Polynomial Extrapolation and EIEMLA

The technology for extrapolating sequences of iterates generated by some iterative process is well developed. The work of [16, 75] and others is testimony to that. Although the context is different in that they assume that only the sequence of iterates, i.e. the approximate vector solutions are known, (A, $\mathbf{b}$ and the iterative process itself are assumed unknown), ultimately they are looking for better approximate solutions using extrapolation. For completeness, therefore, we compare here the Minimal Polynomial Extrapolation (MPE) approach, one of the most popular such approaches, against our own EIEMLA when the generating iterative process is Orthodir. For the benefit of the reader, we recall briefly MPE which is derived using the differences between every two consecutive vector solutions. We follow the derivation of MPE given in [37, 77] to solve SLE's.

Since we only have the sequence of iterates $\mathbf{x}_j, j = 0, 1, ...,$ we consider the system

$$\mathbf{x}_{j+1} = A\mathbf{x}_j, \tag{3.31}$$

for $j = 0, 1, \ldots$ Write

$$\begin{aligned} \mathbf{u}_j &= \Delta\mathbf{x}_j = \mathbf{x}_{j+1} - \mathbf{x}_j, \\ \mathbf{v}_j &= \Delta^2\mathbf{x}_j = \Delta\mathbf{u}_j = \mathbf{u}_{j+1} - \mathbf{u}_j. \end{aligned} \tag{3.32}$$

For a fixed integer k, we define matrices whose columns are the vectors of differences, [75],

$$\begin{aligned} U &= [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{k-1}], \\ V &= [\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{k-1}]. \end{aligned} \tag{3.33}$$

We define vector $\mathbf{c} = [c_0, c_1, \ldots, c_{k-1}, 1]'$ by

$$\mathbf{c} = \begin{bmatrix} -U^{+}\mathbf{u}_k \\ 1 \end{bmatrix} \tag{3.34}$$

The solution of system 3.31, $\mathbf{s}$, is obtained by

$$\mathbf{s} = X\mathbf{c}/d, \tag{3.35}$$

where $X = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k]$, and $d = \mathbf{1}'_{k+1}\mathbf{c}$, where $\mathbf{1}$ is a vector of 1's. Here $\mathbf{s}$ is a weighted average of the $\mathbf{x}_j$'s, with weights determined by the coefficients of the minimal polynomial $P(\lambda)$ of A with respect to $\mathbf{u}_0$, [75], i.e. the unique monic polynomial of least degree such that

$$P(A)\mathbf{u}_0 = 0. \tag{3.36}$$

Table 3.10: *Residual norms of iterates generated by MPE and those generated by EIEM in Orthodir over 100 iterations; random x*

Dimension	Orthodir	MPE in Orthodir	EIEM in Orthodir
n	$\|\|\mathbf{r}_{ortho}\|\|$	$\|\|\mathbf{r}_{MPE}\|\|$	$\|\|\mathbf{r}_{EIEM}\|\|$
1000	$2.2003E+04$	$4.0460E+02$	7.0560
2000	$4.7374E+02$	$6.1937E+04$	1.8774
3000	$2.6217E+02$	$9.1533E+02$	3.4173
4000	32.5071	$2.0483E+02$	2.7530
5000	7.1466	$5.3636E+02$	2.4345
6000	$6.8731E+02$	$1.0545E+04$	2.2929
7000	2.0905	$1.8810E+03$	6.7562
8000	8.8099	$1.5891E+03$	5.5186
9000	5.6547	62.1401	0.1466
10000	$4.2341E+02$	$3.1794E+03$	7.7580
20000	*INF*	$7.1836E+03$	1.8035
30000	$6.1941E+02$	$2.9073E+02$	12.3647
40000	*INF*	$4.0582E+02$	5.8027
50000	11.3670	$9.77997E+02$	2.9005
60000	$1.0451E+02$	$3.1176E+05$	10.6191
70000	28.8041	$7.1713E+02$	4.3908

As said earlier, MPE generates new points which converge, under certain conditions to the true solution in the limit. Here, we compare it to EIEMLA on several problems with $\mathbf{b} = A\mathbf{x}$, where (i) the entries of $\mathbf{x}$ are chosen randomly between 0 and 1, and (ii) $\mathbf{x} = [11\ldots1]'$.

Results are recorded in Tables 3.10 and 3.11. As can be seen, EIEMLA consistently found the solution, in both cases, with smaller residual norms than MPE.

3.5 Summary

We have introduced the EIEMLA method for solving SLE's as a novel strategy to avoid breakdown in Lanczos-type algorithms. It takes advantage of the situation where the residual norms of the iterates generated by Lanczos-type algorithms fluctuate. The sequence of entries of some of these iterates, which are normally

Table 3.11: *Residual norms of iterates generated by MPE and those generated by EIEM in Orthodir over 100 iterations;* $\mathbf{x} = [1, 1, \ldots, 1]'$

Dimension	Orthodir	MPE in Orthodir	EIEM in Orthodir
n	$\|\|\mathbf{r}_{ortho}\|\|$	$\|\|\mathbf{r}_{MPE}\|\|$	$\|\|\mathbf{r}_{EIEM}\|\|$
1000	0.7822	7.3761	0.7306
2000	0.5420	46.7904	0.3171
3000	$2.1440E + 12$	$1.6771E + 12$	0.3631
4000	0.7903	9.9223	0.1453
5000	0.9815	20.5103	0.4604
6000	2.2391	88.8236	0.7925
7000	$4.4812E + 03$	$3.0922E + 02$	0.5479
8000	7.4063	5.1315	0.5634
9000	2.8746	57.4447	0.8081
10000	0.7822	7.3761	0.7306
20000	7.3996	6.4415	0.2816
30000	$1.3825E + 02$	$8.1974E + 02$	0.4201
40000	2.8530	4.3648	0.9119
50000	7.7647	12.9841	0.6299
60000	$3.0733E + 04$	$5.9747E + 04$	0.7303
70000	0.4211	20.9988	0.3394

associated with a small residual norm, form a pattern which leads to a model function. Embedding this function into the Lanczos-type algorithm used yields, through interpolation and extrapolation, a better approximate solution than all of iterates generated by the original algorithms.

We have shown theoretically that the residual norm of the approximate solution generated by EIEMLA is always smaller than all of the previous iterates generated by the Lanczos-type algorithms used. This has been confirmed empirically by solving SLE's of dimensions ranging from 1000 to 70000. Further, we have concluded that, although increasing the number of the iterations from 100 to 200, potentially causes breakdown in Lanczos-type algorithms, frequently, EIEMLA produces a better approximate solution. Moreover, for completeness, we have looked briefly at the so called Minimum Polynomial Extrapolation approach which builds up new approximate solutions to SLE's from sequences of iterates generated by some iterative process. MPE is then pitched against EIEMLA on a number of problems of varying dimensions. EIEMLA comes on top.

It is to note that the length of the sequence to use by EIEMLA and indeed other approaches such as MPE is arbitrary. This suggests a further investigation to find the optimum number of iterations before breakdown occurs, in our case when Lanczos-type algorithms are used.

The implementation of EIEM into a stable Lanczos-type algorithm does not make a significant improvement. This warrants more investigations, particularly since there are other regression approaches that can be tried.

Chapter 4

Restarting Lanczos-type Algorithms from the Iterate Generated by EIEMLA

4.1 Introduction

In Chapter 3, we have shown that EIEMLA enables to capture the properties of the sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ of iterates generated by a Lanczos-type algorithm by regressing upon this sequence of points. The regression model found is then used to generate some points, $\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, \ldots, \mathbf{x}_s$, where $s \geq > k+1$, which have better residual norms than any other points in the previous sequence. However, because we assume that the regression model captures the properties of the Lanczos sequence, these points belong to that sequence.

Although $\min\{||\mathbf{r}_l||\} \leq ||\mathbf{r}_k||$, where $l = k+1, k+2, \ldots, s$, the minimum residual norm of $\mathbf{x}_l$ is still not low enough to indicate convergence. This is obvious since, Lanczos-type algorithms typically take more than n iterations to converge, where n is the dimensions of the problems. Here obviously we assume $n >> 100$ or k.

In this chapter, we investigate using one of these new points as a restarting point for a Lanczos-type algorithm. There are some new points generated by

EIEMLA that can be used as a restarting point, however, we use the better one, called $\mathbf{x}_{model}$, i.e. the one with the minimum residual norm, to get a better result. This point is then used to generate a finite sequence of points, say again a 100. The regression is applied again and a new point is generated. This process continues until a stopping criterion is satisfied.

4.2 Restarting EIEMLA

As explained in Chapter 2, restarting a Lanczos algorithm requires an iterate to restart with. The iterate can be obtained either directly by the Lanczos algorithm itself, such as RLLastIt and RLMinRes (see Subsections 2.1.3 and 2.1.4), or by modifying a sequence generated by the algorithm, such as RLMedVal (see Subsection 2.1.5). In this particular restarting, called REIEMLA, we take the iterate generated by EIEMLA as a starting point. It is as illustrated in Figure 4.1. First, a Lanczos-type algorithm generates the sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$, with assuming it breaks after k iterations. The sequence is then regressed to get a model function. Using this function, we generate a solution, ${\mathbf{x}_{model}}^{(1)}$. Next, we restart the Lanczos-type algorithm from this solution to get another sequence of iterates. We regress again to get ${\mathbf{x}_{model}}^{(2)}$. It is continued until ${\mathbf{x}_{model}}^{(m)}$ is achieved, and the corresponding residual norm is less than the given tolerance.

By Theorem 2.1.1 of Chapter 2, we know that the restarting framework allows Lanczos-type algorithms to generate better iterates. This is expected here, too. This restarting approach is described as Algorithm 11.

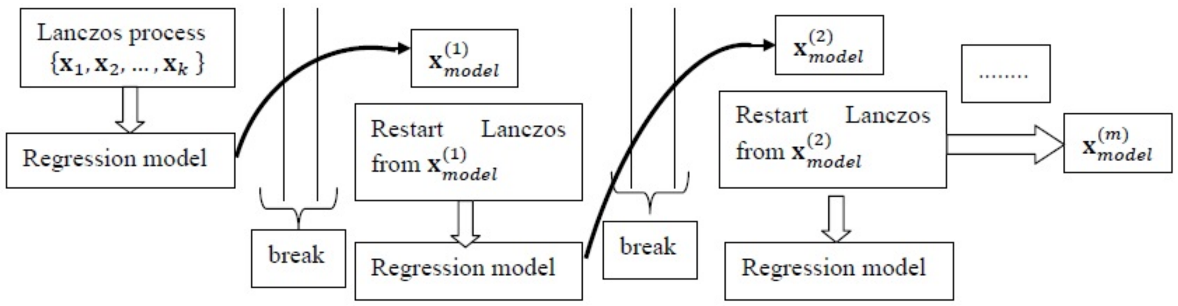

Lanczos process $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$
Regression model
break
$\mathbf{x}^{(1)}_{model}$
Restart Lanczos from $\mathbf{x}^{(1)}_{model}$
Regression model
break
$\mathbf{x}^{(2)}_{model}$
Restart Lanczos from $\mathbf{x}^{(2)}_{model}$
Regression model
....
$\mathbf{x}^{(m)}_{model}$

Figure 4.1: *The process of REIEMLA on SLE's*

Algorithm 11 REIEMLA

```
1: Initialization.    Choose $\mathbf{x}_0$ and $\mathbf{y}$. Set $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{y}_0 = \mathbf{y}$, and $\mathbf{z}_0 = \mathbf{r}_0$.
2: Fix the number of iterations to, say $k$, and the tolerance, $\epsilon$, to $1E - 13$.
3: Run EIEMLA for $k$ iterations. Obtain a sequence of iterates $\{\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, \ldots, \mathbf{x}_s\}$,
   where $s \geq> k + 1$, and calculate the residual norms of these iterates.
4: Compute the minimum of the residual norms, name it as $||\mathbf{r}_{model}||$.
5: if $||\mathbf{r}_{model}|| \leq \epsilon$ then
6:     The solution is obtained, i.e. the iterate which is associated with this residual
       norm, name it as $\mathbf{x}_{model}$.
7:     Stop.
8: else
9:     Initialize the algorithm with
           $\mathbf{x}_0 = \mathbf{x}_{model}$
           $\mathbf{y} = \mathbf{b} - A\mathbf{x}_0$
10:    Go to 3.
11: end if
12: Take $\mathbf{x}_{model}$ as the approximate solution.
13: Stop.
```

4.3 Numerical Results Using Sparse Matrix Algebra

Experiments have been carried out using five implementations of REIEMLA, including restarting EIEM (REIEM) Orthores, REIEM Orthodir, REIEM Orthomin, REIEM A_8B_8, and REIEM A_{12}. The aim is to look at the performance of these EIEMLA's when they are put on the restarting framework. Particularly, we will look at the robustness and the efficiency of each algorithm. The problems solved range from 1000 to 400000 variables. We use sparse matrix technology which is built in Matlab. Recall that a sparse matrix is one with most of its entries being zero, [78].

In this experiment, the block matrix A is modified with the sparse technique (see Appendix B). The values of δ are chosen as 0.2 and 5 only. These values lead to problems with matrices having large and small condition numbers. We use a cycle of 100 iterations to restart. In the following, we will discuss the results on problems for particular values of δ.

4.3.1 REIEMLA on SLE's with $\delta = 0.2$

The results of this particular case are recorded in Table 4.1 and some residual norms are captured in Figure 4.2, 4.3, and 4.4.

Overall, REIEM A_8B_8 found more accurate approximate solutions. This can be seen in Table (4.1), particularly for dimensions between 60000 and 400000. It also showed the best performance in term of efficiency; it consistently took the shortest time on all problems. The second best performance came from REIEM Orthodir. The rest of methods had mixed performances on both accuracy and efficiency.

Figures 4.2, 4.3, 4.4, and 4.5 represent the residual norms of all solutions generated by all considered algorithms for dimensions ranging from 1000 to 8000, 9000 to 70000, and 80000 to 400000, respectively. We can see there, that most of figures show that the red, blue, pink, and yellow curves, corresponding to REIEM Orthodir, REIEM Orthores, REIEM A_8B_8, and REIEM A_{12}, respectively, have a similar shape. The green curve, on the other hand, which represents REIEM Orthomin, appears on top of the other curves for some problems, such as on Figures from 4.4(c) to 4.5(d). It means that it failed to reach the prescribed convergence tolerance.

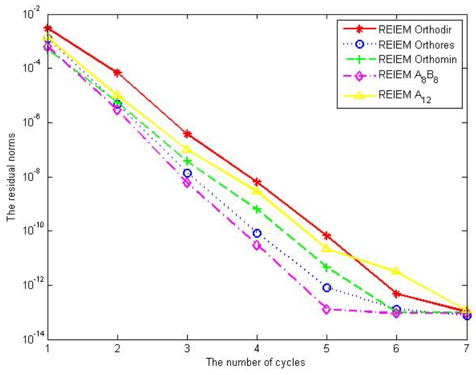
Table 4.1: *REIEMLA's results on SLE's of different dimensions ($\delta = 0.2$).*

Dim	REIEM Orthodir			REIEM Orthores			REIEM Orthomin			REIEM A_8B_8			REIEM A_{12}		
n	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*
1000	$1.0429E-13$	2.7754	7	$7.4768E-14$	2.7204	7	$1.0055E-13$	2.4289	7	$8.8728E-14$	2.3128	7	$1.1678E-13$	2.8407	7
2000	$1.8022E-13$	4.7902	5	$1.3016E-13$	4.7256	5	$9.7620E-14$	4.7388	5	$8.9780E-14$	4.5694	5	$2.6154E-13$	4.8302	5
3000	$9.5558E-14$	8.2847	6	$1.9059E-13$	8.1635	6	$1.8697E-13$	8.2978	6	$1.3804E-13$	7.8779	6	$6.6643E-13$	8.3374	6
4000	$1.5314E-13$	11.1859	6	$9.0616E-13$	11.0619	6	$1.5536E-12$	11.2805	6	$8.7363E-14$	10.6054	6	$2.4647E-13$	11.2323	6
5000	$8.8714E-14$	13.8305	6	$1.7117E-13$	13.6183	6	$1.1540E-13$	13.6849	6	$8.9895E-14$	13.3218	6	$1.5963E-13$	13.9249	6
6000	$9.3580E-14$	16.6379	6	$2.5422E-13$	16.4647	6	$1.3951E-13$	16.5377	6	$1.1735E-13$	14.9176	6	$4.5703E-13$	16.6322	6
7000	$8.8991E-14$	21.5419	7	$9.4050E-14$	21.2915	7	$8.7140E-14$	21.4752	7	$1.0280E-13$	16.4627	6	$1.1965E-13$	21.7140	7
8000	$7.0246E-14$	26.1215	7	$1.7535E-13$	26.6733	7	$9.9997E-14$	27.6451	7	$1.0766E-13$	24.4211	7	$4.0340E-13$	28.7450	7
9000	$1.2724E-13$	34.7815	6	$2.3651E-13$	34.5213	6	$2.7370E-13$	36.4677	6	$9.3160E-14$	32.6551	6	$1.3555E-13$	34.5217	6
10000	$1.1770E-13$	43.3406	7	$1.8943E-13$	42.9775	7	$8.5523E-14$	42.2791	7	$1.1804E-13$	36.6519	7	$1.3061E-13$	43.7932	7
20000	$1.3307E-13$	78.4202	6	$2.5346E-13$	75.5837	6	$1.3043E-13$	76.6608	6	$9.6495E-14$	74.1202	6	$2.1849E-13$	77.1374	6
30000	$7.0624E-14$	$1.3807E+02$	8	$1.3414E-13$	$1.3828E+02$	8	$1.2435E-13$	$1.1271E+02$	8	$1.3085E-13$	$1.2985E+02$	8	$8.3067E-14$	$1.4636E+02$	8
40000	$8.7169E-14$	$1.6587E+02$	7	$2.0691E-13$	$1.6405E+02$	7	$1.1421E-13$	$1.6513E+02$	7	$1.5730E-13$	$1.2543E+02$	7	$1.3643E-13$	$1.6427E+02$	7
50000	$8.8856E-14$	$2.0350E+02$	7	$1.9407E-13$	$2.0388E+02$	7	$1.1142E-13$	$2.0585E+02$	7	$1.1251E-13$	$1.7703E+02$	7	$1.4125E-13$	$2.0489E+02$	7
60000	$1.0238E-13$	$2.2516E+03$	6	$2.5612E-13$	$2.2548E+03$	6	$5.7458E-13$	$1.8015E+03$	6	$5.7665E-14$	$1.7265E+03$	6	$1.0136E-13$	$2.6891E+03$	6
70000	$1.1291E-13$	$2.3504E+03$	6	$2.5633E-13$	$2.3568E+03$	6	$2.3774E-07$	$2.8314E+03$	6	$5.4051E-14$	$2.3329E+03$	6	$8.5146E-12$	$9.6349E+02$	6
80000	$1.1291E-13$	$3.7782E+02$	6	$5.5238E-13$	$3.7811E+02$	6	$9.5690E-09$	$3.7915E+02$	6	$6.9059E-14$	$3.8115E+02$	6	$2.5610E-13$	$3.7799E+02$	6
90000	$1.2560E-13$	$4.1922E+02$	7	$3.5634E-13$	$4.1968E+02$	7	4.4348	$4.2066E+02$	6	$5.5320E-14$	$4.2106E+02$	7	$4.6976E-11$	$4.2059E+02$	7
100000	$6.7572E-14$	$1.9632E+03$	6	$2.7009E-13$	$1.9678E+03$	6	$2.2545E-07$	$2.3319E+03$	6	$1.8846E-14$	$2.0884E+03$	6	$2.0253E-11$	$1.7952E+03$	6
200000	$1.5334E-13$	$3.0293E+03$	6	$2.4225E-12$	$3.0356E+03$	6	$6.3091E-09$	$2.8280E+03$	6	$1.8846E-14$	$2.7299E+03$	6	$1.8905E-13$	$3.1738E+03$	6
300000	$3.5587E-13$	$1.4159E+03$	6	$1.3904E-11$	$1.3917E+03$	6	$2.9327E-08$	$1.4026E+03$	6	$9.1562E-14$	$1.4166E+03$	6	$3.4152E-13$	$1.4245E+03$	6
400000	$1.0917E-13$	$3.0293E+03$	7	$2.8929E-13$	$2.1188E+03$	7	0.9752	$2.1174E+03$	7	$6.3743E-14$	$2.0909E+03$	7	$1.7095E-11$	$2.1298E+03$	7

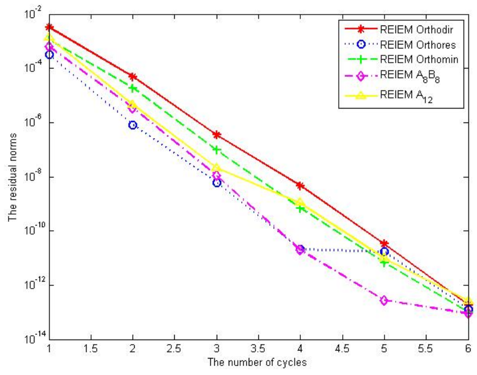
4.3.2 REIEMLA on SLE's with $\delta = 5$

Slightly different from the previous case, according to the information available in Table 4.2, REIEM A_{12} produced better approximate solutions than the others, for all dimensions from 1000 to 50000. But, it took similar computation time to the others. For large dimensions, as usual, REIEM A_8B_8 was the best, in terms of both accuracy and efficiency. REIEM Orthomin, interestingly, performed better this time than in the previous case. Although it could not meet the prescribed tolerance, it produced approximate solutions with residual norms which are mostly $1E-13$. REIEM Orthodir and REIEM Orthores were similar in both accuracy and computation time.

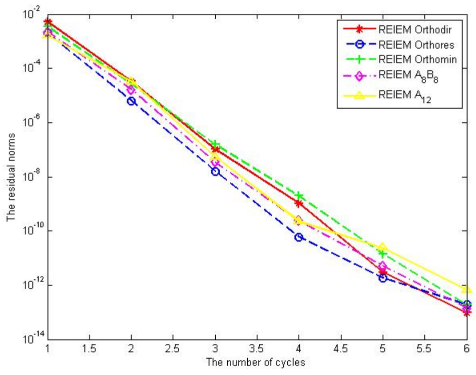
The behaviour of the residual norms of the five algorithms can be seen in Figures 4.6, 4.7, and 4.8. The trend of each curve in this case is different from that of the previous case, particularly for the behaviour of REIEM Orthores and REIEM A_{12}. For instance, in Figures 4.7(f), 4.8(a), and 4.8(b), the blue (REIEM Orthores) and yellow (REIEM A_{12}) curves appear on top. Both curves have similar trend, namely go down at the beginning of restarting, then climb up before falling down again to hit the tolerance. Nevertheless, in Figures 4.9(b), 4.9(c), and 4.9(d), they seem to fluctuate at the end of the restarting process. The green curve (REIEM Orthomin), as usual, is visible at the top of Figures 4.8(c), 4.8(d), 4.8(e), 4.8(f), and 4.9(a).



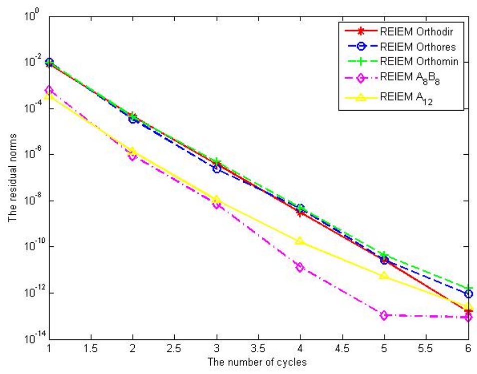
(a) dimensions 1000



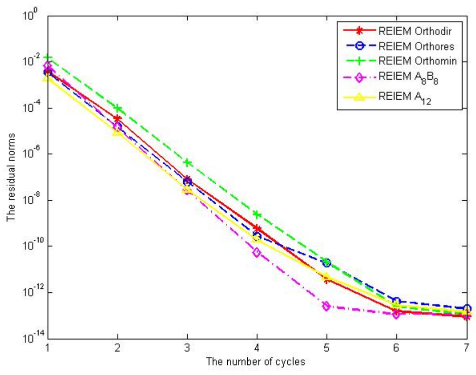
(b) dimensions 2000



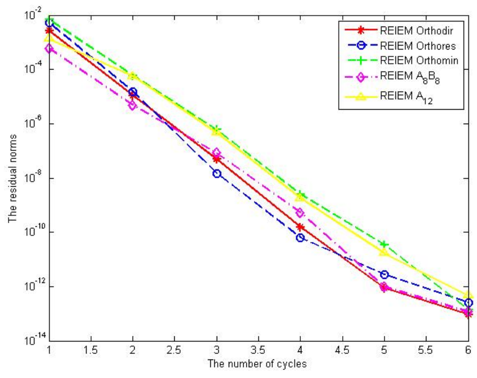
(c) dimensions 3000



(d) dimensions 4000

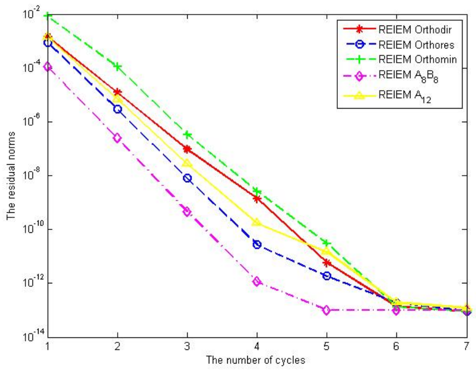


(e) dimensions 5000

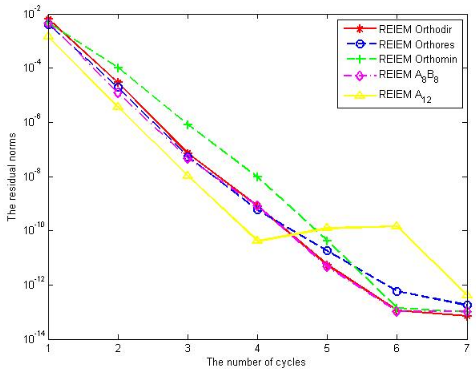


(f) dimensions 6000

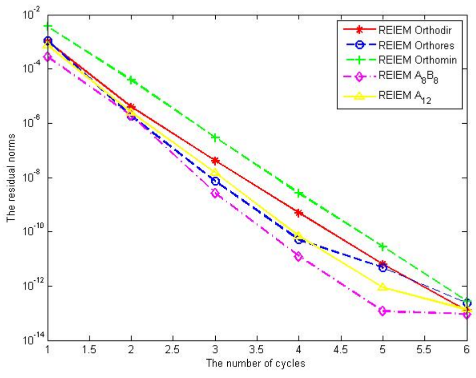
Figure 4.2: *The performance of REIEMLA's for the case of* $\delta = 0.2$, *dimensions 1000 to 6000*



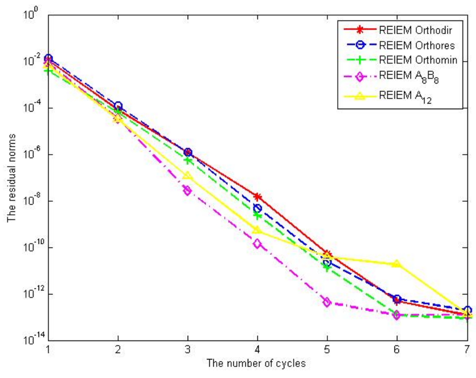
(a) dimensions 7000



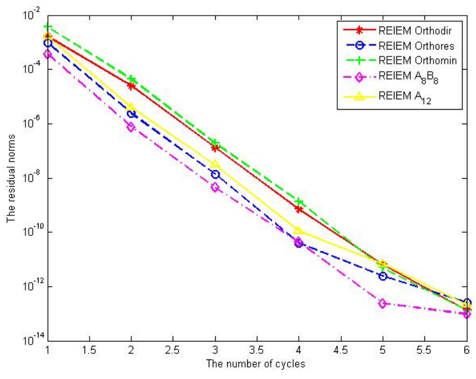
(b) dimensions 8000



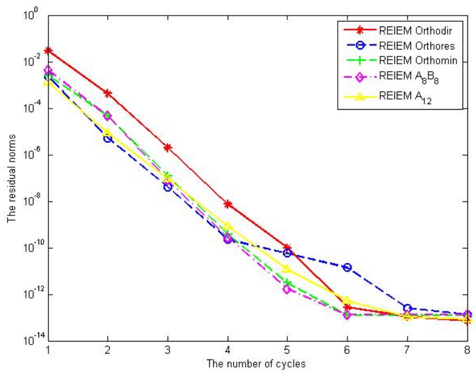
(c) dimensions 9000



(d) dimensions 10000

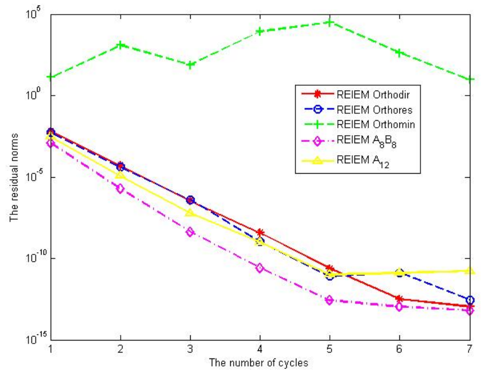


(e) dimensions 20000

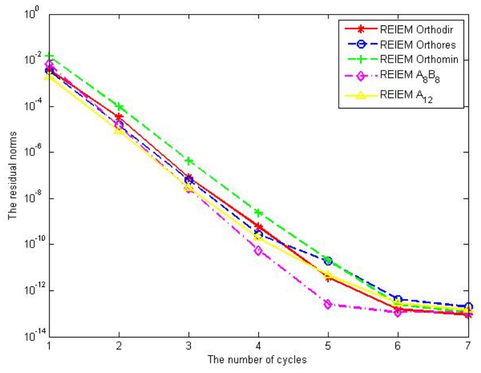


(f) dimensions 30000

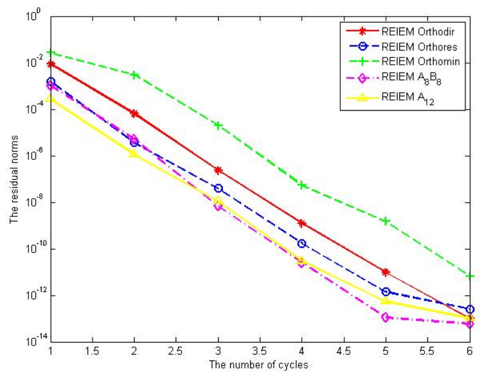
Figure 4.3: *The performance of REIEMLA's for the case of* $\delta = 0.2$, *dimensions 7000 to 30000*



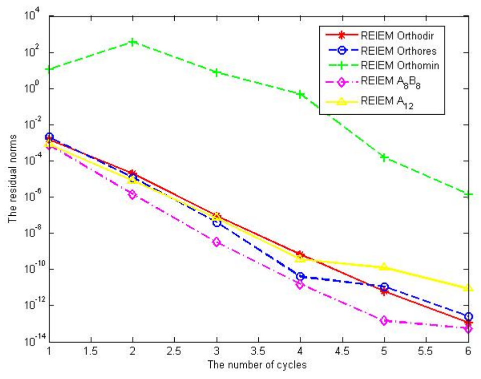
(a) dimensions 40000



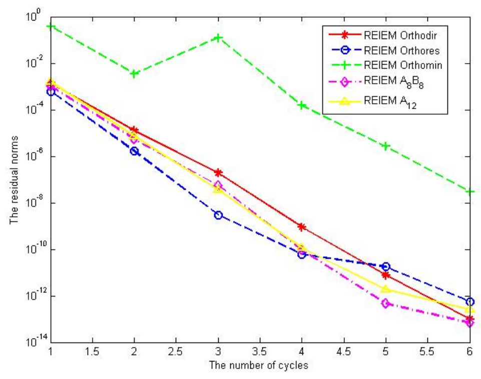
(b) dimensions 50000



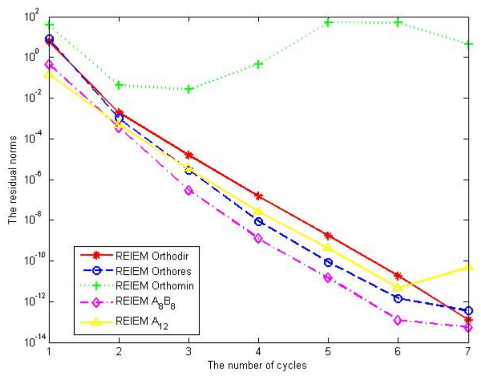
(c) dimensions 60000



(d) dimensions 70000

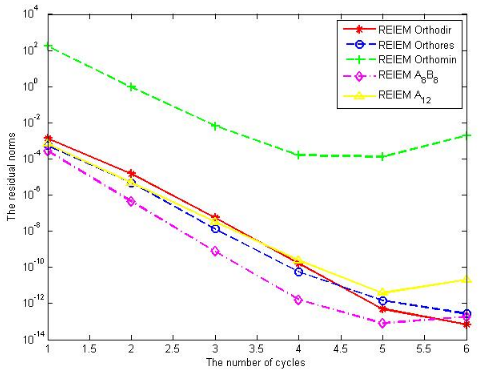


(e) dimensions 80000

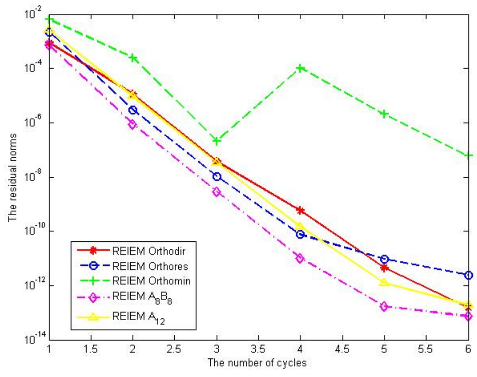


(f) dimensions 90000

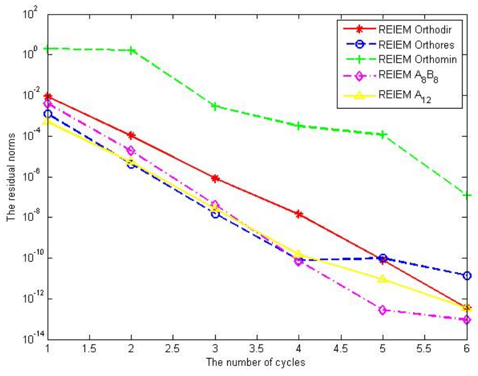
Figure 4.4: *The performance of REIEMLA's for the case of* $\delta = 0.2$*, dimensions 40000 to 90000*



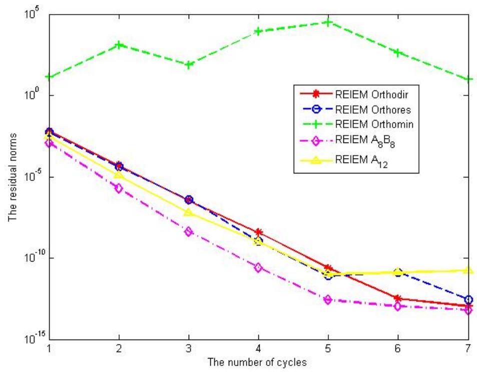
(a) dimensions 100000



(b) dimensions 200000



(c) dimensions 300000



(d) dimensions 400000

Figure 4.5: *The performance of REIEMLA's for the case of* $\delta = 0.2$*, dimensions 100000 to 400000*

Table 4.2: *REIEMLA's results on SLE's of different dimensions ($\delta = 5$)*

Dim	REIEM Orthodir			REIEM Orthores			REIEM Orthomin			REIEM A_8B_8			REIEM A_{12}		
n	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*
1000	$9.2205E-14$	3.6807	3	$7.0954E-13$	2.3924	3	0.1097	2.3823	3	$1.0439E-13$	2.2722	3	$6.2280E-14$	2.7175	3
2000	$1.3970E-13$	4.6865	7	$1.5625E-13$	5.9877	7	$1.3651E-13$	4.6538	7	$1.3778E-13$	4.5001	7	$8.5383E-14$	6.0370	7
3000	$1.4108E-13$	7.3545	8	$1.1046E-13$	9.0448	8	$1.1918E-13$	7.0775	8	$1.0682E-13$	7.7034	8	$9.1630E-14$	10.0885	8
4000	$1.0034E-13$	9.5333	8	$1.3915E-13$	10.5893	8	$1.3337E-13$	9.4175	8	$1.0416E-13$	10.3267	8	$9.8464E-14$	13.4137	8
5000	$1.7616E-13$	12.0375	9	$1.3020E-13$	14.8674	9	$1.3468E-13$	11.8063	9	$1.0835E-13$	11.379	9	$9.1978E-14$	18.0339	9
6000	$1.8162E-13$	14.5665	8	$1.0912E-13$	19.8708	8	$1.4272E-13$	14.2930	8	$1.3804E-13$	13.8562	8	$8.7310E-14$	20.1106	8
7000	$1.1937E-13$	16.5713	8	$1.2771E-13$	18.6329	8	$1.3908E-13$	16.4030	8	$1.0015E-13$	18.1232	8	$8.0583E-14$	22.9056	8
8000	$1.5750E-13$	18.9288	9	$1.3732E-13$	26.4188	9	$1.3564E-13$	18.8584	9	$1.2937E-13$	18.1689	9	$9.7520E-14$	28.9404	9
9000	$1.3838E-13$	21.4294	9	$1.1833E-13$	26.8324	9	$1.0973E-13$	23.9670	9	$1.4105E-13$	20.6244	9	$8.1333E-14$	32.8872	9
10000	$1.2431E-13$	26.7284	11	$1.3849E-13$	29.5336	11	$1.5222E-13$	23.2247	11	$1.5229E-13$	22.6042	11	$8.9171E-14$	42.2168	11
20000	$1.1944E-13$	63.2272	10	$1.3334E-13$	88.5874	10	$1.2719E-13$	63.0043	10	$1.2001E-13$	61.3925	10	$9.7934E-14$	$1.0101E+02$	10
30000	$1.4298E-13$	$1.0852E+02$	13	$1.5161E-13$	$1.7418E+02$	13	$1.3633E-13$	$1.0848E+02$	13	$1.3836E-13$	$1.0556E+02$	13	$9.5331E-14$	$1.9779E+02$	13
40000	$1.6153E-13$	$1.4578E+02$	21	$1.4877E-13$	$3.7430E+02$	21	$1.4449E-13$	$1.4678E+02$	21	$1.4733E-13$	$1.4134E+02$	21	$9.9270E-14$	$4.0423E+02$	21
50000	$1.4321E-13$	$2.0199E+02$	26	$1.5623E-13$	$4.4489E+02$	26	$1.3927E-13$	$2.2374E+02$	26	$1.4451E-13$	$1.9683E+02$	26	$9.8712E-14$	$6.0989E+02$	26
60000	$8.3824E-14$	$2.7679E+02$	6	$8.7462E-11$	$2.7732E+02$	6	$1.1849E-13$	$2.7861E+02$	6	$7.8359E-14$	$2.7907E+02$	6	$1.0358E-13$	$2.7788E+02$	6
70000	$2.9511E-13$	$3.8868E+04$	4	$7.3915E-14$	$2.7901E+04$	4	$2.1817E-06$	$5.1809E+04$	4	$6.0670E-14$	$1.4090E+04$	4	$2.6256E-13$	$7.5767E+04$	4
80000	$8.8285E-14$	$3.6535E+02$	6	$8.8453E-12$	$3.6616E+02$	6	$1.4822E-12$	$3.7085E+02$	6	$8.4575E-14$	$3.6946E+02$	4	$3.7197E-13$	$3.7199E+02$	6
90000	$9.6464E-14$	$4.1400E+02$	6	$1.4580E-11$	$4.1426E+02$	6	$3.5623E-12$	$4.2111E+02$	6	$8.9727E-14$	$4.1873E+02$	6	$5.6663E-11$	$4.2041E+02$	6
100000	$1.0683E-13$	$4.6281E+02$	6	$1.6064E-11$	$4.6057E+02$	6	$3.7188E-13$	$4.6751E+02$	6	$9.5076E-14$	$4.6447E+02$	6	$2.2224E-10$	$4.6956E+02$	6
200000	$9.7746E-14$	$4.2975E+03$	9	$1.1827E-11$	$4.2766E+03$	9	$1.1180E-13$	$5.0559E+03$	9	$9.7836E-14$	$4.2813E+03$	9	$7.5621E-13$	$4.9268E+03$	9
300000	$9.9296E-14$	$4.6323E+03$	13	$2.8400E-13$	$5.3922E+03$	13	$1.2935E-13$	$4.2806E+03$	13	$9.9620E-14$	$4.0059E+03$	13	$5.7770E-13$	$4.6589E+03$	13
400000	$5.5405E-13$	$6.8255E+03$	5	$5.5062E-09$	$5.1609E+03$	5	$4.4937E-12$	$4.7509E+03$	5	$4.4577E-13$	$4.1279E+03$	5	$1.9636E-12$	$4.8695E+03$	5

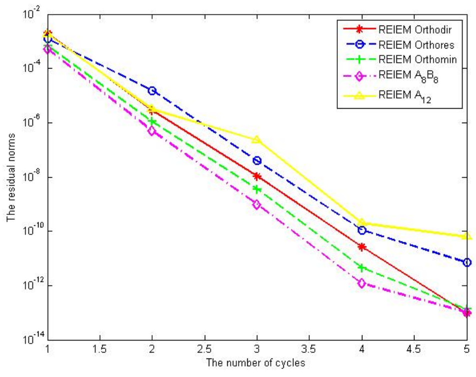
4.4 Comparison of REIEMLA, RLastIt, RLMinRes, and RLMedVal

This section compares REIEMLA and other restarting of Lanczos-type algorithms discussed in Chapter 2, including RLLastIt, RLMinRes, and RLMedVal. As explained in Section 2.1.1, for matrices with big condition numbers, such as those with $\delta = 0.0$, $\delta = 0.2$, and $\delta = 0.5$, RLMinRes experimentally performed the best compared with RLLastIt and RLMedVal. On matrices with $\delta = 5$ and $\delta = 8$, on the other hand, RLMedVal was the best. The aim of this study is to look at which starting point improves the performance of Lanczos-type algorithms. In addition, the stability of RLLastIt, RLMinRes, and RLMedVal will be checked for problems ranging from 10000 to 1000000 dimensions, which are significantly larger than those in Chapter 2.

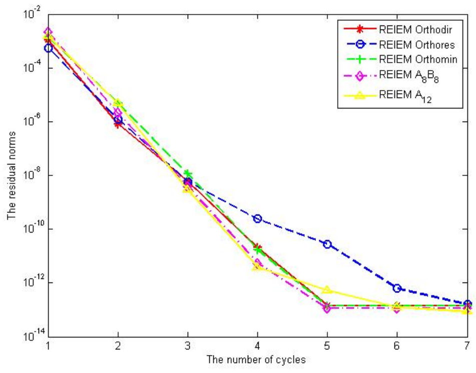
4.4.1 RLLastIt Orthodir, RLMinRes Orthodir, and RLMedVal Orthodir vs REIEM Orthodir

In this section, we compare RLLastIt Orthodir, RLMedVal Orthodir, RLMinRes Orthodir, and REIEM Orthodir. Findings of this particular comparison are in Tables 4.3 and 4.4. The residual norms of the approximate solutions generated by these restartings are captured in Figures 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, and 4.16.

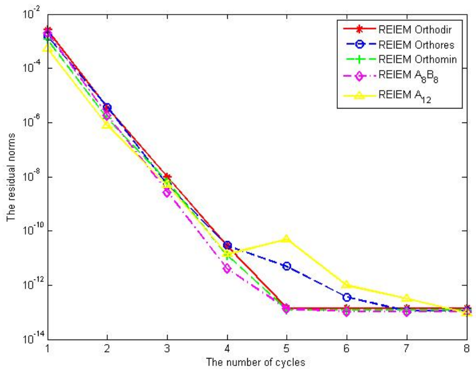
According to Table 4.3, overall, for $\delta = 0.2$, REIEM Orthodir produced more accurate approximate solutions than RLLastIt Orthodir, RLMedVal Orthodir, and RLMinRes Orthodir. However, it was the slowest. RLMinRes Orthodir was the second best for accuracy and it was the fastest. RLLastIt Orthodir, on the other hand, was the worst overall. It still suffered from breakdown (see dimensions



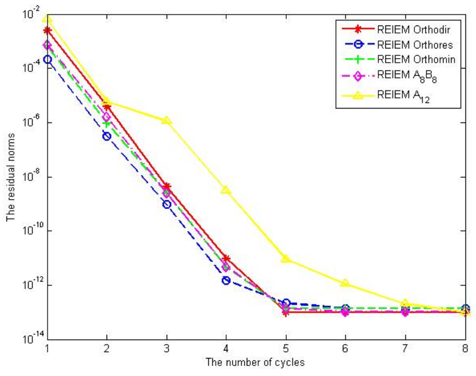
(a) dimensions 1000



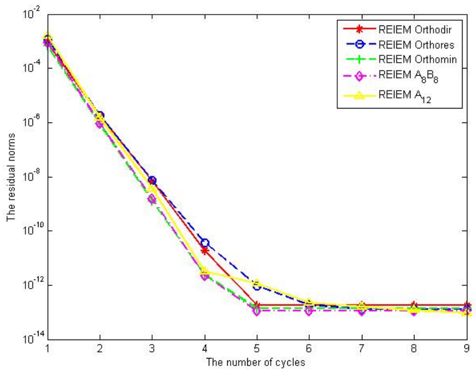
(b) dimensions 2000



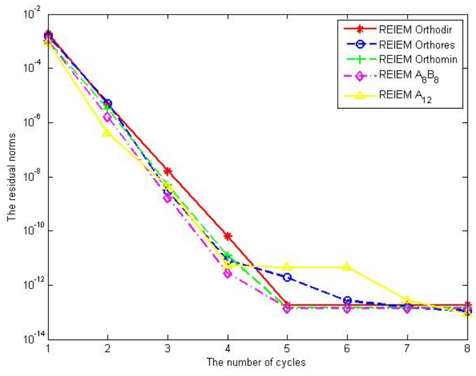
(c) dimensions 3000



(d) dimensions 4000

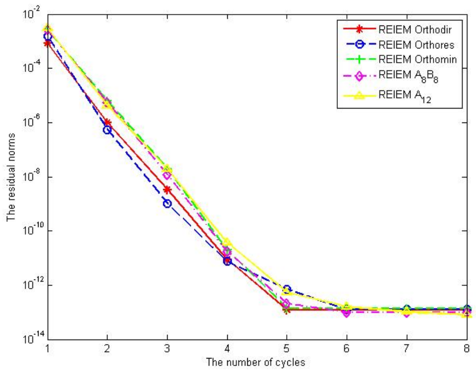


(e) dimensions 5000

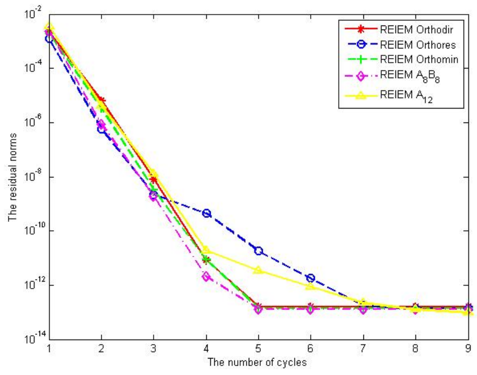


(f) dimensions 6000

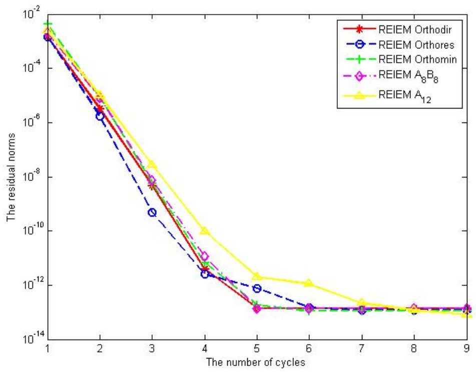
Figure 4.6: *The performance of REIEMLA's for the case of $\delta = 5$, dimensions 1000 to 6000*



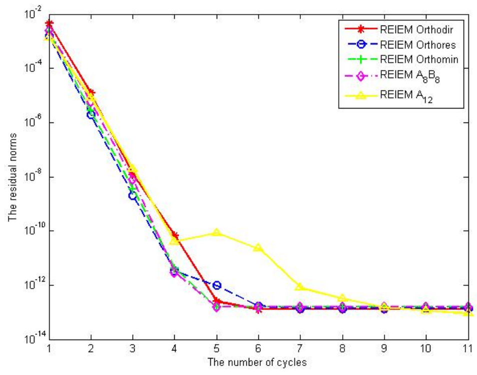
(a) dimensions 7000



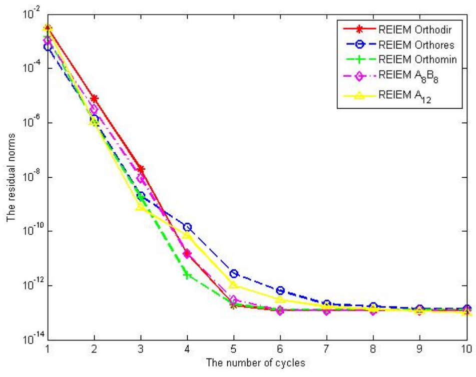
(b) dimensions 8000



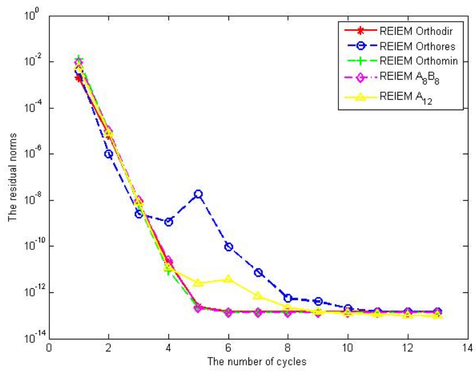
(c) dimensions 9000



(d) dimensions 10000

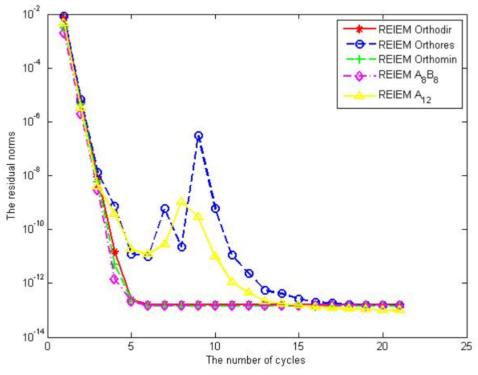


(e) dimensions 20000

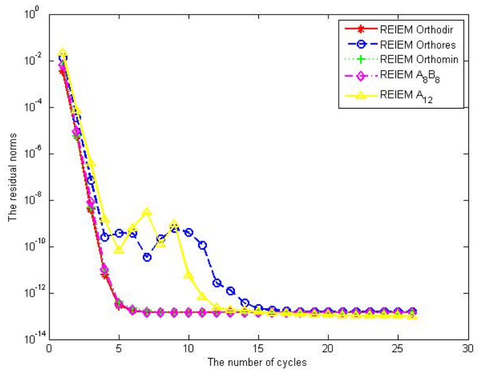


(f) dimensions 30000

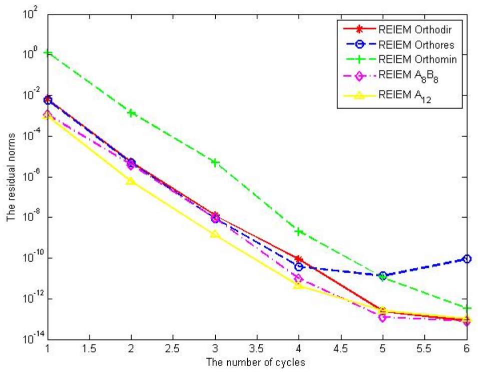
Figure 4.7: *The performance of REIEMLA's for the case of $\delta = 5$, dimensions 7000 to 30000*



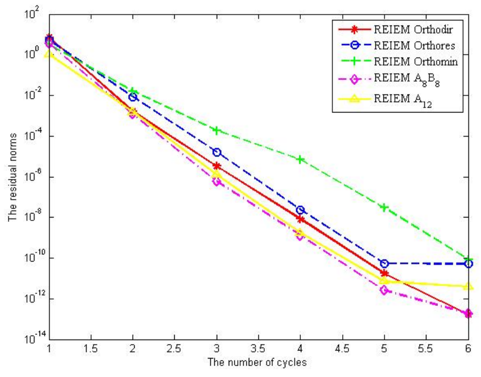
(a) dimensions 40000



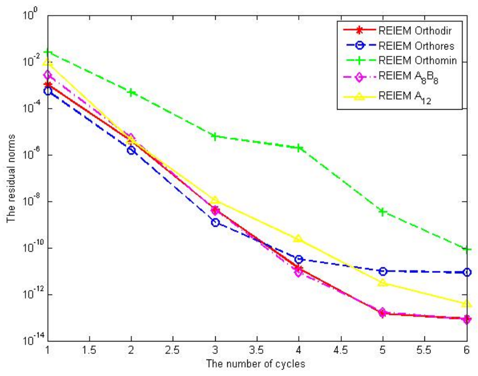
(b) dimensions 50000



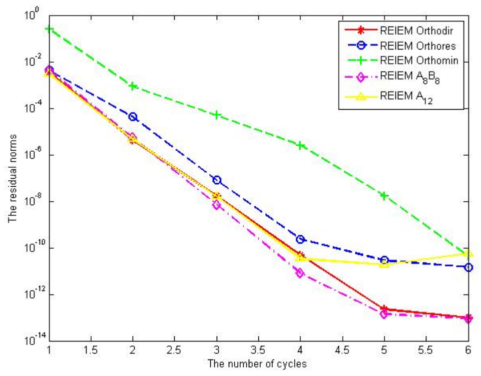
(c) dimensions 60000



(d) dimensions 70000

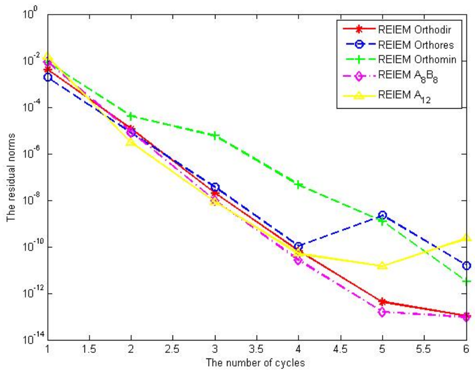


(e) dimensions 80000

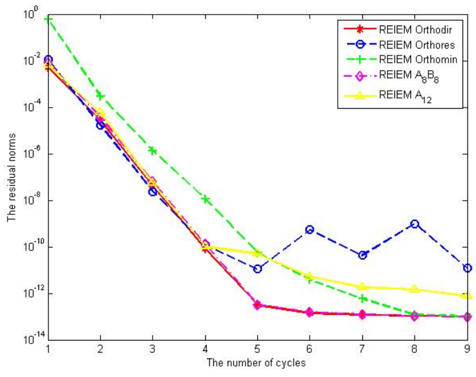


(f) dimensions 90000

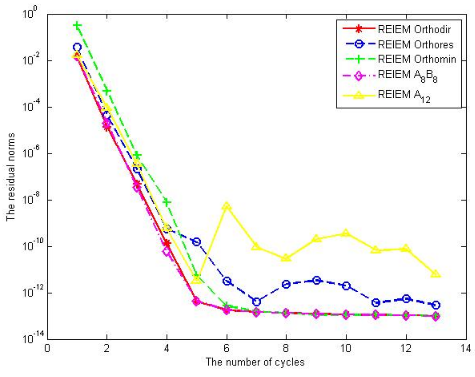
Figure 4.8: *The performance of REIEMLA's for the case of $\delta = 5$, dimensions 40000 to 90000*



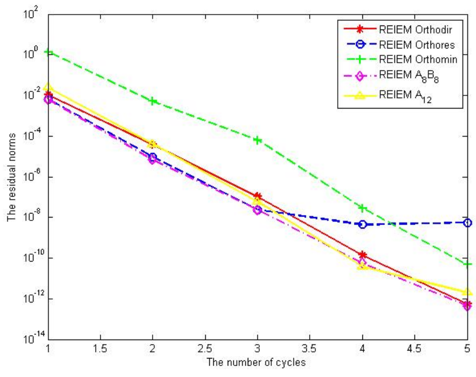
(a) dimensions 100000



(b) dimensions 200000



(c) dimensions 300000



(d) dimensions 400000

Figure 4.9: *The performance of REIEMLA's for the case of* $\delta = 5$, *dimensions 100000 to 400000*

20000, 60000, 90000 column). Yet, for dimensions 1000000, RLLastIt Orthodir produced an approximate solution with a residual norm of 299.2. RLMedVal Orthodir was the third best on accuracy. The computational time of RLMedVal Orthodir is rather low, compared to that of REIEM Orthodir.

For $\delta = 5$, as can be seen in Table 4.4, the trend is slightly different. RLMedVal Orthodir was the best on accuracy, followed by REIEM Orthodir, RLMinRes Orthodir, and RLLastIt Orthodir in that order. For instance, for problems with dimensions between 10000 and 100000, RLMedVal Orthodir consistently managed the approximate solutions with residual norms which satisfied the tolerance. However, for dimensions from 200000 to 100000, the residual norms of the approximate solutions were slightly greater than the tolerance. The residual norms of the approximate solutions produced by RLMinRes Orthodir and REIEM Orthodir, on the other hand, have never satisfied the tolerance. In terms of computing time, the order is the same as in the previous case, where REIEM Orthodir was the slowest, and RLMinRes Orthodir was the fastest . RLMedVal Orthodir time was double that of RLLastIt Orthodir. As expected, no restarting suffered from breakdown.

The comparison of REIEM Orthodir, RLLastIt Orthodir, RLMedVal Orthodir, and RLMinRes Orthodir for different values of δ can be seen in some figures above mentioned. Figures in the first column show the behaviour of 4 restartings for $\delta = 0.2$, while those in the second column show that for $\delta = 5$. We can see in the first column, that for most problems, the red curve, which represents RLLastIt Orthodir, is on top. It indicates that this restarting failed to achieve approximate solutions with the required tolerance. On the second column, in contrast, the red curve along with other curves hit small residual norms.

Table 4.3: *Comparison of RLLastIt Orthodir, RLMinRes Orthodir, RLMedVal Orthodir, and REIEM Orthodir on SLE's with* $\delta = 0.2$.

Dim	RLLastIt Orthodir			RLMedVal Orthodir			RLMinRes Orthodir			REIEM Orthodir		
n	$\lVert r_{last}\rVert$	$T(s)$	cycles*	$\lVert r_{medval}\rVert$	$T(s)$	cycles*	$\lVert r_{min}\rVert$	$T(s)$	cycles*	$\lVert r_{model}\rVert$	$T(s)$	cycles*
10000	$1.0197E+03$	1.6417	8	$8.2954E-14$	3.4869	8	$6.3812E-14$	1.6030	8	$7.2723E-14$	46.8517	8
20000	*NaN*	1.7079	7	*NaN*	8.2981	7	$1.3651E-13$	3.1302	7	$1.3778E-13$	83.4271	7
30000	$1.1431E-05$	4.5129	8	$1.0587E-13$	9.5411	8	$5.6777E-14$	3.36974	8	$7.0624E-14$	$1.3811E+02$	8
40000	$1.0034E-13$	5.1161	7	$1.3915E-13$	11.6754	7	$1.3337E-13$	15.0736	7	$1.0416E-13$	$1.4671E+02$	7
50000	$1.9219E-11$	5.6047	7	$3.0603E-13$	11.9592	7	$1.2119E-13$	4.9212	7	$8.8856E-14$	$1.6158E+02$	7
60000	*NaN*	2.7684	2	*NaN*	16.8510	2	$9.3471E-14$	5.7720	8	$9.4736E-14$	$2.1146E+02$	8
70000	$8.6199E-04$	7.1957	7	$5.1839E-13$	16.1976	7	$1.1174E-13$	6.3761	7	$8.8579E-14$	$2.2149E+02$	7
80000	$1.0629E-13$	8.3211	8	$1.3325E-13$	20.0473	8	$1.3564E-13$	6.5941	8	$8.7312E-14$	$2.8178E+02$	8
90000	*NaN*	15.0871	4	$2.2863E-13$	22.4101	8	$6.7621E-14$	9.1351	8	$9.0221E-14$	$3.1943E+02$	8
100000	$8.4367E-06$	19.8926	6	$2.7951E-12$	38.6984	6	$1.4457E-13$	17.7201	6	$6.7572E-14$	$4.6079E+02$	6
200000	$2.2371E-07$	53.0010	7	$3.0373E-13$	92.9957	7	$1.0271E-13$	42.4940	7	$7.3917E-14$	$1.0452E+03$	7
300000	$3.6456E-05$	84.0492	8	$2.9710E-13$	$1.3644E+02$	8	$9.7195E-14$	69.0920	8	$5.7715E-14$	$1.7206E+03$	8
400000	$1.0269E-06$	$1.2527E+02$	21	$4.0323E-13$	$2.0462E+02$	21	$1.0179E-13$	$1.0265E+02$	21	$6.2753E-14$	$2.2990E+03$	21
500000	4.0095	$1.2553E+02$	7	$4.9181E-13$	$2.2944E+02$	7	$1.1028E-13$	$1.0562E+02$	7	$7.5120E-14$	$2.5773E+03$	7
600000	$2.3792E-08$	$2.1644E+02$	8	$3.3380E-13$	$3.3367E+02$	8	$1.1534E-13$	$1.6817E+02$	8	$6.9701E-14$	$3.5818E+03$	8
700000	$8.8101E-04$	$9.3461E+02$	8	$4.8528E-13$	$1.1483E+03$	8	$1.4392E-13$	$1.3074E+04$	8	$9.6797E-14$	$1.1499E+02$	8
800000	0.0018	$1.0005E+03$	8	$3.1252E-13$	$1.3431E+03$	8	$1.3669E-13$	$8.7973E+02$	8	$9.0274E-14$	$1.3790E+04$	8
900000	$1.7343E-06$	$2.9709E+02$	9	$3.5152E-13$	$4.9551E+02$	9	$1.0190E-13$	$2.0792E+02$	9	$7.8603E-14$	$5.6509E+03$	9
1000000	299.2002	$3.4414E+02$	10	$2.7200E-13$	$5.7312E+02$	10	$1.1225E-13$	$2.7021E+02$	10	$7.8631E-14$	$5.7529E+03$	10

Table 4.4: *Comparison of RLLastIt Orthodir, RLMinRes Orthodir, RLMedVal Orthodir, and REIEM Orthodir on SLE's with* $\delta = 5$.

Dim	RLLastIt Orthodir			RLMedVal Orthodir			RLMinRes Orthodir			REIEM Orthodir		
n	$\|\|r_{last}\|\|$	$T(s)$	cycles*	$\|\|r_{medval}\|\|$	$T(s)$	cycles*	$\|\|r_{min}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*
10000	$1.5282E-13$	1.5647	7	$8.6507E-14$	2.9339	7	$1.1236E-13$	1.1503	7	$1.2431E-13$	37.4049	7
20000	$2.5421E-13$	2.0033	7	$9.2670E-14$	5.3713	7	$1.8305E-13$	1.9031	7	$1.1944E-13$	74.7341	7
30000	$2.5585E-13$	3.6599	8	$9.3863E-14$	8.9491	8	$1.3291E-13$	2.8207	8	$1.4298E-13$	$1.1495E+02$	8
40000	$2.1163E-13$	4.1743	10	$9.4974E-14$	13.2775	10	$1.4665E-13$	4.3929	10	$1.6153E-13$	$1.4879E+02$	10
50000	$2.1957E-13$	5.6835	11	$9.4983E-13$	17.7897	11	$1.4795E-13$	5.1027	11	$1.4321E-13$	$2.0707E+02$	11
60000	$2.3518E-13$	6.6643	12	$9.8109E-14$	21.1287	12	$1.5621E-13$	5.9157	12	$1.5946E-13$	$2.5977E+02$	12
70000	$1.4588E-12$	7.5442	13	$9.9303E-14$	26.2691	13	$1.6399E-13$	6.8447	13	$1.5147E-13$	$3.2813E+02$	13
80000	$3.2856E-13$	8.3211	16	$9.9896E-14$	20.0473	16	$1.0001E-13$	6.5941	16	$1.5778E-13$	$3.2813E+02$	16
90000	$1.0512E-13$	9.7338	19	$9.8565E-14$	37.3427	19	$1.5962E-13$	8.9901	19	$1.6593E-13$	$3.5163E+02$	19
100000	$1.1403E-13$	10.2525	19	$9.9112E-14$	37.9995	19	$1.1024E-13$	10.0486	19	$1.5772E-13$	$4.0715E+02$	19
200000	$2.6513E-11$	23.6909	5	$7.3801E-13$	43.8722	5	$6.9504E-13$	23.9239	5	$6.0369E-13$	$5.1194E+02$	5
300000	$2.5185E-10$	34.5799	5	$7.8724E-13$	64.6559	5	$3.5092E-13$	33.9495	5	$1.6898E-12$	$7.7716E+02$	5
400000	$2.5430E-13$	61.3019	6	$7.8724E-13$	$1.2496E+02$	6	$2.2746E-13$	62.6279	6	$4.9443E-13$	$1.5736E+03$	6
500000	$2.0284E-12$	74.2925	5	$1.1518E-12$	$1.4259E+02$	5	$3.3174E-13$	73.4990	5	$7.9202E-13$	$1.7319E+03$	5
600000	$9.7777E-11$	75.6925	5	$1.6607E-12$	$1.3715E+02$	5	$3.6769E-13$	76.7997	5	$8.4074E-13$	$1.5827E+03$	5
700000	$1.8568E-12$	$7.2134E+02$	6	$7.8956E-13$	$1.1184E+03$	6	$1.0062E-13$	$5.1049E+02$	6	$6.9662E-13$	$1.3265E+04$	6
800000	$1.2368E-11$	$3.4720E+02$	6	$1.2724E-12$	$5.6019E+02$	6	$1.1644E-13$	$3.1915E+02$	6	$1.0454E-12$	$4.6898E+03$	6
900000	$1.1575E-13$	$1.4965E+03$	7	$6.5957E-13$	$2.5649E+03$	7	$1.1459E-13$	$1.0995E+03$	7	$6.1413E-13$	$2.9890E+04$	7
1000000	$2.8962E-12$	$6.2909E+02$	6	$1.0127E-12$	$8.2985E+02$	6	$1.1586E-13$	$5.1711E+02$	6	$7.9511E-13$	$6.3018E+03$	6

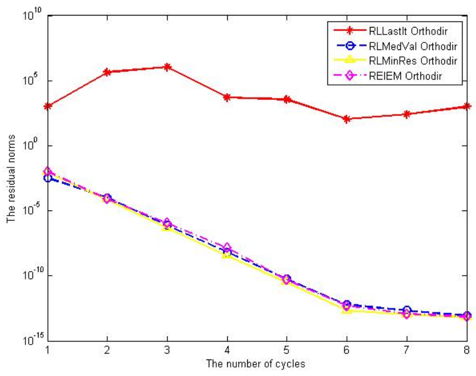
4.4.2 RLLastIt A_{12}, RLMinRes A_{12}, and RLMedVal A_{12} vs REIEM A_{12}

Here we consider A_{12}, [31], which according to the author, is more robust than other Lanczos-type algorithms. Typically, A_{12} takes longer to breakdown than Lanczos-type algorithms found in [3]. In fact, when we implemented REIEM A_{12} (see Section 4.3), it also consistently showed a better performance than other Lanczos-type algorithm. All of the findings are recorded in Tables 4.5 and 4.6 which respectively present results for cases with $\delta = 0.2$ and $\delta = 5$.

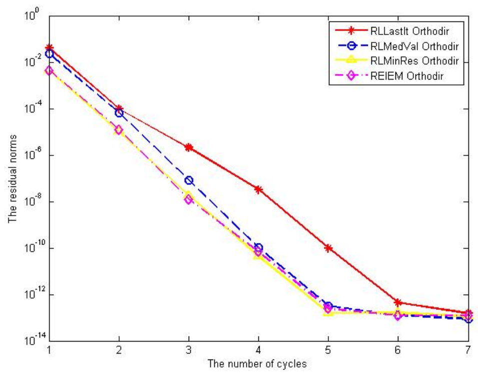
According Table 4.5, RLLastIt A_{12}, RLMinRes A_{12}, and REIEM A_{12} were similarly robust. RLMedVal A_{12}, in contrast, was less accurate on most of problems; it also suffered from breakdown for dimension 20000. In terms of computing time, REIEM A_{12} typically was the slowest, while RLMinRes A_{12} was the fastest. RLLastIt A_{12} and RLMedVal A_{12} were in the middle.

The behaviour of the four restartings for the particular value $\delta = 5$, is rather different from that shown in the above commented results. We can see in Table 4.6, in general, RLMinRes A_{12} was more accurate than other restartings, though the residual norms of the iterates were still larger than the tolerance. It is followed by REIEM A_{12}, RLMedVal A_{12}, and RLLastIt A_{12}. The trend in time consuming is the same as the previous case.

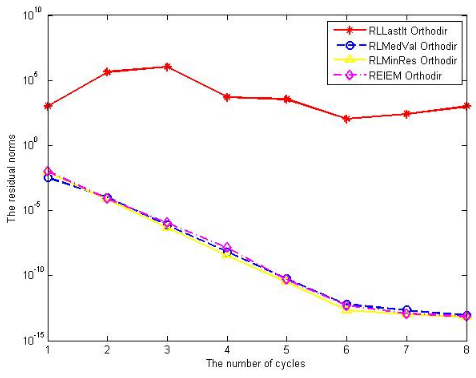
Figures 4.17 , 4.18, 4.19, 4.20, 4.21, 4.22, and 4.23 show the accuracy of solutions generated by RLLastIt A_{12}, RLMedVal A_{12}, RLMinRes A_{12}, and REIEM A_{12}, for both $\delta = 0.2$ and $\delta = 5$, respectively. For instance, in the first column of the figures, the blue curve appears over other curves for most problems, whereas in the second column, the red curve is found on the top for most of problems. The



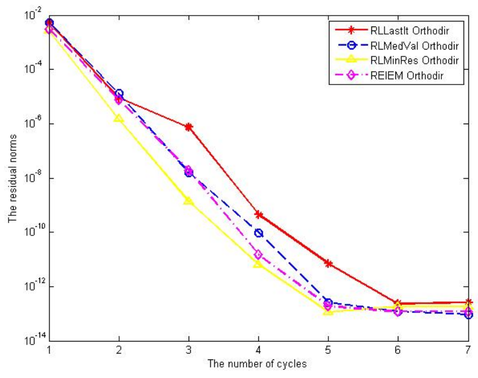
(a) dimensions 10000; $\delta = 0.2$



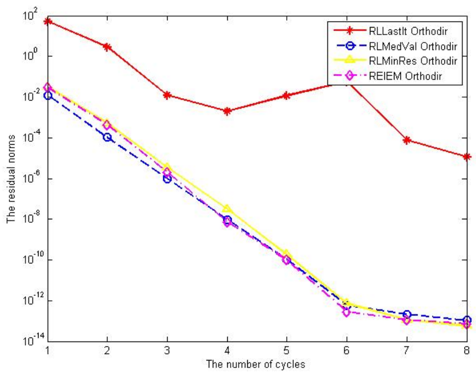
(b) dimensions 10000; $\delta = 5$



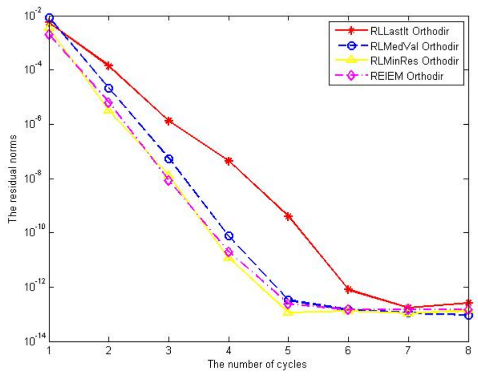
(c) dimensions 20000; $\delta = 0.2$



(d) dimensions 20000; $\delta = 0.2$

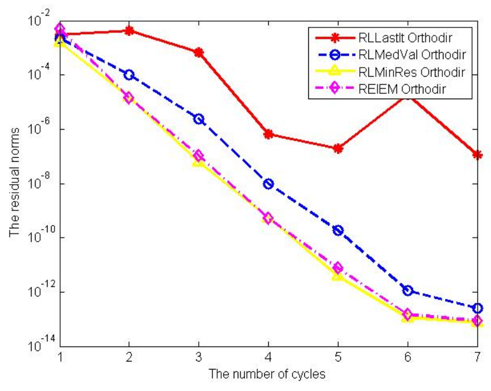


(e) dimensions 30000; $\delta = 0.2$

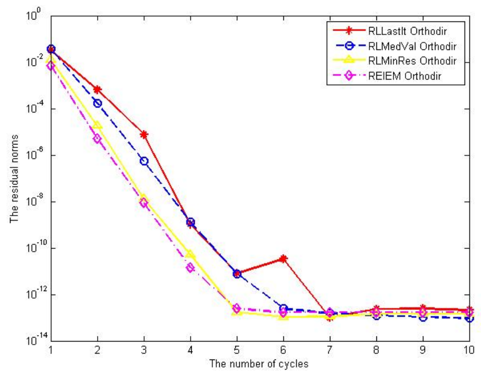


(f) dimensions 30000; $\delta = 5$

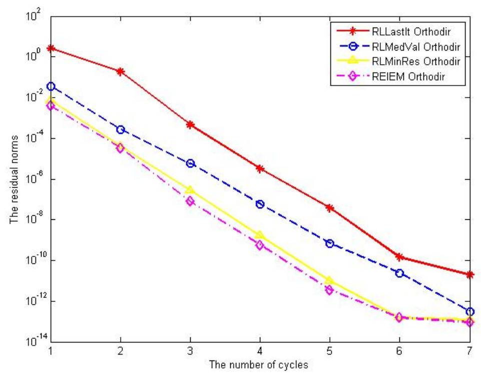
Figure 4.10: *The performances of RLLastIt Orthodir, RLMinres Orthodir, RLMedVal Orthodir and REIEM Orthodir on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 10000 to 30000*



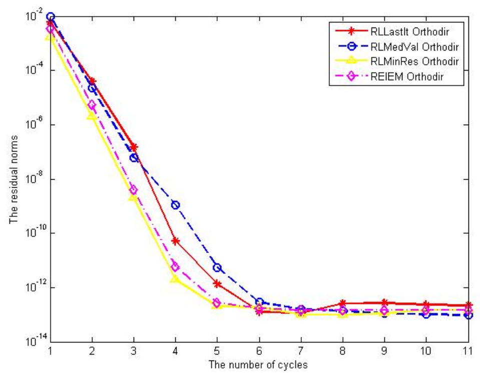
(a) dimensions 40000; δ = 0.2



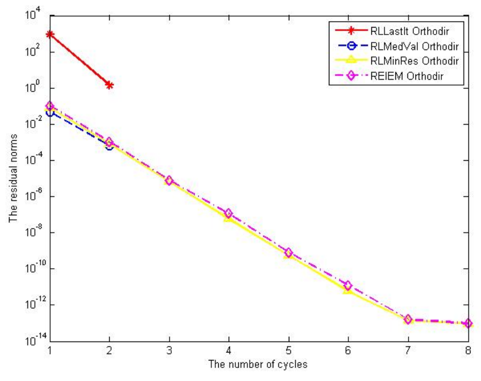
(b) dimensions 40000; δ = 5



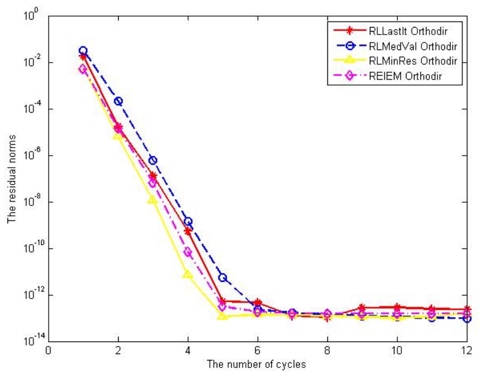
(c) dimensions 50000; δ = 0.2



(d) dimensions 50000; δ = 5

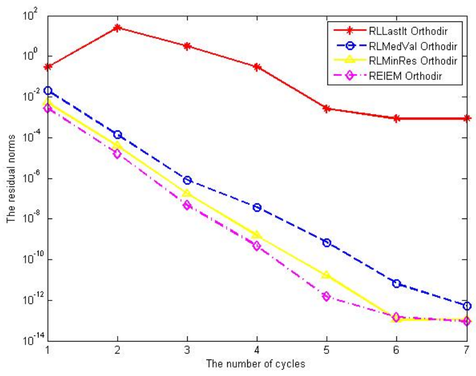


(e) dimensions 60000; δ = 0.2

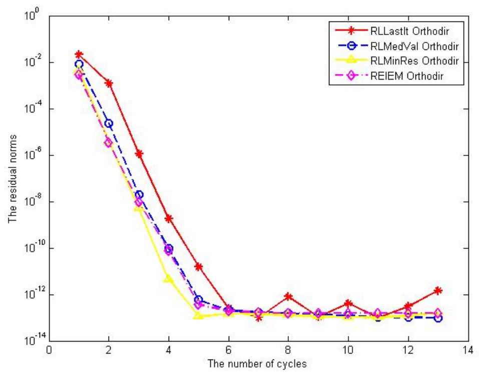


(f) dimensions 60000; δ = 0.2

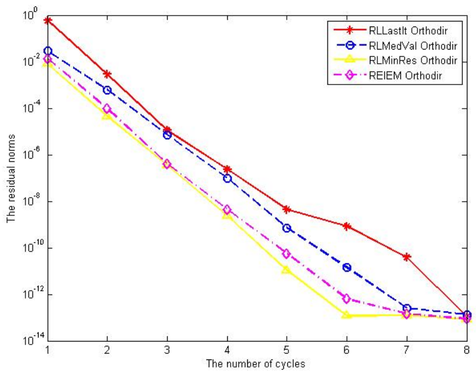
Figure 4.11: *The performances of RLLastIt Orthodir, RLMinres Orthodir, RLMedVal Orthodir and REIEM Orthodir on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 40000 to 60000*



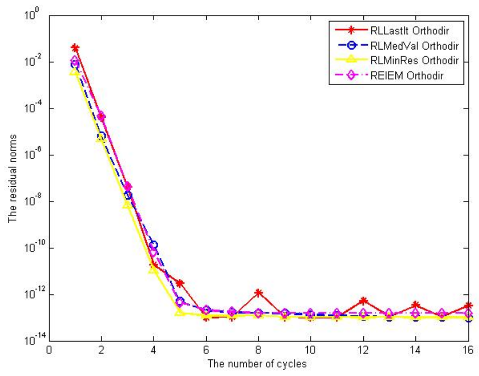
(a) dimensions 70000; $\delta = 0.2$



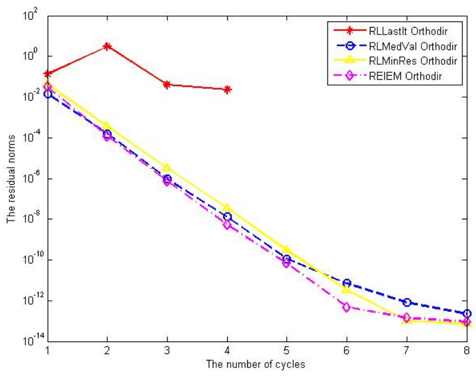
(b) dimensions 70000; $\delta = 5$



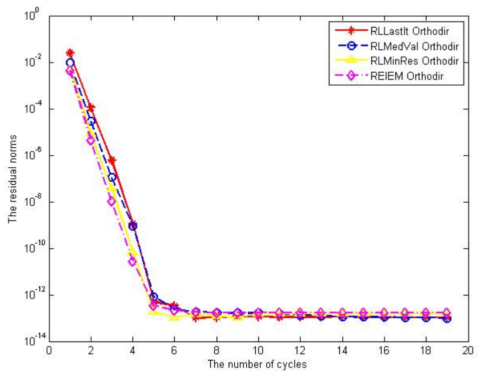
(c) dimensions 80000; $\delta = 0.2$



(d) dimensions 80000; $\delta = 5$

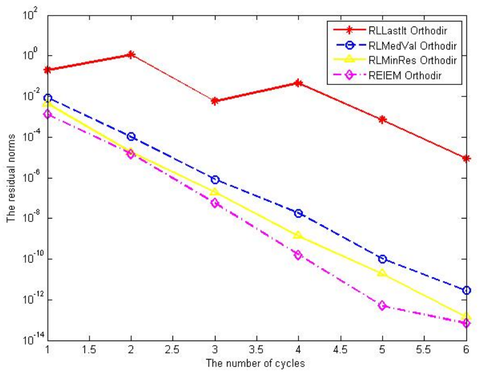


(e) dimensions 90000; $\delta = 0.2$

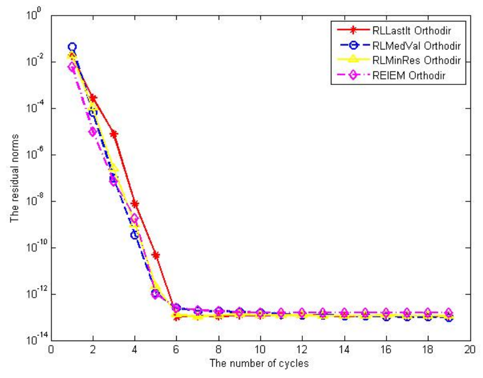


(f) dimensions 90000; $\delta = 5$

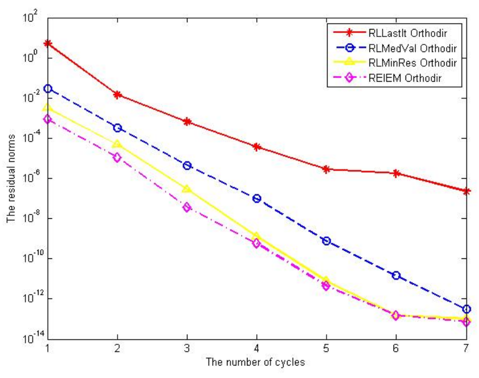
Figure 4.12: *The performances of RLLastIt Orthodir, RLMinres Orthodir, RLMedVal Orthodir and REIEM Orthodir on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 70000 to 90000*



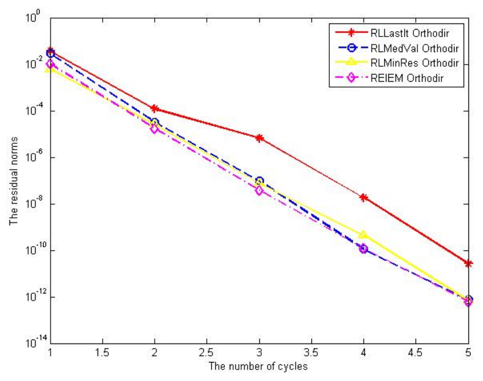
(a) dimensions 100000; $\delta = 0.2$



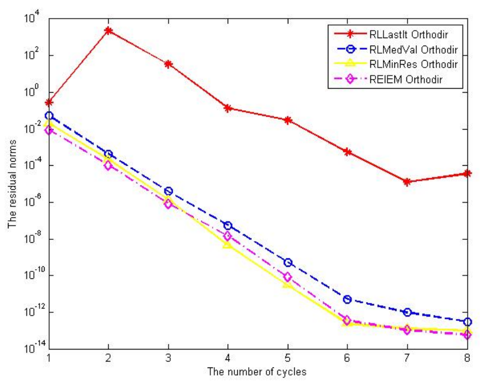
(b) dimensions 100000; $\delta = 0.2$



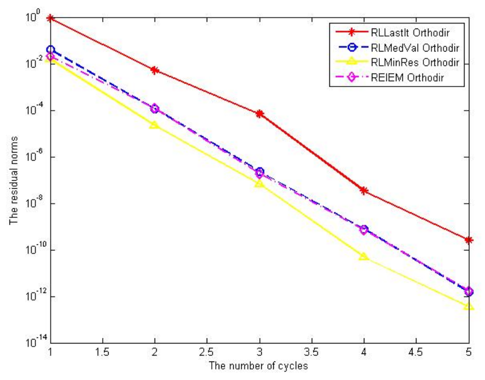
(c) dimensions 200000; $\delta = 0.2$



(d) dimensions 200000; $\delta = 5$

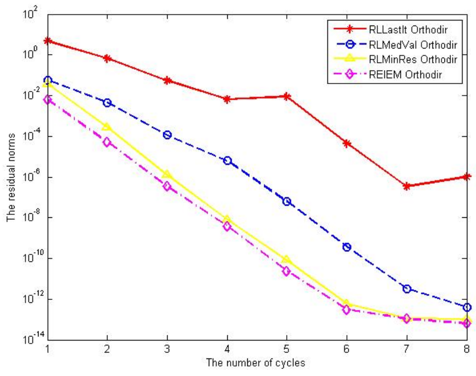


(e) dimensions 300000; $\delta = 0.2$

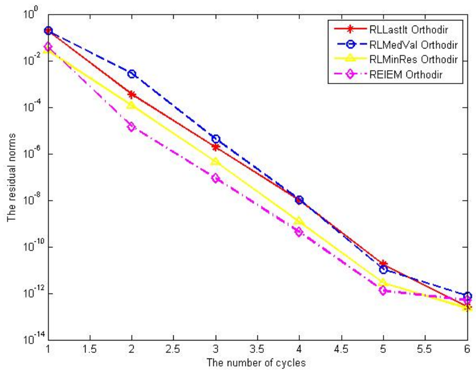


(f) dimensions 300000; $\delta = 5$

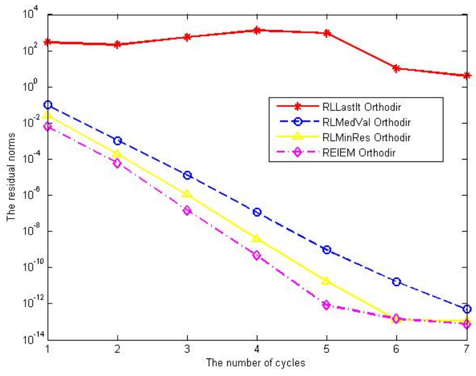
Figure 4.13: *The performances of RLLastIt Orthodir, RLMinres Orthodir, RLMedVal Orthodir and REIEM Orthodir on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 100000 to 300000*



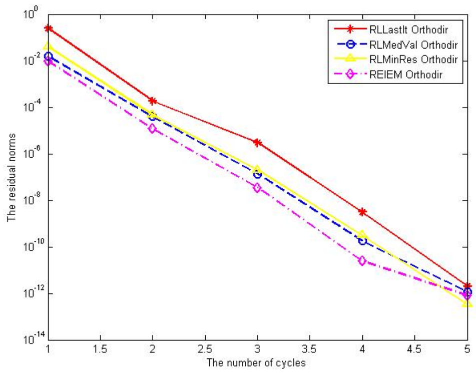
(a) dimensions 400000; $\delta = 0.2$



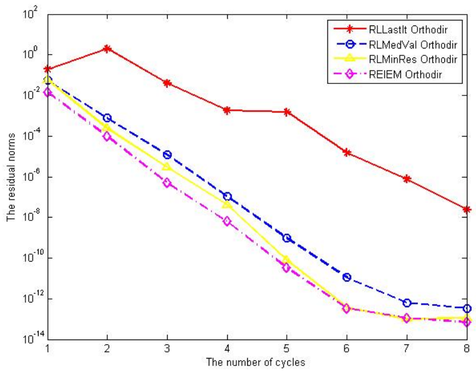
(b) dimensions 400000; $\delta = 5$



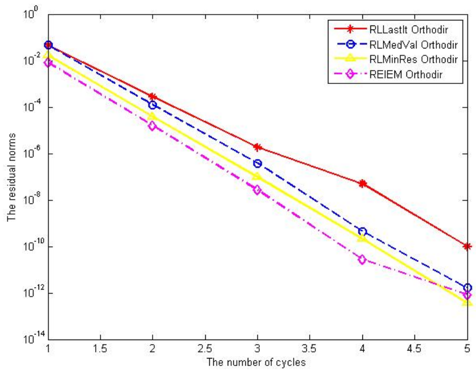
(c) dimensions 500000; $\delta = 0.2$



(d) dimensions 500000; $\delta = 0.2$

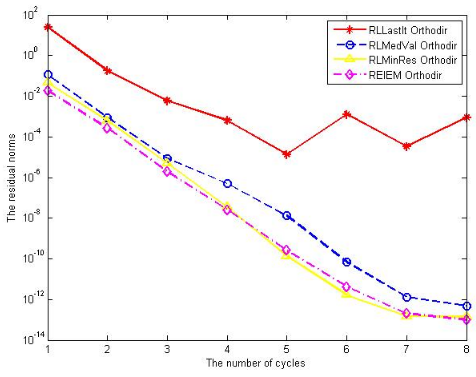


(e) dimensions 600000; $\delta = 0.2$

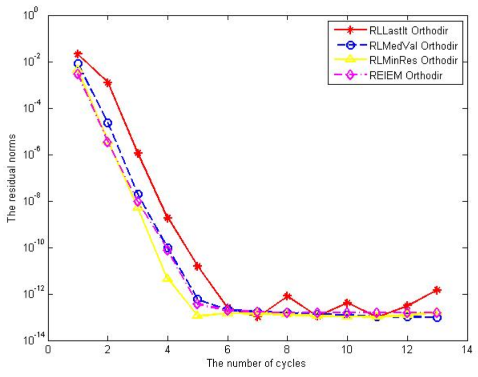


(f) dimensions 600000; $\delta = 5$

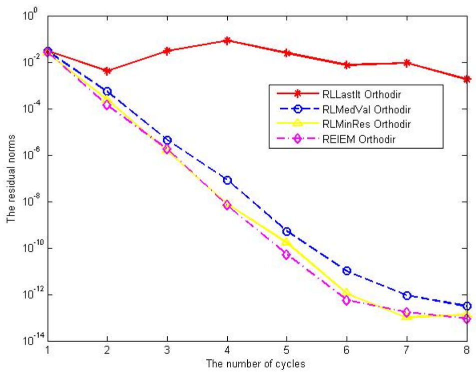
Figure 4.14: *The performances of RLLastIt Orthodir, RLMinres Orthodir, RLMedVal Orthodir and REIEM Orthodir on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 400000 to 600000*



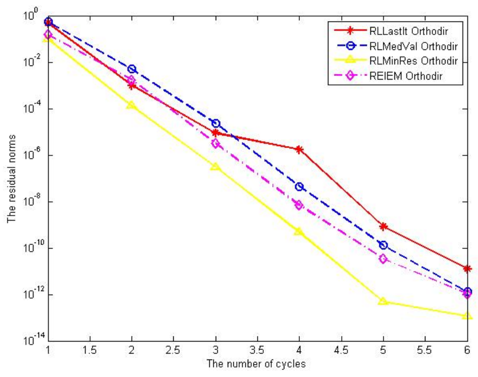
(a) dimensions 700000; $\delta = 0.2$



(b) dimensions 700000; $\delta = 5$

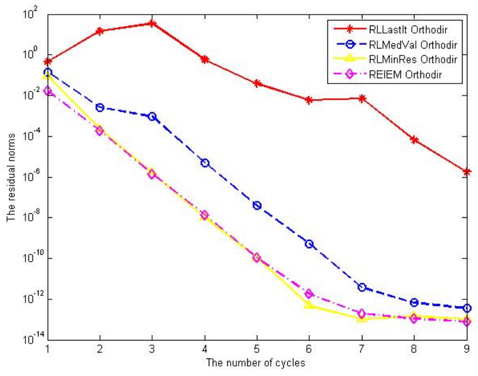


(c) dimensions 800000; $\delta = 0.2$

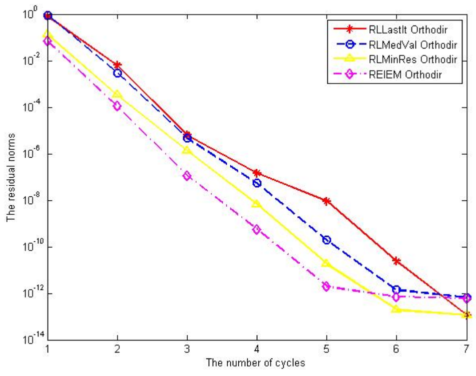


(d) dimensions 800000; $\delta = 5$

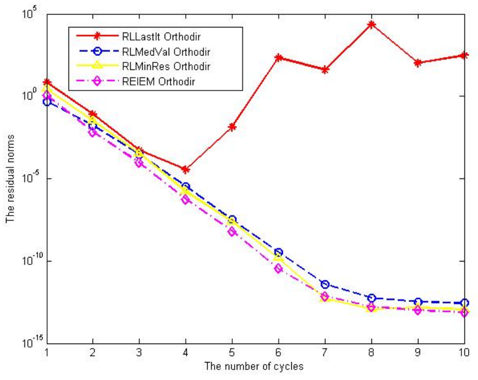
Figure 4.15: *The performances of RLLastIt Orthodir, RLMinres Orthodir, RLMedVal Orthodir and REIEM Orthodir on SLE's for* $\delta = 0.2$ *and* $\delta = 5$*, dimensions 700000 and 800000*



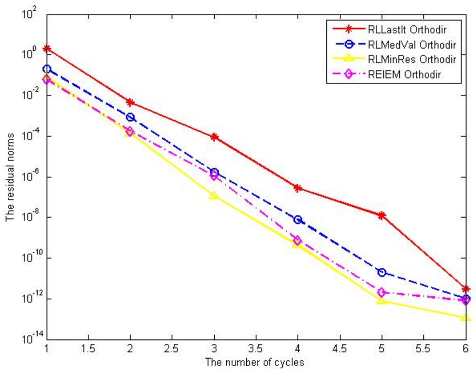
(a) dimensions 900000; $\delta = 0.2$



(b) dimensions 900000; $\delta = 0.2$



(c) dimensions 1000000; $\delta = 0.2$



(d) dimensions 1000000; $\delta = 5$

Figure 4.16: *The performances of RLLastIt Orthodir, RLMinres Orthodir, RLMedVal Orthodir and REIEM Orthodir on SLE's for* $\delta = 0.2$ *and* $\delta = 5$*, dimensions 900000 and 1000000*

other thing to notice is that the yellow curve which is in both the first and the second column, appears below the pink curve for most problems.

Table 4.5: *Comparison of RLLastIt A_{12}, RLMinRes A_{12}, RLMedVal A_{12}, and REIEM A_{12} on SLE's with $\delta = 0.2$*

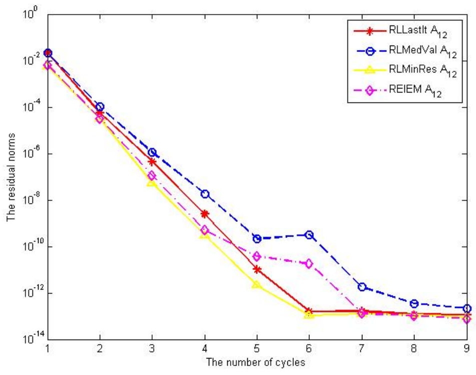
Dim	RLLastIt A_{12}			RLMedVal A_{12}			RLMinRes A_{12}			REIEM A_{12}		
n	$\|r_{last}\|$	$T(s)$	cycles*	$\|r_{medval}\|$	$T(s)$	cycles*	$\|r_{min}\|$	$T(s)$	cycles*	$\|r_{model}\|$	$T(s)$	cycles*
10000	$1.1395E-13$	0.9111	9	$2.1911E-13$	2.8311	9	$9.1387E-14$	0.9705	9	$7.7202E-14$	38.6083	9
20000	$1.0572E-13$	2.0039	8	*NaN*	4.6817	3	$8.2156E-14$	1.8019	8	$1.0677E-13$	71.0239	8
30000	$1.2036E-13$	2.8893	8	$1.0614E-11$	7.8855	8	$1.4478E-13$	2.9199	8	$8.3067E-14$	$1.0552E+02$	8
40000	$5.6816E-14$	3.7705	12	$8.4546E-14$	13.0142	12	$1.0515E-13$	3.8425	12	$1.1670E-13$	$1.4082E+02$	12
50000	$2.7123E-13$	5.1914	8	$7.7261E-13$	12.6740	8	$9.8339E-14$	4.7926	8	$1.1965E-13$	$1.7529E+02$	8
60000	$6.4499E-14$	6.1521	10	$2.2694E-13$	17.7543	10	$5.7380E-14$	5.4171	10	$1.3297E-13$	$1.9357E+02$	10
70000	$6.2368E-14$	10.1749	12	$5.8986E-13$	27.0398	12	$9.4701E-14$	8.7917	12	$1.1024E-13$	$2.8047E+02$	12
80000	$3.9066E-14$	9.1172	19	$9.9861E-14$	36.4875	19	$1.2571E-13$	10.5607	19	$1.0825E-13$	$3.3930E+02$	19
90000	$2.0307E-13$	10.4320	9	$2.7902E-13$	24.9229	9	$1.0952E-13$	10.2247	9	$9.1989E-14$	$3.4409E+02$	9
100000	$6.5739E-14$	11.3657	19	$9.7209E-14$	44.9519	19	$1.3900E-13$	10.2565	19	$1.0604E-13$	$7.3209+02$	19
200000	$1.8157E-13$	33.5512	11	$1.7998E-13$	72.9424	11	$9.2556E-14$	24.9164	11	$1.1353E-13$	$9.1544E+02$	11
300000	$5.9051E-14$	40.1802	15	$2.8456E-13$	$1.4153E+02$	15	$9.1524E-14$	52.3172	15	$1.5013E-13$	$1.8099E+03$	15
400000	$8.3224E-14$	$1.0201E+02$	14	$1.6404E-13$	$1.7227E+02$	14	$9.7657E-14$	81.5764	14	$1.2943E-13$	$7.5495E+03$	14
500000	$1.1030E-13$	$6.4965E+02$	11	$4.0941E-13$	$4.7080E+02$	11	$1.2828E-13$	$1.7966E+02$	11	$9.6065E-14$	$2.5773E+03$	11
600000	$1.0211E-13$	$1.5877E+02$	11	$3.6880E-13$	$3.7680E+02$	11	$1.0665E-13$	$2.0973E+02$	11	$9.0823E-14$	$4.4756E+03$	11
700000	$1.0126E-13$	$2.8527E+02$	14	$2.8299E-13$	$7.0311E+02$	14	$1.1074E-13$	$2.2525E+02$	14	$9.9619E-14$	$6.9622E+03$	14
800000	$2.3916E-13$	$3.5925E+02$	11	$4.2084E-11$	$6.2091E+02$	11	$1.0075E-13$	$3.1237E+02$	11	$9.8629E-14$	$5.9613E+03$	11
900000	$1.2672E-12$	$4.2183E+02$	12	$3.2157E-13$	$6.0596E+02$	12	$1.1223E-13$	$3.5926E+02$	12	$9.7263E-14$	$7.1965E+03$	12
1000000	$9.7307E-14$	$3.5967E+02$	14	$3.2669E-13$	$7.5548E+02$	14	$1.1015E-13$	$3.5034E+02$	14	$1.0235E-13$	$8.9431E+03$	14

Table 4.6: *Comparison of RLLastIt A_{12}, RLMinRes A_{12}, RLMedVal A_{12}, and REIEM A_{12} on SLE's with $\delta = 5$*

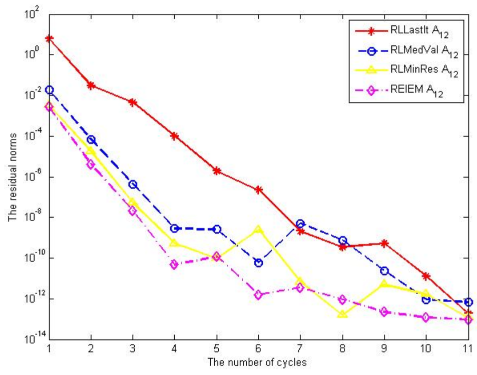
Dim	RLLastIt A_{12}			RLMedVal A_{12}			RLMinRes A_{12}			REIEM A_{12}		
n	$\|\|r_{last}\|\|$	$T(s)$	cycles*	$\|\|r_{medval}\|\|$	$T(s)$	cycles*	$\|\|r_{min}\|\|$	$T(s)$	cycles*	$\|\|r_{model}\|\|$	$T(s)$	cycles*
10000	$1.8281E-13$	7.3084	11	$6.7944E-13$	5.6726	11	$1.1966E-13$	3.2605	11	$9.6361E-14$	37.4127	11
20000	$9.1375E-09$	4.8755	11	$6.4979E-13$	7.6063	11	$6.2927E-13$	4.1526	11	$9.1728E-14$	71.9489	11
30000	$7.0341E-10$	8.4317	11	$1.1904E-13$	10.7368	11	$7.3341E-12$	6.5926	11	$9.7945E-14$	$1.0635E+02$	11
40000	$3.2725E-11$	17.2950	15	$1.0309E-13$	18.5392	15	$6.1552E-10$	15.3649	15	$9.8554E-14$	$1.8631E+02$	15
50000	$5.0559E-08$	44.5313	27	$2.9991E-12$	45.2498	27	$4.1688E-12$	31.2646	27	$9.8652E-14$	$3.8606E+02$	27
60000	$1.0285E-07$	7.9078	7	$2.1248E-12$	16.7972	7	$1.047E-13$	8.2064	7	$2.5346E-13$	$2.4950E+02$	7
70000	$4.3529E-07$	8.6247	8	$1.1799E-08$	19.8241	8	$1.1393E-13$	8.6232	8	$1.0651E-10$	$2.5008E+02$	8
80000	$4.6898E-10$	10.1676	9	$2.3376E-08$	22.9596	9	$1.0965E-13$	9.6831	9	$2.9196E-13$	$3.0086E+02$	9
90000	$4.0313E-08$	7.6963	5	$8.2471E-08$	16.6028	5	$1.1498E-13$	7.0518	5	$2.5325E-12$	$2.1959E+02$	5
100000	$1.4261E-09$	15.3556	11	$3.3158E-10$	34.2262	11	$1.1657E-13$	13.9603	11	$1.8055E-13$	$4.2915+02$	11
200000	$2.2120E-09$	26.9593	6	$1.3579E-10$	49.8357	6	$1.3313E-13$	27.2543	6	$2.5797E-12$	$5.6340E+03$	6
300000	$3.6275E-08$	60.8363	9	$4.2473E-12$	$1.1266E+02$	9	$1.1711E-13$	60.1272	9	$9.1608E-12$	$1.3719E+03$	9
400000	$1.2404E-10$	94.5036	9	$1.9132E-12$	$1.6330E+02$	9	$1.2487E-13$	95.3229	9	$2.8754E-12$	$4.4937E+03$	9
500000	$2.2187E-13$	$1.7289E+02$	8	$4.4922E-11$	$3.0280E+02$	8	$2.4364E-13$	$1.9042E+02$	8	$6.1720E-12$	$2.9615E+03$	8
600000	$5.4853E-10$	$1.5086E+03$	6	$5.4888E-12$	$1.2023E+03$	6	$1.7595E-13$	$7.0673E+02$	6	$1.1055E-10$	$7.1069E+03$	6
700000	$2.9782E-08$	$3.0449E+02$	10	$7.8564E-10$	$5.2249E+02$	10	$2.0764E-13$	$3.0176E+02$	10	$1.0550E-10$	$4.9793E+03$	10
800000	$2.5890E-07$	$2.0886E+02$	5	$1.4378E-09$	$3.4398E+02$	5	$4.0240E-13$	$2.3000E+02$	5	$6.7190E-12$	$3.3461E+03$	5
900000	$3.4270E-10$	$4.0327E+03$	6	$2.9562E-10$	$4.6774E+03$	6	$1.3948E-13$	$3.8044E+03$	6	$2.1753E-12$	$3.5051E+04$	6
1000000	$4.7081E-08$	$1.3421E+03$	14	$1.5497E-09$	$1.8816E+03$	14	$2.6710E-11$	$1.4518E+03$	14	$6.8502E-13$	$1.2079E+04$	14

4.5 Summary

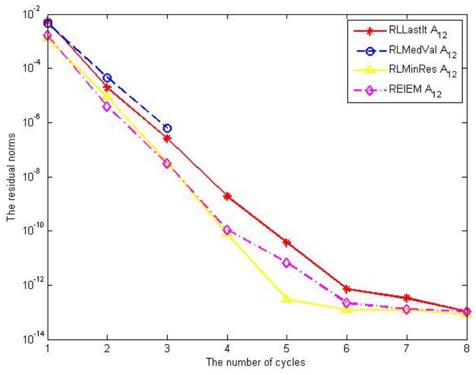
Restarting from the iterate generated by EIEMLA's (REIEMLA's) have been implemented. They include REIEM Orthodir, REIEM Orthores, REIEM Orthomin, REIEM A_8B_8, and REIEM A_{12}. This kind of restarting uses an iterate generated by EIEMLA as a starting point which is different from either those investigated in Chapter 2 or in [29, 30], which is a novelty of our works. We can conclude here that REIEMLA produced the best results as can be seen in Tables 4.3 and 4.4 of Section 4.4, so they do agree with the theory expanded in Chapter 3. This means that the method is comparatively better in terms of quality of solution. However, given the time overheads required to find the regression model and restarting, both REIEM Orthodir and REIEM A_{12} (as REIMLA) are not efficient in terms of computing time.



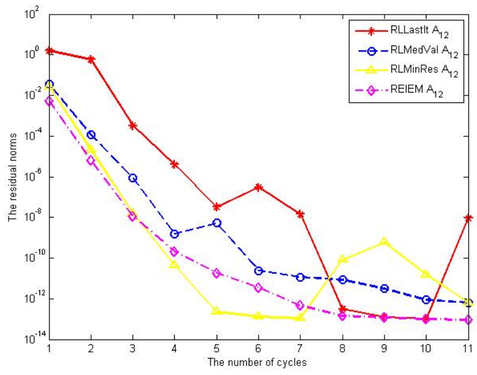
(a) dimensions 10000; $\delta = 0.2$



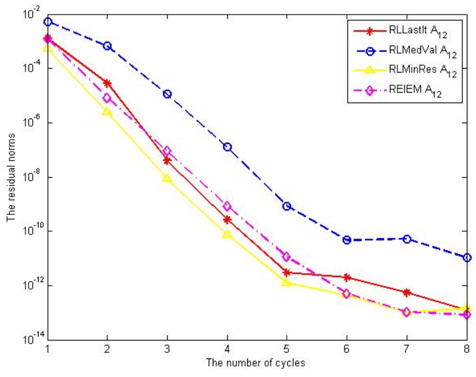
(b) dimensions 10000; $\delta = 5$



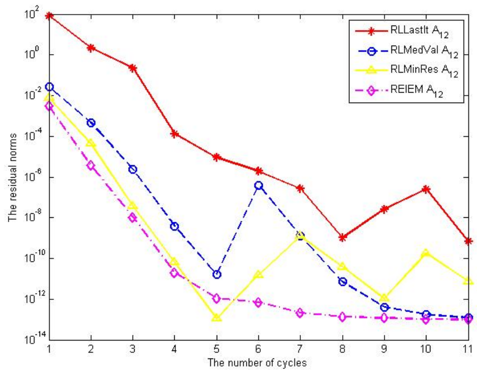
(c) dimensions 20000; $\delta = 0.2$



(d) dimensions 20000; $\delta = 5$

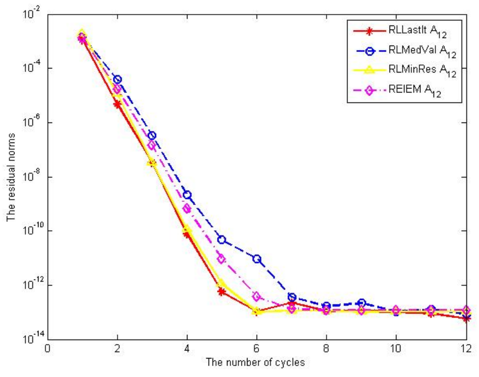


(e) dimensions 30000; $\delta = 0.2$

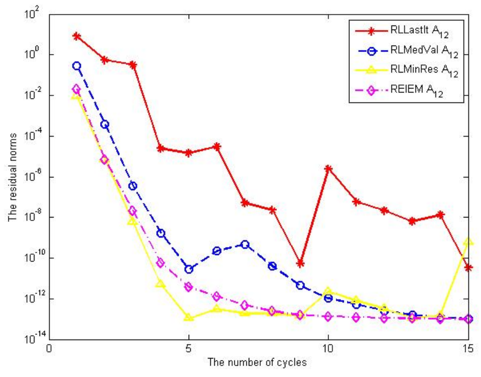


(f) dimensions 30000; $\delta = 5$

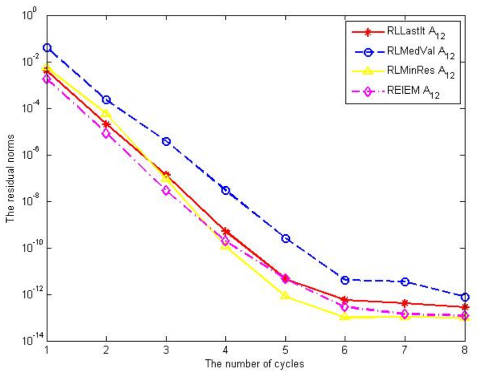
Figure 4.17: *The performances of RLLastIt A_{12}, RLMinres A_{12}, RLMedVal A_{12} and REIEM A_{12} on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 10000 to 30000*



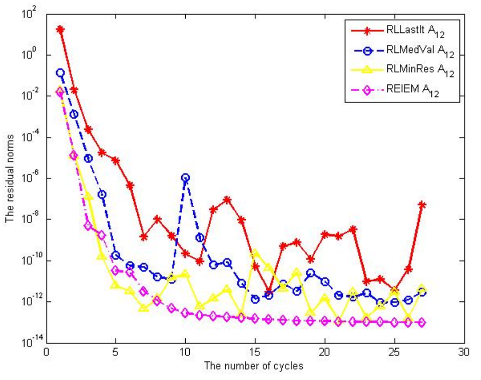
(a) dimensions 40000; $\delta = 0.2$



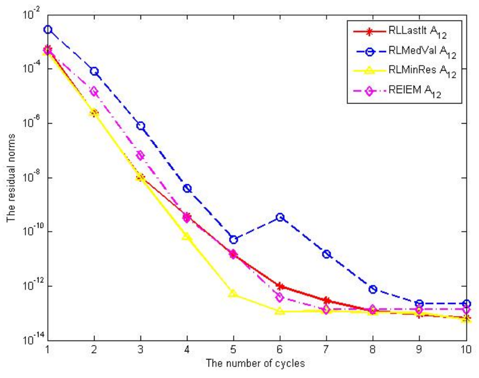
(b) dimensions 40000; $\delta = 5$



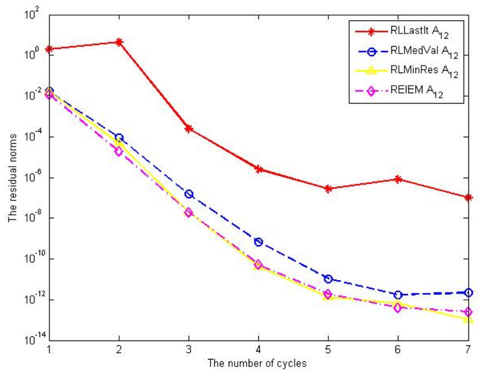
(c) dimensions 50000; $\delta = 0.2$



(d) dimensions 50000; $\delta = 5$

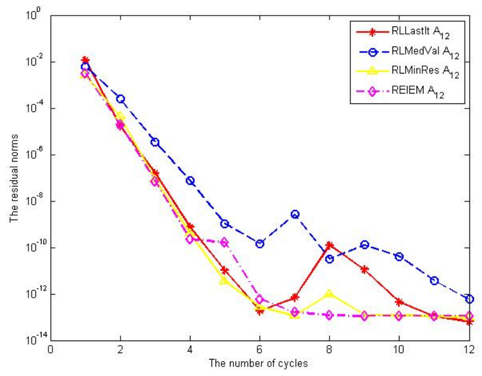


(e) dimensions 60000; $\delta = 0.2$

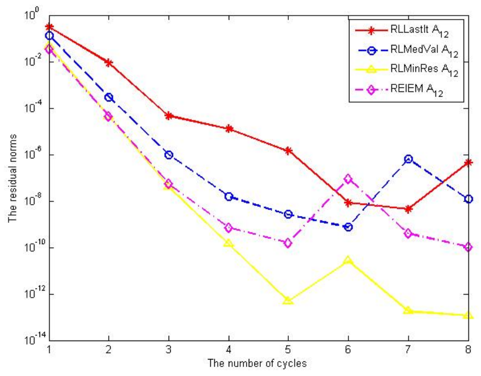


(f) dimensions 60000; $\delta = 5$

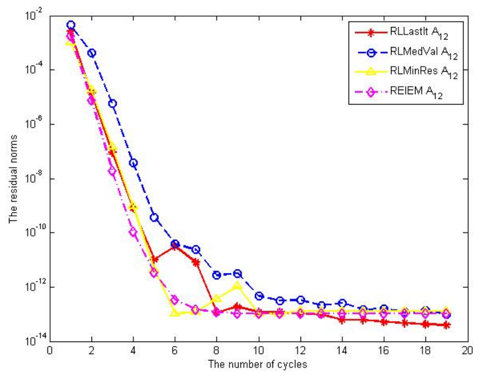
Figure 4.18: *The performances of RLLastIt A_{12}, RLMinres A_{12}, RLMedVal A_{12} and REIEM A_{12} on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 40000 to 60000*



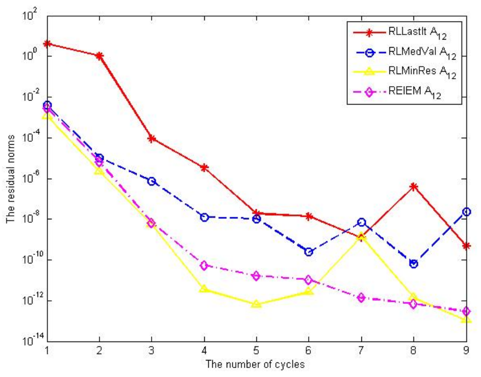
(a) dimensions 70000; $\delta = 0.2$



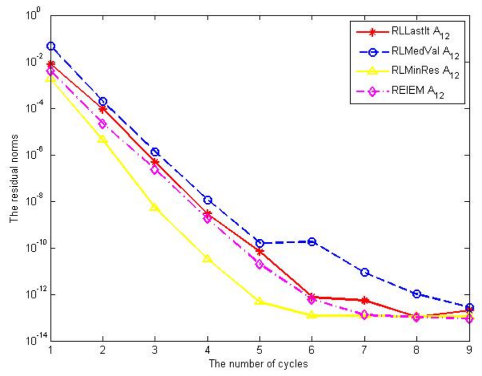
(b) dimensions 70000; $\delta = 5$



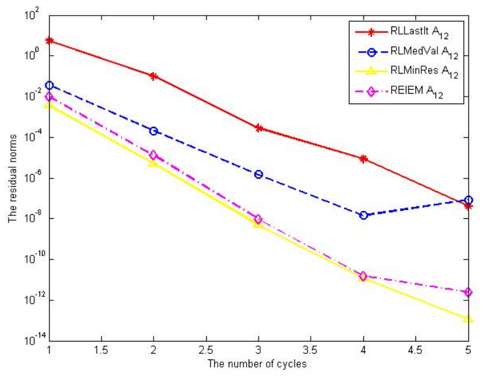
(c) dimensions 80000; $\delta = 0.2$



(d) dimensions 80000; $\delta = 5$

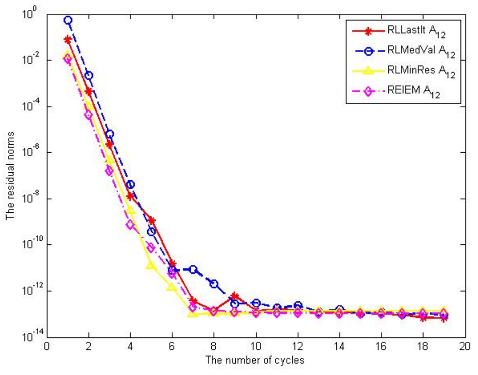


(e) dimensions 90000; $\delta = 0.2$

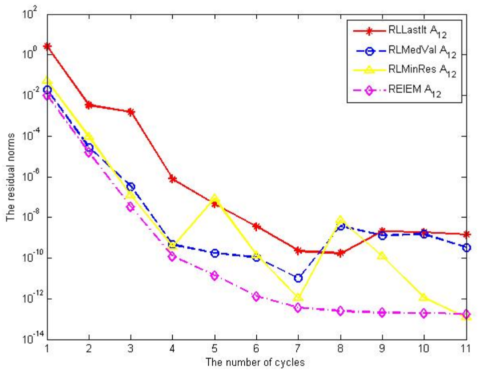


(f) dimensions 90000; $\delta = 5$

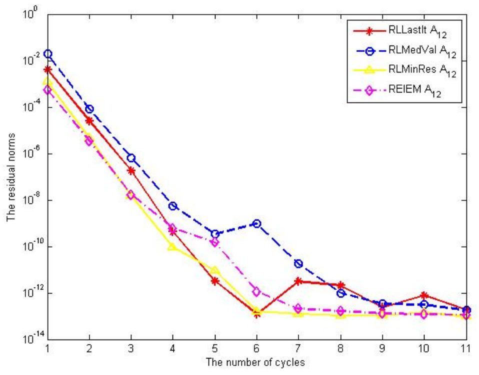
Figure 4.19: *The performances of RLLastIt A_{12}, RLMinres A_{12}, RLMedVal A_{12} and REIEM A_{12} on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 70000 to 90000*



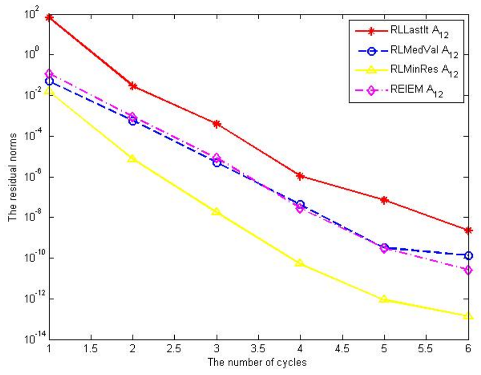
(a) dimensions 100000; $\delta = 0.2$



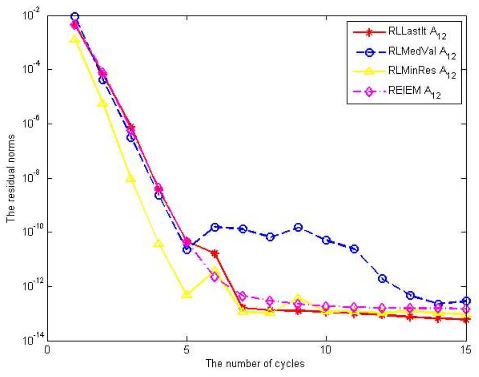
(b) dimensions 100000; $\delta = 5$



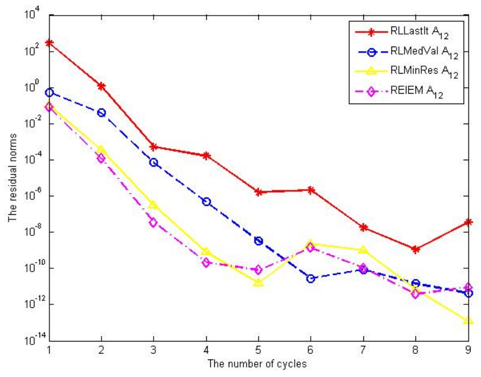
(c) dimensions 200000; $\delta = 0.2$



(d) dimensions 200000; $\delta = 5$

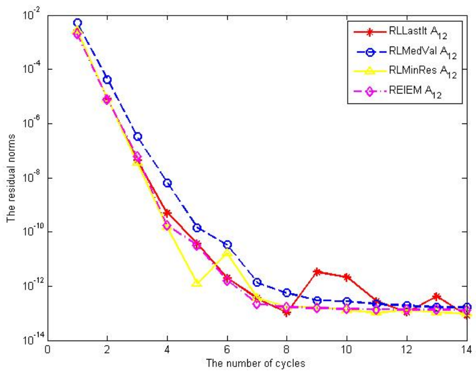


(e) dimensions 300000; $\delta = 0.2$

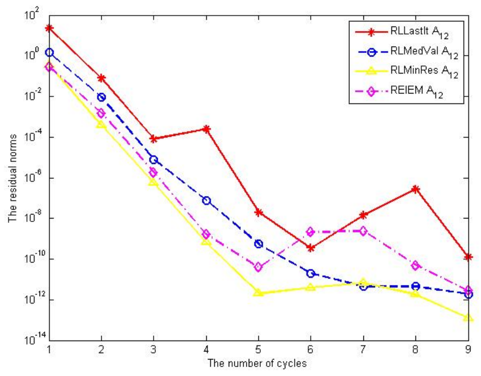


(f) dimensions 300000; $\delta = 5$

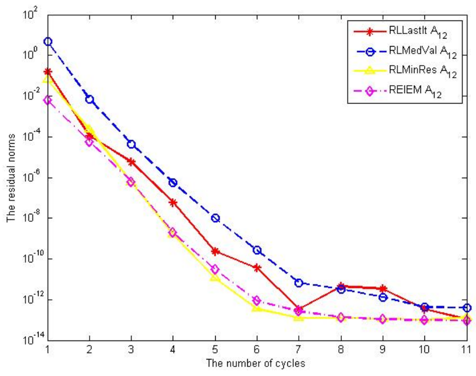
Figure 4.20: *The performances of RLLastIt A_{12}, RLMinres A_{12}, RLMedVal A_{12} and REIEM A_{12} on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 100000 to 300000*



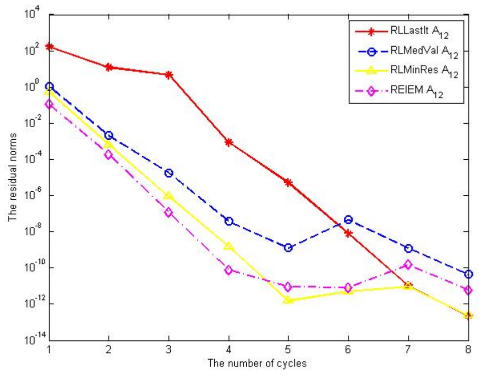
(a) dimensions 400000; $\delta = 0.2$



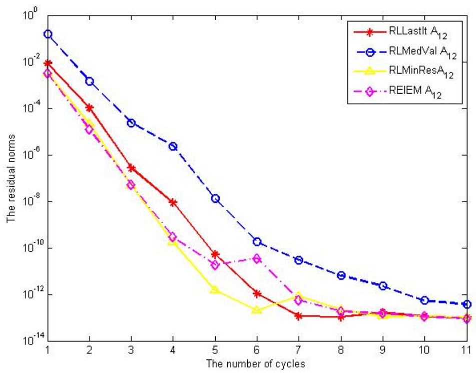
(b) dimensions 400000; $\delta = 5$



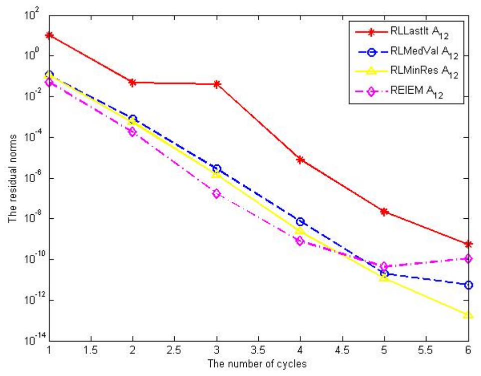
(c) dimensions 500000; $\delta = 0.2$



(d) dimensions 500000; $\delta = 5$

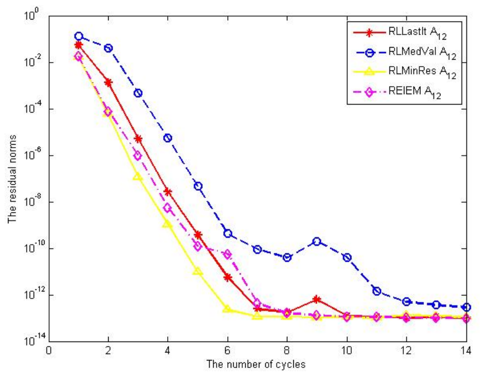


(e) dimensions 600000; $\delta = 0.2$

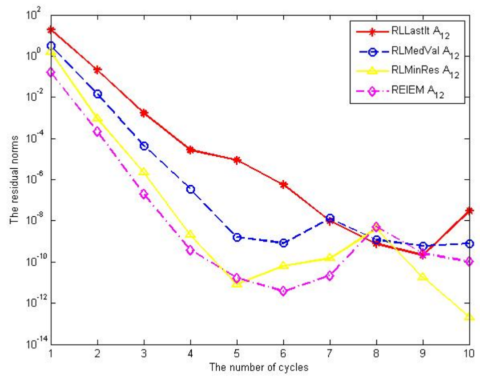


(f) dimensions 600000; $\delta = 5$

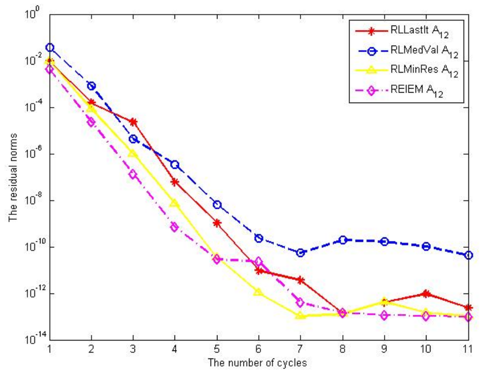
Figure 4.21: *The performances of RLLastIt A_{12}, RLMinres A_{12}, RLMedVal A_{12} and REIEM A_{12} on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 400000 to 600000*



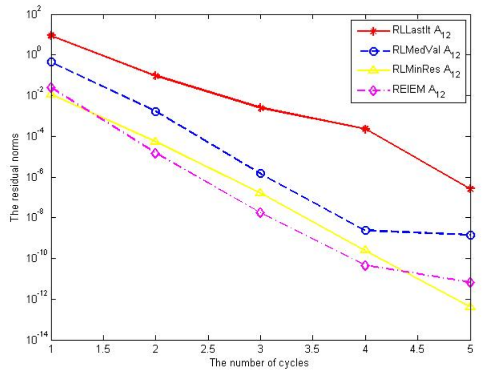
(a) dimensions 700000; $\delta = 0.2$



(b) dimensions 700000; $\delta = 5$

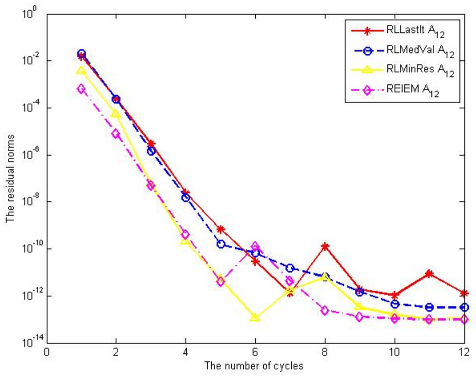


(c) dimensions 800000; $\delta = 0.2$

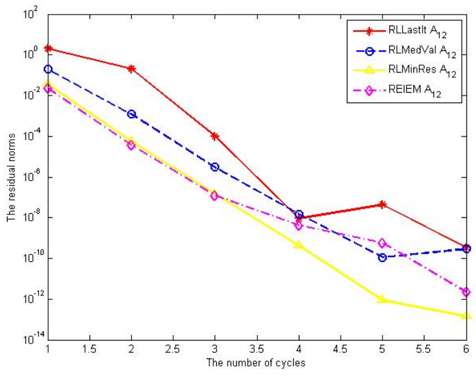


(d) dimensions 800000; $\delta = 5$

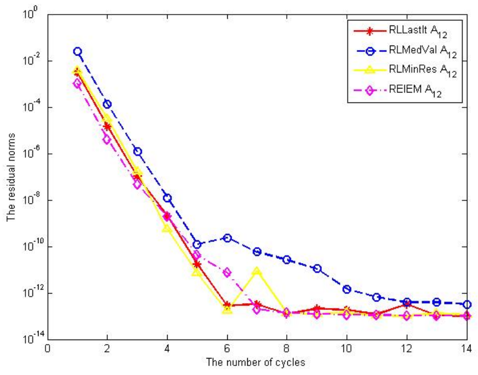
Figure 4.22: *The performances of RLLastIt A_{12}, RLMinres A_{12}, RLMedVal A_{12} and REIEM A_{12} on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 700000 and 800000*



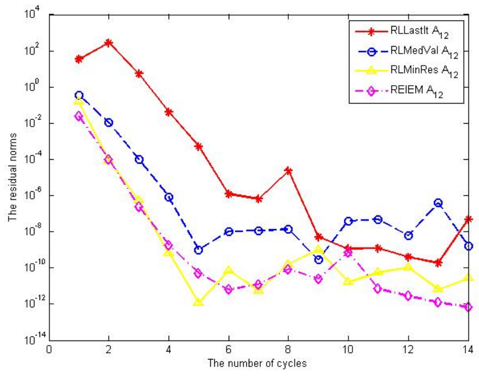
(a) dimensions 900000; $\delta = 0.2$



(b) dimensions 900000; $\delta = 5$



(c) dimensions 1000000; $\delta = 0.2$



(d) dimensions 1000000; $\delta = 5$

Figure 4.23: *The performances of RLLastIt A_{12}, RLMinres A_{12}, RLMedVal A_{12} and REIEM A_{12} on SLE's for $\delta = 0.2$ and $\delta = 5$, dimensions 900000 to 1000000*

Chapter 5

Solving Large Scale SLE's on a Cloud Computing Platform

Large scale SLE's occur routinely in a variety of applications ranging from engineering to finance and economics. It is therefore, important to show that our algorithms can handle such problems. Here, we investigate the parallel implementation on the Cloud to solve SLE's up to a million variables.

5.1 Parallel Computing

Parallel processing and programming are now well developed to the point where even a novice can solve their problems on parallel and distributed computing platforms. Recall that the aim of parallel processing is to speed-up the solution of problems and applications where traditional sequential programming is not effective. However, we must be aware that parallel processing does not always deliver because applications and problems are not necessarily pre-disposed to parallelisation. In other words, if an application is inherently difficult to segment and solve in parallel, no interesting speed-ups can be achieved. It is also important to know that in a distributed/parallel machine, not all processors (nodes) are necessarily of the same specification. Some may be faster than others because of design, age, configuration, operating system etc. This means that the parallel machine may not perform as one expects. It is also important to recall Amdahl's Law, [1], which stipulates that a parallel machine is only as fast as its slowest node! This is a universal law which applies in other areas. For instance, a chain is only as strong as its weakest link. Other issues have a great effect on performance as well, outside the quality of the processors themselves (hardware); these include load-balancing, good programming, code optimisation and other things, [68]. The platform deals with it behind the scenes. However, it still remains to understand the protocols of the parallel processing provider be they the local platform or outside and commercial one, [69].

The solution of large scale SLE's is time consuming, particularly on single processor machines. Here we will try to exploit multiple processors and powerful vector-oriented hardware to tackle this issue. In general, parallel algorithms can be created by reformulating standard algorithms or by discovering new ones, [42]. The implementation of parallel iterative methods for solving systems of linear equations in high dimensions and other applications is well developed, [25, 60, 63, 64, 70, 80]. Here, we rely on a parallel environment provided in Matlab to implement our algorithms which are described in previous chapters.

5.1.1 Parallel Computing in Matlab

Parallel Matlab is an extension of Matlab that takes advantage of multi-core desktop machines and clusters. It allows us to solve large problems in parallel and to handle data-intensive problems using multi-core processors, Graphics Processing Units (GPU), and computer clusters, [57]. According to [57], by default, the toolbox provides twelve workers to execute applications locally on a multi-core desktop. It means that without recoding, it allows to run the same application on a computer cluster or a grid computing service. Matlab has built-in functions for exploiting parallelism; (1) *parfor-loop* is used to run parallel algorithms tasks on multiple processors, and (2) SPMD, or Single Program Multiple Data, is used to handle large data sets and data parallel algorithms. Typically, the *parfor-loop* is used when all iterations are completely independent of each other. SPMD, however, requires communication or synchronization between workers when running a program on multiple data sets.

Running a program in parallel Matlab can be done locally on a PC using the Parallel Computing Toolbox (PCT), and remotely on a cluster using the Matlab Distributed Computing Server (MDCS). On a local machine, the PCT takes advantage of a maximum of 8 cores for running a program on a desktop, while MDCS controls parallel execution of MATLAB on a cluster with tens or hundreds of cores. The MDCS plays an important role in speeding up Matlab programs by running them in a high-performance computing environment, such as those now offered by Amazon EC2 and other cloud computing services, [57]. To solve larger problems, it also can be used to scale up the computer cluster on the cloud computing services. When using MDCS on the cloud computing services, we should be aware of the license arrangements. The users should refer to [55] to understand how MDCS works on the cloud services and should read [56] to understand several licensing options for MDCS.

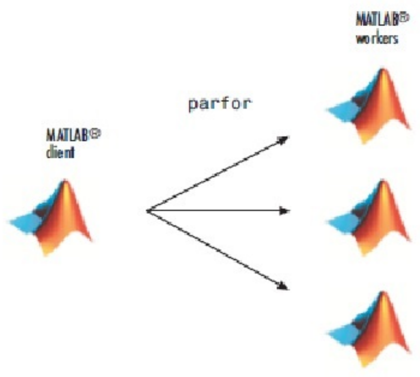


Figure 5.1: *The concept of parallel tasks on several CPUs, [57]*

5.1.2 Understanding the *parfor*-loop Function in Parallel Matlab

The *parfor*-loop is useful in situations where many loop iterations of a simple calculation are required. It divides the loop iterations into groups so that each worker executes some portion of the total number of iterations. The *parfor*-loop is also useful when the loop iterations take a long time to execute, because the workers can execute iterations simultaneously, [57]. In principal, when using the *parfor*-loop, the data is sent from the client to workers, and the results are sent back to the client and pieced together. This is illustrated in Figure 5.1.

To begin using PCT Matlab, for instance the *parfor*-loop, we first open the Matlab pool. This reserves a collection of Matlab workers sessions to run the loop iterations. To do so, we write the command :

```
matlabpool open 4
```

Matlab returns the following :

Starting matlabpool using the 'local' configuration ... connected to 4 labs.

It means that there are 4 workers active at the moment.

Next, we call our program which contains *parfor*-loop in it. Suppose we have the following code

```
FOR $i = 1 : 1024$
    $A(i) = sin(i * 2 * \pi/1024)$;
    plot(A);
END
```

We can modify the code in order to run it in parallel by using *parfor* statement as follows

```
PARFOR $i = 1 : 1024$
    $A(i) = sin(i * 2 * \pi/1024)$;
    plot(A);
END
```

The only difference is the keyword *parfor* which replaces *for*. Note here that each iteration must be completely independent of all other iterations. In the example above, the worker calculating the value of $A(100)$ might not be the same worker calculating $A(500)$. Also, in parallel processing, we can not guarantee a given order of calculations, i.e. $A(900)$ might be calculated before $A(400)$, [57].

The parallel process is ended by writing instructions in the command window.

```
MATLABPOOL CLOSE
```

5.2 Running EIEMLA in Parallel

We are concerned with the stability of the new approach when solving large scale problems (up to 1000000 variables). To run EIEMLA using the cloud computing service, the code should first be parallelized. Often parallelisation is done by hand by the user as in [70]. Here, the pain of parallelisation is taken away by

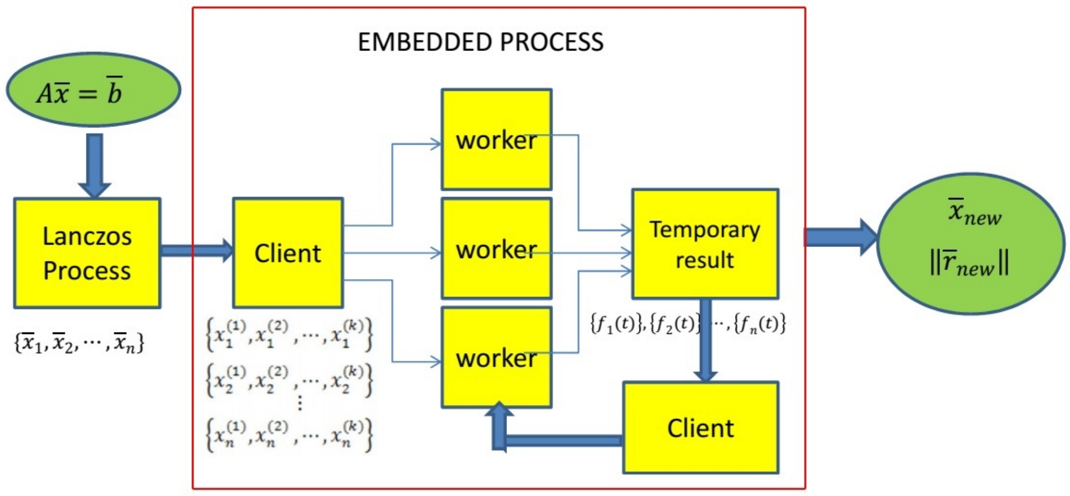

EMBEDDED PROCESS
$A\overline{x} = \overline{b}$
Lanczos Process
$\{\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_n\}$
Client
$\{x_1^{(1)}, x_1^{(2)}, \cdots, x_1^{(k)}\}$
$\{x_2^{(1)}, x_2^{(2)}, \cdots, x_2^{(k)}\}$
$\{x_n^{(1)}, x_n^{(2)}, \cdots, x_n^{(k)}\}$
worker
worker
worker
Temporary result
$\{f_1(t)\}, \{f_2(t)\}, \cdots, \{f_n(t)\}$
Client
$\overline{x}_{new}$
$\|\overline{r}_{new}\|$

Figure 5.2: *The embedded process in Lanczos algorithms*

the parallel environment. In Matlab, it is achieved using the *parfor*-loop function. This is illustrated in Figure 5.2.

First, the system $A\mathbf{x} = \mathbf{b}$ is processed by Lanczos algorithm to generate a sequence $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}$, of approximate solutions. The sequence is then used as an input in the embedded process. In the client box, the re-arranged sequence is sent to the workers where the interpolation and extrapolation of the sequence by using PCHIP, [35], is carried out. There are n data sets that need to be interpolated in this stage. As said earlier, the order of computations is not preset. The amount of processing carried out on a given worker depends on the speed of the processor and the load balancing implemented by the master processor. After the whole embedded process is finished, the new approximate solution and the corresponding residual norm are produced and then sent back to the client as the final output.

5.3 Numerical Results

This section presents the numerical results of the implementation of EIEMLA in a parallel system. The comparison of the computation time between using *for*-loop and *parfor*-loop is presented. We also present results obtained by restarting EIEMLA (REIEMLA) in a parallel environment. We particularly compare the performance of REIEMLA, in terms of CPU time, when run on a parallel machine and on a sequential one. The test problems are solved under MATLAB 2012b on Unix0 system provided by the University of Essex which includes hardware that consists of 4 x AMD Opteron(tm) processors with 2.20 GHz speed and 48 cores, 128 GB RAM, and a 1000 Mbps ethernet interface. The Matlab PCT license is available in this system with a maximum of 8 local workers. For running the sequential algorithm, we used Matlab 2013a on a machine with 12 GB RAM.

5.3.1 Embedded Interpolation and Extrapolation Model in Orthodir Running on Parallel Systems

Table 5.1 presents the computation time of the embedded interpolation and extrapolation model (EIEM) Orthodir algorithm in both parallel and serial environments. We solved several problems of dimensions ranging from 10000 to 1000000. As we can see here, the table consists of 5 columns each of which presents respectively the dimensions of the problems, the residual norms, the computation time of EIEM Orthodir with *for*-loops and *parfor*-loops, and the speed up. The speed up is calculated by taking the ratio of EIEM Orthodir with *for*-loops CPU times and those of *parfor*-loops.

In general, the use of *parfor*-loops is able to reduce the execution time signifi-

Table 5.1: *Comparison of EIEM Orthodir in parallel and sequential environments*

Dim	Residual Norm	Processing Time (sec.)		Speed up
n	$\|\|\mathbf{r}_{model}\|\|$	Sequential EIEM Orthodir	Parallel EIEM Orthodir	x (times)
10000	4.0804	5.2229	1.2959	4.03
20000	2.0776	10.5466	2.2446	4.69
30000	14.4011	15.8540	3.2166	4.92
40000	6.7584	20.8902	3.9129	5.34
50000	3.3782	26.2809	4.9604	5.29
60000	12.3680	31.5569	5.8258	5.42
70000	5.1140	36.7257	6.8410	5.37
80000	1.3536	41.9938	7.8136	5.37
90000	5.8349	47.4259	8.8534	5.36
100000	1.0606	52.5293	9.7401	5.39
200000	5.0090	105.7368	19.2961	5.48
300000	30.4925	157.4330	28.4763	5.53
400000	30.8407	210.8508	38.6017	5.46
500000	44.7364	262.4477	49.2533	5.33
600000	71.9335	316.1252	56.3174	5.61
700000	62.2338	368.3247	65.9906	5.58
800000	16.7394	422.8909	74.7826	5.65
900000	34.0717	476.6024	83.9225	5.67
1000000	18.1782	549.0482	101.2656	5.42

cantly. To solve SLE's with dimensions 10000, for instance, EIEM Orthodir with *parfor*-loops runs four times faster than with *for*-loops. In addition, for dimensions 20000 and 30000, the speed up is 5 fold. For dimensions 40000 to 1000000, the parallel program is 5 times, sometimes 6 times, faster than the sequential one. These comparisons are clearly seen in Figure 5.3.

5.3.2 REIEMLA in Parallel Systems

In this section, we report on experiments with restarting EIEM (REIEM) Orthodir on a parallel architecture. We used *parfor*-loops as a means to parallelize our codes. Comparisons are on computation times of the algorithms with *parfor*-loop and *for*-loops. Here again, problems are ranging from 100000 to 1000000 dimensions.

It can be seen in Table 5.2, that in general, REIEM Orthodir runs in parallel significantly faster than sequentially. For instance, speed up is 4.04 for dimensions

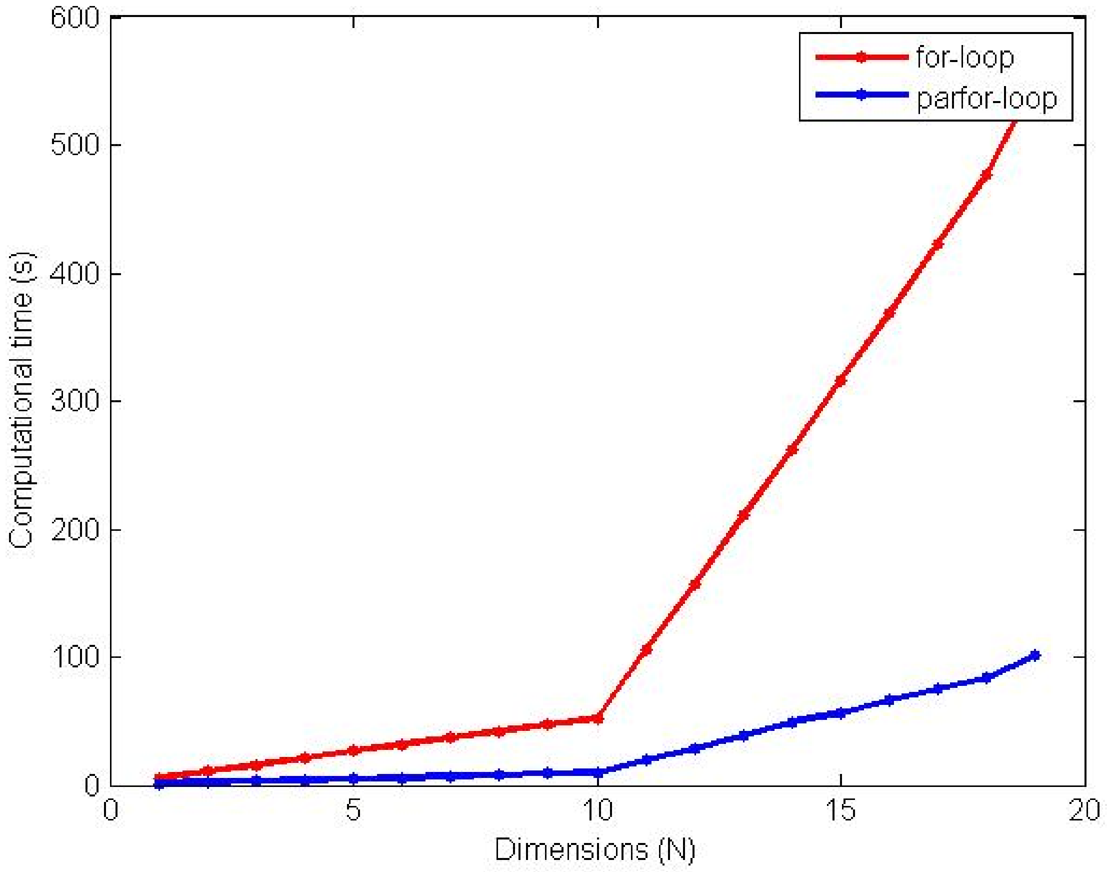


Figure 5.3: *Comparison of for-loop and parfor-loop execution speeds*

Table 5.2: *Comparison of REIEM Orthodir in parallel and in sequential environments*

Dim	Residual Norms	Processing Time (sec.)		Speed up
n	$\|\|\mathbf{r}_{model}\|\|$	Sequential REIEM Orthodir	Parallel REIEM Orthodir	x (times)
100000	$6.7572E-14$	$3.9749E+02$	98.3595	4.04
200000	$7.3917E-14$	$9.0306E+02$	$2.3294E+02$	3.88
300000	$5.7715E-14$	$1.0328E+03$	$3.6675E+02$	2.82
400000	$6.2753E-14$	$3.3668E+04$	$4.9720E+02$	6.77
500000	$7.5120E-14$	$3.5596E+03$	$5.5249E+02$	6.44
600000	$6.9701E-14$	$2.8829E+03$	$7.2046E+02$	4.00
700000	$9.6797E-14$	$4.0071E+03$	$8.8016E+02$	4.55
800000	$9.0274E-14$	$4.5529E+03$	$9.8848E+02$	4.61
900000	$7.8603E-14$	$5.5275E+03$	$1.1893E+03$	4.65
1000000	$7.8631E-14$	$6.1636E+03$	$1.2373E+03$	4.98

100000, meaning REIEM Orthodir solved the problems in a parallel environment 4 times faster than restarting in a sequential environment. Also, when solving 400000 and 500000 variables, REIEM Orthodir with *parfor*-loops was about 7 times and 6 times faster respectively than that with *for*-loops.

5.4 Execution of EIEMLA and REIEMLA on the Cloud Computing Platform

In this section, we run the parallel codes on the cloud platform. We first look at the background of cloud computing.

5.4.1 Cloud Computing

It its simplest from Landis, [53], gave this definition: "Cloud computing is a style of computing on the Internet, rather than on a PC. It simply means that we can log onto a website to do whatever we might normally do on a PC". He added that cloud computing enables us to do all of our computing on the Internet such as installing, upgrading, uploading, downloading, backing up and otherwise managing physical hardware, operating systems and software. Furthermore, with cloud computing, the actual processing and computing is done by remote servers (or virtual servers) and the PC is mainly used as a way to run a Web browser, [53]

"Cloud computing is defined as a subscription-based service where we can obtain networked storage space and computer resources", according to [45]. These services include incorporating infrastructure as a service (IIaS), the platform as a service (PaS), and software as a service (SaS).

Buyya et al. [15] stated that "Cloud computing has been coined as an umbrella term to describe a category of sophisticated on-demand computing services initially offered by commercial providers, such as Amazon, Google, and Microsoft. It denotes a model on which a computing infrastructure is viewed as a "cloud", from which business and individuals access applications from anywhere in the world on demand".

According to [72], cloud computing has some essential characteristics such as on-demand self-service, broad network access, resource pooling, rapid elasticity, and measure service. The explanation of each characteristic can be referred to the author.

There are different types of cloud platforms that we can subscribe to: (1) public cloud, which can be accessed by any subscriber with an internet connection and access to the cloud space; (2) private cloud, which is established for a specific group or organization and limits access to just that group; (3) community cloud, which is shared among two or more organization that have similar cloud requirements; and (4) hybrid cloud, which is essentially a combination of at least two clouds, where the cloud included are a mixture of public, private , or community, [45, 53].

See Figures 5.4 for an illustration of the concept of cloud computing.

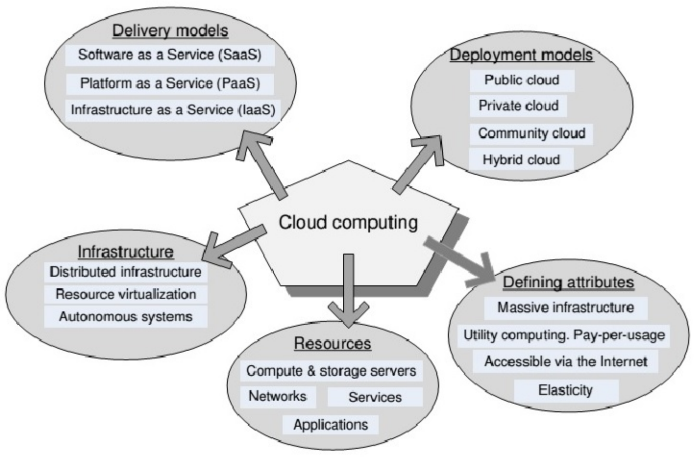
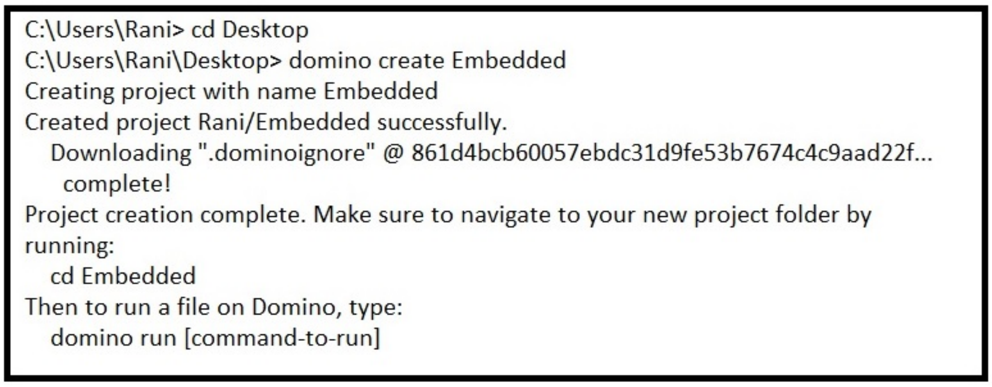


Figure 5.4: *The concept of cloud computing*

5.4.2 Domino Data Lab as a Provider

Domino Data Lab, founded by Nick Elprin in 2014, is a new cloud service that allows us to run R, Python, and Matlab codes, [27]. In this article, it has been claimed: ”Domino is a platform-as-a-service (PaaS) for data analysis, to equip a larger group of users with functionality that has typically been inaccessible to people without engineering abilities and a massive amount of time to set up infrastructure and plumbing”. He also stated : ”Domino addresses three core areas of functionality: (1) It runs R code (or Python, Julia, Matlab , and more) on the cloud without any set-up or configuration. Domino handles Amazon Machine Images (AMI) and package management, job distribution and secure data transfer. It allows us to change our hardware with one-click, or to distribute our analysis across multiple machines; (2) It automatically keeps a revision history of our projectcode, data, and results, so we can browse and reproduce past work.



```
C:\Users\Rani> cd Desktop
C:\Users\Rani\Desktop> domino create Embedded
Creating project with name Embedded
Created project Rani/Embedded successfully.
   Downloading ".dominoignore" @ 861d4bcb60057ebdc31d9fe53b7674c4c9aad22f...
     complete!
Project creation complete. Make sure to navigate to your new project folder by
running:
   cd Embedded
Then to run a file on Domino, type:
   domino run [command-to-run]
```

Figure 5.5: *Creating a project on Domino*

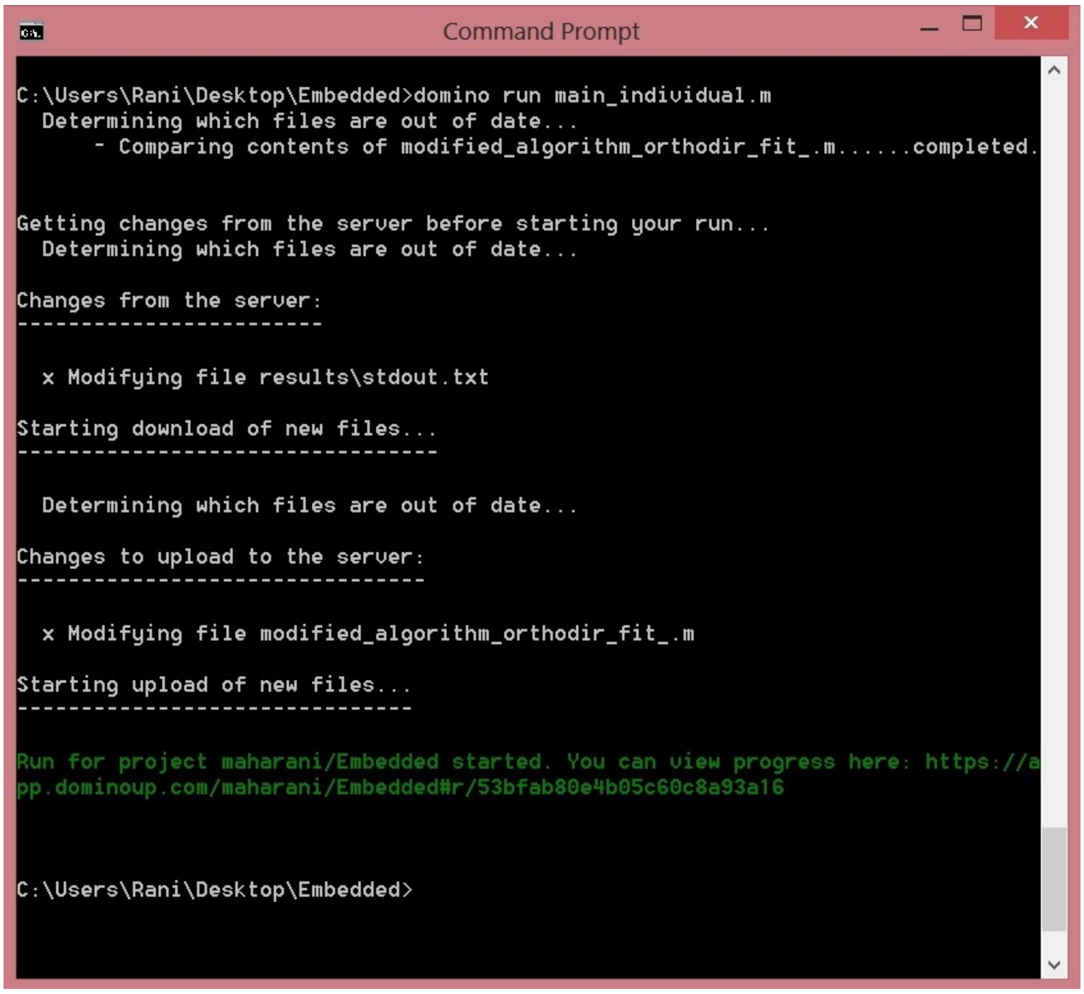
Unlike traditional source control, Domino tracks large data files, and creates a first-class association between our results (e.g. charts) and the code/data that produced them; (3) It facilitates collaboration so it is easy to share results and co-author analyses". Readers who are interested to understand how to access Domino may find informations in [28].

5.5 EIEMLA on Domino Cloud Platform

The Domino cloud was used in testing the algorithms. There are two different ways of initiating a code execution Domino, namely (1) using command Windows, and (2) directly using Domino websites.

In the first case, a Domino window is used to create project name inside which files may be added in relation to the algorithm. An execution command file is used to perform the computational run. Figure 5.5 illustrates the creating a project entitled "Embedded" and Figure 5.6 demonstrates the sequence of execution.

In the second case, the Domino website is used to execute a code directly from some menus which are available on it. Figure 5.7 illustrates the Domino



```
C:\Users\Rani\Desktop\Embedded>domino run main_individual.m
  Determining which files are out of date...
      - Comparing contents of modified_algorithm_orthodir_fit_.m......completed.


Getting changes from the server before starting your run...
  Determining which files are out of date...

Changes from the server:
------------------------

  x Modifying file results\stdout.txt

Starting download of new files...
---------------------------------

  Determining which files are out of date...

Changes to upload to the server:
--------------------------------

  x Modifying file modified_algorithm_orthodir_fit_.m

Starting upload of new files...
-------------------------------

Run for project maharani/Embedded started. You can view progress here: https://a
pp.dominoup.com/maharani/Embedded#r/53bfab80e4b05c60c8a93a16



C:\Users\Rani\Desktop\Embedded>
```

Figure 5.6: *Running a project on Domino*

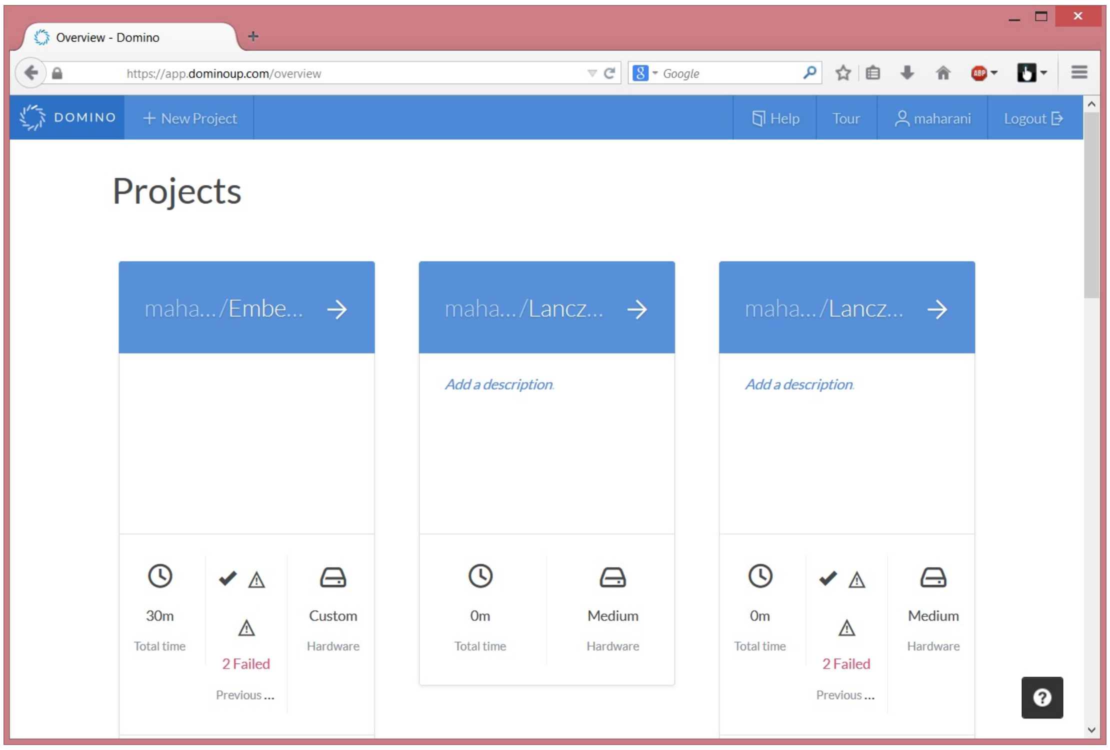


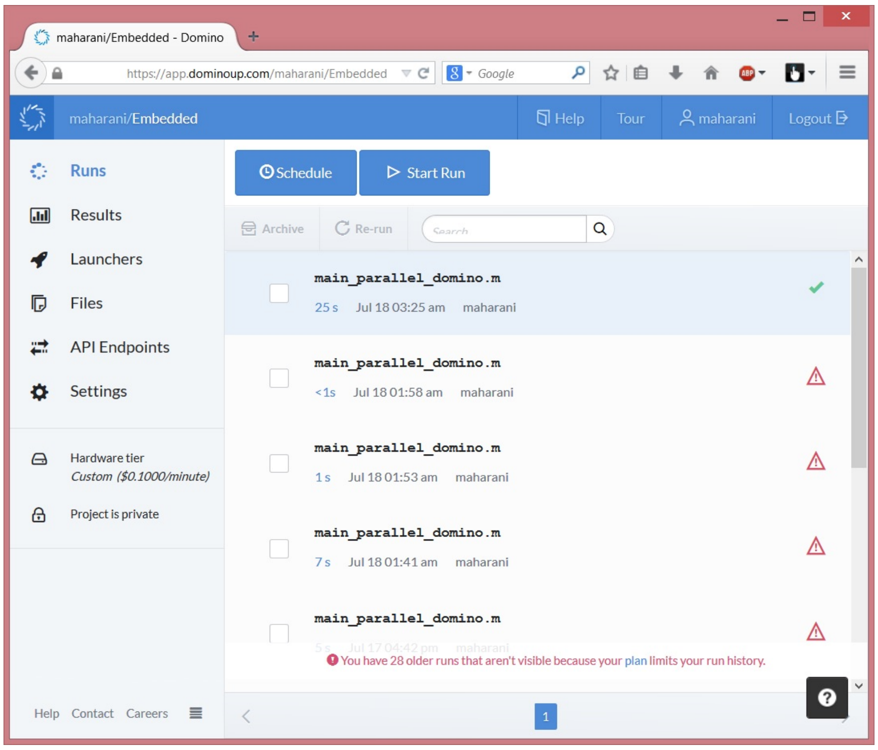
Figure 5.7: *The general view of Domino*

performance when we opened our account. The project entitled ”Embedded” has been set as well as some related files.

Figure 5.8 describes some menus available on Domino, such as ”Runs”, ”Results”, ”Launchers”, ”APIEndpoints”, and ”Settings”. Also, as can be seen on the figure, the last two menus at the bottom show our usage of hardware on Domino and our project property. Here we discuss some Domino properties which are related to our study.

Figure 5.9 shows the ”Files” menu which contains some codes such as ”main parallel domino.m”, and figure 5.10 demonstrates the ”Settings” menu which enables us to choose the variety of hardwares to run our codes.

To run a code on Domino, we simplify click on ”Runs” menu, as can be seen in Figure 5.11, which is also the charge of the service started. If the execution is



main_parallel_domino.m
25 s Jul 18 03:25 am maharani

main_parallel_domino.m
<1s Jul 18 01:58 am maharani

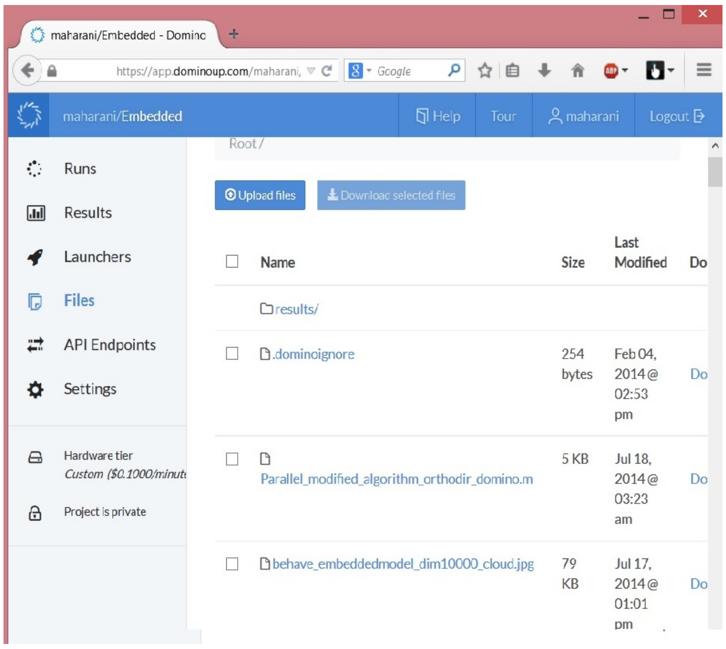
main_parallel_domino.m
1 s Jul 18 01:53 am maharani

main_parallel_domino.m
7 s Jul 18 01:41 am maharani

main_parallel_domino.m

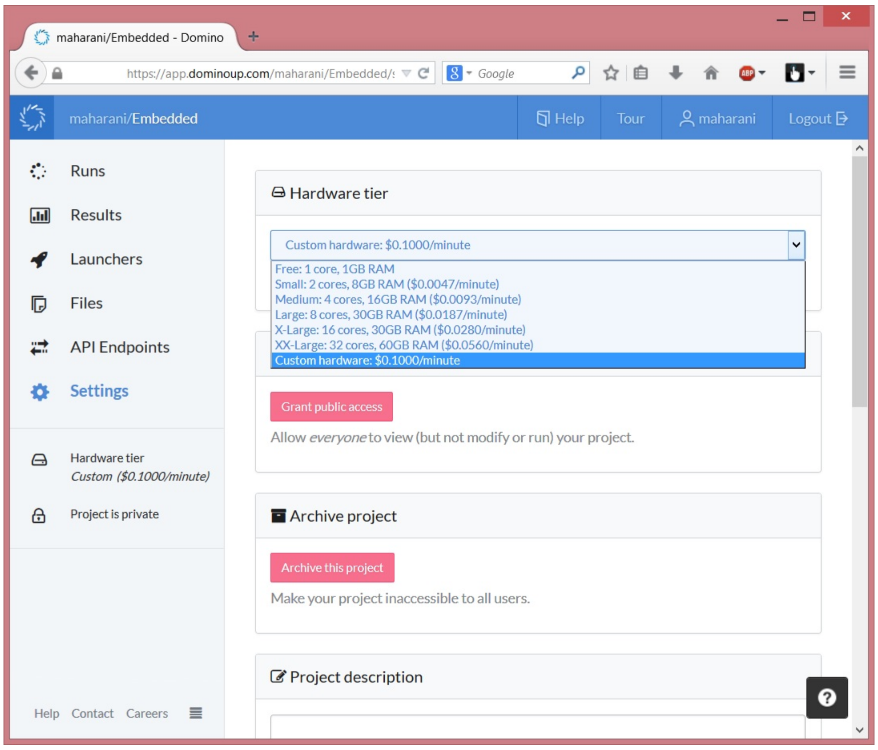
You have 28 older runs that aren't visible because your plan limits your run history.

Figure 5.8: *The Domino menu*



Name	Size	Last Modified	Do
results/			
.dominoignore	254 bytes	Feb 04, 2014 @ 02:53 pm	Do
Parallel_modified_algorithm_orthodir_domino.m	5 KB	Jul 18, 2014 @ 03:23 am	Do
behave_embeddedmodel_dim10000_cloud.jpg	79 KB	Jul 17, 2014 @ 01:01 pm	Do

Figure 5.9: *The Domino files menu*



Hardware tier
Custom ($0.1000/minute)

Project is private

Hardware tier

Custom hardware: $0.1000/minute

- Free: 1 core, 1GB RAM
- Small: 2 cores, 8GB RAM ($0.0047/minute)
- Medium: 4 cores, 16GB RAM ($0.0093/minute)
- Large: 8 cores, 30GB RAM ($0.0187/minute)
- X-Large: 16 cores, 30GB RAM ($0.0280/minute)
- XX-Large: 32 cores, 60GB RAM ($0.0560/minute)
- Custom hardware: $0.1000/minute

Grant public access

Allow *everyone* to view (but not modify or run) your project.

Archive project

Archive this project

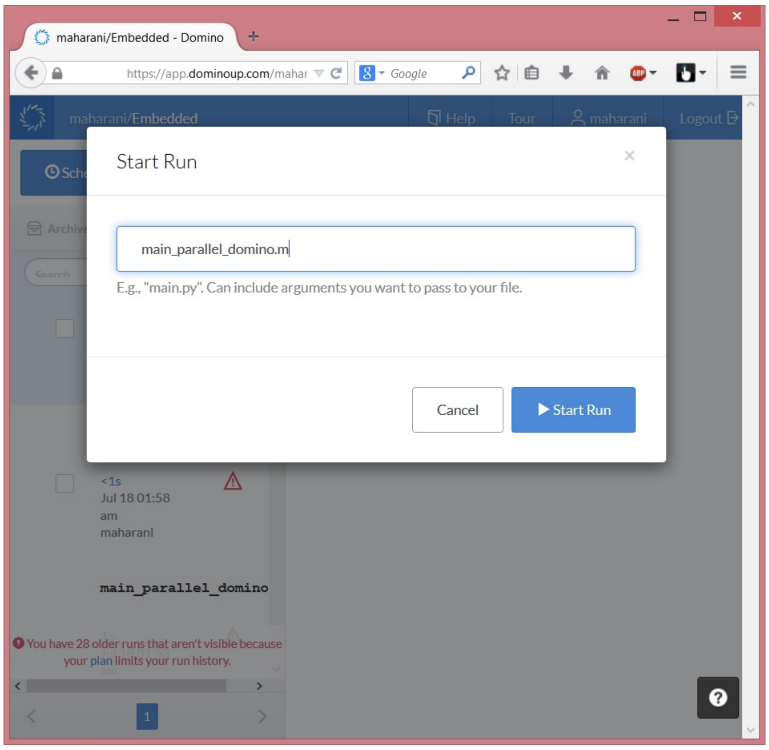
Make your project inaccessible to all users.

Project description

Figure 5.10: *The setting menu on Domino*

successful, we will get a notice as in Figure 5.12, and the results can be viewed afterwards (Figures 5.13 and 5.14). As mentioned earlier, we also receive the notifications of our results via email.

Domino has several options of hardwares, from 1 GB RAM and 1 core only, to the XX large which contains 60 GB RAM and 32 cores. If the users need more RAM than those available the Domino team will set up a special hardware, the so-called ”Custome hardware”. In this study, we used the X-large and the XX-large hardware which respectively contain 16 and 32 cores with 30 and 60 GB RAM. We compared with Unix2 available at the university of Essex super computer with 48 cores and 256 GB shared RAM.



main_parallel_domino.m

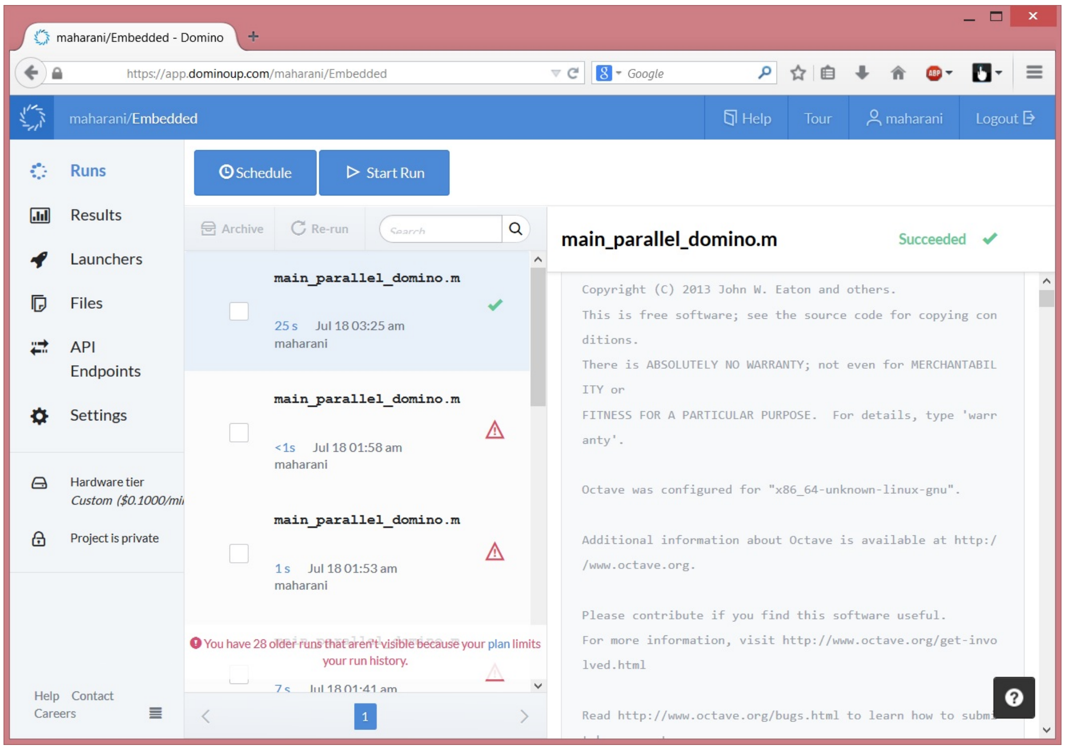
E.g., "main.py". Can include arguments you want to pass to your file.

<1s Jul 18 01:58 am maharani

main_parallel_domino

You have 28 older runs that aren't visible because your plan limits your run history.

Figure 5.11: *The Domino runs menu*



Hardware tier *Custom ($0.1000/mi*

Project is private

main_parallel_domino.m
25 s Jul 18 03:25 am
maharani

main_parallel_domino.m
<1s Jul 18 01:58 am
maharani

main_parallel_domino.m
1 s Jul 18 01:53 am
maharani

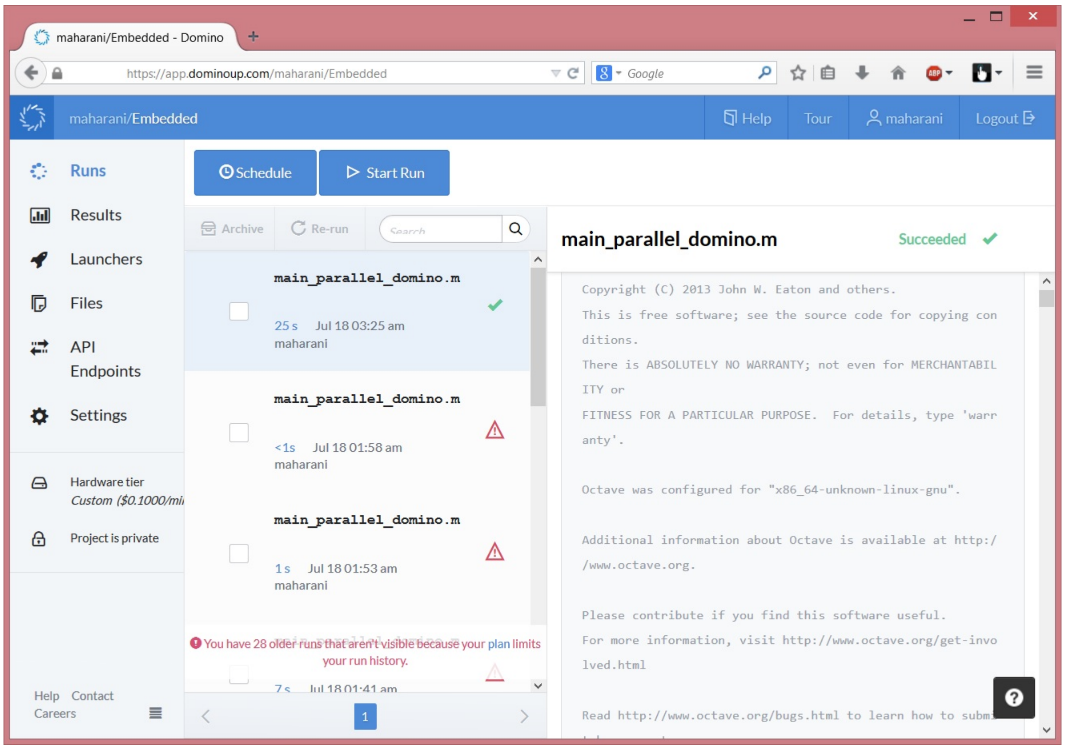
You have 28 older runs that aren't visible because your plan limits your run history.

7 s Jul 18 01:41 am

main_parallel_domino.m Succeeded ✓

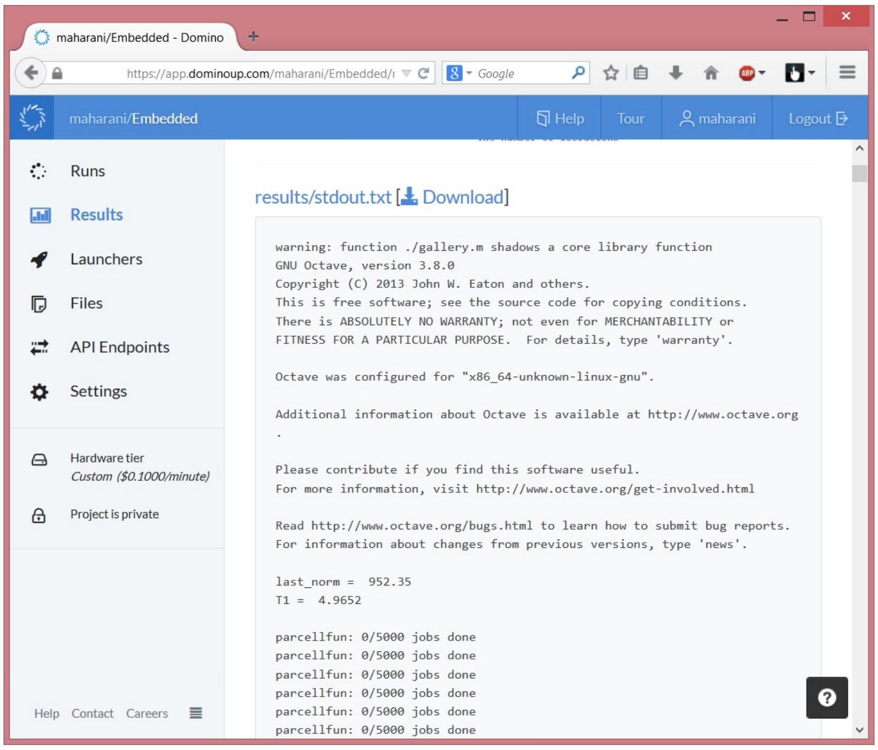
```
Copyright (C) 2013 John W. Eaton and others.
This is free software; see the source code for copying con
ditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABIL
ITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warr
anty'.

Octave was configured for "x86_64-unknown-linux-gnu".

Additional information about Octave is available at http:/
/www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-invo
lved.html

Read http://www.octave.org/bugs.html to learn how to submi
```

Figure 5.12: *The notifications of our running*



```
Copyright (C) 2013 John W. Eaton and others.
This is free software; see the source code for copying con
ditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABIL
ITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warr
anty'.

Octave was configured for "x86_64-unknown-linux-gnu".

Additional information about Octave is available at http:/
/www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-invo
lved.html

Read http://www.octave.org/bugs.html to learn how to subm
```

Figure 5.13: *The figure results on Domino*



```
warning: function ./gallery.m shadows a core library function
GNU Octave, version 3.8.0
Copyright (C) 2013 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-unknown-linux-gnu".

Additional information about Octave is available at http://www.octave.org
.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

last_norm =  952.35
T1 =  4.9652

parcellfun: 0/5000 jobs done
parcellfun: 0/5000 jobs done
parcellfun: 0/5000 jobs done
parcellfun: 0/5000 jobs done
parcellfun: 0/5000 jobs done
parcellfun: 0/5000 jobs done
```

Figure 5.14: *The text result on Domino*

Table 5.3: *Performance of EIEM Orthodir on the Cloud when solving SLEs with* $\delta = 0.2$

Dim	Computational Time (seconds)	
n	Domino Cloud (16 cores)	Domino Cloud (32 cores)
100000	61.234	64.613
200000	$1.2294E+02$	$1.3160E+02$
300000	$1.8518E+02$	$1.9988E+02$
400000	$1.9288E+02$	$2.0844E+02$
500000	$3.0886E+02$	$3.1468E+02$
600000	$3.6125E+02$	$4.1542E+02$
700000	$4.4078E+02$	$4.4470E+02$
800000	$4.9064E+02$	$4.9886E+02$
900000	$5.4645E+02$	$4.8818E+02$
1000000	$5.9574E+02$	$5.6232E+02$

5.5.1 EIEM Orthodir on Domino Cloud : Numerical Results

The results are recorded in Tables 5.3 and 5.4. Note, Speed up 1* is the ratio of the Domino Cloud with 16 cores CPU time and the CPU time of the local parallel machine; Speed 2* is the ratio of Domino Cloud with 32 cores CPU time and that of this same local parallel machine. This platform is the Unix2 machine of the University of Essex which supports Matlab. The way we access it is similar to the way we access the Domino cloud platform or any cloud providers for that matter. Here we present experimental results comparing performance of EIEM Orthodir on this local machine and on the Domino cloud computing.

As can be seen in Table 5.3, in most cases, the processing time of 32 cores on Domino cloud is slower than 16 cores. This appears, for instance, on dimensions ranging from 100000, to 800000. For problems of dimensions 900000 and 1000000, however, the 32 cores is slightly faster than the 16 cores. This means that more processors do not necessarily translate into performance because of many factors including communication costs.

Table 5.4: *CPU times of EIEM Orthodir on the local platform and on the Domino Cloud*

Dim	Local Platform (8 workers)	Speed up 1*	Speed up 2*
n	Computational Time (seconds))	x(times)	x(times)
100000	9.7401	6.29	6.63
200000	19.2961	6.37	6.82
300000	28.4763	6.5	7.02
400000	38.6017	4.9	5.39
500000	49.2533	6.27	6.39
600000	56.3174	6.41	7.38
700000	65.9906	6.68	6.74
800000	74.7826	6.56	6.67
900000	83.9225	6.51	5.82
1000000	101.2656	5.88	5.55

Interestingly, the local platform is more efficient than the Domino Cloud, with both 16 cores and 32 cores, as can be seen in Table 5.4. The processing time in the local machine is consistently less than that of the Domino Cloud. For instance, when solving 100000 dimensional problems, the execution time of the EIEM Orthodir on the university machine is 6.29x faster than the processing time on the Domino cloud with 16 cores. This factor is even bigger when we used 32 cores on Domino cloud; it is 6.6x faster. Speed up 1 and speed up 2, however, fell to 4.9 and 5.39 respectively, when solving 400000 problems. The rest of the results show the same trend.

5.5.2 REIEM Orthodir on the Domino Cloud: Numerical Results

Looking at Table 5.5, overall, the trend is similar to that of the previous section; the execution time on the University Cloud is consistently less than that on Domino Cloud. One thing to highlight is that, the 32 cores on Domino Cloud seems to be slower than the 16 cores in some cases, while in some others it is faster. For instance, for dimensions ranging from 50000 to 80000, 100000, 300000, and 400000,

Table 5.5: *Performance of Parallel REIEM Orthodir on SLEs with* $\delta = 0.2$

Dim	Computational Time (sec.)			Speed up 1*	Speed up 2*
n	Local Platform (8 nodes)	Domino Cloud (16 cores)	Domino Cloud (32 cores)	x(times)	x(times)
50000	49.9953	$2.48E + 02$	$2.79E + 02$	4.96	5.58
60000	66.3108	$2.99E + 02$	$3.37E + 02$	4.51	5.08
70000	75.5876	$3.17E + 02$	$3.56E + 02$	4.19	4.71
80000	79.946	$4.02E + 02$	$4.46E + 02$	5.03	5.58
90000	100.6326	$4.75E + 02$	$4.65E + 02$	4.72	4.62
100000	98.3595	$4.67E + 02$	$5.11E + 02$	4.75	5.19
200000	$2.3294E + 02$	$1.052E + 03$	$9.97E + 02$	4.52	4.28
300000	$3.6675E + 02$	$1.544E + 03$	$1.650E + 03$	4.21	4.49
400000	$4.9720E + 02$	$2.325E + 03$	$2.409E + 03$	4.68	4.85
500000	$5.5249E + 02$	$2.866E + 03$	$2.650E + 03$	5.19	4.79

the 32 cores is slower than the 16 cores. However, for dimensions 90000, 200000, and 500000, the 32 cores is faster than the 16 cores. So far, we do not have enough evidence to explain why this is the case, although communications overheads, and sharing of the platform with other users may be the reason. We also suspect that our code is accessing some shared resource (e.g., a global matrix). So, it is possible that the underlying operating system is doing some locking to prevent multiple threads from accessing that resource at the same time. If that is really happening, then more threads could slow things down, [26].

5.6 Summary

In this chapter, we contribute in implementing our new approach EIEMLA and REIEMLA using parallel as well as cloud computing. Regarding the first, the experimental results show that the use of *parfor*-loop instead of *for*-loop is able to speed up the computation. For further work, it is suggested to use other parallel environments of such as SPMD, and CUDA GPU, [57].

Secondly, we have implemented EIEM Orthodir as well as REIEM Orthodir on the cloud. We used the local parallel platform at the University of Essex and Domino Cloud provider. The experimental results show that the use of local platform is more efficient than the Domino Cloud. This shows that often it is not necessary to embark on expensive ventures to get good performance.

In conclusion, it can be said that parallel processing improves the performance of our algorithms and presents and edge over the sequential implementation of these algorithms. It is also unwarranted to assume that running parallel codes on platforms with a large number of processors will deliver better speed up than platforms with a low number of nodes. This has been substantiated in this chapters where 16 nodes do better than 32.

It is also notable that we have managed to solve with algorithms we have introduced here, large scale SLE's with up to 10^6 variables.

Chapter 6

Conclusion and Further Work

6.1 Conclusion

The objectives set out in Chapter 1 have been largely achieved. We have reviewed the extensive literature concerning the Lanczos process, Lanczos-type algorithms, their implementation and their failings in particular the phenomenon known as breakdown. We have then looked at a particular approach to avoiding breakdown, namely restarting, and investigated the quality of starting points for restarting Lanczos-type algorithms to treat breakdown issue. It is important since the starting point affects the behaviour and performance of the Lanczos process. We considered three such points; the last iterate before breakdown occurs (algorithm RLLastIt), the iterate with the lowest residual norm found before breakdown (algorithm RlMinRes, and the iterate whose entries are from the median value of all iterates generated so far (algorithm RlMedVal). Restarting from the iterate with the lowest residual norm among those generated in the previous cycle, i.e. RLMinRes, showed the best performance in our experiments. A simple explanation could be that such points are nearer the true solution by virtue of their residual norm which is low.

Our experience with running Lanczos-type algorithms supports the idea that trends in sequences of iterates exist and can be exploited. These trends can be seen when the iterates, and more precisely their entries, are represented in the Parallel Coordinate System (PCS). To exploit these trends, as explained in Chapter 3, we resorted to regression. Polynomial interpolation is used to build a model that captures the trend and extrapolation is then used to generate new iterates which fall within the sequence of iterates that the Lanczos-type algorithm would generate. This approach has been implemented as the Embedded Interpolation and Extrapolation Model in Lanczos-type algorithms (EIEMLA). EIEMLA is a good device for combating breakdown. The quality of the solutions it generates is shown theoretically to be at least as good as that of the iterates on which regression is based. This is from the point of view of the residual norm as a measure of quality. Large scale problems, ranging from 1,000 to 70,000 dimensions have been solved with EIEMLA. This work contributed to the improvement of stability of Lanczos process for solving a large scale problems.

We have also investigated restarting with output from the EIEMLA; the resulting algorithm is referred to as REIEMLA. This algorithm allowed us to solve problems of up to 400,000 dimensions, coping well with the breakdown in the process. Since EIEMLA and REIEMLA can work for any Lanczos-type algorithm, we considered many of them including Orthodir, Orthores, Orthomin, A_8B_8, and REIEM A_{12}. Some test results can be found in Section 4.3.

Since the numerical solution of large scale SLE's is a time consuming exercise, we considered parallel processing to speed up the computational process. This has been addressed in Chapter 5 via the use of parallel platforms provided by Cloud Computing organisations such as Domino. Our codes written in Matlab lend themselves to parallelisation through the use of a facility provided by Matlab called Parfor. This allowed us to run EIEMLA and REIEMLA in parallel over a number of nodes up to 64. Computational gains were made although not when using 64nodes. Clearly, there is a limit to how much can be gained in this way. A more explicit parallelisation of the code, i.e. without relying on Parfor, using PVM or MPI, may prove beneficial in this respect.

Moreover, our study has successfully addressed methods for solving SLE's with larger sizes than those that have been reported in the literature particularly for the numerically stable strategies, namely restarting and switching, [29].

It is fair to say that our objective have been met on whole. The suggested methods, however, are robust in that breakdown is avoided and convergence is often achieved, but not as efficient as basic restarting and switching algorithms.

6.2 Further Works

There are a number of issues that could be followed up in this project. These include

- Testing the algorithms put forward on the eigenvalue problem; note that the Lanczos process has initially been designed for this problem.
- Some entries of iterates generated by Lanczos-type algorithms seem to settle down to their final values early on; can a test be devised to recognise such entries and take advantage of that computationally?

- In EIEMLA, the returned functions $f_i(t^*)$, for $i = 1, 2, \ldots, n$, are of the polynomial type. Can other functions work as well or better?

- Explore the possibility to extend this method to Non-Linear Systems of Equations (NLSE's).

- In all the restarting approaches considered, the number of iterations before restart is arbitrary except when restarting follows a breakdown. Using an appropriate number of iterations close to when breakdown occurs would be beneficial since we may find the true solution or get closer to it faster.

- Exploring further parallel implementations of the algorithms put forward in this thesis. An explicit parallel implementation which does not rely on ready facilities such as Parfor of Matlab may lead to greater speed-ups.

- Investigate running the algorithms on GPU parallel platforms.

Bibliography

[1] Gene M. Amdahl. Validity of the Single Processor Approach to Achieving LArge Scale Computing Capabilities. In *AFIPS '67 (Spring) Proceedings*, pages 483–485, International Business Machines Corporation, Sunnyvale, California, April 1967.

[2] N. Douglas Arnold. *A Concise Introduction to Numerical Analysis*. School of Mathematics, University of Minnesota, Minneapolis, 2001.

[3] C. Baheux. New Implementations of Lanczos Method. *Journal Of Computational and Applied Mathematics*, 57:3–15, 1995.

[4] R. Barret, M. Berry, T.F. Cahn, J. Demmel, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and V.A. Van der Vorst. *Templates for the Solutions of Linear Systems : Building Block for Iterative Methods* . SIAM, Philadelphia, PA, 1994.

[5] C. Brezinski. *Projection Methods for Systems of Equations*. Elsevier Science, 1997.

[6] C. Brezinski and H. Sadok. Lanczos-type Algorthms for Solving Systems of Linear Equation. *Applied Numerical Mathematics*, 11:443–473, 1993.

[7] C. Brezinski and R. Zaglia. A New Presentation of Orthogonal Polynomials

with Applications to Their Computation. *Numerical Algorithms*, 1:207–221, 1991.

[8] C. Brezinski and R. Zaglia. Breakdowns in the Computation of Orthogonal Polynomials. *Nonlinear Numerical Methods and Rational Approximation*, pages 49–59, 1994.

[9] C. Brezinski and R. Zaglia. Look-Ahead in Bi-CGSTAB and Other Methods for Linear Systems. *BIT*, 35:169–201, 1995.

[10] C. Brezinski and R. Zaglia. Breakdowns in the Implementation of Lanczos Method for Solving Linear Systems. *Comput. Math. Appl.*, 33:31–44, 1997.

[11] C. Brezinski, R. Zaglia, and H. Sadok. Avoiding Breakdown and Near-Breakdown in Lanczos-type Algorithms. *Numerical Algorithms*, 1:261–284, 1991.

[12] C. Brezinski, R. Zaglia, and H. Sadok. A Breakdown-Free Lanczos-type Algorithm for Solving Linear Systems. *Numerical Mathematics*, 63:29–38, 1992.

[13] C. Brezinski, R. Zaglia, and H. Sadok. The Matrix and Polynomial Approaches to Lanczos-type Algorithms. *Journal Of Computational and Applied Mathematics*, 123:241–260, 2000.

[14] C. Brezinski, R. Zaglia, and H. Sadok. A Review of Formal Orthogonality in Lanczos-based Method. *Journal of Computational and Applied Mathematics*, 140:81–98, 2002.

[15] Rajkumar Buyya, James Broberg, and Andrej Goscinski. *Cloud Computing Principles and Paradigms*. John Wiley and Sons, New Jersey, 2011.

[16] S. Cabay and Jackson L.W. A Polynomial Extrapolation Method for Finding Limits and Antilimits of Vector Sequences. *SIAM Journal on Numerical Analysis*, 13(5):734–752, 1976.

[17] B. N. Datta. *Numerical Linear Algebra and Applications*. Philadelphia: Society for Industrial and Applied Mathematics, 2010.

[18] F. David, McAllister and A. Roulier John. Algorithms for Computing Shape Preserving Spline Interpolates to Data. *Mathematics of Computation*, 31(139):717–725, 1977.

[19] Jr David M. Young. *Iterative Methods for Solving Partial Difference Equations of Elliptic type*. PhD thesis, Department of Mathematics, Harvard University, Cambridge, 1950.

[20] Carl de Boor. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.

[21] R. Delbourgo and J.A. Gregory. Shape Preserving Piecewise Rational Interpolation. *SIAM Journal on Scientific and Statistical Computing*, 10, 1983.

[22] J. W. Demmel. *Applied Numerical Linear Algebra*. Philadelphia: Society for Industrial and Applied Mathematics, 1997.

[23] Paul Dierckx. *Curve and Surface Fitting with Splines*. Clarendon Press, Oxford, 1995.

[24] Andre Draux. Formal Orthogonal Polynomials Revisited. Applications. *Numerical Algorithms*, 11:143–158, 1996.

[25] L.S. Duff and H.A. Van der Vorst. Developments and Trends in the Parallel Solution of Linear Systems. *Parallel Computing*, 25:1931–1970, 1999.

[26] Nick Elprin. Private discussion, 11 August 2014.

[27] Nick Elprin. Domino: A Platform-as-a-Service for Industrialized Data Analysis, 2014.

[28] Nick Elprin. Domino: A Platform-as-a-Service for Industrialized Data Analysis, 2014.

[29] M. Farooq. *New Lanczos-type Algorithms and Their Implementation*. PhD thesis, Department of Mathematical Sciences, University of Essex, 2011.

[30] M. Farooq and A. Salhi. A Preemptive Restarting Approach to Beating Inherent Instability. *Iranian Journal of Science and Technology Transaction a Science*, 12, 2012.

[31] M. Farooq and A. Salhi. New Recurrence Relationships between Orthogonal Polynomials which Lead to New Lanczos-type Algorithms. *Journal of Prime Research in Mathematics*, 8:61–75, 2012.

[32] M. Farooq and A. Salhi. A Switching Approach to Avoid Breakdown in Lanczos-type Algorithms. *Applied Mathematics and Information Sciences*, 8:2161–2169, 2014.

[33] Stephen Few. Multivariate Analysis Using Parallel Coordinates. The article is part of a series that he began in July 2006 with the article entitled : An Introduction to Visual Multivariate Analysis, September 2006.

[34] F. N. Fritsch and J. Butland. A Method for Constructing Local Monotone Piecewise Cubic Interpolants. *SIAM J. Sci. Stat. Comput.*, 5(2):300–304, 1984.

[35] F. N. Fritsch and R.E. Carlson. Monotone Piecewise Cubic Interpolation . *SIAM Journal Numerical Analysis*, 17(2):239–248, 1980.

[36] Lin Fu Sen. Piecewise Polynomial Interpolation, March 2014. Scientific Computing, Fall 2007.

[37] Walter Gander, J.Martin Gander, and Felix Kwok. *Scientific Computing: An Introduction Using Maple and Matlab*. Springer, 2010.

[38] Micahel L. Overton Gene H. Golub. Convergence of a two-stage Richardson Iterative Procedure for Solving Systems of Linear Equations. *Numerical Analysis*, 912:125–139, 1982.

[39] P. Grave-Morris. A Look-Around Lanczos Algorithm for Solving a System of Linear Equations. *Numerical Algorithm*, 15:247–274, 1997.

[40] E. Guennouni. A Unified Approach to Some Strategies for the Treatment of Breakdown in Lanczos-type Algorithms. *Application Mathematics*, 26:477–488, 1999.

[41] Martin H. Gutknecht. *Lanczos-type Solvers for Nonsymmetric Linear Systems of Equations*. Cambridge University Press, 1997.

[42] Don Heller. A Survey of Parallel Algorithms in Numerical Linear Algebra. Technical Report 1-1-1976, Technical report, Department of Computer Science, Carnegie Mellon University, February 1976.

[43] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 2002.

[44] Malik Z. Hussain and Muhammad Sarfraz. Positivity-Preserving Interpolation of Positive Data by Rational Cubics. *Journal of Computational and Applied Mathematics*, 218:446–458, 2008.

[45] Alexa Huth and James Cebula. The Basics of Cloud Computing. Technical report, Technical report, Produced for US-CERT, Carnegie Mellon University, 2011.

[46] M.James Hyman. Accurate Monotonicity Preserving Cubic Interpolation. *SIAM J. Sci. Stat. Comput.*, 4(4):645–654, 1983.

[47] Wayne Joubert. Lanczos Methods for the Solution of Nonsymmetric Systems of Linear Equations. *SIAM J. Matrix Anal.Appl*, 1992.

[48] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, Philadelphia, PA, 1995.

[49] R. Kosara. Parallel Coordinates, May 2012.

[50] I.B. Kvasov. *Methods of Shape-Preserving Spline Approximation*. World Scientific, Singapore, 2000.

[51] C. Lanczos. An Iteration Method for The Solution of the Eigenvalue Problem of Linear Differential and Integral Operator. *J.Res.Natl.Bur.Stand*, 45:255–282, 1950.

[52] C. Lanczos. Solution of Systems of Linear Equations by Minimized Iterations. *Journal of Research of the National Bureau of Standards*, 49:33–53, 1952.

[53] C. Landis and D. Blacharski. *Cloud Computing Made Easy*. Virtual Global Inc., 2013.

[54] R. H. Magnus and Eduard Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6), December 1952.

[55] MathWorks. *MATLAB Distributed Computing Server : Cloud Center User's Guide*. The Math Works Inc, 2013.

[56] MathWorks. *MATLAB Distributed Computing Server : System Administrator's Guide*. The Math Works Inc, 2013.

[57] MathWorks. *Parallel Computing Toolbox User's Guide*. The Math Works Inc, 2013.

[58] G. Meurant. *Computer Solution of Large Linear Systems*. Elsevier North Holand, 2006.

[59] Ronald B. Morgan. On Restarting the Arnoldi Method for Large Nonsymmetric Eigenvalue Problems. *Mathematics of Computation*, 65(215):1213–1230, 1996.

[60] I. Popovyan. Efficient Parallelization of Lanczos type Algorithms. *IACR Cryptology ePrint Archieve*, 416, 2011.

[61] P.M. Prenter. *Splines and Variational Methods*. John Wiley and Sons, New York, 1975.

[62] IEEE Computer Society Press, editor. *Parallel Coordinates : a Tool for Visualizing Multi-Dimensional Geometry*, 1990.

[63] K. Rajalakshami. Parallel Algorithm for Solving Large Systems of Simultaneous Linear Equations. *IJCSNS*, 9(7):276–279, July 2009.

[64] M. Rashid and Jon Crowcroft. Parallel Iterative Solution Method for Large Sparse Linear Equation Systems. Technical Report 650, Technical report, University of Cambridge, Computer Laboratory, October 2005.

[65] Amos Ron. Cubic Hermite Spline Interpolation II, October 2010.

[66] Y. Saad. Krylov Subspace Method for Solving Large Unsymmetric Linear Systems. *Mathematics of Computations*, 37(155), July 1981.

[67] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Philadelphia: Society for Industrial and Applied Mathematics, 2003.

[68] Abdellah Salhi. Load-Balancing in Flood-Fill Configures Applications. Technical Report 95/6, Department of Business Studies, University of Edinburgh, UK, 1995.

[69] Abdellah Salhi. On parallel processing issues: communications models, programming paradigms, load-balancing etc..., March 2015.

[70] Abdellah. Salhi, Les G. Proll, and David Rios Insua. Parallelising an Optimisation-based Framework for Sensitivity Analysis in Multi-Criteria Decision Making. Technical Report 98-15, Mathematics Department, University of Essex, UK, October 1998.

[71] A. Samsul and Noorhana. Y. Seabed Loging Data Curve Fitting Using Cubic Splines. *Applied Mathematical Sciences*, 7(81):4015–4026, 2013.

[72] E.Y. Sarna, David. *Implementing and Developing Cloud Computing Applications*. Taylor and Francis Group, Boca Raton, FL, 2011.

[73] R. Shanon, T. Holland, and A. Quigley. Multivariate Graph Drawing Using Parallel Coordinate Visualization. Technical report, University College Dublin, September 2008.

[74] A. Sidi, F. William Ford, and A. David Smith. Acceleration of Convergence of Vector Sequences. *SIAM Journal on Numerical Analysis*, 23(1):178–196, 1986.

[75] A. Sidi, F. William Ford, and A. David Smith. Extrapolation Methods for Vector Sequences. *SIAM Journal on Numerical Analysis*, 29(2):199–233, 1987.

[76] D.C. Sorensen. Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations. Technical Report NASI-19480, Technical report, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, May 1996.

[77] Yoshio Takane. The MPE (Minimal Polynomial Extrapolation) and Vector-Epsilon Methods: Numerical Demonstration of their Equivalence in Certain

Cases. In *Proceeding of Symposium on New Developments in Multivariate Computational Statistics and Behaviourmetrics*, pages 89–97, Osaka University, 2009. In K. Adachi et al. (Eds).

[78] MathWorks Team. Sparse Matrices, 2014.

[79] Stan Tomov. Discretization of PDEs and Tools for th Parallel Solution of the Resulting Systems, March 2010.

[80] Audun Torp. *Sparse Linear Algebra on a GPU with Applications to Flow in Porous Media*. PhD thesis, Department of Physics and Mathematics, Norwegian University of Science and Technology, 2009.

[81] H.A. Van der Vorst. The Superlinear Convergence Behaviour of GMRES. *Journal of Computational and Applied Mathematics*, 48:327–341, 1993.

[82] G. Wolberg and A. Itzik. Monotonic Cubic Spline Interpolation.

[83] David M. Young and Kang C. Jea. On the Simplification of Generalized Conjugate-Gradient Methods for Non-Symmetrizable Linear Systems. *Elsevier journal, Linear Algebra and its Applications*, 52–53:399–417, 1983.

Appendix A

Several Basic Concepts

A.1 Original Lanczos Method

Lanczos proposed a method for solving the eigenvalue problems, [51], where a $n \times n$ matrix can be transformed into a tridiagonal one to simplify the problem. Lanczos method is then extended to find the solution of a systems of linear equations. We briefly define some concepts and show how Lanczos algorithm is derived by orthogonalizing the natural basis of the Krylov subspace, [22]. Let us consider a system of n linear equations in n unknowns .

$$A\mathbf{x} = \mathbf{b} \tag{A.1.1}$$

where $A \in R^{n \times n}$ and $\mathbf{b} \in \mathbf{R}^n$. We can transform the system (A.1.1) into:

$$A\mathbf{z} = \mathbf{y} \tag{A.1.2}$$

for a given $\mathbf{z}$. Thus, we consider the matrix A in the sequence $\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots$ and compute the $\mathbf{y}'s$ as :

$$\begin{aligned}
\mathbf{y}_1 &= \mathbf{b} \\
\mathbf{y}_2 &= A\mathbf{y}_1 \\
\mathbf{y}_3 &= A\mathbf{y}_2 = A^2\mathbf{y}_1 \\
&\vdots \\
\mathbf{y}_n &= A\mathbf{y}_{n-1} = A^{n-1}\mathbf{y}_1 \\
\mathbf{y}_{n+1} &= A\mathbf{y}_n = A^n\mathbf{y}_1
\end{aligned}$$

We set $K = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]$. Then, we can write

$$AK = [A\mathbf{y}_1, A\mathbf{y}_2, \ldots, A\mathbf{y}_n] = [\mathbf{y}_2, \mathbf{y}_3, \ldots, \mathbf{y}_n, A^n\mathbf{y}_1]. \tag{A.1.3}$$

Assuming K is a non-singular. If we pre multiply both sides of (A.1.3) by K^{-1}, then we obtain

$$K^{-1}AK = K^{-1}[\mathbf{y}_2, \mathbf{y}_3, \ldots, \mathbf{y}_n, A^n\mathbf{y}_1], \tag{A.1.4}$$

so that for the last column we have vector $K^{-1}A^n\mathbf{y}_1$. Let $c = -K^{-1}A^n\mathbf{y}_1$. Then we can write

$$AK = K[\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n, -c] = KC, \tag{A.1.5}$$

Using the information that K is non-singular, then we obtain

$$C = K^{-1}AK = \begin{pmatrix} 0 & 0 & 0 & \dots & -c_1 \\ 1 & 0 & 0 & \dots & -c_2 \\ 0 & 1 & \dots & 0 & -c_3 \\ \vdots & \vdots & 1 & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -c_n \end{pmatrix} \qquad \text{(A.1.6)}$$

where $\mathbf{e}_i$ is the i_th column of the identity matrix. In fact, C is a *Companion* matrix and has the characteristic polynomial :

$$p(x) = x^n + \sum_{i=1}^{n} c_i x^{i-1}. \qquad \text{(A.1.7)}$$

The *eigenvalue* of A can be evaluated by reducing it into matrix C an considering $p(x) = 0$. In practice, the form (A.1.6) is not easy to be calculated since finding c requires $(n-1)$ matrix-vector multiplication by A and then solving a linear system with K. We can not guarantee solving a linear system with K will be easier than solving the original problem. Therefore, we can consider an orthogonal matrix Q to replace matrix K such that every column of K and Q span the same space which is called a *Krylov Subspace*. The *Krylov* subspace itself is defined by $K_n(A, \mathbf{b}) = span(\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \dots, A^{n-1}\mathbf{b})$. Let us follow the procedure below, [22]:

- Perform the QR decomposition K of $K = QR$, where Q is an orthogonal matrix and R is an upper triangular matrix. Then, we obtain:

$$\begin{aligned}
C &= K^{-1}AK \\
&= QR^{-1}A(QR) \\
&= R^{-1}Q^{-1}AQR \\
&= R^{-1}Q^{T}AQR, \quad \text{since } Q \text{ is orthogonal} \\
RC &= Q^{T}AQR, \quad \text{or} \\
Q^{T}AQ &= RCR^{-1} \equiv H \qquad \text{(A.1.8)}
\end{aligned}$$

Since R is upper triangular and C is upper *Hessenberg*, then H is also upper *Hessenberg*. In other words, the matrix A can be reduced to an upper *Hessenberg* matrix using the orthogonal transformation. For a symmetric case, in particular, if A is a symmetric matrix, so is $H = Q^{T}AQ$ and we write $T = Q^{T}AQ$, instead of H, where T is a tridiagonal matrix (lower *Hessenberg*).

- Compute the columns of Q one at time : Let $Q = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n]$. Since $H = Q^{T}AQ$ yields $AQ = QH$, then we obtain

$$A\mathbf{q}_j = \sum_{i=1}^{(j+1)} h_{i,j}\ \mathbf{q}_i \qquad \text{(A.1.9)}$$

  Also, since $\mathbf{q}_i$ are orthonormal, we can pre-multiply both sides of (A.1.9) by ${\mathbf{q}_m}^{T}$ to get

$$\mathbf{q}_m^{T}A\mathbf{q}_j = \sum_{i=1}^{j+1} h_{i,j}\ {\mathbf{q}_m}^{T}q_i = h_{m,j}, \quad 1 \le m \le j \qquad \text{(A.1.10)}$$

  Furthermore, from (A.1.9) we also get

$$h_{j+1,j}\ \mathbf{q}_{j+1} = A\mathbf{q}_j - \sum_{i=1}^{j} h_{i,j}\ \mathbf{q}_i \qquad \text{(A.1.11)}$$

Now we have two forms of (A.1.9) and (A.1.10) which lead to the *Arnoldi* and the

Lanczos methods as follows.

Algorithm 1. The Arnoldi Method

Inputs : (i) A, an $n \times n$ matrix, (ii) $\mathbf{b}$, an n -vector, (iii) k, a positive integer less than or equal to n.

Outputs : (i) a set of $(k + 1)$ orthonormal vectors $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_{k+1}$, (ii) a $(k + 1) \times k$ Hessenberg matrix $H = (h_{ij})$.

Step 0. Normalize the vector $\mathbf{q}_1$ by using formula

$$\mathbf{q}_1 = \frac{\mathbf{b}}{||\mathbf{b}||_2}$$

Step 1

```
for $i = 0 : k$ do
    $z = A\mathbf{q}_i$
    for $i = 0 : j$ do
        $h_{i,j} = \mathbf{q}_i{}^T\mathbf{z}$
        $\mathbf{z} = \mathbf{z} - h_{i,j}\,\mathbf{q}_i$
    end for
    $h_{j+1,j} = ||\mathbf{z}||_2$
    if $h_{j+1,j} = 0$, stop.
    $\mathbf{q}_{j+1} = \mathbf{z}/h_{j+1,j}$
end for
```

In case matrix A is symmetric, $H = T$ is a tri-diagonal matrix of the forms

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \dots & 0 \\ 0 & \beta_2 & \alpha_3 & \beta_3 & \ddots \\ \vdots & \vdots & \ddots & \ddots & \beta_{n-1} \\ 0 & 0 & \dots & \beta_{n-1} & \alpha_n \end{pmatrix}$$

Hence, we have $AQ = QT$. Equating column j on both sides, then we obtain

$$A\mathbf{q}_j = \beta_{j-1}\,\mathbf{q}_{j-1} + \alpha_j\,\mathbf{q}_j + \beta_j\,\mathbf{q}_{j+1}. \tag{A.1.12}$$

Since Q is orthonormal, pre-multiplying both sides of (A.1.12) by $\mathbf{q}_j^T$ yields :

$$\begin{aligned} \mathbf{q}_j^T A\mathbf{q}_j &= \beta_{j-1}\mathbf{q}_j^T\mathbf{q}_{j-1} + \alpha_j\mathbf{q}_j^T\mathbf{q}_j + \beta_j\mathbf{q}_j^T\mathbf{q}_{j+1} \\ &= 0 + \alpha_j + 0 = \alpha_j \end{aligned}$$

Thus, we have the different form of vector q for the Lanczos algorithm as follows.

Algorithm 2. The Lanczos Method

Inputs : (i) A, an $n \times n$ matrix, (ii) $\mathbf{b}$, an n -vector, (iii) k, a positive integer less than or equal to n.

Outputs : (i) a set of $(k+1)$ orthonormal vectors $\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_{k+1}\}$, (ii) the entries α_j and β_j of the symmetric tridiagonal matrix T_k.

Step 0. Set $\beta_0 = 0$, $\mathbf{q}_0 = 0$, and $\mathbf{q}_1 = \frac{\mathbf{b}}{\|\mathbf{b}\|_2}$.

Step 1

```
1: for $j = 1, 2, \dots, k$ do
2:     $\mathbf{z}_j = A\mathbf{q}_j$
```

```
3:     $\alpha_j = \mathbf{q}_j^T \mathbf{z}_j$
4:     $\mathbf{z}_j = \mathbf{z}_j - \alpha_j \mathbf{q}_j$
5:     $\beta_{j+1} = ||\mathbf{z}||_2$
6:     if $\beta_j = 0$ , quit.
7:     $\mathbf{q}_{j+1} = \frac{\mathbf{z}_j}{\beta_{j+1}}$
8:  end for
```

Appendix B

Numerical Results

B.1 EIEMLA Implementation

B.1.1 EIEM Orthodir Algorithm: $\delta = 0.5$

Slightly different from the case of $\delta = 0.2$, Lanczos Orthodir performs better when solving the SLE's using 100 iterations. It can be seen in both Tables B.1 and B.2 that there is no *Inf* value in the tables, which means that there is no breakdown occurs in this particular case. We also highlight here that the decrease numbers in both tables are varies among the SLE's. The extreme decrease number appears in dimensions 3000, 5000, 9000, and 70,000 for 100 iteration; where at the average, the residuals norm of the solution generated by the Orthodir with the EIEM are about 5 times smaller than the lowest residual norm of the iterate generated by Orthodir.

The percentage decrease of the improvement of using 200 iterations were calculated in Table B.3 and the differences between the two are clearly seen in Figure (B.1).

Table B.1: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 100 iterations, $\delta = 0.5$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	1.8783	1.8214	0.8441	2.1578	0.5258
2000	5.5838	0.9944	0.4717	2.1081	1.5697
3000	$1.0451E+02$	4.7064	1.0741	4.9849	2.8035
4000	7.1519	1.5904	0.5783	2.7501	2.8300
5000	$1.0223E+01$	3.5704	0.7241	4.9308	6.0809
6000	$5.7273E+01$	2.9810	0.9278	3.2130	8.1089
7000	8.3892	0.6367	0.1770	3.5972	10.5194
8000	4.6183	2.4319	0.6789	3.5821	13.1138
9000	$6.0832E-01$	0.5488	0.0982	5.5886	16.0586
10000	$7.934E+01$	4.7519	1.3328	3.5654	17.538
20000	$2.0231E+01$	2.5695	0.7952	3.2321	67.9624
30000	3.8310	2.4905	0.8328	2.9905	$3.5762E+02$
40000	$1.1499E+02$	10.7960	3.8297	2.8326	$5.8288E+02$
50000	$1.6340E+01$	4.1429	1.5021	2.7581	$1.0264E+03$
60000	$7.0046E+01$	34.5209	10.5869	3.2607	$1.5219E+03$
70000	$3.3002E+02$	17.4342	3.9145	4.4538	$4.7551E+03$

Table B.2: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 200 iterations, $\delta = 0.5$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$2.2942E+06$	1.3881	0.4077	3.4047	0.7443
2000	$1.6882E+07$	0.9944	0.3608	2.7561	1.8638
3000	$3.7536E+07$	4.7064	1.5181	3.1002	3.3828
4000	$3.5346E+08$	1.5904	0.5771	2.7558	5.375
5000	$5.7211E+05$	3.5704	0.8789	4.0624	6.7559
6000	$2.1347E+03$	2.9810	1.0816	2.7576	9.6289
7000	$1.6233E+08$	0.6367	0.1912	3.3300	13.3599
8000	$7.9534E+04$	2.4319	0.7959	3.0589	14.2452
9000	$9.7211E+1$	0.3483	0.0944	3.6896	19.6231
10000	$7.934E+01$	4.7519	1.5626	3.0410	15.8286
20000	$1.8967E+07$	2.5695	0.9323	2.7561	86.9935
30000	$2.3109E+04$	2.4905	0.8328	2.9905	$9.9763E+02$
40000	$1.2275E+09$	10.2165	3.0543	3.3450	$1.0778E+03$
50000	$7.3204E+04$	2.7272	0.8135	3.3524	$1.4354E+03$
60000	$1.8828E+04$	31.2924	8.7724	3.5671	$2.3105E+03$
70000	$2.6128E+07$	17.4342	3.9145	4.4538	$4.7551E+03$

Table B.3: *The comparison of the EIEM Orthodir implementation for 100 and 200 iterations, for the case of $\delta = 0.5$*

Dimension	$\|\mathbf{r}_{model}\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.8441	0.4077	51.7
2000	0.4717	0.3608	23.5
3000	1.0741	1.5181	41.3
4000	0.5783	0.5771	0
5000	0.7241	0.8789	21.3
6000	0.9278	1.0816	16.6
7000	0.1770	0.1912	8.02
8000	0.678	0.7959	17.4
9000	0.0982	0.0944	3.9
10000	1.3328	1.5626	17.2
20000	0.7952	0.9323	17.2
30000	0.8328	0.8328	0
40000	3.8297	3.0543	20.2
50000	1.5021	0.8135	45.8
60000	10.5869	10.7341	0
70000	3.914	3.91452	0

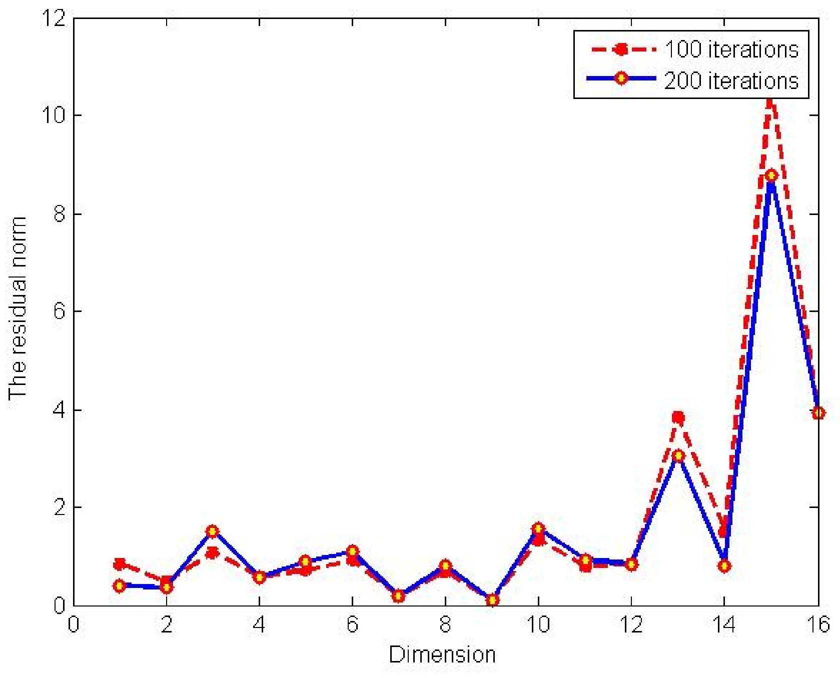


Figure B.1: *The behaviour of residual norms of the iterates generated by EIEM Orthodir for 100 and 200 iteration; the case of $\delta = 0.5$*

B.1.2 EIEM Orthodir Algorithm : $\delta = 0.8$

Similar to the previous case, the original Orthodir also performs better in this particular case. There is only one problem where the breakdown occurs, namely when solving SLE's with 8000 dimensions (see Table B.5). If we look at the decrease number in Table B.4, the significant number appears in dimensions 70000; where the residual norm of the Orthodir with the EIEM is 6.5 times smaller that the residual norm of the iterates generated by Lanczos Orthodir. This factor remains the same when we increased the iterations up to 200 (see Table B.5. Some extreme decrease numbers also appears in dimensions 5000, 9000, and 40,000 for 100 iterations; where the residual norm of the iterates generated by Orthodir with the EIEM are respectively about 5 times smaller than the residual norm of the iterates generated by the original Orthodir. The explanations of Table B.6 and Figure B.2 are similar to the previous cases.

B.1.3 EIEM Orthodir Algorithm : $\delta = 5$

As can be seen Tables B.7 and B.8 that there is no breakdown occurs. In fact, the improvement of the residual norms as a result in applying the EIEM in Orthodir appear in all problems. For instance, in dimensions 2000, 6000, 9000, and 10000, the residual norm of the iterates generated by the Orthodir with the EIEM were about four times smaller than those of generated by original Orthodir. Other problems seem similar with the average of the decrease number is about 3. Increasing the iterations up to 200, however, does not make significantly difference on the residual norms in most of problems (see Table B.9. For instance, when solv-

Table B.4: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 100 iterations, $\delta = 0.8$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$2.5926E+01$	0.2369	0.0860	2.7547	0.5059
2000	$7.8379E-02$	0.0505	0.0164	3.0793	1.318
3000	$5.1712E+02$	1.1776	0.2529	4.6564	2.4002
4000	$2.5821E+01$	1.0768	0.3907	2.7861	3.7333
5000	3.7132	0.9833	0.3240	4.9308	5.9803
6000	$2.0941E+04$	103.6807	34.6200	2.9948	8.0119
7000	1.0877	0.4585	0.1544	2.9696	10.2601
8000	$6.9498E+02$	29.8434	9.8285	3.0364	13.0358
9000	$1.8962E+02$	25.9947	5.0312	5.1667	14.8423
10000	$5.5303E+01$	4.7815	1.4725	3.2472	18.0211
20000	$1.1411E+01$	1.0164	0.2688	3.7813	66.0835
30000	$2.9739E+03$	213.3377	84.6039	2.5216	$2.0982E+02$
40000	$2.3134E+02$	100.3546	23.4275	4.2836	$1.2308E+03$
50000	$3.8614E+01$	18.1606	6.5847	2.7579	$1.1691E+03$
60000	$1.5264E+02$	47.9573	17.3872	2.7582	$1.9340E+03$
70000	$2.8888E+04$	33.4808	5.1349	6.5201	$3.2625E+03$

Table B.5: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 200 iterations, $\delta = 0.8$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$4.6289E+08$	0.2369	0.0860	2.7547	0.7443
2000	$5.1205E+03$	0.0505	0.0164	3.0793	1.6161
3000	$1.2108E+06$	1.1776	0.2529	4.6564	3.4088
4000	$7.8175E+08$	0.9525	0.2848	3.3445	5.0934
5000	$8.8165E-01$	0.5217	0.1559	3.3419	7.4046
6000	$6.6716E+07$	103.6807	37.6200	2.7559	10.2061
7000	1.0877	0.4585	0.1544	2.9696	9.7593
8000	*INF*	29.8434	10.8285	2.7560*	18.4311
9000	$6.6210E+4$	22.0634	6.5959	3.3450	21.119
10000	$8.8350E+01$	2.3929	0.7154	3.3448	25.99416
20000	$8.9179E+05$	1.0164	0.3688	2.7559	94.1729
30000	$1.0942E+10$	233.3377	83.6039	2.7909	$5.6078E+02$
40000	$4.5991E+09$	100.3546	23.4275	4.2836	$9.7072E+02$
50000	$6.5309E+06$	18.1606	6.5847	2.7579	$1.6362E+03$
60000	$6.8522E+03$	42.4838	14.5559	2.9187	$3.9056E+03$
70000	$4.7767E+07$	33.4808	5.1349	6.5202	$3.4996E+03$

Table B.6: *The comparison of the EIEM Orthodir implementation for 100 and 200 iterations, for the case of* $\delta = 0.8$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.0860	0.0860	0
2000	0.0164	0.0164	0
3000	0.2529	0.2529	0
4000	0.3907	0.2848	27.1
5000	0.3240	0.1559	51.9
6000	34.6200	37.6200	8.7
7000	0.1544	0.1544	0
8000	9.8285	10.8285	10.2
9000	5.0312	6.5959	31.1
10000	1.4725	0.7154	51.4
20000	0.2688	0.3688	37.2
30000	84.603	83.6039	1.2
40000	23.4275	23.4275	0
50000	6.58471	6.5847	0
60000	17.3872	14.5559	16.3
70000	5.1349	5.1349	0

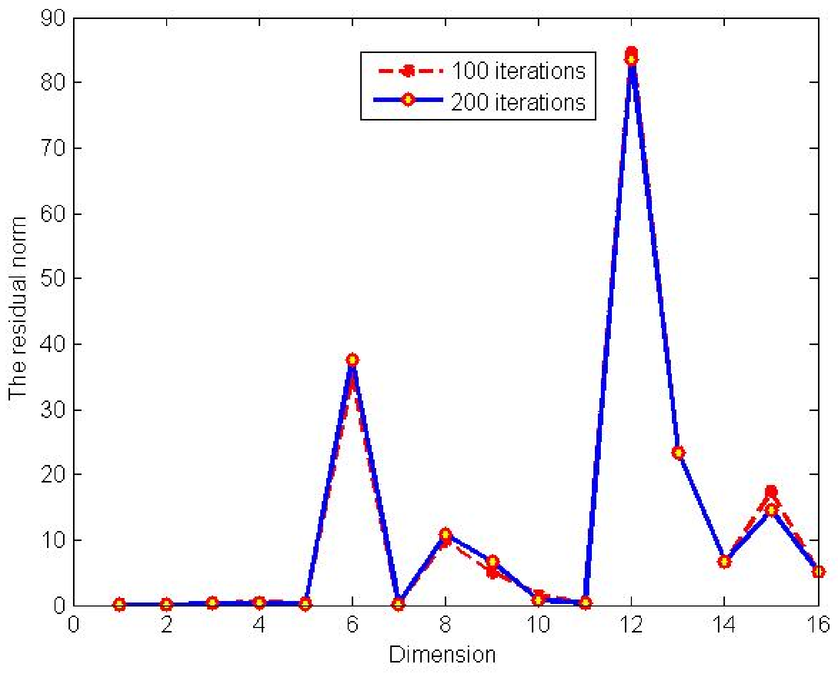


Figure B.2: *The behaviour of residual norms of the iterates generated by EIEM Orthodir for 100 and 200 iteration; the case of* $\delta = 0.8$

Table B.7: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 100 iterations, $\delta = 5$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$1.0992E+01$	2.9154	0.8082	3.6073	0.5196
2000	$5.4979E+01$	4.4439	1.0776	4.1239	1.3042
3000	$9.9974E+01$	7.4355	2.0682	3.5952	2.4186
4000	$2.1689E+01$	5.8785	1.8285	3.2149	3.7985
5000	3.2033	3.8900	1.1868	3.2799	5.4061
6000	$1.0238E+01$	6.5297	1.5716	4.1548	9.8097
7000	$4.0638E+01$	2.1982	0.6719	3.2716	13.1614
8000	$1.0653E+02$	6.5665	1.9771	3.3213	16.6412
9000	2.7395	1.7444	0.3711	4.7006	21.2535
10000	$6.5232E+01$	15.1933	3.3391	4.5501	24.6426
20000	$3.0549E+01$	7.3298	2.2799	3.2149	91.5777
30000	$3.6463E+01$	11.0830	3.6449	3.0407	$4.1047E+02$
40000	$4.7312E+02$	17.7709	6.4433	2.7580	$6.2475E+02$
50000	$1.8226E+01$	4.3368	1.2724	3.4084	$1.9825E+03$
60000	$9.6296E+01$	25.4886	9.2417	2.7579	$1.9826E+03$
70000	$2.6510E+01$	17.9748	6.1187	2.9377	$2.1842E+03$

ing large scale problems, i.e.dimensions 30000, 50000, and 70000, the percentage decrease reached respectively 59%, 37%, and 62%. The differences of using bot 100 and 200 iterations can also be seen in Figure B.3.

B.1.4 EIEM Orthodir Algorithm : $\delta = 8$

In this particular case, Lanczos Orthodir performs well to find a good approximate solution without facing breakdown. The EIEM in the algorithm is also worth it. The significant improvement appears in dimensions 8000, where the decrease number is about 4.12 which means that the residual norm of the model iterate is about 4 times smaller than the lowest residual norm of the iterate generated by original Orthodir (see Table B.10). Other problems also behave similarly where the average of residual norm is three times smaller than the lowest residual norm

Table B.8: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 200 iterations, $\delta = 5$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	4.1236	2.9154	0.6928	4.2081	0.87623
2000	$2.4843E + 03$	4.4439	1.0776	4.1239	1.5255
3000	$2.3513E + 01$	7.4355	2.0682	3.5952	2.7852
4000	$2.0947E + 01$	5.8785	1.8285	3.2149	5.2364
5000	$5.0856E + 01$	3.8900	1.4464	2.6894	6.7514
6000	$1.0238E + 01$	6.5297	1.5716	4.1548	9.8097
7000	$4.0638E + 01$	2.1982	0.6719	3.2716	13.1614
8000	$1.0653E + 02$	6.5665	1.9771	3.3213	16.6412
9000	2.7395	1.7444	0.3711	4.7006	21.2535
10000	$6.5232E + 01$	15.1933	3.3391	4.5501	24.6426
20000	$3.0549E + 01$	7.3298	2.2799	3.215	91.5777
30000	$1.1460E + 02$	4.5718	1.4835	3.0818	$4.6338E + 02$
40000	$2.3758E + 01$	17.771	6.4433	2.7581	$1.2039E + 03$
50000	$3.4672E + 01$	2.2300	0.8086	2.7599	$1.3201E + 03$
60000	$1.7553E + 02$	25.4886	9.1417	2.7881	$2.6501E + 03$
70000	$2.7121E + 01$	6.4980	2.3514	2.7635	$3.2511E + 03$

Table B.9: *The comparison of the EIEM Orthodir implementation for 100 and 200 iterations, for the case of $\delta = 5$*

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.8082	0.6928	14.27
2000	1.0776	1.0776	0
3000	2.0682	2.0682	0
4000	1.8285	1.8285	0
5000	1.1868	1.4464	21.87
6000	1.5716	1.5716	0
7000	0.6719	0.6719	0
8000	1.9771	1.9771	0
9000	0.3711	0.3711	0
10000	3.339	3.339	0
20000	2.2799	2.2799	0
30000	3.6449	1.4835	59.3
40000	6.4433	6.4433	0
50000	1.2724	0.8086	36.45
60000	9.2417	9.2417	0
70000	6.1187	2.3514	61.57

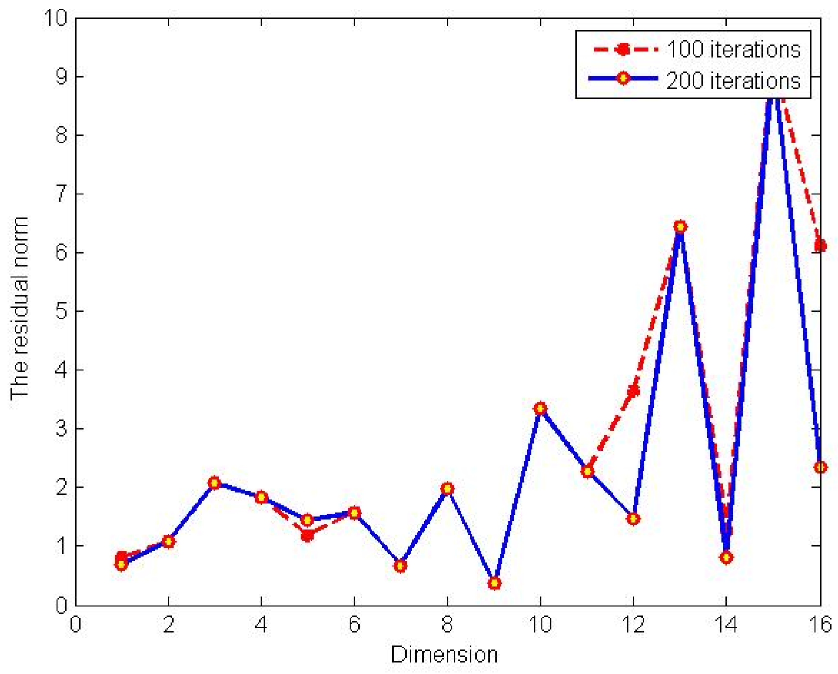


Figure B.3: *The behaviour of residual norms of the iterates generated by EIEM Orthodir for 100 and 200 iteration; the case of* $\delta = 5$

of the previous iterates.

Similar to the previous cases, increasing the iterations up to 200 behaves well. According the information in Table B.12, the percentage decrease reached about 56% in dimensions 30000 when the iterations increased. It followed by dimensions 10000 and 5000 which hit 46% and 43% respectively. For large dimensions, i.e. 40000 and 50000, the percentage decrease were about 34% respectively. In dimensions 2000, however, the residual norm of the model iterate increased from 0.4887 to 2.57, or about 81% increased, when we increased the iterations from 100 to 200. All of the behavious differences by increasing the iterations from 100 to 200 are captured in Figure B.4.

Table B.10: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 100 iterations, $\delta = 8$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	5.9686	2.3831	0.6694	3.5600	0.6919
2000	$2.7284E+01$	1.2911	0.4887	2.6380	1.7023
3000	$1.9115E+01$	21.9595	8.0749	2.7195	3.0265
4000	$1.1805E+02$	20.6092	7.5124	2.7434	4.6145
5000	$6.8169E+01$	26.7770	9.1505	2.9263	6.5283
6000	$3.8709E+01$	15.6265	5.6164	2.7823	8.7754
7000	$2.0905E+01$	17.8229	6.5562	2.7185	11.3243
8000	$4.4907E+01$	19.6936	4.7774	4.1222	14.2098
9000	$4.6669E+01$	30.6000	10.3877	2.9458	17.4346
10000	$7.2719E+01$	16.9018	6.2070	2.7230	20.9478
20000	$5.0573E+01$	33.9200	12.2611	2.7665	73.3194
30000	$6.8594E+01$	44.9832	15.7708	2.8523	$6.4489E+02$
40000	$1.1651E+02$	79.6834	28.8917	2.7580	$1.1728E+03$
50000	$1.1204E+02$	42.7284	15.0339	2.8421	$1.0261E+03$
60000	$9.6487E+01$	40.6506	12.0781	3.3656	$2.7166E+03$
70000	$1.8619E+02$	136.2499	38.2304	3.5639	$2.1882E+03$

Table B.11: *Comparison between the residual norms of the iterates generated by the original Orthodir algorithm and those generated by EIEM in Orthodir algorithm for 200 iterations, $\delta = 8$*

Dim	Orthodir		Orthodir with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	7.7057	1.2911	0.4887	2.6419	0.7337
2000	$3.7015E+01$	6.8989	2.5700	2.6844	1.9011
3000	$3.0822E+01$	20.7521	7.5666	2.7426	3.3701
4000	$6.0728E+01$	20.6092	7.9124	2.6047	5.1737
5000	$1.5362E+01$	15.3617	5.2462	2.9282	7.2108
6000	$9.0598E+01$	15.6265	5.6164	2.7823	9.8097
7000	$6.3885E+02$	17.8229	6.9562	2.5622	12.7826
8000	$2.5438E+01$	18.0309	6.6433	2.7141	15.9878
9000	$1.3081E+02$	25.0825	9.2643	2.7074	19.7902
10000	$1.8444E+04$	8.8147	3.3365	2.6419	23.9232
20000	$6.9922E+01$	33.9200	12.2611	2.7665	84.2992
30000	$6.0592E+01$	20.7679	6.9638	2.9823	$5.87948E+02$
40000	$5.6718E+01$	52.5156	19.0412	2.7579	$9.8724E+03$
50000	$6.4700E+02$	38.9168	14.0688	2.7662	$1.2107E+03$
60000	56.4558	40.6505	12.0781	3.3656	$2.6501E+03$
70000	$2.7660E+02$	105.4887	38.2482	2.7580	$2.6126E+03$

Table B.12: *The comparison of the EIEM Orthodir implementation for 100 and 200 iterations, for the case of* $\delta = 8$

Dimension	$\|\mathbf{r}_{model}\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.6694	0.4887	26.99
2000	0.4887	2.5700	80.98
3000	8.0749	7.5666	6.7
4000	7.5124	7.9125	0
5000	9.1505	5.2462	42.67
6000	5.6164	5.6164	0
7000	6.5562	6.9562	0
8000	4.7774	6.6433	39.06
9000	10.3877	9.2643	10.81
10000	6.2070	3.3365	46.25
20000	12.2611	12.2611	0
30000	15.7708	6.9638	55.84
40000	28.8917	19.0412	34.09
50000	15.0339	14.0688	34.09
60000	12.0781	12.0781	0
70000	38.2304	38.2304	0

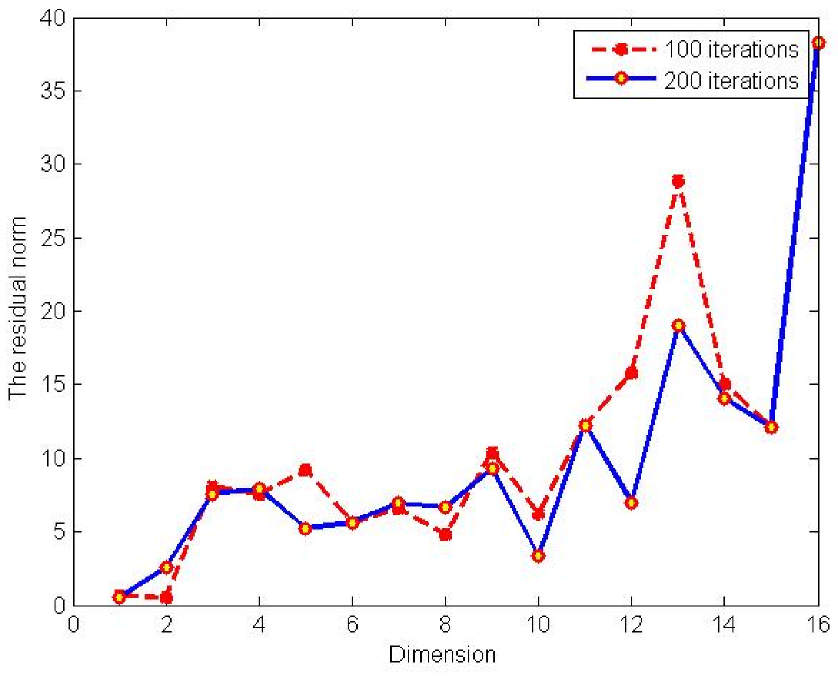


Figure B.4: *The behaviour of residual norms of the iterates generated by EIEM Orthodir for 100 and 200 iteration; the case of* $\delta = 8$

B.1.5 EIEM Orthomin Algorithm : $\delta = 0.5$

It can be seen in both Tables B.13 and B.14 that there is no *Inf* value in the residual norms column, which means that there is no breakdown occurs in this particular case. We also highlight here that the decrease numbers in both tables are mostly about 2 and 3. It means that the residual norms of the iterates generated by Orthomin with the EIEM are mostly about two and three times smaller than the lowest residual norms of all iterates generated by the original Orthomin.

The improvement of the residual norms of the iterates generated by the EIEM in Orthomin as a result in increasing the iterations up to 200 can be seen in Table B.15. This improvement is measured by computing the percentage decrease. For instance, when solving 1000 dimensional problem, the residual norm of the iterate generated by the EIEM in Orthomin decreased about 27% when we increased the iterations Also, for dimension 30000, the percentage decrease was about 56%. In contrast, the percentage increase was about 41%, when solving dimensions 2000. This means that the residual norm of the iterate generated by Orthomin with the EIEM using 100 iterations tends to go up when the iterations increased. Other cases remain a stable when we increased the iterations.

B.1.6 EIEM Orthomin Algorithm: $\delta = 0.8$

Similar to the previous cases, there is no breakdown occurs in this particular case. In fact, there is improvement on the residual norm when using the EIEM in Orthomin (see Tables B.16 and B.17). It can be seen on the decrease column on both tables. For instance, the residual norms of the iterates generated by the EIEM in

Table B.13: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 100 iterations, $\delta = 0.5$*

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	5.9686	2.3831	0.6694	3.5600	0.6919
2000	$2.7284E+01$	1.2911	0.4887	2.6380	1.7023
3000	$1.9115E+01$	21.9595	8.0749	2.7195	3.0265
4000	6.3662	0.7867	0.2572	3.0587	8.6011
5000	1.7354	0.4538	0.1115	4.0699	12.8472
6000	$3.8709E+01$	15.6265	5.6164	2.7823	8.7754
7000	$2.0905E+01$	17.8229	6.5562	2.7185	11.3243
8000	$4.4907E+01$	19.6936	4.7774	4.1222	14.2098
9000	$4.6669E+01$	30.6000	10.3877	2.9458	17.4346
10000	$7.2719E+01$	16.9018	6.2070	2.7230	20.9478
20000	$5.0573E+01$	33.9200	12.2611	2.7665	73.3194
30000	$6.8594E+01$	44.9832	15.7708	2.8523	$6.4489E+02$
40000	$1.1651E+02$	79.6834	28.8917	2.7580	$1.1728E+03$
50000	$1.1204E+02$	42.7284	15.0339	2.8421	$1.0261E+03$
60000	$9.6487E+01$	40.6506	12.0781	3.3656	$2.7166E+03$
70000	$1.8619E+02$	136.2499	38.2304	3.5639	$2.1882E+03$

Table B.14: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 200 iterations, $\delta = 0.5$*

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	7.7057	1.2911	0.4887*	2.6419	0.7337
2000	$3.7015E+01$	6.8989	2.5700*	2.6844	1.9011
3000	$3.0822E+01$	20.7521	7.5666*	2.7426	3.3701
4000	9.2220	0.7867	0.2572	3.0587	11.0919
5000	3.0015	0.4538	0.1639*	2.7688	18.7678
6000	$9.0598E+01$	15.6265	5.6164	2.7823	9.8097
7000	$6.3885E+02$	17.8229	6.9562	2.5622	12.7826
8000	$2.5438E+01$	18.0309	6.6433*	2.7141	15.9878
9000	$1.3081E+02$	25.0825	9.2643*	2.7074	19.7902
10000	$1.8444E+04$	8.8147	3.3365*	2.6419	23.9232
20000	$6.9922E+01$	33.9200	12.2611	2.7665	84.2992
30000	$6.0592E+01$	20.7679	6.9638*	2.9823	$5.87948E+02$
40000	$5.6718E+01$	52.5156	19.0412	2.7579	$9.8724E+03$
50000	$6.4700E+02$	38.9168	14.0688	2.7662	$1.2107E+03$
60000	56.4558	40.6505	12.0781	3.3656	$2.6501E+03$
70000	$2.7660E+02$	105.4887	38.2482	2.7580	$2.6126E+03$

Table B.15: *The comparison of the EIEM Orthomin implementation for 100 and 200 iterations, for the case of* $\delta = 0.5$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.6694	0.4887	26.99
2000	0.4887	2.5700	80.98
3000	8.0749	7.5666	6.7
4000	0.2572	0.2572	0
5000	0.1115	0.1639	42.67
6000	5.6164	5.6164	0
7000	6.5562	6.9562	0
8000	4.7774	6.6433	39.06
9000	10.3877	9.2643	10.81
10000	6.2070	3.3365	46.25
20000	12.2611	12.2611	0
30000	15.7708	6.9638	55.84
40000	28.8917	19.0412	34.09
50000	15.0339	14.0688	34.09
60000	12.0781	12.0781	0
70000	38.2304	38.2304	0

Orthomin are respectively 10 times and 14 times smaller than the minimum residual norm of the iterates generated by the original Orthomin when solving SLE's dimensions 8000 (for 100 iterations) and dimensions 40000 (for 200 iterations). Other problems in both tables have similar decrease numbers, i.e. about 3, 4, and 5.

In Table B.18, the residual norms of some iterates generated by Orthomin with the EIEM decreased by increasing the iterations up to 200. For instance, for problems with dimensions 1000, 2000, 3000, 5000, and 70000, the percentage decrease are respectively 34%, 20%, 20%, 23%, and 88%. In contrast, for dimensions 8000 and 9000, the percentage increase are respectively 74% and 64%. These behaviour are also captured in Figure B.6.

Table B.16: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 100 iterations,* $\delta = 0.8$

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	1.5595	0.2748	0.0804	3.4179	0.7867
2000	$6.7846E+03$	1.7972	0.8318	2.1606	2.4571
3000	$5.6030E+02$	4.9488	1.4539	3.4038	5.0375
4000	$1.3552E+03$	0.9227	0.4300	2.1458	8.4023
5000	$2.8683E+01$	1.4975	0.5978	2.5050	12.6682
6000	$1.2369E+03$	11.1826	4.7875	2.3358	14.1291
7000	$1.1719E+02$	5.6363	2.0804	2.7092	24.2852
8000	$5.7674E+03$	8.4599	0.7982	10.5987	32.0787
9000	$4.0409E+01$	4.1724	0.5599	7.4520	40.1987
10000	$2.1855E+02$	2.7536	0.8760	3.1434	49.0421
20000	$3.7715E+01$	3.1585	1.3718	2.3024	1.8571
30000	$8.8924E+05$	63.7004	23.7422	2.6830	$1.2914E+03$
40000	$4.6453E+03$	250.0413	17.4236	14.3507	$1.2199E+03$
50000	$6.1251E+02$	17.4355	1.6002	10.8958	$1.9549E+03$
60000	$1.7492E+02$	8.3668	3.1185	2.6829	$2.9044E+03$
70000	2.0426	2.0426	0.6613	3.0888	$5.9227E+03$

Table B.17: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 200 iterations,* $\delta = 0.8$

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$9.6541E+01$	0.2748	0.0532	5.1654	1.2095
2000	$1.0237E+05$	1.7972	0.6699	2.6844	4.0583
3000	$6.9982E+02$	3.1618	1.1585	2.7292	6.7418
4000	$5.8047E+01$	0.9227	0.3539	2.6072	13.9629
5000	$6.5222E+02$	1.4975	0.4581	3.2689	13.9149
6000	$1.2369E+03$	11.1826	4.0680	2.7489	13.4683
7000	$3.5167E+02$	5.6363	2.0008	2.81701	24.722
8000	$1.5177E+03$	8.4599	3.0532	2.7708	53.7407
9000	$2.2164E+03$	4.1724	1.5351	2.7179	55.2054
10000	$2.1855E+02$	2.7536	0.9263	2.9727	44.4763
20000	$1.8459E+02$	3.1585	1.1572	2.7294	$2.4995E+02$
30000	$1.9198E+03$	63.7004	21.7422*	2.9298	$1.1511E+02$
40000	$1.0280E+05$	250.0413	17.4236	14.3507	$2.0618E+03$
50000	$8.7788E+05$	17.4355	1.6002	10.8958	$2.7704E+03$
60000	$2.5536E+04$	8.3668	3.0185	2.7718	$8.6801E+03$
70000	$4.5035E-01$	0.0220	0.0782	0.2813	$8.219E+03$

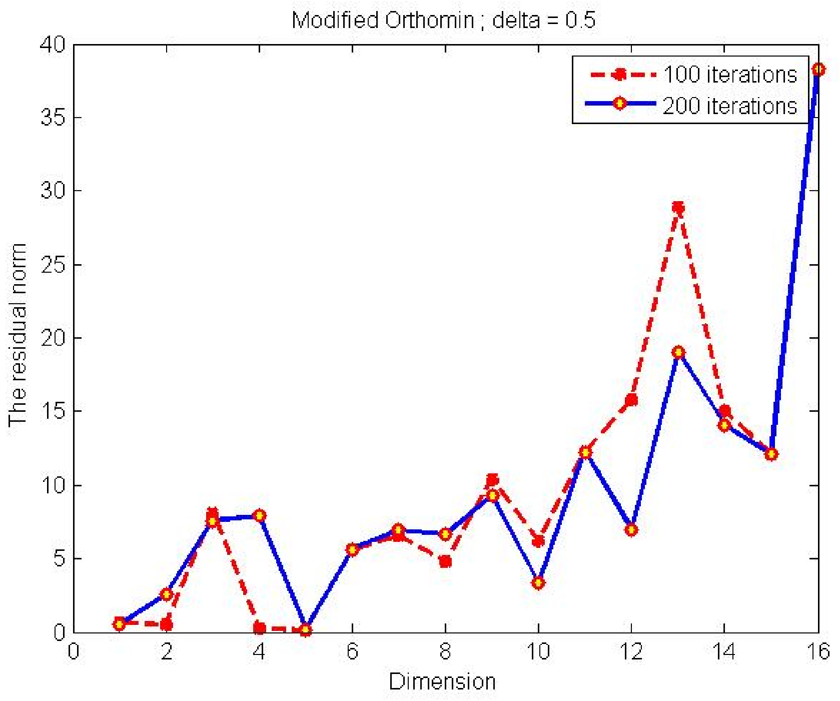


Figure B.5: *The behaviour of residual norms of the iterates generated by EIEM Orthomin for 100 and 200 iteration; the case of* $\delta = 0.5$

B.1.7 EIEM Orthomin Algorithm: $\delta = 5$

Slightly different from the previous cases, the improvement of the residual norms after imposing the EIEM in Orthomin algorithm are not so significant (see the decrease numbers on Tables B.16 and B.17). Also, increasing the iterations up to 200, does not make significant differences on those residual norms. For instance, when solving dimensions 10000 and 20000, the percentage decrease reached respectively 44%, and 42% , according to Table B.21. Only one problem where increasing the iterations up to 200 make the residual norm of the iterate generated by the EIEM in Orthomin increase, i.e. in dimension 6000 with the percentage increase is 72%. It can be seen clearly in Figure B.7 .

Table B.18: *The comparison of the EIEM Orthomin implementation for 100 and 200 iterations, for the case of* $\delta = 0.8$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.0804	0.0532	33.83
2000	0.8318	0.6699	19.46
3000	1.4539	1.1585	20.31
4000	0.4300	0.3539	17.7
5000	0.5978	0.4581	23.39
6000	4.7875	4.0680	15.03
7000	2.0804	2.0804	0
8000	0.7982	3.0532	73.86
9000	0.5599	1.5351	63.5
10000	0.8760	0.9263	5.74
20000	1.3718	1.1572	18.5
30000	23.7422	21.7422	9.2
40000	17.4236	17.4236	0
50000	1.6002	1.6002	0
60000	3.1185	3.1185	0
70000	0.6613	0.0782	88.17

Table B.19: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 100 iterations,* $\delta = 5$

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$1.9590E+01$	2.4785	0.9238	2.6829	1.088
2000	8.3932	0.8013	0.2987	2.6835	3.5769
3000	$2.1651E+02$	3.4239	1.1762	2.9109	4.3619
4000	$7.9942E+01$	2.3600	0.8796	2.6830	9.8006
5000	$6.2883E+01$	2.7989	1.0232	2.7354	12.0658
6000	$4.2912E+01$	4.4958	1.5756	2.8534	17.0101
7000	$1.0665E+02$	5.6275	2.1975	2.5609	22.6653
8000	$2.5449E+01$	5.4889	2.0258	2.7095	29.2024
9000	$6.2325E+01$	9.8697	2.6786	3.6846	38.1748
10000	$1.9821E+01$	6.6903	2.2936	2.9169	47.7449
20000	$8.8445E+01$	12.0115	4.3769	2.7443	$1.8559E+02$
30000	$3.0467E+02$	33.5764	11.5145	2.916	$9.6699E+02$
40000	$1.6658E+02$	6.9695	2.4015	2.9021	$1.8796E+03$
50000	$6.8389E+01$	16.7751	6.1524	2.7265	$2.6762E+03$
60000	$2.5799E+01$	18.7636	4.3952	4.2691	$4.4875E+03$
70000	$1.0788E+01$	10.4984	2.8920	3.6301	$4.0578E+03$

Table B.20: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 200 iterations, $\delta = 5$*

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$1.9590E + 01$	2.4785	0.9238	2.68291	1.088
2000	$5.2662E + 01$	0.8013	0.2987	2.6835	3.5769
3000	7.6491	3.0467	1.0356	2.9413	7.6088
4000	$3.1456E + 01$	2.3600	0.8796	2.683	12.5411
5000	$2.3570E + 01$	2.7989	1.0232	2.7354	19.5972
6000	$3.5480E + 01$	4.4958	1.5756	2.8534	27.0477
7000	$7.3411E + 01$	5.6275	2.1975	2.5609	36.1136
8000	$3.6774E + 02$	5.4889	2.0258	2.7095	47.5767
9000	$1.5230E + 01$	9.8697	2.6786	3.6846	57.2231
10000	4.5607	3.7117	1.2834	2.89219	70.5569
20000	$2.1882E + 02$	7.0947	2.5443	2.7885	$2.8146E + 02$
30000	$7.9728E + 02$	33.5764	11.5145	2.916	$9.8124E + 02$
40000	$4.7231E + 01$	6.9695	2.4015	2.9021	$2.3112E + 03$
50000	$2.0034E + 02$	16.7751	6.1524	2.7265	$3.3636E + 03$
60000	$4.7045E + 01$	12.5977	4.3754	2.8792	$5.13353E + 03$
70000	$8.2325E + 01$	10.4984	2.8920	3.6302	$7.7634E + 03$

Table B.21: *The comparison of the EIEM Orthomin implementation for 100 and 200 iterations, for the case of $\delta = 5$*

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.9238	0.9238	0
2000	0.2987	0.2987	0
3000	1.1762	1.0356	13.7
4000	0.8796	0.8796	0
5000	1.0232	1.0232	0
6000	1.5756	5.6164	71.9
7000	2.1975	2.1975	0
8000	2.0258	2.0258	0
9000	2.6786	2.6786	0
10000	2.2936	1.2834	44.06
20000	4.3769	2.5443	41.87
30000	11.5145	11.5145	0
40000	2.4015	2.4015	0
50000	6.1524	6.1524	0
60000	4.3952	4.3754	0
70000	2.8920	2.8920	0

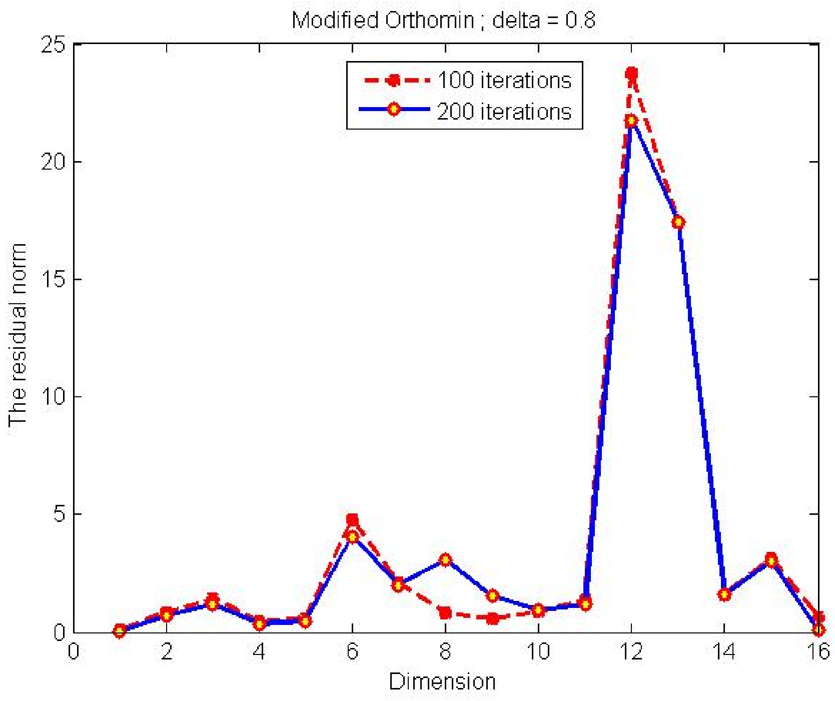


Figure B.6: *The behaviour of residual norms of the iterates generated by EIEM Orthomin for 100 and 200 iteration; the case of* $\delta = 0.8$

B.1.8 EIEM Orthomin Algorithm: $\delta = 8$

In this particular case, Lanczos Orthomin performs well to find a good approximate solution without facing breakdown. Applying the EIEM in the algorithm is also worth it. It is indicated by the decrease numbers in both Tables B.22 and B.23 which is about 3; which means that the residual norm the iterate generated by the EIEM is 3 times smaller than the lowest residual norm of the iterate generated by the original Orthomin.

According the information in Table B.24, for dimensions 1000 and 9000, the percentage decreased about 53% and 51% respectively when the iterations were increased up to 200. The percentage, however, increased for dimensions 2000 and 20000 when the iterations increased, with the percentage increased are respec-

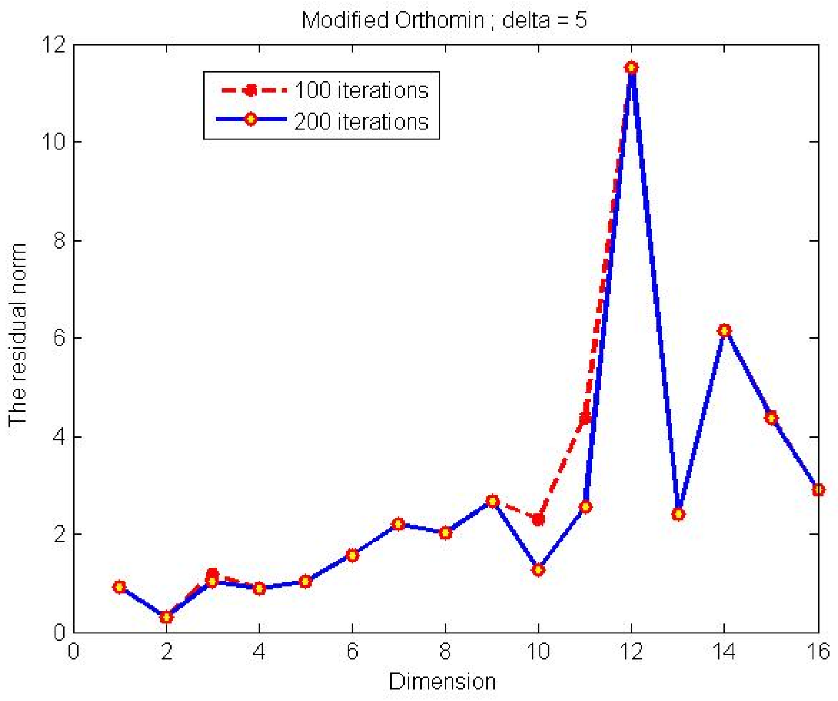


Figure B.7: *The behaviour of residual norms of the iterates generated by EIEM Orthomin for 100 and 200 iteration; the case of* $\delta = 5$

tively 15% and 17%. All of the behaviour differences by increasing the iterations from 100 to 200 are captured in Figure B.8.

B.1.9 EIEM Orthores Algorithm: $\delta = 0.2$

Similar to the previous subsection, in this subsection, we will look at some results of running EIEM in Orthores algorithm, for both 100 and 200 iterations. They are provided in Tables B.25 and B.26. including some improvements of the residual norms of the iterates that might be made. might be made In particular we Similar to the previous subsection, Based on the informations available in Table B.25, overall, the EIEM in Orthores algorithm finds a good solution with a small residual norm even smaller than all of the previous iterates. We noticed here that in this particular case, the Orthores does not experience breakdown for both 100 and 200 iterations. The decrease number also improved in most problems, par-

Table B.22: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 100 iterations, $\delta = 8$*

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$1.8994E+02$	6.1281	2.2841	2.6829	0.8016
2000	$7.7451E+03$	4.0691	1.3167	3.0906	2.5555
3000	$4.7707E+01$	12.3334	4.5569	2.7065	5.16
4000	$1.5484E+02$	15.0122	5.2953	2.8350	8.6441
5000	$2.0429E+01$	9.0788	3.3338	2.7233	12.9375
6000	$8.6403E+01$	23.0560	8.8934	2.5925	18.1627
7000	$7.5573E+02$	20.4577	6.6249	3.0880	24.5629
8000	$1.5668E+02$	24.2072	9.5224	2.5421	29.5651
9000	$4.6034E+01$	18.2494	6.4019	2.8506	37.4057
10000	$5.9648E+02$	34.1580	11.7313	2.9117	46.0312
20000	$1.9880E+02$	76.5965	19.8233	3.8639	$3.2280E+02$
30000	$2.6202E+01$	26.2026	7.1209	3.6797	$9.6138E+02$
40000	$1.7379E+02$	57.4135	22.3990	2.5632	$1.3874E+03$
50000	$3.5316E+02$	108.8633	37.1745	2.9284	$4.3481E+03$
60000	$2.1466E+03$	31.2251	12.6381	2.4707	$4.0689E+03$
70000	$1.5691E+02$	63.8185	20.0467	3.1835	$4.7489E+03$

Table B.23: *Comparison between the residual norms of the iterates generated by the original Orthomin algorithm and those generated by EIEM in Orthomin algorithm for 200 iterations, $\delta = 8$*

Dim	Orthomin		Orthomin with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$3.7592E+01$	3.4218	$1.0754*$	3.1819	0.907
2000	$3.7015E+01$	4.0691	$1.5166*$	2.6830	2.9374
3000	$7.8998E+02$	12.3334	$3.5969*$	3.4289	6.2075
4000	$7.3054E+01$	15.0122	4.5953	3.2669	10.1714
5000	$1.1803E+03$	9.0788	3.3438	2.7151	15.776
6000	$4.9839E+01$	23.0560	8.6144	2.7823	22.4854
7000	$3.2196E+02$	20.4577	6.6249	3.0880	30.1689
8000	$1.0137E+02$	24.2072	9.3224	2.5967	39.1885
9000	$4.8830E+01$	9.3011	$3.1667*$	2.9371	50.5056
10000	$9.5600E+03$	34.1580	11.7313	3.1830	60.5066
20000	$1.4116E+02$	69.0569	$23.2909*$	2.9649	$4.4228E+02$
30000	$4.7941E+02$	26.2026	7.1209	3.6797	$8.7217E+02$
40000	$6.2237E+02$	57.4135	22.3990	2.9596	$2.7652E+03$
50000	$1.2707E+03$	108.8633	37.1745	2.9284	$4.8148E+03$
60000	$5.6175E+02$	31.2251	12.6381	3.2398	$4.9722E+03$
70000	$1.1303E+02$	63.8185	20.0467	3.1835	$6.9434E+03$

Table B.24: *The comparison of the EIEM Orthomin implementation for 100 and 200 iterations, for the case of* $\delta = 8$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	2.2841	1.0754	52.92
2000	1.3167	1.5166	15.18
3000	4.5569	4.5569	0
4000	5.2953	4.5953	13.2
5000	3.3338	3.3338	0
6000	8.8934	8.8934	0
7000	6.6249	6.6249	0
8000	9.5224	9.5224	0
9000	6.4019	3.1667	50.53
10000	11.7313	11.7313	0
20000	19.8233	23.2909	17.49
30000	7.1209	7.1209	0
40000	22.3990	22.3990	0
50000	37.1745	37.1745	0
60000	12.6381	12.6381	0
70000	20.0467	20.0467	0

ticularly when using 200 iterations. For instance, when solving 3000 dimensions with 100 iterations, the residual norm of the model solution is 7 times smaller than the lowest residual norm of the iterates generated by Orthores process. This factor remained the same when using 200 iterations. Furthermore, when solving dimensions 5000 using 100 iterations, the decrease number is about 3.7. This factor rose up significantly to 6.24 when we increased the iterations. This trend also occurs in large dimensions, such as in dimensions 20000, 40000, and 60000, where the decrease numbers when using 100 iterations are respectively 3.9, 3.5, and 4.2.

We also highlight here that the improvement as a result in increasing the iterations in the algorithm almost occurs in most problems (see Table B.27). For

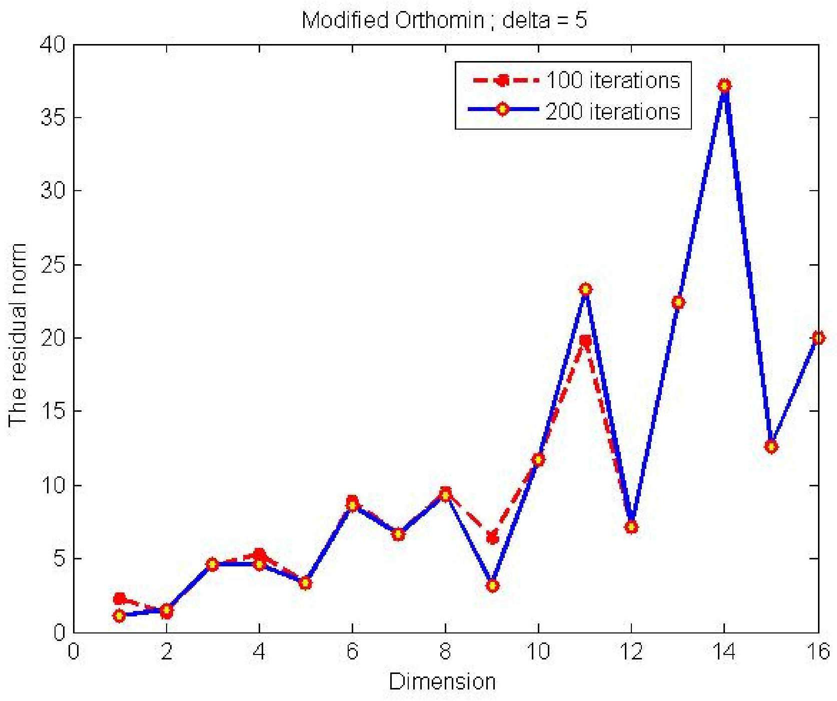


Figure B.8: *The behaviour of residual norms of the iterates generated by EIEM Orthomin for 100 and 200 iteration; the case of* $\delta = 8$

instance, when solving dimensions 1000, 6000, and 7000, the percentage decrease appear significantly which are about 92%, 93%, and 96%. Note, the percentage decrease/increase in this table describes how far the model residual norm decrease/increase from 100 iterations to 200 iterations. Furthermore, in large dimensions such as 20000, 30000, and 60000, the model residual norms when using 100 iterations are respectively about 1.32, 0.26, and 18.66, rose up considerably to 0.04, 0.05, and 0.04 respectively. In other words, the percentage decrease in these cases are about 96%, 81%, and 99% respectively. Other cases have the percentage decrase/increase are zero, which means that there is no improvement of the model residual norms when we increased the the iterations. The improvements of the residual norms as a results in increasing the iterations is illustrated in Figure B.9.

Table B.25: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 100 iterations,* $\delta = 0.2$

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	2.4674	0.5201	0.1740	2.9891	0.698
2000	2.2724	0.9674	0.1986	4.8711	1.4259
3000	$4.8757E+03$	39.6925	5.4254	7.3161	2.3106
4000	$1.43872E-01$	0.1439	0.0480	2.9979	4.0736
5000	$5.5639E+01$	26.7770	0.8679	3.7053	5.3188
6000	2.9042	2.9042	1.1009	2.6380	7.2979
7000	$1.0182E+01$	2.8798	1.0745	2.6801	9.4892
8000	$1.4693E+01$	14.1454	5.3622	2.6379	11.9616
9000	4.6987	1.7645	0.4689	3.7631	14.99
10000	$1.5058E+01$	10.3933	3.8568	2.6948	18.0196
20000	$2.4341E+02$	5.1874	1.3162	3.9412	64.3696
30000	$9.1983E-01$	0.9057	0.2654	3.4126	$1.6457E+02$
40000	$1.2038E+02$	20.5881	5.7758	3.5646	$1.7665E+03$
50000	$1.5064E-01$	0.1011	0.0299	3.3813	$1.5649E+03$
60000	$4.2012E+02$	77.9312	18.6620	4.1759	$3.1967E+03$
70000	5.4393	3.7597	1.3637	2.7569	$3.5734E+03$

Table B.26: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 200 iterations,* $\delta = 0.2$

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$7.1654E-02$	0.0425	0.0137	3.1022	0.7924
2000	$4.6799E-01$	0.0918	0.0221	4.1539	1.9527
3000	$4.8757E+03$	39.6925	5.4254	7.3161	2.6044
4000	$7.9202E-02$	0.0477	0.0113	4.2212	7.1707
5000	1.1375	0.6078	0.0973	6.2467	7.2108
6000	$6.8539E-01$	0.2202	0.0777	2.8339	9.9596
7000	3.4682	0.1095	0.0415	2.6386	13.1762
8000	$9.0036E-01$	0.3853	0.0804	4.7923	16.7186
9000	$5.0894E-01$	0.2649	0.0953	2.7796	21.9828
10000	8.08204	3.5266	0.9218	3.8258	28.2759
20000	$6.9922E+01$	0.1812	0.0456	3.9737	$1.0085E+02$
30000	$7.5801E-01$	0.1835	0.0510	3.5980	$4.8194E+02$
40000	$5.6718E+01$	20.5881	5.7758	2.7579	$9.8724E+03$
50000	$1.506E-01$	0.1011	0.0299	3.3813	$1.5649E+03$
60000	$1.7953E-01$	0.1511	0.0442	3.4186	$3.2919E+03$
70000	$1.4044E+01$	1.5592	0.4582	3.4029	$3.3566E+03$

Table B.27: *The comparison of the EIEM Orthores implementation for 100 and 200 iterations, for the case of $\delta = 0.2$*

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.1740	0.0137	92.13
2000	0.1986	0.0221	88.87
3000	5.4254	5.4254	0
4000	0.0480	0.0113	76.46
5000	0.8679	0.0973	88.79
6000	1.1009	0.0777	92.94
7000	1.0745	0.0415	96.14
8000	5.3622	0.0804	98.5
9000	0.4689	0.0953	79.68
10000	3.8568	0.9218	76.09
20000	1.3162	0.0456	96.54
30000	0.2654	0.0510	80.78
40000	5.7758	5.7758	0
50000	0.0299	0.0299	0
60000	18.6620	0.0442	99.76
70000	1.3637	0.4582	66.4

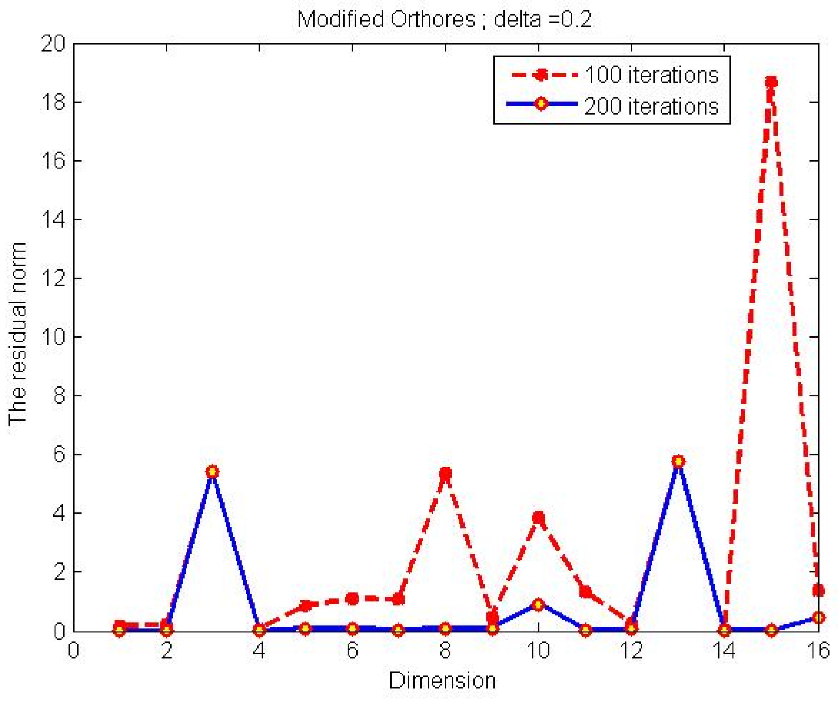


Figure B.9: *The behaviour of residual norms of the iterates generated by EIEM Orthores for 100 and 200 iteration; the case of $\delta = 0.2$*

B.1.10 EIEM Orthores Algorithm: $\delta = 0.5$

According to Tables B.28 and B.29, the decrease numbers mostly show a significant improvement when the EIEM applied, particularly when we increased the number of iterations. For instance, for dimensions 1000 and 2000, the residual norms of the model solution are about 4 times smaller respectively, than the minimum residual norm of the iterate generated by the original Orthores. Furthermore, for a large problems, such as dimensions 20000, 40000, and 60000, the decrease numbers are about 4.1, 4, and 4.7 respectively. It means that the residual norms of the model solutions are about respectively 4,4, and 5 times smaller than the lowest residual norm of the previous iterates.

The changing of the residual norms of the model solutions as a result in increasing the iterations can be seen in Table B.30 as well as in Figure B.10. We can see here that the positive changing occurs in almost all of problems, only three problems with no changing. For instance, the highest percentage decrease of the residual norm of the model solution was obtained when solving 1000 dimensions problem with 95%. It followed by dimensions 4000, 6000, and 8000 with respectively 94%, 94, and 93% of decrease. For high scale problems, however, the percentage decrease seems went down. For instance, in dimensions 40000, 50000, and 60000, the percentage decrease are about 54%, 87%, and 64% respectively. Only the highest dimensional problem gets a high percentage decrease which is about 94%.

Table B.28: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 100 iterations,* $\delta = 0.5$

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$6.4489E-02$	0.0375	0.0095	3.9474	0.6182
2000	$6.8363E-01$	0.1686	0.0423	3.9858	1.3122
3000	$1.6074E+02$	0.2312	0.0876	2.6393	2.3003
4000	$9.1293E-01$	0.9129	0.3174	2.8762	3.7671
5000	$2.4654E-01$	0.1761	0.0464	3.7953	5.3663
6000	3.1706	0.2218	0.0658	3.3708	7.3712
7000	$1.6443E-01$	0.0802	0.0304	2.6382	8.4712
8000	3.4744	0.1608	0.0609	2.6404	11.8802
9000	$4.7236E-02$	0.0472	0.0156	3.0256	14.6132
10000	$8.4735E-01$	0.1311	0.0468	2.8013	17.7345
20000	8.9266	3.3489	0.8250	4.0593	64.6193
30000	$6.8594E+01$	0.3350	0.0941	3.5600	$1.5220E+02$
40000	4.7609	0.8977	0.2217	4.0492	$6.2563E+03$
50000	$6.8238E-02$	0.0652	0.0241	2.7054	$1.0261E+03$
60000	$5.0281E-01$	0.5028	0.1058	4.7527	$2.7166E+03$
70000	1.3202	1.0184	0.3561	2.8599	$3.2102E+03$

Table B.29: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 200 iterations,* $\delta = 0.5$

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$4.1693E-03$	0.0015	$4.3499E-04$	3.44836	0.7556
2000	$6.8364E-01$	0.1686	0.0423	3.9858	1.6578
3000	$1.6074E+02$	0.2312	0.0876	2.6393	3.3701
4000	$7.8225E-02$	0.0640	0.0191	3.3508	5.6576
5000	$9.4144E-02$	0.0284	0.0067	4.2388	7.9878
6000	$4.3752E-02$	0.0256	0.0040	6.4	11.1991
7000	$5.0291E-02$	0.0295	0.0050	5.9	14.1003
8000	$3.9382E-02$	0.0130	0.0046	2.8261	17.7754
9000	$3.0444E-02$	0.0096	0.0030	3.2	22.1213
10000	6.1736	0.0500	0.0079	6.3291	27.6473
20000	$8.8953E-01$	0.5075	0.1422	3.5689	89.9022
30000	1.7087	0.3350	0.0941	3.5600	$2.3071E+02$
40000	$6.8325E-01$	0.5429	0.1030	5.2709	$9.6040E+03$
50000	$3.4279E-02$	0.0329	0.0032	10.2813	$1.5552E+03$
60000	$5.7642E-01$	0.1875	0.0380	4.9342	$4.1907E+03$
70000	$1.6495E-01$	0.0736	0.0220	3.3455	$4.7222E+03$

Table B.30: *The comparison of the EIEM Orthores implementation for 100 and 200 iteration, for the case of* $\delta = 0.5$

Dimension	$\lvert\lvert\mathbf{r}_{model}\rvert\rvert$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.0095	$4.3499E-04$	95.42
2000	0.0423	0.0423	0
3000	0.0876	0.0876	0
4000	0.3174	0.0191	93.98
5000	0.0464	0.0067	85.56
6000	0.0658	0.0040	93.92
7000	0.0304	0.0050	83.55
8000	0.0609	0.0046	92.45
9000	0.0156	0.0030	80.77
10000	0.0468	0.0079	83.12
20000	0.8250	0.1422	82.76
30000	0.0941	0.0941	0
40000	0.2217	0.1030	53.54
50000	0.0241	0.0032	86.72
60000	0.1058	0.0380	64.08
70000	0.3561	0.0220	93.82

B.1.11 EIEM Orthores Algorithm: $\delta = 0.8$

If we look at the decrease numbers in Tables B.31 and B.31, the significant numbers appear in dimensions 60000 (for 100 iterations) which is 43.5, and in dimensions 7000 and 30000 (for 200 iterations) which are respectively 35.8 and 27.4 It means that the residual norms of the model solutions are smaller than the minimum residual norms of the iterates generated by the Orthores algorithm. Another fantastic numbers also appear in dimensions 3000 and 40000 (for 100 iterations), which are 6.9 and 6 .2 respectively, and for dimensions 60000, 9000, and 2000, which are about 13.7, 11.7, and 10.6 respectively.

Based on the information on Table B.33, the highest percentage decrease is obtained when solving 8000 problem which is about 100% decreased. There

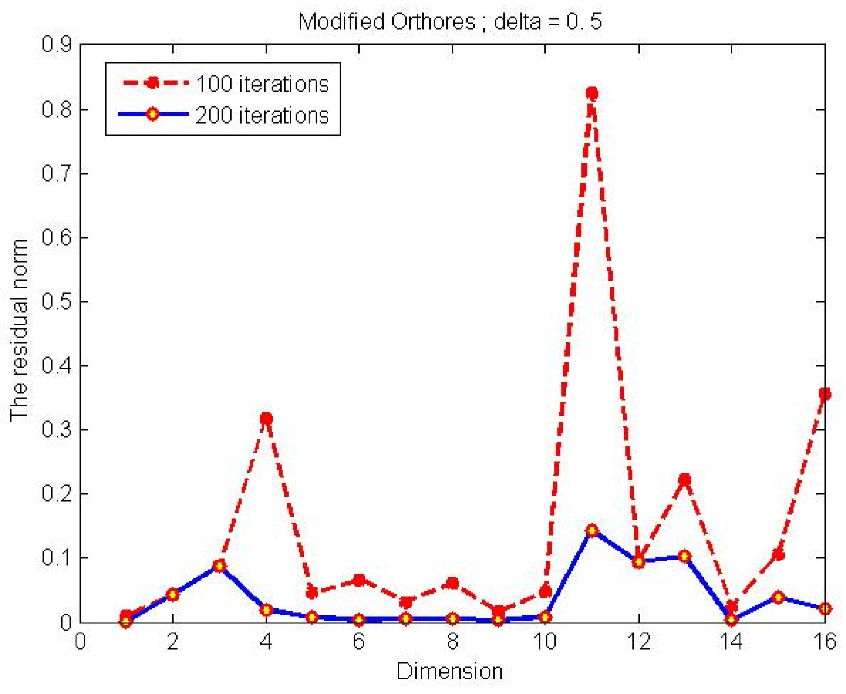


Figure B.10: *The behaviour of residual norms of the iterates generated by EIEM Orthores for 100 and 200 iteration; the case of* $\delta = 0.5$

are some problems which reached 99% of decreasing when we increased the iterations, namely dimensions 3000, 4000, 7000, 20000, 30000, and 60000. Other problems have the percentage decrease which are about 90%. The behaviour of these residual norms are captured in Figure B.6.

B.1.12 EIEM Orthores Algorithm: $\delta = 5$

In this particular case, increasing the iterations up to 200 does not make a significant changing of the residual norms od the model solutions in most problems. The improvement is only appeared however, when solving large scale problems, i.e. for dimensions 10000 and 30000, where the percentage decrease reached up to 95% and 98% respectively, according to Table B.36. It followed by dimensions 40000 and 60000 with the percentage decrease are about 82% and 81% respectively. The largest problem, surprisingly, only obtained 59% of decreasing. This

Table B.31: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 100 iterations,* $\delta = 0.8$

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$3.7530E-02$	0.0082	0.0031	2.6452	0.6676
2000	$1.3082E+01$	0.6285	0.1783	3.5249	1.548
3000	$1.8496E+02$	21.2092	3.0499	6.9541	2.7656
4000	1.6492	1.3626	0.3561	3.8265	4.2714
5000	$2.3216E+01$	3.2991	1.2506	2.6380	5.814
6000	$1.0248E-01$	0.0984	0.0339	2.9027	7.6747
7000	$1.1685E-01$	0.0724	0.0230	3.1478	10.1714
8000	9.5833	4.4633	1.6919	2.6380	12.7683
9000	1.5925	0.2431	0.0723	3.3624	16.0025
10000	1.2868	0.3596	0.0739	4.8660	19.3439
20000	$2.8193E+01$	0.7484	0.2837	2.6379	68.7558
30000	$1.4483E-01$	0.1448	0.0424	3.4151	$1.6629E+02$
40000	$2.6822E-01$	0.2682	0.0433	6.1939	$8.3926E+02$
50000	$4.6801E+04$	71.2207	26.3431	2.7036	$1.3590E+03$
60000	7.2134	7.2134	0.1658	43.5066	$1.9154E+03$
70000	$5.5968E-01$	0.5597	0.1588	3.5246	$2.1882E+03$

Table B.32: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 200 iterations,* $\delta = 0.8$

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	0.0034	0.0011	$1.4073E-04$	7.8164	0.7337
2000	$1.4715E-01$	0.0251	0.0063	3.9841	1.7895
3000	$3.7463E-02$	0.0244	0.0035	6.9714	3.3701
4000	$6.1324E-03$	0.0041	0.0012	3.4167	5.4888
5000	$3.8861E+01$	2.0343	0.3769	5.3975	6.7686
6000	$1.0248E-01$	0.0984	0.0339	2.9027	7.9037
7000	$9.4114E-03$	0.0061	$1.7048E-04$	35.7981	15.8099
8000	$5.1757E-01$	0.2053	0.0331	6.2024	17.5319
9000	$4.3398E-02$	0.0129	0.0011	11.7273	24.1204
10000	$1.2177E-02$	0.0043	0.0013	3.3077	30.6657
20000	$0189E-02$	0.0062	$5.836E-04$	10.6237	84.2992
30000	$1.1425E-02$	0.0048	$1.7522E-04$	27.3973	$2.3564E+02$
40000	$9.2999E-02$	0.0332	0.0051	6.5098	$5.5072E+03$
50000	$6.4700E+02$	2.7680	0.6599	4.1946	$1.2514.9E+04$
60000	$1.7145E-02$	0.0107	$7.8074E-04$	13.7049	$1.8012E+04$
70000	$1.2817E-01$	0.0400	0.0079	5.0632	$1.8906E+04$

Table B.33: *The comparison of the EIEM Orthores implementation for 100 and 200 iterations, for the case of* $\delta = 0.8$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.0031	$1.4073E-04$	95.46
2000	0.1783	0.0063	96.47
3000	3.0499	0.0035	99.89
4000	0.3561	0.0012	99.66
5000	1.2506	0.3769	69.86
6000	0.0339	0.0339	0
7000	0.0230	$1.7048E-04$	99.26
8000	1.6919	$1.7522E-04$	99.98
9000	0.0723	0.0331	54.36
10000	0.0739	0.0011	98.51
20000	0.2837	$5.836E-04$	99.79
30000	0.0424	$1.7522E-04$	99.58
40000	0.0433	0.0051	88.22
50000	26.3431	0.6599	97.49
60000	0.1658	$7.8074E-04$	99.53
70000	0.1588	0.0079	95.03

behaviour can be seen clearly in Figure B.12.

B.1.13 EIEM Orthores Algorithm: $\delta = 8$

In this particular case, imposing the EIEM in the algorithm is also worth it. The decrease numbers in both Tables B.37 and B.38 were varies ; ranging between 3 and 7. For instance, the highest decrease number was obtained when solving 8000 problem with 100 iterations, i.e. it was approximately 6.98. It means that the residual norm of the model solution is about 7 times smaller than the lowest residual norm of all of the previous iterates generated by the original Orthores. When using 200 iterations, however, it was achieved for solving dimensions 40000 with the decrease number was about 7.01. The lowest decrease numbers of both using 100 and 200 iterations (i.e. respectively 2.66 and 2.95) were obtained when solving dimensions 1000 and 2000 respectively.

Table B.34: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 100 iterations, $\delta = 5$*

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$1.3199E+01$	2.7546	1.0190	2.7032	0.5031
2000	2.5579	0.7233	0.2742	2.6379	1.3359
3000	$6.4057E+01$	4.1652	1.5789	2.6380	2.5045
4000	$3.8162E+02$	$0.9450n$	0.3357	2.7434	3.9903
5000	$1.7846E+01$	2.7709	0.9516	2.9118	5.8211
6000	$1.4266E+02$	4.1558	1.5754	2.6379	7.9541
7000	$3.8086E+01$	6.7323	1.8338	3.6712	10.5452
8000	$8.9181E+02$	5.6826	1.1945	4.7573	13.2839
9000	$2.8539E+01$	3.0953	1.1734	2.6379	16.5919
10000	$1.7500E+02$	3.5354	1.3197	2.6789	20.1626
20000	$1.3453E+02$	4.3751	1.2507	3.4981	67.2667
30000	$3.2474E+02$	14.4268	5.4688	2.6380	$1.5113E+02$
40000	$1.4301E+01$	3.9251	1.4879	2.6380	$8.6406E+03$
50000	$6.8157E+02$	52.7413	19.8173	2.6614	$8.5284E+03$
60000	$4.7690E+01$	40.9302	12.6033	3.2476	$1.1920E+04$
70000	$1.5397E+01$	12.2256	4.6344	2.6380	$1.7828E+04$

Table B.35: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 200 iterations, $\delta = 5$*

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$2.3456E-01$	0.2013	0.0602	3.3439	0.6424
2000	2.0746	0.7233	0.2742	2.6378	1.7676
3000	1.3875	1.3875	0.4148	3.3449	3.4424
4000	$6.0728E+01$	0.9450	0.3357	2.8150	5.1737
5000	$3.8175E+01$	2.7709	0.9516	2.9118	5.5528
6000	$6.2632E+01$	4.1558	1.5754	2.6379	8.2538
7000	$1.7872E+01$	1.0734	0.2773	3.8709	15.7251
8000	7.9552	1.6116	0.4818	3.3449	19.6892
9000	$2.6287E+01$	3.0953	1.1734	2.6379	24.5682
10000	$2.3455E-01$	0.2013	0.0602	3.3439	23.9232
20000	3.5497	2.1630	0.5980	3.6171	$1.0006E+02$
30000	$2.8936E-01$	0.2894	0.0865	3.3457	$2.2438E+02$
40000	4.5911	0.8883	0.2656	3.3445	$1.5960E+04$
50000	$8.5273E+01$	40.6797	12.1614	3.3449	$1.6936E+04$
60000	$4.3547E+01$	9.6274	2.4449	3.9377	$2.6050E+04$
70000	$2.0613E+01$	7.9062	1.8882	4.1872	$3.6839E+04$

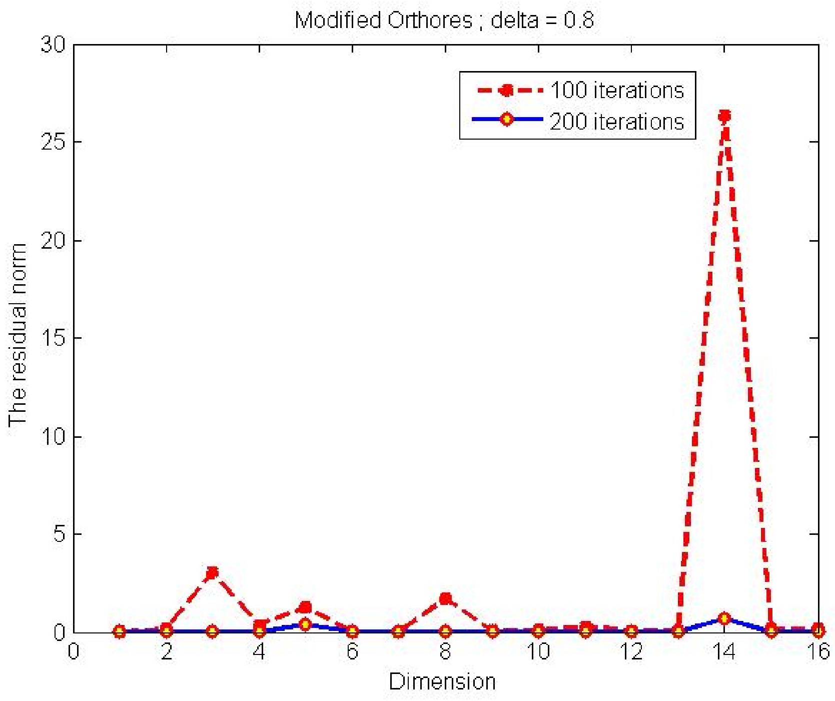


Figure B.11: *The behaviour of residual norms of the iterates generated by EIEM Orthores for 100 and 200 iteration; the case of* $\delta = 0.8$

The percentage decrease of the residual norms of the model solutions as a result in increasing the iterations from 100 to 200 was put in Table B.39. The highest of the percentage decrease, for instance, was reached when solving dimensions 30000 of the problem; i.e. up to 96%. It means that the residual norm of the iterate generated by the EIEM was about 96% decrease when the iterations increased up to 200. The lowest of the percentage decrease, however, was obtained when solving 20000 dimensional problem which was 40%. Interestingly, about 3% of percentage increase when solving 40000 dimensional problem, and there is no changing of the two residual norms when solving dimensions 2000. All of the behaviour of the residual norms of the model iterate by increasing the iterations from 100 to 200 are captured in Figure B.13.

Table B.36: *The comparison of the EIEM Orthores implementation for 100 and 200 iterations, for the case of* $\delta = 5$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	1.0190	0.0602	94.09
2000	0.2742	0.2742	0
3000	1.5789	0.4148	73.73
4000	0.3357	0.3357	0
5000	0.9516	0.9516	0
6000	1.5754	1.5754	0
7000	1.8338	0.2773	84.87
8000	1.1945	0.4818	59.67
9000	1.1734	1.1734	0
10000	1.3197	0.0602	95.44
20000	1.2507	0.5980	52.19
30000	5.4688	0.0865	98.42
40000	1.4879	0.2656	82.15
50000	19.8173	12.1614	38.63
60000	12.6033	2.4449	80.6
70000	4.6344	1.8882	59.26

Table B.37: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 100 iterations,* $\delta = 8$

Dim	Orthores		Orthores with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$1.1182E+02$	12.1604	4.5689	2.6616	0.6034
2000	$3.0349E+02$	6.2720	2.1240	2.9529	1.3823
3000	$2.8841E+01$	19.4297	5.6712	3.4260	2.5522
4000	9.4642	9.4643	1.3299	3.6292	3.8659
5000	5.9627	4.8265	9.1505	2.9263	5.4138
6000	$2.5258E+01$	20.2821	7.1811	2.8244	7.5282
7000	$1.1174E+01$	4.3063	1.3689	3.1458	9.78
8000	$1.1336E+01$	11.3369	1.6239	6.9813	12.2216
9000	$1.046E+02$	21.6714	5.3962	4.0160	15.0955
10000	$5.5428E+02$	12.0280	4.5595	2.6380	18.2792
20000	$2.7060E+02$	50.9992	14.7468	3.4583	64.8147
30000	$1.8367E+03$	67.5386	19.3364	3.4928	$1.7858E+02$
40000	$7.4530E+01$	48.7460	18.4784	2.6379	$3.5438E+03$
50000	$2.2490E+02$	113.0310	40.0828	2.8199	$5.6528E+03$
60000	$1.4372E+02$	59.8770	21.4780	2.7878	$9.7571E+03$
70000	$5.5557E+01$	31.3792	9.1514	3.4288	$1.0918E+04$

Table B.38: *Comparison between the residual norms of the iterates generated by the original Orthores algorithm and those generated by EIEM in Orthores algorithm for 200 iterations, $\delta = 8$*

Dim	Orthores		Orthores with EIEM		
n	$\lVert\mathbf{r}_k\rVert$	$\lVert\mathbf{r}_m\rVert$	$\lVert\mathbf{r}_{model}\rVert$	Decrease	Time(s)
1000	$1.4196E+01$	3.2824	0.7113	4.6147	0.6395
2000	$4.9479E+01$	6.2720	2.1240	2.9529	1.7602
3000	$2.9939E+01$	11.2040	2.7154	4.1261	3.2809
4000	3.5545	2.4593	0.7352	3.3451	5.2631
5000	6.3762	2.6509	0.6040	4.3889	7.5837
6000	$5.0436E+01$	15.5616	4.1370	3.7616	10.4562
7000	$2.4665E-01$	0.2440	0.0729	3.3471	13.903
8000	2.7090	1.5714	0.4203	3.7388	17.6931
9000	2.4041	1.7006	0.4992	3.4067	21.9681
10000	$1.0140E+01$	3.7178	1.0948	3.3959	26.9081
20000	$8.5786E+02$	30.3630	8.8066	3.4478	$1.0045E+02$
30000	$1.1301E+01$	2.8657	0.7814	3.6674	$4.7771E+02$
40000	23.4966	52.5156	19.0412	7.0181	$2.6114E+03$
50000	$2.4468E+01$	20.9844	5.4758	3.8322	$1.9505E+03$
60000	$8.5716E+01$	29.7359	6.7185	4.4259	$5.5161E+03$
70000	$3.8253E+02$	18.6180	3.7744	$4.9327'$	$1.5804E+03$

Table B.39: *The comparison of the EIEM Orthores implementation for 100 and 200 iterations, for the case of $\delta = 8$*

Dimension	$\lVert\mathbf{r}_{model}\rVert$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	4.5689	0.7113	84.43
2000	2.1240	2.1240	0
3000	5.6712	2.7154	52.12
4000	1.3299	0.7352	44.72
5000	9.1505	0.6040	93.4
6000	7.1811	4.1370	42.39
7000	1.3689	0.0729	94.67
8000	1.6239	0.4203	74.12
9000	5.3962	0.4992	90.75
10000	4.5595	1.0948	75.99
20000	14.7468	8.8066	40.28
30000	19.3364	0.7814	95.96
40000	18.4784	19.0412	3.05
50000	40.0828	5.4758	86.34
60000	21.4780	6.7185	68.72
70000	9.1514	3.7744	58.76

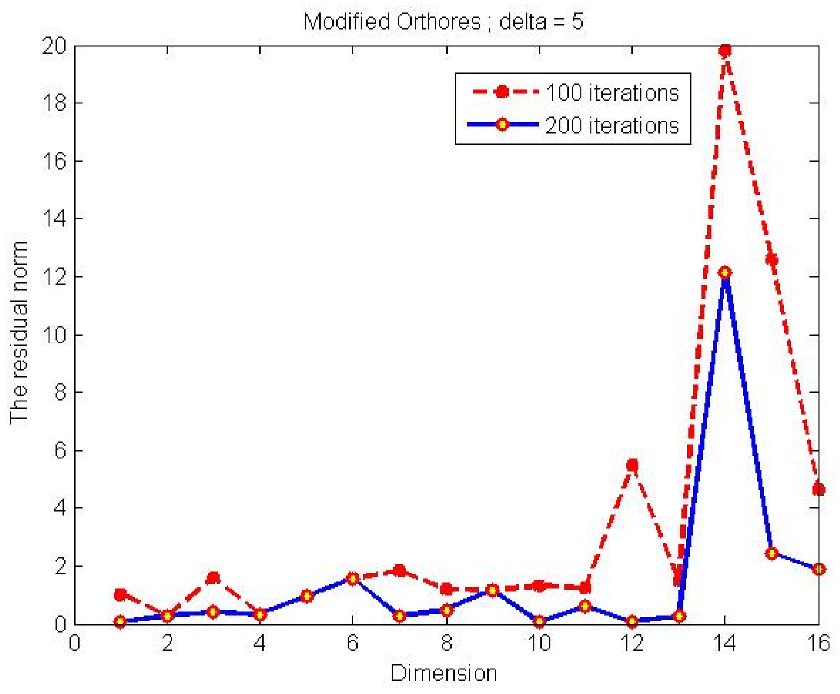


Figure B.12: *The behaviour of residual norms of the iterates generated by EIEM Orthores for 100 and 200 iteration; the case of* $\delta = 5$

B.1.14 EIEM A12 Algorithm: $\delta = 0.5$

It can be seen in both Tables B.40 and B.41 that there is no breakdown occurs in this particular case. We also highlight here that the decrease numbers in both tables mostly show the improvement when the EIEM applied in A12, particularly when we increased the number of iterations. For instance, in dimensions 20000 and 30000, the decrease numbers are about 4.3 and 4.7 respectively. It means that they are 4 and 5 times smaller than the lowest residual norms of the iterates generated by the original A12.

The improvement of the residual norms of the model solutions as a result in increasing the iterations can be seen in Table B.42 as well as Figure B.14. We can see here that the positive changing occurs in almost all of problems, only three

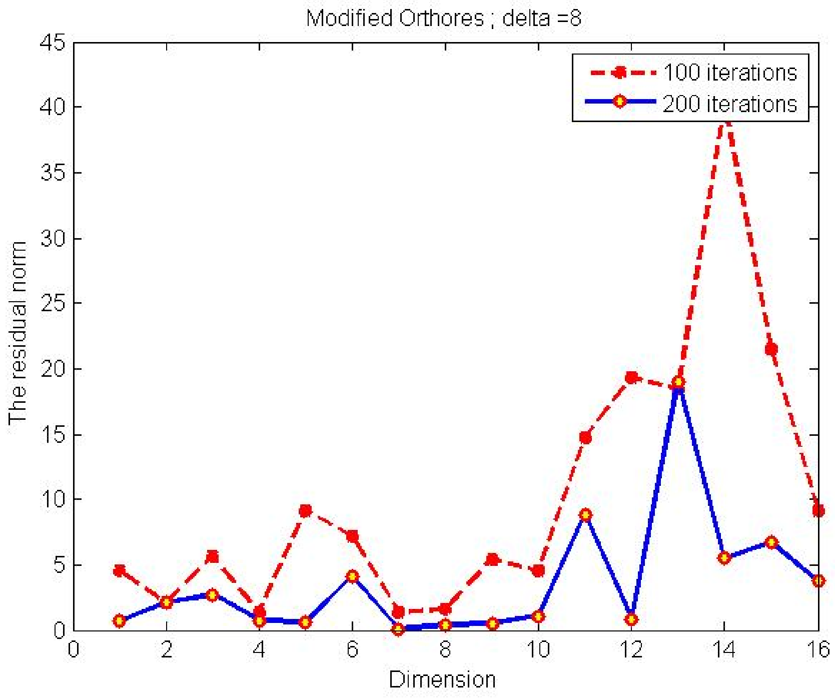


Figure B.13: *The behaviour of residual norms of the iterates generated by EIEM Orthores for 100 and 200 iteration; the case of* $\delta = 8$

problems with no changing. For instance, the highest percentage decrease of the model residual norm was obtained when solving 10000 dimensions problem with 72%. It followed by dimensions 8000, 50000, and 6000 with respectively 61%, 59%, and 53% of decrease. For high scale problems, however, the percentage decrease seems go down. For instance, in dimensions 60000, 70000, the percentage decrease are about 28% and 8% respectively.

B.1.15 EIEM A12 Algorithm: $\delta = 0.8$

In this particular case, the A12 algorithm performs better to solve the SLEs than the previous cases. It is indicated by some residual norms of the iterates generated by the original A12 which are smaller than those were in other types algorithms. This, interestingly, affects the improvement of the residual norms generated by the A12 algorithm with the EIEM. It can be seen in Table B.43, the decrease num-

Table B.40: *Comparison between the residual norms of the iterates generated by the original A12 algorithm and those generated by EIEM in A12 algorithm for 100 iterations,* $\delta = 0.5$

Dim	A12		A12 with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$5.9969E-02$	0.0600	0.0227	2.6432	0.5615
2000	0.5972	0.1120	0.0352	3.1818	1.6065
3000	0.1005	0.0556	0.0311	1.7878	2.999
4000	0.1407	0.0701	0.0245	2.8612	4.9411
5000	0.4651	0.1919	0.0635	3.0221	7.7084
6000	$2.5333E-02$	0.0253	0.0077	3.2857	10.8655
7000	7.8441	1.7013	0.6238	2.7273	13.6886
8000	3.1706	0.9685	0.4671	2.0734	17.8277
9000	0.3243	0.1412	0.0350	4.0343	22.3657
10000	0.2215	0.1808	0.0584	3.0959	26.8259
20000	2.6856	0.7996	0.1849	4.3245	99.5463
30000	0.8136	0.6000	0.1275	4.7059	$1.9590E+02$
40000	0.5553	0.3511	0.1320	2.6598	$3.1594E+03$
50000	1.7156	1.6121	0.4702	3.4285	$5.3506E+03$
60000	2.3382	2.3382	0.7862	2.9741	$7.2492E+03$
70000	0.4022	0.4022	0.1486	2.7066	$9.883E+03$

Table B.41: *Comparison between the residual norms of the iterates generated by the original A12 algorithm and those generated by EIEM in A12 algorithm for 200 iterations,* $\delta = 0.5$

Dim	A12		A12 with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$8.7937E-02$	0.0409	0.0141	2.9007	0.8212
2000	0.4517	0.1120	0.0352	3.18182	1.9448
3000	0.6605	0.0358	0.0134	2.6716	3.6717
4000	0.3924	0.0600	0.0173	3.4682	6.0855
5000	1.7385	0.1728	0.0533	3.2420	8.4621
6000	$5.8629E-02$	0.0122	0.0036	3.3889	11.7
7000	4.9305	1.7013	0.6238	2.7273	15.2835
8000	0.5975	0.4901	0.1842	2.6607	20.0747
9000	1.0783	0.1412	0.0350	4.0343	24.8756
10000	1.4388	0.0545	0.0161	3.3851	30.6666
20000	5.8252	0.7330	0.1443	5.0797	$1.2194E+02$
30000	0.2857	0.2504	0.0749	3.3431	$2.4338E+02$
40000	0.2324	0.2072	0.0685	3.0248	$2.3348E+03$
50000	2.6544	1.0521	0.1934	5.4400	$6.2173E+03$
60000	2.2387	2.0271	0.5683	3.5669	$1.0506E+04$
70000	0.3794	0.3661	0.1372	2.6684	$1.3011E+04$

Table B.42: *The comparison of the EIEM A12 implementation for 100 and 200 iterations, for the case of* $\delta = 0.5$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.0227	0.0141	37.89
2000	0.0352	0.0352	0
3000	0.0311	0.0134	56.91
4000	0.0245	0.0173	29.39
5000	0.0635	0.0533	16.06
6000	0.0077	0.0036	53.25
7000	0.6238	0.6238	0
8000	0.4671	0.1842	60.56
9000	0.0350	0.0350	0
10000	0.0584	0.0161	72.43
20000	0.1849	0.1443	21.96
30000	0.1275	0.0749	41.25
40000	0.1320	0.0685	48.11
50000	0.4702	0.1934	58.87
60000	0.7862	0.5683	27.72
70000	0.1486	0.1372	7.67

ber of dimensions 9000 is about 8.35; which means that the residual norm of the model solution is 8 times smaller than the minimum residual norm of the iterate generated by the algorithm without the embedded model. Similarly, when solving 6000, the decrease number is about 4.8. This trend, however, does not appear in Table B.44, or when we use 200 iterations. Here, the decrease number seems steady at the value of 3. Yet, for dimensions 5000 and 6000, the decrease number are only 1.5 and 1.2 respectively.

The comparison of the residual norms of the model solutions when using 100 and 200 iterations can be seen Table B.45. According to the table, the highest of the percentage decrease was obtained when solving 30000 dimensional problems, i.e. about 95%. It is followed by dimensions 3000 which is about 84%, dimensions

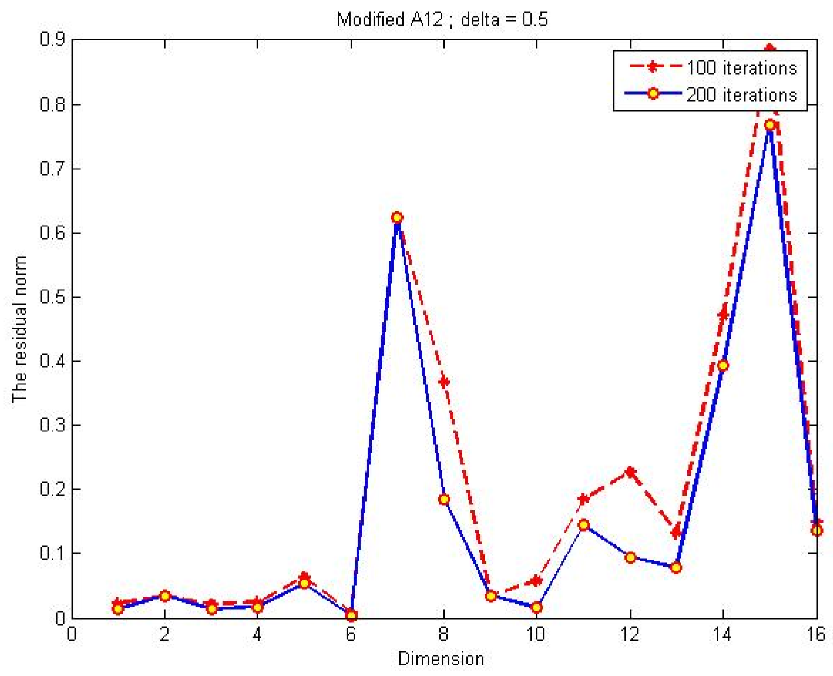


Figure B.14: *The behaviour of residual norms of the iterates generated by EIEM A12 for 100 and 200 iteration; the case of* $\delta = 0.5$

8000 which is about 81%, dimensions 60000 and 70000 which are about 77%, and dimensions 10000 which is about 75%. Only two cases with no improvement, i.e. dimensions 4000 and 40000. The behaviour of these residual norms is captured in Figure B.15.

B.1.16 EIEM A12 Algorithm: $\delta = 5$

As we can see in Table B.46 and Table B.47 that the breakdown does not occur in A12 algorithm using both 100 and 200 iterations. Slightly different from the previous cases that the improvement of the residual norms after imposing the EIEM in the A12 algorithm are not too significant. The highest of the decrease factor appears in dimensions 40000 which is about 4.1. The lowest one is about 1.8 for dimensions 1000. Other problems seem have similar value which are about 3. These values also do not change significantly when we increased the iterations up to 200.

Table B.43: *Comparison between the residual norms of the iterates generated by the original A12 algorithm and those generated by EIEM in A12 algorithm for 100 iterations,* $\delta = 0.8$

Dim	A12		A12 with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	0.0066	0.0049	0.0019	2.5789	0.5547
2000	0.9513	0.1751	0.0491	3.5662	1.5953
3000	0.0254	0.0224	0.0082	2.7317	3.0537
4000	1.7353	0.2097	0.0570	3.6789	4.7695
5000	0.0083	0.0083	0.0031	2.6774	7.09
6000	0.6732	0.5791	0.1195	4.8460	9.7174
7000	0.0403	0.0169	0.0054	3.1296	12.7549
8000	0.5499	0.2931	0.1311	2.2357	16.2716
9000	0.0395	0.0192	0.0023	8.3478	20.2395
10000	0.9086	0.2477	0.0778	3.1838	24.6824
20000	0.1439	0.1438	0.0404	3.5594	92.0772
30000	8.6236	0.0471	0.0176	2.6761	$1.9519E+02$
40000	39.3774	1.5916	0.5033	3.1623	$1.9553E+03$
50000	0.2102	0.0956	0.0352	2.7159	$4.7054E+03$
60000	1.7048	0.4816	0.1682	2.8633	$5.2172E+03$
70000	0.2262	0.0872	0.0302	2.8874	$9.9424E+03$

Table B.44: *Comparison between the residual norms of the iterates generated by the original A12 algorithm and those generated by EIEM in A12 algorithm for 200 iterations,* $\delta = 0.8$

Dim	A12		A12 with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	$2.2189E-03$	0.0022	$7.9498E-04$	2.7674	0.6237
2000	0.2326	0.0653	0.0215	3.0372	1.9429
3000	0.1253	0.0110	0.0013	2.6191	3.5862
4000	0.3424	0.2097	0.0570	3.6789	5.8755
5000	$3.8001E-03$	0.0034	0.0023	1.4783	8.6905
6000	5.4971	0.2346	0.1889	1.2419	12.0094
7000	1.4572	0.0169	0.0074	2.2838	16.1133
8000	0.1124	0.0882	0.0240	3.675	20.1751
9000	$4.6968E-03$	0.0030	0.0011	2.7273	25.3264
10000	0.2705	0.0622	0.0191	3.2566	30.7491
20000	0.2949	0.0883	0.0298	2.9631	1.1584
30000	$5.7830E-03$	0.0028	$8.1290E-04$	3.4445	$2.5266E+02$
40000	15.8617	1.5916	0.5033	3.1623	$4.7842E+03$
50000	$7.4341E-02$	0.0373	0.0117	3.1880	$6.9698E+03$
60000	0.1201	0.1201	0.0378	3.1773	$8.7285E+03$
70000	$2.5516E-02$	0.0255	0.0069	3.6957	$1.2646E+04$

Table B.45: *The comparison of the EIEM A12 implementation for 100 and 200 iterations, for the case of* $\delta = 0.8$

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.0019	$7.9498E-04$	58.21
2000	0.0491	0.0215	56.21
3000	0.0082	0.0013	84.15
4000	0.0570	0.0570	0
5000	0.0031	0.0023	25.81
6000	0.1195	0.0889	25.61
7000	0.0054	0.0074	37.03
8000	0.1311	0.0240	81.69
9000	0.0023	0.0011	10.81
10000	0.0778	0.0191	75.45
20000	0.0404	0.0298	26.24
30000	0.0176	$8.1290E-04$	95.38
40000	0.5033	0.5033	0
50000	0.0352	0.0117	66.76
60000	0.1682	0.0378	77.53
70000	0.0302	0.0069	77.15

As mentioned above that increasing the iterations does not affect significantly in the residual norm. We can see in Table B.48, that most cases have 0% of increase/decrease, which means that there is no improvement of the model solutions when we increased the iteration from 100 to 200. there are only few improvement of the residual norms of the model solutions, i.e. for dimensions 7000 and 8000 with the percentage decrease of 11% for both cases. This behaviour can be seen clearly in Figure B.16.

Table B.46: *Comparison between the residual norms of the iterates generated by the original A12 algorithm and those generated by EIEM in A12 algorithm for 100 iterations,* $\delta = 5$

Dim	A12		A12 with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	6.5886	0.5906	0.3239	1.8234	0.7064
2000	12.908	0.9519	0.2324	4.0959	1.9039
3000	84.3474	1.8317	0.6791	2.6973	3.5194
4000	$1.6182E+03$	5.0002	1.6106	3.1046	5.4087
5000	33.0722	0.4479	0.1296	3.4560	7.8716
6000	31.0726	0.7091	0.2660	2.6658	10.8356
7000	$3.1264E+03$	6.9314	2.7275	2.5413	13.9267
8000	$6.2828E+02$	4.9514	1.9969	2.4795	17.7319
9000	$1.0063E+02$	0.6662	0.2070	3.2184	21.7344
10000	$4.2937E+02$	4.0062	0.9987	4.0114	26.333
20000	42.0875	3.9950	1.5016	2.6605	95.9888
30000	6.9736	2.5030	0.8488	2.9489	$2.0412E+02$
40000	$3.5057E+05$	8.9589	2.1763	4.1166	$3.1556E+03$
50000	58.5051	6.4544	2.2965	2.8105	$4.8648E+03$
60000	74.8813	3.0424	1.0533	2.8885	$3.7087E+03$
70000	$6.6705E+02$	3.6396	0.9315	3.9073	$6.3261E+03$

Table B.47: *Comparison between the residual norms of the iterates generated by the original A12 algorithm and those generated by EIEM in A12 algorithm for 200 iterations,* $\delta = 5$

Dim	A12		A12 with EIEM		
n	$\|\|\mathbf{r}_k\|\|$	$\|\|\mathbf{r}_m\|\|$	$\|\|\mathbf{r}_{model}\|\|$	Decrease	Time(s)
1000	0.1222	0.5906	0.3239	2.7611	0.5649
2000	14.7744	0.9519	0.2324	4.0959	1.665
3000	$1.5413E+02$	1.8317	0.6791	2.6973	3.1346
4000	$3.9391E+02$	5.0002	1.6106	3.1046	5.0613
5000	22.2256	0.4479	0.1296	3.4560	7.4937
6000	8.8872	0.7091	0.2660	2.6658	10.1608
7000	$5.7503E+02$	6.9314	2.4275	2.8554	13.5824
8000	$7.3417E+01$	4.9514	1.7749	2.7897	17.2963
9000	36.4292	0.6662	0.2070	3.2184	21.6702
10000	$2.7589E+02$	4.0062	0.9987	4.0114	26.1259
20000	$2.2992E+02$	3.9950	1.5016	2.6605	98.787
30000	68.6757	2.5030	0.8488	2.9489	$2.1067E+02$
40000	$2.2052E+03$	8.9589	2.1763	4.1166	$2.1074E+03$
50000	$1.1818E+02$	6.4544	2.5583	2.5229	$5.0833E+03$
60000	$1.7504E+02$	3.0424	1.0848	2.8046	$6.6581E+03$
70000	$4.5829E+02$	3.6396	1.0377	3.5074	$7.198E+03$

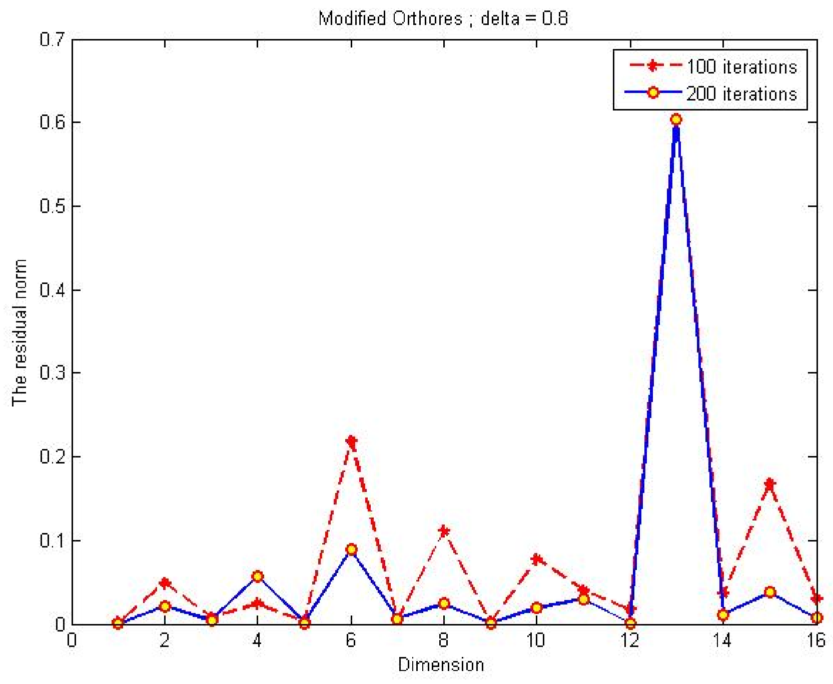


Figure B.15: *The behaviour of residual norms of the iterates generated by EIEM A12 for 100 and 200 iteration; the case of $\delta = 0.8$*

Table B.48: *The comparison of the EIEM A12 implementation for 100 and 200 iterations, for the case of $\delta = 5$*

Dimension	$\|\|\mathbf{r}_{model}\|\|$		Percentage Decrease
n	100 iterations	200 iterations	%
1000	0.3239	0.3239	0
2000	0.2324	0.2324	0
3000	0.6791	0.6791	0
4000	1.6106	1.6106	0
5000	0.1296	0.1296	0
6000	0.2660	0.2660	0
7000	2.7275	2.4275	10.99
8000	1.9969	1.7749	11.11
9000	0.2070	0.2070	0
10000	0.9987	0.9987	0
20000	1.5016	1.5016	0
30000	0.8488	0.8488	0
40000	2.1763	2.1763	0
50000	2.2965	2.5583	11.39
60000	1.0533	1.0848	2.9
70000	0.9315	1.0377	10.23

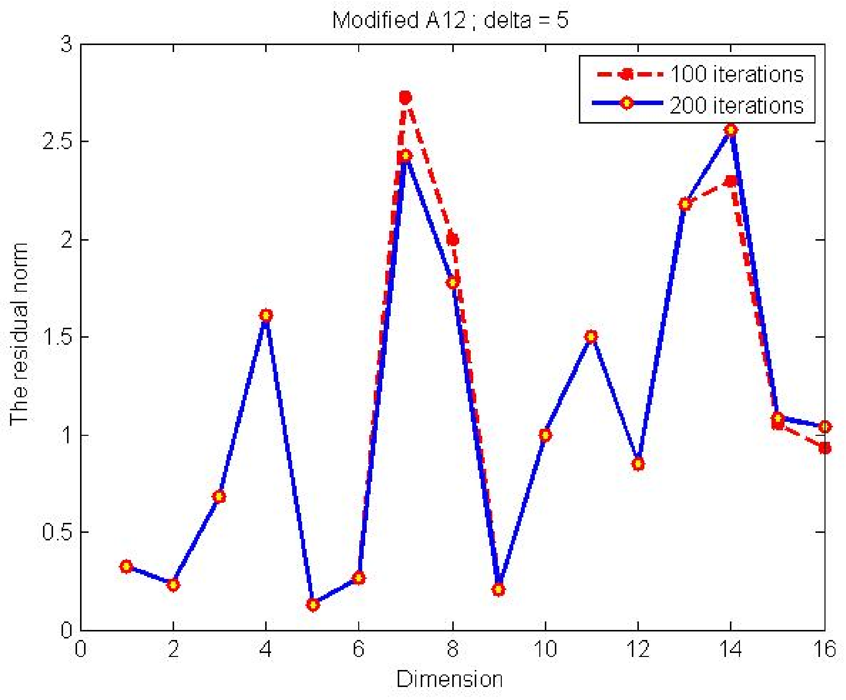


Figure B.16: *The behaviour of residual norms of the iterates generated by EIEM A12 for 100 and 200 iteration; the case of* $\delta = 5$

Appendix C

Algorithms

C.1 Lanczos-type Algorithms and Their Modifications

C.1.1 Orthodir

```
function [sol,last_norm] = algorithm_orthodir(A,b,x,y,num_it)

tol = 1e-13;
n = size(A);
x(:,1) = x;
y(:,1) = y;
r(:,1) = b - A*x(:,1);
z(:,1) = r(:,1);
res_norm(:,1) = norm(r(:,1));

d(:,1) = A*z(:,1);
A_{2} = -(y(:,1)'* r(:,1))/(y(:,1)'*d(:,1));
x(:,2) = x(:,1) - A_{2}*z(:,1);
r(:,2) = r(:,1)+ A_{2}*d(:,1);
%u(:,2) = x(:,2) - x(:,1);
res_norm(:,2) = norm(r(:,2));

At = A';
y(:,2) = At*y(:,1);
B_{2} = 0;
C_{2} = (-(y(:,2)'*d(:,1)))/(y(:,1)'*d(:,1));
z(:,2) = d(:,1) + C_{2}*z(:,1);
k = 2;
while (norm(r(:,k))>tol) && (k <= num_it)
 d(:,k) = A*z(:,k);
 A_{k+1} = (-(y(:,k)'* r(:,k)))/(y(:,k)'*d(:,k));
 x(:,k+1) = x(:,k) - A_{k+1}*z(:,k);
 r(:,k+1) = r(:,k) + A_{k+1}*d(:,k);
 y(:,k+1) = At*y(:,k);
 B_{k+1} = (-(y(:,k)'*d(:,k)))/(y(:,k)'*z(:,k-1));
 C_{k+1} = ((-B_{k+1}*(y(:,k)'* d(:,k-1)))- (y(:,k+1)'* d(:,k)))/(y(:,k)'*d(:,k));
 z(:,k+1) = d(:,k)+ B_{k+1}*z(:,k-1) + C_{k+1}*z(:,k);
    %u(:,k+1) = x(:,k+1) - x(:,k);
 res_norm(:,k) = norm(r(:,k));
 k = k+1;

end
disp('----------------------');
disp('Solution is ');
%x(:,end-1) = [];
sol = x(:,end);
disp('');
disp('----------------------');
disp('The norm of the residual is ');
last_norm = res_norm(:,end)
```

C.1.2 RLMedVal Algorithm

```
function [med_val,norm_med] = algorithm_orthodir_modify_1(A,b,x,y,num_it)

tol = 1e-13;
n = size(A);

x(:,1) = x;
y(:,1) = y;
r(:,1) = b - A*x(:,1);
z(:,1) = r(:,1);
res_norm(:,1) = norm(r(:,1));
med(:,1) = 0;
%rd(:,1) = 0;
d(:,1) = A*z(:,1);
A_{2} = -(y(:,1)'* r(:,1))/(y(:,1)'*d(:,1));
x(:,2) = x(:,1) - A_{2}*z(:,1);
r(:,2) = r(:,1)+ A_{2}*d(:,1);
res_norm(:,2) = norm(r(:,2));
At = A';
y(:,2) = At*y(:,1);
B_{2} = 0;
C_{2} = (-(y(:,2)'*d(:,1)))/(y(:,1)'*d(:,1));
z(:,2) = d(:,1) + C_{2}*z(:,1);
k = 2;
while (norm(r(:,k))>tol) && (k <= num_it)
 d(:,k) = A*z(:,k);
 A_{k+1} = (-(y(:,k)'* r(:,k)))/(y(:,k)'*d(:,k));
 x(:,k+1) = x(:,k) - A_{k+1}*z(:,k);
 r(:,k+1) = r(:,k) + A_{k+1}*d(:,k);
 y(:,k+1) = At*y(:,k);
 B_{k+1} = (-(y(:,k)'*d(:,k)))/(y(:,k)'*z(:,k-1));
 C_{k+1} =  ((-B_{k+1}*(y(:,k)'* d(:,k-1)))- (y(:,k+1)'* d(:,k)))/(y(:,k)'*d(:,k));
 z(:,k+1) = d(:,k)+ B_{k+1}*z(:,k-1) + C_{k+1}*z(:,k);
```

```
    res_norm(:,k) = norm(r(:,k));
    k = k+1;

end
disp('----------------------');
disp('Solution is ');
data_sol = x(:,1:k);
for i = 1:n
  med(:,i) = median(data_sol(i,1:k));
  %mn(:,i) = mean(data_sol(i,1:k));
end
med_val = med(:,1:n)';
%mn_val = mn(:,1:n)';
res_med = b - A*med_val;
norm_med = norm(res_med)
%res_mn = b - A*mn_val;
%norm_mn = norm(res_mn)
```

C.1.3 RLMinRes Algorithm

```
function [sol_mn,norm_mn] = algorithm_orthodir_modify_2(A,b,x,y,num_it)

tol = 1e-13;

x(:,1) = x;
y(:,1) = y;
r(:,1) = b - A*x(:,1);
z(:,1) = r(:,1);
res_norm(:,1) = norm(r(:,1));
d(:,1) = A*z(:,1);
A_{2} = -(y(:,1)'* r(:,1))/(y(:,1)'*d(:,1));
x(:,2) = x(:,1) - A_{2}*z(:,1);
r(:,2) = r(:,1)+ A_{2}*d(:,1);
res_norm(:,2) = norm(r(:,2));
At = A';
y(:,2) = At*y(:,1);
B_{2} = 0;
C_{2} = (-(y(:,2)'*d(:,1)))/(y(:,1)'*d(:,1));
z(:,2) = d(:,1) + C_{2}*z(:,1);
k = 2;
while (norm(r(:,k))>tol) && (k <= num_it)
 d(:,k) = A*z(:,k);
 A_{k+1} = (-(y(:,k)'* r(:,k)))/(y(:,k)'*d(:,k));
 x(:,k+1) = x(:,k) - A_{k+1}*z(:,k);
 r(:,k+1) = r(:,k) + A_{k+1}*d(:,k);
 y(:,k+1) = At*y(:,k);
 B_{k+1} = (-(y(:,k)'*d(:,k)))/(y(:,k)'*z(:,k-1));
 C_{k+1} =  ((-B_{k+1}*(y(:,k)'* d(:,k-1)))- (y(:,k+1)'* d(:,k)))/(y(:,k)'*d(:,k));
 z(:,k+1) = d(:,k)+ B_{k+1}*z(:,k-1) + C_{k+1}*z(:,k);
 res_norm(:,k) = norm(r(:,k));
 k = k+1;

end
data_sol = x(:,1:k);
data_norm = res_norm(:);
[norm_mn,index] = min(data_norm)
disp('----------------------');
disp('The minimum residual norm is ');
norm_mn
disp('The Min Solution is ');
sol_mn = data_sol(:,index);
```