COMPUTATION AND ANALYSIS OF SPECTRA OF LARGE UNDIRECTED NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGE ERDEM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
SCIENTIFIC COMPUTING

JUNE 2010

Approval of the thesis:

COMPUTATION AND ANALYSIS OF SPECTRA OF LARGE UNDIRECTED NETWORKS

submitted by **ÖZGE ERDEM** in partial fulfillment of the requirements for the degree of **Master of Science in Department of Scientific Computing, Middle East Technical University** by,

Prof. Dr. Ersan Akyıldız _____
Dean, Graduate School of **Applied Mathematics**

Prof. Dr. Bülent Karasözen _____
Head of Department, **Scientific Computing**

Prof. Dr. Bülent Karasözen _____
Supervisor, **Department of Mathematics, METU**

Prof. Dr. Jürgen Jost _____
Co-supervisor, **Max-Planck Institute for Mathematics in Sciences, Leipzig, Germany**

Examining Committee Members:

Prof. Dr. Gerhard Wilhelm Weber _____
Institute of Applied Mathematics, METU

Prof. Dr. Bülent Karaözen _____
Department of Mathematics & Institute of Applied Mathematics, METU

Assoc. Prof. Dr. Ömür Uğur _____
Institute of Applied Mathematics, METU

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ÖZGE ERDEM

Signature :

ABSTRACT

COMPUTATION AND ANALYSIS OF SPECTRA OF LARGE UNDIRECTED NETWORKS

Erdem, Özge

M.S., Department of Scientific Computing

Supervisor : Prof. Dr. Bülent Karasözen

Co-Supervisor : Prof. Dr. Jürgen Jost

June 2010, 91 pages

Many interacting complex systems in biology, in physics, in technology and social systems, can be represented in a form of large networks. These large networks are mathematically represented by graphs. A graph is represented usually by the adjacency or the Laplacian matrix. Important features of the underlying structure and dynamics of them can be extracted from the analysis of the spectrum of the graphs. Spectral analysis of the so called normalized Laplacian of large networks became popular in the recent years. The Laplacian matrices of the empirical networks are in form of unstructured large sparse matrices. The aim of this thesis is the comparison of different eigenvalue solvers for large sparse symmetric matrices which arise from the graph theoretical representation of undirected networks. The spectrum of the normalized Laplacian is in the interval $[0, 2]$ and the multiplicity of the eigenvalue 1 plays a particularly important role for the network analysis. Moreover, the spectral analysis of protein-protein interaction networks has revealed that these networks have a different distribution type than other model networks such as scale free networks. In this respect, the eigenvalue solvers implementing the well-known implicitly restarted Arnoldi method, Lanczos method, Krylov–Schur and Jacobi Davidson methods are investigated. They exist as MATLAB routines and

are included in some freely available packages. The performances of different eigenvalue solvers SPEIG, AHBEIGS, IRBLEIGS, EIGIFP, LANEIG, JDQR, JDCG in MATLAB and the library SLEPc in C++ were tested for matrices of size between 100-13000 and are compared in terms of accuracy and computing time. The accuracy of the eigenvalue solvers are validated for the Paley graphs with known eigenvalues and are compared for large empirical networks using the residual plots and spectral density plots are computed.

Keywords: Empirical networks, Undirected graphs, Spectral Graph Theory, Eigenvalue Solvers

ÖZ

YÖNSÜZ GENİŞ AĞLARIN SPEKTRUM HESAPLAMALARI VE ANALİZLERİ

Erdem, Özge
Yüksek Lisans, Bilimsel Hesaplama Bölümü
Tez Yöneticisi : Prof. Dr. Bülent Karasözen
Ortak Tez Yöneticisi : Prof. Dr. Jürgen Jost

Haziran 2010, 91 sayfa

Biyoloji, fizik, teknoloji ve sosyal sistemler gibi alanlarda yer alan kompleks sistemler büyük ağlar şeklinde gösterilebilinirler. Bu ağlar, daha sonra matematiksel olarak çizgeler şeklinde ifade edilirler. Çizgeler, Adjacency ve Laplace matrisleriyle gösterilebilinecekleri bulunmuştur. Böylece sistemlerin yapılarının ve dinamiklerinin önemli özellikleri matrislerin spektrum analizlerinden çıkarılır. Büyük ağların birimleştirilmiş Laplace matrislerinin analizi son yıllarda popüler hale gelmiştir. Bu matrisler genellikle 0 ların çoğunlukta olduğu ve düzensiz bir yapıya sahiptirler. Bu tezin amacı farklı özdeğer çözücülerin yönsüz ağların simetrik matrisler şeklinde gösterimlerinden doğan büyük ve sparse yapıdaki matrisler üzerindeki performanslarını karşılaştırmaktır. Birimleştirilmiş Laplace matrisinin özdeğerleri [0,2] aralığındadır ve 1 özdeğerlerinin çokluğu (multiplicity) ağların analizinde önemli rol oynamaktadır. Ayrıca protein etkileşim ağları üzerinde yapılan çalışmalar sonucunda bu ağların mevcut olarak bilinen model ağlara göre daha farklı bir spektral dağılıma sahip olduğu bulunmuştur. Bu çerçevede, dolaylı olarak yeniden başlayan Arnoldi(IRA), blok Lanczos, Krylov–Schur ve Jacobi–Davidson metotlarına dayanan özdeğer çözücüler araştırılmıştır. Bu çözücüler MATLAB rutini olarak ve ücretsiz yazılımlar halinde bulunmaktadır. MATLAB da yazılmış özdeğer

çözücülerinden, SPEIG, AHBEIGS, IRBLEIGS, EIGIFP, LANEIG, JDQR, JDCG, ve C++ da yazılmış olan SLEPc paketinin performansları büyüklükleri 100 ile 13000 arasında değişen matrisler üzerinde kesinlik ve süreleri açısından karşılaştırıldı. Bu bazda özdeğerleri bilinen Paley çizgeleri ile ampirik ağlardan doğan çizgelerin birimleştirilmiş Laplace matrisleri kullanıldı. Paley çizgelerinin özdeğerleri için hata figürleri, ampirik ağların özdeğerleri için ise residual figürleri ile spektral dağılımlarının gösteren figürler oluşturuldu.

Anahtar Kelimeler: Ampirik ağlar, Yönsüz Çizgeler, Çizge Teorisi, Özdeğer Çözücüler

ACKNOWLEDGMENTS

I would like to show my gratitude to all people who supported me during the completion of this thesis. First of all, I am heartily grateful to my supervisor Professor Dr. Bülent Karasözen for introducing me this subject and his guidance through this study from initial to final level. I would like to thank him for his valuable suggestions and careful examination of this thesis.

I also owe my deepest gratitude to Professor Dr. Jürgen Jost. He has made available his support in a number of ways starting from hosting me at Max Planck Institute and sharing his valuable ideas and advices on the research results. It was a great pleasure to have the opportunity for being a member of such an important scientific society.

I also would like to thank to Assist. Professor Dr. Ömür Ugur for his helps and orientations for implementations in the operating system Linux. Warmest thanks to Mario Thüne for providing the necessary data sets and valuable discussions in MPI.

Finally, special thanks to my family for their support and love. I really appreciate them for their patience and kindness to me.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

The power grid system of a country, the supply chain bringing products from world to a country, air traffic system, WWW system, biological networks like protein-protein interaction networks are examples of complex systems from different application areas. Despite the differences in the origins of these problems, they all have an important common feature: their elements (components) interact with each other and compose a hierarchy of subsystems. This hierarchy is determined through following certain rules or dynamics defined by external and internal influences. As a result of this behavior, the interactions inherit characteristic properties such as adaptation, evolution and uncertainty. In addition, the hierarchy of components constitutes the internal structure of systems. Therefore in order to understand the interactions, it is important to analyze the dynamics of structural properties of complex systems in networked form. The study of large networks became very popular in various disciplines, like mathematics, physics, biology and sociology. Many methodologies, theories, schemes are introduced for modeling, manipulating, understanding and analyzing the networks. The analysis of networks, based on graph theory, is an emerging and promising branch of science that captures important characteristics of complex systems. Networks are analyzed using various graph theoretical tools connected with algebra, topology, dynamical systems and matrix theory. In this respect, sample models are constructed in order to mimic or reconstruct the behavior of real world models; structural relations between parameters are found; relations between spectrum of the matrices describing the graph structures are exploited. All these methodologies aim to capture the qualitative information about the structure of networks. They somehow achieve this but in a restricted sense: only some certain qualitative properties can be revealed. The spectrum of the normalized Laplacian of a graph plays an important role in this respect. The distribution of eigenvalues reflects important qualitative features of

graphs. It was found that some certain evolutionary processes leave characteristic traces in the spectrum. Then this result is used to reconstruct model networks aiming to mimic the behavior of real networks.

Recent results about spectral properties of normalized Laplacian matrix have given rise to the practical problem of determining the spectrum of a matrix. Although, in theory there is a simple expression for calculating eigenvalues and eigenvectors of matrices, in practice this is a very difficult task to accomplish especially for large sparse matrices. Direct solvers which attempt to solve the problem in finite steps are not applicable in case of large, sparse matrices. Therefore iterative eigenvalue solvers are usually used. The most widely used iterative eigenvalue solvers are appropriate for eigenvalue problems with dense, small and structured matrices. On the other hand, matrices arising from network applications are large, sparse (mostly consist of zeros) and do not have a regular structure. Therefore, usual algorithms are not efficient for matrices arising from networks. Methods taking the advantages of sparsity structure should be used instead of them. Krylov subspace algorithms and Jacobi Davidson type methods are two important classes of these algorithms. They exploit the sparsity of matrices only requiring matrix vector multiplications.

Several eigenvalue packages (will be referred as eigensolver or eigenvalue solver) depending on Krylov and Jacobi–Davidson methods are developed and most of them are freely available on Internet. They differ in various aspects such as, the programming language (C++, Fortan or MATLAB are the most widely used), the versions of the methods or simply the design of algorithms. For example, the most popular and widely used sparse matrix eigensolver is ARPACK. It is written in FORTRAN (there is also a C++ interface) and implements a famous Krylov subspace method: Implicitly Restarted Arnoldi algorithm.

This thesis focuses on computing and analyzing the spectrum of normalized Laplacian matrices of networks. The eigenvalue distribution of this matrix reveals important information about the structure and dynamics of the network. Evolutionary processes for the network leaves certain traces in the spectrum. For these cases, computing the whole spectrum is an objective and efficient eigenvalue solvers are required to achieve this. The goal of this work is to provide a comparison of the already existing sparse eigenvalue solvers. They are based on Krylov subspace algorithms and Jacobi Davidson methods and written in MATLAB and $C++$.

The outline of the thesis is as follows:

In Chapter 2, basic definitions and elementary notions of graph theory will be given. In Section 2.2, first necessary tools for spectral analysis of a graph is introduced then general and recent results about relations between spectrum of normalized Laplacian matrix and the structure of networks is given.

In Chapter 3, iterative eigenvalue solvers based on Krylov subspace and Jacobi–Davidson methods are introduced with their numerical properties. In addition, different variants of the methods (implicitly restarted Arnoldi (IRA), Lanczos methods and their block variants, Krylov–Schur method, inverse free preconditioned Lanczos algorithm and Jacobi–Davidson algorithm with conjugate gradient method) are included.

Software packages in MATLAB like SPEIG, AHBEIGS, IRBLEIGS, EIGIFP, LANEIG, JDQR, JDCG and the package SLEPc in C++ are described in Chapter 4.

In Chapter 5, the spectrum of the normalized Laplacian matrices of Paley graphs with known eigenvalues and some empirical networks, like protein - protein interaction networks, are computed and the performance of the eigenvalue solvers are compared with respect to the accuracy using the residual plots and to computing times and spectral density plots with the Gaussian and Lorenz kernels are computed. The thesis ends with some conclusions.

CHAPTER 2

LARGE NETWORKS AND SPECTRAL GRAPH THEORY

Networks are useful tools for analyzing complex systems. A complex system has components interacting with each other. These interactions generally follow some rules and dynamics that are changing over time. The structure of the systems are shaped by adapting the relations between changes and interactions of components. On the other hand, structure also effects the interactions. In brief, both the dynamics and the structure of networks are affected from each other. Therefore in order to understand the internal dynamics of the systems, it is necessary to investigate the structural properties of the network [14]. In this respect, network theory consists of methods and tools for doing such analyzes.

Today, many real systems from different applications such as in sociology, in biology, in physics, in chemistry and in information technologies can be represented by networks. Networks represented by movie actors, company directors, scientific co-authorship, telephone calls, emails, friendships and sexual contacts are considered as social networks; citation, word co-occurrence and world wide web are information networks; internet, power grids, train routes, electronic circuits and software packages can be considered as technological networks; metabolic reactions, protein-protein interaction (PPI), gene regulation and food web are classified as biological networks. These networks are analyzed in order to understand the qualitative features of the data that they are describing. For example, sexual contacts networks are analyzed to understand the spread of infection of diseases. Email networks are used in order to understand how viruses are spread over. (For various examples and extended research, see [9, 10, 14, 51].)

The study of complex systems in the form of networks depends on graph theory (see for example [19]). The components of the complex systems can be considered as vertices and

relations or interactions between these components can be assigned as edges of the graph. Therefore, the first issue in analyzing systems with graph theoretical tools is to decide what would be the vertices and what would be the edges (the construction scheme). The initial proof about this subject can be traced back to 1735, when Leonard Euler declared the solution of Könisberg bridges problem through graphical representation. In this famous problem, the lands are assigned as vertices and bridges are assigned as edges. Although this problem is much smaller in size than today's problems, it is an important step in the history of graph theory.

The primary focus of analyzing networks has two aspects. The first aspect includes the introduction of qualitative and quantitative parameters. Many graph theoretical tools such as degree distribution, path length, diameter, clustering coefficient, centrality, betweennes, chromatic number are introduced in order to capture and reveal the structural properties of networks [18, 19, 33, 43, 51]. The question here is which parameters reveal global properties and which parameters are more graph invariant. For example, how difficult it is to break the graph into disjoint components [21] or how difficult it is to synchronize the coupled dynamics operating at individual vertices [44]. However, it should be noticed that these parameters can capture only some parts of structural properties of graphs. They do not give information about all the qualitative properties of networks. More information about graph invariants and parameters can be found in [18, 33, 51]. The second aspect about analyzing networks includes development of model networks such as random graphs [64], scale free networks [15] and small-world networks [76]. These models are constructed in order to help to understand the characteristic structures or behaviors of real networks. They are also used to understand the interaction of the parameters described in the first aspect. For example, in [27] random graph models are constructed inspiring from social networks. In this model, every pair of nodes have a probability p of being connected. Interesting properties of these graphs with different values of p are investigated in [28] and [29]. Moreover, the question that asks if the constructed model networks can capture the properties of real networks after fitting certain parameters is currently being investigated.

Among various graph analyzing tools spectral graph theory is important in the sense that it combines algebraic matrix theory and linear algebra with graph theory [21, 22, 23]. The main goal of spectral graph theory is to investigate structural properties of networks from the spectrum of the matrices describing the graph structure of the network. It tries to answer that

in what limits the spectrum informs us about the qualitative characteristics of networks or if it is possible to make a classification of the networks with respect to their spectrums. In other words, is it possible to trace back the structural properties of graphs from their spectrum?

In the early stages of spectral graph theory, adjacency matrices are used for analyzing networks. Relations between qualitative properties and eigenvalues of adjacency matrices are found [33, 51]. In recent years, a new approach from geometric perspective is realized. The analogy between spectral graph theory and Riemannian geometry has led this subject to enter a new era. The combinations of algebraic spectral methods and Riemannian geometry in graph theory introduced the Laplacian matrix [8, 21]. The new researches on Laplacian matrix of a graph has revealed that spectrum of Laplacian is a powerful tool in analyzing networks. It reflects the global properties of the graphs better than the adjacency matrices [8, 10, 11, 12].

In [77] the characteristic properties of the spectrum of random matrices are studied. A new law called Wigner's semi circle law was introduced (for details see [8] and [77]). Then the studies on Erdös and Rényi's random graphs showed that the distribution of eigenvalues of adjacency matrices of these graphs follow the semi-circle law [31], whereas spectral distributions of adjacency matrices of Barabási and Albert's scale-free graphs follow a power law distribution [34]. Later on, in [23] it is found that spectral distributions of different matrices of a graph may follow different rules. For example, the spectrum of adjacency matrix of a graph may obey a power law whereas the distribution of eigenvalues of Laplacian matrix may follow Wigner's semicircle law [21]. In this study, the primary focus will be on spectral distribution of Laplacian matrices of graphs.

In this Chapter recent studies about spectral graph theory are summarized by emphasizing especially spectral properties of the Laplacian matrix. The first Section includes basic definitions related with graph theory. These definitions will be used through the rest of the Chapter. Second section consists of two subsections. In the first part, the basic tools of spectral graph theory like the connectivity matrices are introduced. Some general and recent results about relations between parameters and spectrum of connectivity matrices are given in the second part. Examples of spectra of graphs are presented in the last Section.

2.1 Basics of Graph Theory

In the following Section, the definitions in [43], Chapter 1 and [8] will be made used of.

Definition 2.1 *A **graph** $G = (V, E)$ consists of two sets V and E such that the elements of V are joined by the elements of E. The elements of V are called vertices and the elements of E are called edges. Each edge has one or two vertices associated to it which are called endpoints. The **order** of a graph is the cardinality of its vertex set and the **size** of a graph is the cardinality of its edge set.*

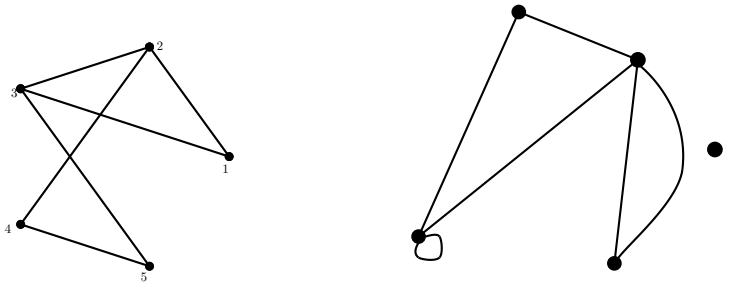
Definition 2.2 ***Adjacent edges** are two edges that have a common endpoint. An edge joining two distinct end points is called a **proper edge**. A **multi-edge** is a collection of two or more edges having same endpoints. A **loop** is an edge that joins a single vertex to itself.*

Definition 2.3 *A vertex v is an end point of an edge e, in other words v is said to be incident on e and e is said to be incident on v. A vertex u is adjacent to a vertex v if they are joined by an edge. Two adjacent vertices, i and j, are called **neighbors**, is denoted by $i \sim j$. The **edge multiplicity** is the number of edges between two vertices. The **degree** of a vertex v is the number of edges incident to v plus twice the number of self-loops. In some context, valence is also used instead of degree.*

For example in Figure 2.1 (a), the graph has vertex set $V = \{1, 2, 3, 4, 5\}$ with edges between them. The neighboring vertices are $1 \sim 2$, $1 \sim 3$, $3 \sim 2$, $4 \sim 2$, $3 \sim 5$, $4 \sim 5$ and the edge multiplicity is one for all vertices. The degrees with respect to vertex numbers are $2, 3, 3, 2, 2$.

Definition 2.4 *One vertex forms a trivial graph. A graph without loops and multi-edges is called **simple graph**. Other graphs in which multi-edges and loops are existed are called **general graphs**.*

In Figure 2.1, both graphs have 5 vertices and 6 edges. The first figure is an example of a connected simple graph whereas the second figure is an example of a general graph with two components. In the picture on the right, one vertex is connected to itself, it is a loop. Moreover, two vertices has multiple edges between them and one vertex is isolated.



(a) (b)

Figure 2.1: (a) A simple graph (b) A general graph

Simple graphs are easier to analyze than general graphs. Because of their simple structure, most of researches emphasize on simple graphs. In this study, simple graphs are dealt with rather than general graphs. From now on, unless the opposite is specified, the word graph will refer to a simple graph through this thesis.

Definition 2.5 *A **directed graph** (Digraph) is a graph each of whose edges have a direction. The direction to an edge is assigned by naming the endpoints of the edge as head and tail. Then the edge is said to be directed from its tail to its head. The indegree of a vertex v in a digraph is the number of edges directed to v and the outdegree of v is the number of edges directed from v. If there are no directions assigned to the edges, the graph is called **undirected graph**.*

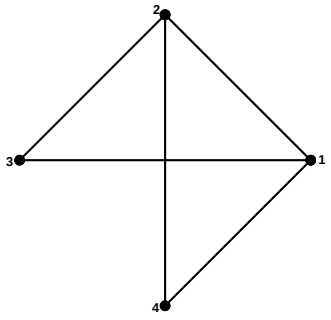


Figure 2.2: A directed graph with 4 vertices and 5 edges

The graph in Figure 2.2 is a directed graph with 4 vertices. The directions of vertices are:

$$1 \to 2, \quad 1 \to 4, \quad 3 \to 1, \quad 4 \to 2, \quad 3 \to 2$$

The out and indegrees of vertices of the directed graph are given below:

Vertex Number	Indegree	Outdegree
1	1	2
2	3	0
3	0	2
4	1	1

Definition 2.6 *A **walk**, W, in a graph G is an alternating sequence of vertices and edges*

$$W = v_0, e_1, v_1, e_1, \ldots, e_n, v_n$$

such that for $j = 1, \ldots, n$ *the vertices* v_{j-1} *and* v_j *are the endpoints of the edge* e_j*. The **length** of a walk is the number of vertices within it. A walk is said to be called **trial** if all edges are distinct. An Eulerian trail in a graph G is a walk that contains each edge of G. When no internal vertex is repeated in a trial it is called a **path**. A closed path is a **cycle**. It is of length at least* 1.

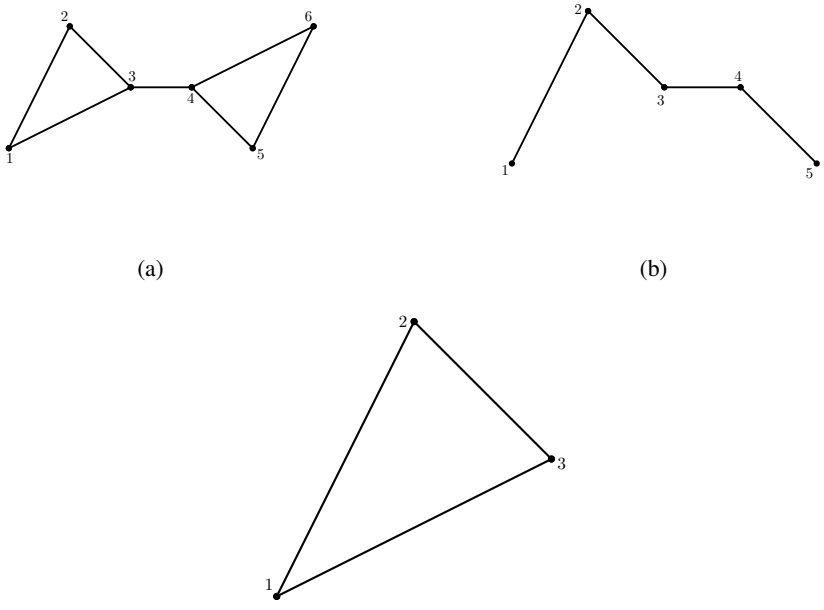


Figure 2.3: (a) A simple graph with 6 vertices with an example of a walk (b) and a cycle (c)

In a simple graph a walk can be represented by listing vertices:

$$W = v_0, v_1, \ldots, v_n$$

such that for $j = 1, 2, \ldots, n$ the vertices v_j and v_{j-1} are adjacent. Here v_0 is called the initial vertex, v_n is the final vertex and all other vertices are called internal vertices.

For example, the walk given in Figure 2.3 can be represented as:

$$W = 1, 2, 3, 4, 5$$

where v_i is represented by numbers i here. 1 is the initial vertex and 5 is the final vertex. Here, W is also a trial and a path. Another example of a walk from the same figure would be:

$$W_a = 2, 3, 4, 6$$

Definition 2.7 *A graph is **connected** if there is walk for every pair of vertices. A digraph is weakly connected if its underlying graph is connected. Strongly connected for a directed graph means there is a directed walk from each vertex to other vertices.*

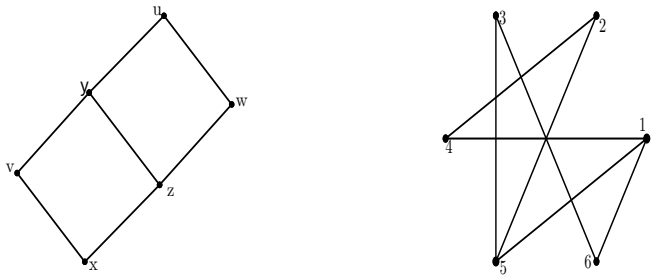
Definition 2.8 *The **distance** between two vertices u and v in a graph is the length of the shortest walk between them. It is denoted by $d(u, v)$. The **eccentricity** of a vertex in a connected graph is the distance to a vertex farthest from v. Maximum eccentricity in a connected graph is the **diameter** of the graph.*

For example the eccentricity of vertex 3 in Figure 2.4(b) is three and the eccentricity of vertex 5 is 2. The diameter of the graph is three.

Definition 2.9 *An **isomorphism** between two graph G and H is a pair of bijections $\phi_V : V_G \to V_H$ and $\phi_E : E_G \to E_H$ such that for every pair of vertices $u, v \in V_G$, the set of edges in E_G joining u and v is mapped bijectively to the set of edges in E_H joining the vertices $\phi(u)$ and $\phi(v)$. Two graphs are **isomorphic** if there is bijection between them.*

The graphs in Figure 2.4 are isomorphic and the relation between vertices are

$$1 \leftrightarrow z \quad 2 \leftrightarrow v$$
$$3 \leftrightarrow u \quad 4 \leftrightarrow x$$
$$5 \leftrightarrow y \quad 6 \leftrightarrow w$$



(a) (b)

Figure 2.4: Isomorphic two graphs

Definition 2.10 *A **subgraph** of a graph G is a graph H such that $V_H \subseteq V_G$ and $E_G \subseteq E_H$. The induced set on vertices $W = \{w_1, \ldots, w_k\}$ has W as its vertex set and contains every edge of G whose end points are in W. It is denoted by G(W). A subgraph H of G is spanning subgraph if the vertex set of H is equal to vertex set of G. The maximal connected subgraph of a graph is called **component**.*

Definition 2.11 *A connected graph without cycles is called a **tree**. A spanning tree of a graph is a spanning subgraph that is a tree.*

Up to here, basic definitions about graph theory are introduced. Some important types of graphs will be announced in the rest of this Section.

Definition 2.12 *A **complete graph** is a simple graph such that every pair of vertices are connected with an edge. It is denoted by K_n where n is the number of vertices.*

Definition 2.13 *A simple or multi–graph is **bipartite** if the vertices can be partitioned into two sets such that no edge joins two vertices in the same set. If the vertex set of the graph can be divided into k sets such that edge joins two vertices in the same set, the graph is called **k-bipartite**.*

Trees are examples of bipartite graphs. Moreover every cycle with even number of vertices are bipartite whereas cycles with odd number of vertices are not. Bipartite graphs are important for graph coloring problems. A graph is k-colorable if and only if it is k-partite.

Definition 2.14 *A **complete bipartite** graph is a simple bipartite graph which each vertex in one set is connected by edges to all vertices in the other set. It is denoted by $K_{m,n}$, where m and n represents the number of vertices in each set and m + n is the total number of vertices in the graph.*

(a) (b)

Figure 2.5: (a) Complete graph of 5 vertices and (b) complete bipartite graph ($K_{5,3}$) of 8 vertices

Another interesting class of graphs is regular graphs in which every vertex has the same degree.

Definition 2.15 *A graph is **regular** if every vertex has the same degree. If the degree is k, it is generally said k-regular. A regular graph $G = (V, E)$ with degree k is strongly regular if every adjacent vertex has the same number of common neighbors a and every nonadjacent vertex has the same number of common neighbors d. A strongly regular graph is generally represented by these numbers:* (n, k, a, d)

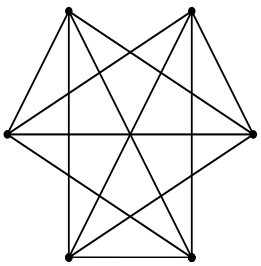


Figure 2.6: Regular graph with 6 vertices and degree 4

2.2 Spectral Graph Theory

Spectral graph theory is an important tool for analyzing the structure of graphs. The analyzes are done through the matrix representations of graphs. In the early days of the spectral graph theory, combination of linear algebra, matrix and graph theory are used to analyze the adjacency matrices. The relations between quantitative parameters and eigenvalues of adjacency matrix were investigated in [33, 51]. In the recent years, a new approach has started to gain popularity. The analogy between Riemannian geometry and spectral graph theory is noticed and this concept brought many new insights and useful tools [21, 22]. Spectrum of normalized Laplacian matrix is one of these useful tools. It is found that the eigenvalues of normalized Laplacian matrix relate well to some graph invariants. Moreover the spectrum reveals important features of the structural inheritance [8, 10, 11].

2.2.1 Connectivity Matrices

A graph $G = (V, E)$ can be represented by various kinds of matrices. Eigenvalues of some of these matrices play an important role in the analysis of graphs. Before going in a detailed analysis of spectrums, the four important matrices will be introduced in this section: Adjacency, Laplacian and two normalized Laplacian matrices.

- **Adjacency Matrix:** The matrix A=$[a_{ij}]$ such that

$$a_{ij} = \begin{cases} 1, & \text{if } ij \text{ is an edge} \\ 0, & \text{otherwise} \end{cases}$$

  is the adjacency matrix of the graph.

- **Laplacian Matrix:** The matrix L=$[l_{ij}]$ such that

$$l_{ij} = \begin{cases} n_i, & \text{if } i = j \\ -1, & \text{if j is an edge} \\ 0, & \text{otherwise} \end{cases}$$

  is the Laplacian matrix of the graph.

- **Normalized Laplacian Matrix:** There are different kinds of normalized Laplacian matrices with respect to normalization factors. Here, two of them will be presented. The first one is in accordance with the matrix introduced by Chung [21].

The matrix $\mathcal{L} = [l_{ij}]$ such that

$$l_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } n_i \neq 0 \\ -\frac{1}{\sqrt{n_i n_j}}, & \text{if } ij \text{ is an edge} \\ 0, & \text{otherwise} \end{cases}$$

The matrix $\Delta = [l_{ij}]$ such that

$$l_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } n_i \neq 0 \\ -\frac{1}{n_j}, & \text{if } ij \text{ is an edge} \\ 0, & \text{otherwise} \end{cases}$$

are normalized Laplacian matrix of the graphs.

Unfortunately, for many graphs, there is no clear relationship between the connectivity matrices. However, the following relations would sometimes be useful especially in designing algorithms:

$$L = D - A,$$

where D is a diagonal matrix with degree on diagonal entries

$$\mathcal{L} = D^{\frac{-1}{2}} L D^{\frac{-1}{2}} = I - D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$$

and there is similarity between two kinds of normalized Laplacian matrices:

$$\Delta = D^{\frac{1}{2}} \mathcal{L} D^{\frac{-1}{2}}$$

Example: The connectivity matrices of the graph in Figure 2.7 are as follows:

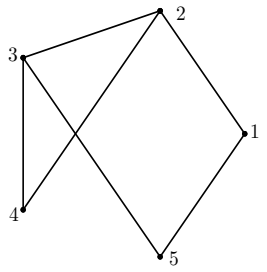


Figure 2.7: A simple graph with 5 vertices

The adjacency matrix

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

The Laplacian matrix is

$$L = \begin{pmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 3 & -1 & -1 \\ 0 & -1 & -1 & 2 & 0 \\ -1 & 0 & -1 & 0 & 2 \end{pmatrix}.$$

The normalized matrices are given by

$$\mathbf{L} = \begin{pmatrix} 2 & -\frac{1}{\sqrt{6}} & 0 & 0 & -\frac{1}{2} \\ -\frac{1}{\sqrt{6}} & 3 & -\frac{1}{3} & -\frac{1}{\sqrt{6}} & 0 \\ 0 & -\frac{1}{3} & 3 & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} \\ 0 & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 2 & 0 \\ -\frac{1}{2} & 0 & -\frac{1}{\sqrt{6}} & 0 & 2 \end{pmatrix}, \ \Delta = \begin{pmatrix} 1 & -\frac{1}{3} & 0 & 0 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{3} & -\frac{1}{3} & 0 \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{1}{3} & -\frac{1}{3} & 1 & 0 \\ -\frac{1}{2} & 0 & -\frac{1}{3} & 0 & 1 \end{pmatrix}.$$

2.2.2 Eigenvalues of Graphs

In graph theory, eigenvalues of graphs refers to the eigenvalues of one of the connectivity matrices that are found from the usual eigenvalue problem:

$$Ax = \lambda x,$$

where A is a suitable connectivity matrix of the graph.

The spectrum of an adjacency matrix carries details about local structural properties of graphs such as the number of edges, triangles, loops or bipartiteness. Moreover, the extremal eigenvalues also provide bounds for some parameters such as chromatic number. The chromatic number of a graph G, $\chi(G)$, is the minimum number of colors required for coloring it. Computing this number is NP-hard problem but a bound is obtained by extremum eigenvalues of adjacency matrix:

$$1 - \frac{\lambda_n}{\lambda_1} \leq \chi(G) \leq 1 + \lambda_n,$$

where λ_1 is the smallest and λ_n is the largest eigenvalue.

The average degree, $\overline{d}$ is the ratio between the sum of the degrees of vertices and the size of the graph. $\overline{d} = \frac{1}{n}\sum_{i\in V(G)} d(i)$. It can not be larger than the largest eigenvalue of the adjacency matrix:

$$\overline{d} \leq \lambda_n.$$

The connected components of the graph can be found by smallest eigenvalues of the Laplacian matrix. In addition, Laplacian matrix can be used to determine the spanning trees of the graphs. In the early days of spectral graph theory, most of the researches have focused on the analysis of adjacency matrix and Laplacian matrix. Detailed analysis of these can be found in [22, 33]. The studies on spectrum of normalized Laplacian are more recent. It is found that the eigenvalues of this matrix carry information about the graph that the other connectivity matrices fail to determine. The evolutionary structure of the network can be traced back from spectral distribution of this matrix and multiplicities of some eigenvalues [8].

In the rest of this section we summarize the recent results about the spectral properties of normalized Laplacian matrix. These results are can be found in [8, 9, 10, 11, 13, 14, 21].

The normalized Laplacian matrix of the graph defined in the previous section can also be interpreted as an operator on the set of vertices. Let $u : V \to \mathbb{R}$ be a real valued function on the vertex set and the inner product be defined as

$$(u, v) = \sum_i n_i u(i) v(i).$$

Then the effect of normalized Laplacian on a function can be reformulated:

$$\Delta u(i) := u(i) - \frac{1}{n_i} \sum_{j, j\sim i} u(j). \tag{2.1}$$

This action of Laplacian on functions reveals three important properties of the operator:

1. Δ is a self adjoint operator with respect to the inner product defined above.
   This property implies that the operator is symmetric. Therefore, the eigenvalues of normalized Laplacian matrix are real.

2. Δ is nonnegative. This implies that all eigenvalues are nonnegative.

3. $\Delta u = 0$ when u is constant. This property implies that the smallest eigenvalue is 0.

The effect of Laplacian as an operator on the functions defined on vertices leads to apply the famous Courant-Fischer theorem to determine the eigenvalues [21]. The theorem is as follows:

Theorem 2.16 *(Courant-Fischer Theorem) Let M be a real symmetric matrix with eigenvalues* $\lambda_0 \leq \lambda_0 - 1 \leq \ldots \leq \lambda_{n-1}$. *Let* $\mathfrak{X}^k$ *denote any k dimensional subspace of* $\mathbb{R}^n$ *and* $x \perp \mathfrak{X}^k$ *signifies that* $x \perp y$ *for all* $y \in \mathfrak{X}^k$. *Then*

$$\lambda_i = \min_{\mathfrak{X}^{n-i-1}} \left(\max_{x \perp \mathfrak{X}^{n-i-1}, x \neq 0} R(X) \right) = \max_{\mathfrak{X}^i} \left(\min_{x \perp \mathfrak{X}^i, x \neq 0} R(x) \right), \tag{2.2}$$

where $R(x) = \frac{x^\top M x}{x^\top x}$ *is the Rayleigh quotient.*

Let the eigenvalues be ordered as

$$\lambda_0 = 0 \leq \lambda \leq \ldots \leq \lambda_{N-1}.$$

If this theorem is adapted to Laplacian matrix, the following expressions would be obtained:

$$\lambda_i = \min_{Y^{n-i-1}} \left(\max_{y \perp Y^{n-i-1}, y \neq 0} \frac{\sum_{i \sim j} (y_i - y_j)^2}{\sum_i y_i^2 d_i} \right). \tag{2.3}$$

From Equation (2.3), the following expressions for λ_1 and λ_{n-1} can be deduced:

$$\lambda_{n-1} = \max_{y \neq 0} \frac{\sum_{i \sim j} (y_i - y_j)^2}{\sum_i y_i^2 d_i}, \tag{2.4}$$

$$\lambda_1 = \min_{y \perp D1, y \neq 0} \frac{\sum_{i \sim j} (y_i - y_j)^2}{\sum_i y_i^2 d_i}. \tag{2.5}$$

These results are consequences of application of Courant-Fischer Theorem to the Laplacian matrix. The eigenvalues are bounded above by 2. The last eigenvalue, λ_{N-1}, equals to 2 if and only if the graph is bipartite. The difference between λ_{N-1} and 2 gives an idea about how far the graph is from being bipartite.

If N is the number of vertices in the graph $G = (V, E)$, then

$$\sum_i \lambda_i \leq N$$

and equality holds if and only if the graph is connected. For connected graphs the equivalence comes from an algebraic fact: the sum of eigenvalues of a matrix equals to trace of the matrix.

Important properties of eigenvalues of normalized Laplacian matrix will be summarized from now on. The multiplicity of the smallest eigenvalue ($\lambda_0 = 0$) gives the number of connected

components of the graph, i.e. if

$$\lambda_0 = \lambda_1 = \ldots = \lambda_{k-1} = 0 \text{ and } \lambda_k > 0$$

then the graph has k-connected components. For connected graphs, λ_1 is an important eigenvalue. The difference between λ_0 and λ_1 carries the information about how difficult it is to separate the graph into its disjoint components. In this respect, Cheeger constant (denoted by h_G) gives a bound for λ_1. Cheeger constant is parameter adapted from Riemannian geometry and defined by:

$$h_G = \inf \left\{ \frac{|E_0|}{\min\{\sum_{i\in\Gamma_1} n_i, \sum_{j\in\Gamma_2} n_j\}} \right\}.$$

The infimum in the definition is taken over E_0 which is a subset whose removal breaks the graph into two disjoint graphs Γ_1 and Γ_2. $|E_0|$ represents the cardinality of the set E_0. Cheeger constant for a graph is important in the sense that it helps to answer the question: for any subset of vertices of a specified volume, how many edges can be guaranteed to connect the subset to the rest of the graph. The definition implies that for any small subset E_0, there are at least $h_G\, vol(E_0)$ edges leaving E_0. The bound on λ_1 is

$$2h_G \geq \lambda_1 \geq \frac{h_G^2}{2}.$$

For a connected graph, the diameter and volume provides a bound for λ_1:

$$\lambda_1 \geq \frac{1}{D\,\mathrm{vol}G},$$

where D represents the diameter and $\mathrm{vol}G = \sum_i n_i$ Moreover, the following expression shows that the diameter can also be bounded by volume and λ_1 by a different expression:

$$D(G) \leq \frac{\log \frac{\mathrm{vol}(G)}{\min d_x}}{\log \frac{1}{1-\lambda_1}}.$$

This inequality can be generalized to find the distance of subgraphs of the graph G. If X and Y are two different subgraphs with a distance of at least 2, the distance between them is defied by

$$d(X, Y) = \min\{d(x, y) : x \in X \text{ and } y \in Y\}.$$

Extremum eigenvalues provide a bound for this value:

$$d(X, Y) \leq \frac{\log \sqrt{\frac{vol(\widehat{X})vol(\widehat{Y})}{vol(X)vol(Y)}}}{\log \frac{\lambda_{n-1}+\lambda_1}{\lambda_{n-1}-\lambda_1}}.$$

Up to now, the general properties of spectrum of normalized Laplacian are presented. Recent studies in [8, 10, 9] and [13] showed that some graphs has peaks at the eigenvalue 1. They investigated the graph evolutionary process related with these peaks. The rest of this section summarizes these results. The proofs of the theorems will not be included, but they can be found in [8].

Combining the Equation (2.1) with the usual eigenvalue Equation, $\Delta u - \lambda u = 0$, results in

$$\frac{1}{n_i} \sum_{j \sim i} u(j) = (1 - \lambda) u(i), \quad \forall\, i = 1, \ldots, n$$

So, if the eigenfunction vanishes at vertices i, the sum of the values of the function on neighbors of i would equal to 0, $\sum_{j \sim i} u(j) = 0$. On the other hand, if $\lambda = 1$, then

$$\sum_{j \sim i} u(j) = 0. \tag{2.6}$$

Therefore, for the eigenvalue 1, there are functions whose sum of values on neighboring vertices is zero. A function u satisfying this property is called balanced solution. The multiplicity of eigenvalue 1 gives the dimension of the set of linearly independent balanced functions on the graph. Moreover, it can be deduced from Equation (2.6) that the multiplicity of eigenvalue 1 equals to the dimension of the kernel of the adjacency matrix of the graph.

Definition 2.17 *A* motif Σ *is a connected small subgraph of the graph* Γ *containing all edges of* Γ *between vertices of* Σ.

In this context, Γ is supposed to be very large when compared with the motif Σ. Now, some theorems about graph operations and their effect on spectrum will be presented.

Theorem 2.18 *Let* Γ^{Σ} *be obtained from* Γ *by adding a copy of the* motif Σ *consisting of vertices* $q_1, \ldots, q_m$ *and connections between them and connecting each* q_α *with* p *does not belong to* Σ *that are neighbors of* p_α. *Then* Γ^{Σ} *possesses the eigenvalue* 1 *with a localized eigenfunction that is nonzero only at* p_α *and* q_α.

Corrollary 2.19 *Let* Γ^{Σ} *be obtained from* Γ *by adding a copy of the* motif Σ^1, *a copy of the* motif Σ *consisting of vertices* $q_1, \ldots, q_m$ *and the corresponding connections between them and connecting each* q_α *with all* p *that are neighbors of* p_α. *Then* Γ^{Σ} *possesses* m *more eigenvalues 1 that* Γ *with a localized eigenfunctions* f_1^α $(\alpha = 1, \ldots, m)$ *that are* 1 *at* p_α, -1 *at* q_α *and zero elsewhere.*

This theorem also holds when Σ is a single vertex. Thus, with the corollary if there is a high multiplicity for the eigenvalue 1, the graph may be evolved by many doubling of vertices or if the graph is constructed by doubling vertices, the multiplicity of eigenvalue 1 would be high. If one wants a high multiplicity at 1, he/she can perform many vertex doubling. This could be done both by doubling a vertex many times or doubling different vertices. But it is claimed that each procedure leaves certain traces in the spectrum [8].

Unfortunately, this result can not be generalized for more general eigenvalues. But the following theorem has some useful implications:

Theorem 2.20 *Let Σ be a motif in Γ. Suppose f satisfies*

$$\frac{1}{n_i} \sum_{j \in \Sigma, j \sim i} f(j) = (1 - \lambda) f(i), \tag{2.7}$$

for all $i \in \Sigma$ and some λ. Then the motif doubling of above theorem produces the graph Γ^Σ with eigenvalue λ and eigenfunction f^{Γ^Σ} agreeing with f on Σ, with $-f$ the double of Σ and identically 0 on the rest of Γ^Σ.

This theorem can be applied to the smallest motif, an edge. Assuming the vertices of the motif are p_1 and p_2, the Equation (2.7) becomes;

$$\begin{aligned} \frac{1}{n_{p_1}} f(p_1) &= (1 - \lambda) f(p_2), \\ \frac{1}{n_{p_2}} f(p_2) &= (1 - \lambda) f(p_1), \end{aligned}$$

and admits the solution $\lambda = 1 \pm \frac{1}{\sqrt{n_{p_1} n_{p_2}}}$.

Therefore, as the degree of vertex increases, the eigenvalues start to gather around 1 more and more. Moreover they would be symmetric around 1. The following theorem is about doubling the entire graph.

Theorem 2.21 *Let Γ_1 and Γ_2 be isomorphic graphs with vertices $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ respectively where p_i corresponds to q_i for all i. Then a graph Γ_0 can be constructed by connecting p_i with q_j whenever $p_j \sim p_i$. If $\lambda_1, \ldots, \lambda_n$ are eigenvalues of Γ_1 and Γ_2, then the new graph has the same eigenvalues as well as the eigenvalue 1 with multiplicity n.*

The next result is about motif joining and works for any eigenvalue.

Theorem 2.22 *Let Γ_1 and Γ_2 be graphs with common eigenvalue λ and corresponding eigenfunctions f_λ^1 and f_λ^2. Assume that $f_\lambda^1(p_1) = 0$ and $f_\lambda^2(p_2) = 0$ for some $p_1 \in \Gamma_1$ and $p_2 \in \Gamma_2$. Then the graph Γ obtained by joining Γ_1 and Γ_2 by identifying p_1 and p_2 also has the same eigenvalue λ with eigenfunction given by f_λ^1 on Γ_1 and f_λ^2 on Γ_2.*

2.3 Spectrum of Some Special Graphs

Definitions of examples of special graphs are given in Section 2.1. Some of these graphs' spectrums can easily be formulated and examples for these graphs are given in this Section.

- *Bipartite Graphs:* The largest eigenvalue of a bipartite graph is 2 and this property is specific to bipartite graphs. Moreover if λ is an eigenvalue of the graph, $2 - \lambda$ is also an eigenvalue for the graph. Therefore the spectrum of a bipartite graph is symmetric.

- *Complete Graphs:* Except the first eigenvalue, all eigenvalues of a complete graph, K_n, are equal:
$$\lambda_1 = \lambda_2 = \ldots = \lambda_{N-1} = \frac{N}{N-1} \text{ and } \lambda_0 = 0.$$

- *Complete Bipartite Graphs:* For a complete bipartite graph $K_{m,n}$, the eigenvalues are $0, 1$ and 2. The multiplicity of 1 is $m + n - 2$.

- *Regular Graphs:* An important property of regular graphs is the following relations between connectivity matrices:
$$\begin{aligned} L &= kI - A, \\ \mathcal{L} &= I - \frac{1}{k}A, \end{aligned}$$
where k is the degree of vertices, A is adjacency, L is Laplacian matrix and $\mathcal{L}$ is normalized Laplacian matrices of graphs.

  There is no explicit formulation for the eigenvalues of this type of graphs. However, the above relations can be adapted to the eigenvalues. Let $\theta_1, \ldots, \theta_n$ be eigenvalues of adjacency matrix. Then
$$\begin{aligned} \text{spectrum}(L) &= \{k - \theta_n, \ldots, k - \theta_1\}, \\ \text{spectrum}(\mathcal{L}) &= \left\{1 - \frac{\theta_n}{k}, \ldots, 1 - \frac{\theta_1}{k}\right\}. \end{aligned} \tag{2.8}$$

Therefore, for a k-regular graph it would be enough to determine eigenvalues of one of connectivity matrices.

- *Strongly Regular Graphs:* As it can be understand from the name strongly regular graphs are a type of regular graphs. Therefore it would be enough to find one matrices' eigenvalues so that the other eigenvalue can be found by (2.9). The eigenvalues of adjacency matrix of a strongly regular graph are determined by the parameters of the graph. Let G be a strongly regular graph with parameters (n, k, a, c), then the eigenvalues of adjacency matrix of G would be:

$$\begin{aligned} \theta &= \frac{(a-c)+\sqrt{\Delta}}{2}, \\ \tau &= \frac{(a-c)-\sqrt{\Delta}}{2}, \end{aligned}$$

where $\Delta = (a-c)^2 + 4(k-c)$with the multiplicities

$$\begin{aligned} m_\theta &= \frac{1}{2}\left((n-1) - \frac{2k+(n-1(a-c))}{\sqrt{\Delta}}\right), \\ m_\tau &= \frac{1}{2}\left((n-1) - \frac{2k+(n-1(a-c))}{\sqrt{\Delta}}\right). \end{aligned}$$

Since the multiplicities must be integers, these formulas can also be used to check whether a strongly regular graph exists with the given parameters or not.

- *Paley Graphs:*

Definition 2.23 *Let q be a prime such that $q \equiv 1$ (mod 4). The **Paley** graph P_q has a vertex set consisting of elements of GF(q). Two vertices in P_q are adjacent if and only if their difference is a square in GF(q). Let p be any prime number. The Paley sum graph, $\widehat{P}_p$, has vertices $0, 1, \ldots, p-1$ and two vertices i and j are adjacent if and only if $i + j$ is a quadratic residue module p. For $p \equiv 3$ mod(4), Paley sum graphs are directed.*

Paley graphs are examples of strongly regular graphs with parameters

$$(q, \frac{q-1}{2}, \frac{q-5}{4}, \frac{q-1}{4}),$$

where q is the number of vertices, $\frac{q-1}{2}$ is the degree of vertices, $\frac{q-5}{4}$ is the number of common neighbors of each adjacent vertex and $\frac{q-1}{4}$ is the number of common neighbors of each nonadjacent vertex.

A property of Paley graphs is that they belong to the family of expander graphs. There is no strict definition about expander graphs. Intuitively, they contain the class of graphs that each subset X of vertex set have many neighbors. In other words, any 'small' subset of vertices has a relatively 'large' neighbors. Although, there is no explicit explanation of concepts small, large and many, some restrictions about these concepts and more information about expanders can be found in [21], Chapter 6.

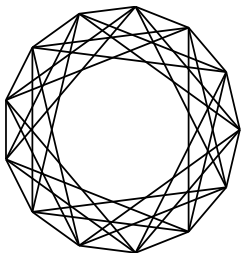


Figure 2.8: Paley graph of 13 vertices

The above formulation of eigenvalues of adjacency matrix can be applied to eigenvalues of adjacency matrix of Paley graphs too. The simplified formulation for them is:

$$\begin{aligned} \theta_1 &= \frac{-1+\sqrt{q}}{2}, \\ \theta_2 &= \frac{-1-\sqrt{q}}{2}. \end{aligned} \tag{2.9}$$

with θ_1 and θ_2 have equal multiplicities $\frac{q-1}{2}$.

Moreover by the relation between matrices of regular graphs, the eigenvalues of Laplacian matrix of Paley graphs can be found:

$$\lambda = 0, \quad \lambda_1 = 1 + \frac{1-\sqrt{q}}{q-1} \quad \text{and} \quad \lambda_2 = 1 + \frac{1+\sqrt{q}}{q-10}, \tag{2.10}$$

with the same multiplicities as above.

2.4 An Example For Real Networks: Protein Protein Interaction Networks

Recent researches on cellular biology have showed that understanding only the functioning of individual cellular components is not enough to explain certain processes within the cell. For example, during cancer researches, although the responsible genes and proteins are found,

their effect on suppressing the growth of cancer can not be maintained. This problem is resolved by a new approach which is an emerging and popular perspective in biological researches nowadays. It includes, taking into account the whole system interacting with the related proteins and genes in addition to investigating specific genes responsible for cancer [17, 73]. Observations showed that many functional activities within the cell arise from interactions between the components such as RNA, DNA, small molecules and proteins. As a result, understanding and analyzing the relations, interactions between these components have become one of the key challenges of biological research in 21th century [16].

Proteins within the cell are responsible for many important tasks. They form the basics of enzymes (which catalyzes and regulates the reactions) and dominate transcriptions, synthesis of some proteins and the translation of information through the cell (a signalling molecule attaches to the matching protein and the signal propagates along a pathway by series of interactions between proteins) [17]. Therefore, analyzing the interactions between proteins have great importance in understanding the vital processes within the cell.

The physical relations between proteins can easily be conceptualized to vertex-edge representation: proteins are assigned as vertices and two vertices are linked each other if the corresponding proteins interact with each other. Moreover, the nature of links determines the type of the graph: directed or undirected. In a directed protein protein interaction (PPI) network, the interactions have follow a certain path. For example, the flow of materials from one protein to another would have a certain path. Therefore the graphical representation of the network would be directed. On the other hand, the links are not assigned by a specific direction in an undirected network as in the case of binding of a protein to another [16]. This graph theoretical representation of PPI networks enabled to reveal the important features such as disassortativity property, high clustering, small world property and robustness against random attacks [16, 17, 72]. Disassortativity [16] property is related with making new connection tendency of vertices. In PPI networks, it is observed that highly connected proteins tend to make connections with proteins having fewer neighbors. Small world property in a graph indicates the existence of short paths among vertices. The existence of this property in PPI networks explains the quick information transformation among proteins. Another interesting property of PPI's is that although they have a high degree of tolerance for removing randomly selected nodes, attacks on highly connected vertices could be very important for the network. In addition, it is also possible to trace back the evolutionary process of PPI's with this graph

representations [8]. It was found that the spectral distribution of related normalized Laplacian matrices of real networks of PPI's have a different structure than network models like Erdös Renyi random graphs, small world graphs, Watts–Strogatz graph or Barabasi Albert graph [8, 10]. This implies these model networks are insufficient to imitate real PPI networks. In other words, different models are required in order to understand specific features of PPI networks.

Duplication of proteins and mutation of connections are the two basic processes that constitute the evolution of PPI networks. They correspond to duplication of vertices and edges respectively and result in different motifs such as triangles or squares in the graph. Moreover these processes are reflected to the spectrum of the normalized Laplacian matrix as high multiplicities at eigenvalue 1 and sometimes relatively less multiplicities at $3/2$ [8, 9, 10]. Many algorithms depending on node duplication process are constructed for modeling PPI's from different aspects (for example see [1]) such as imitating the degree distribution of vertices. However, they are generally insufficient to recover all structural properties of real PPI networks such as clustering coefficient. Inspiring by the evolutionary process behind the PPI networks, a new algorithm is suggested in [10]. Apart from the usual algorithms, the authors made some modifications on the assumption of links between old and duplicated copies. Instead of cross links between old and duplicated vertices, they assumed a low probability preference for second order neighbors as recipients of new connections. They observed a correspondence between spectral distributions of normalized Laplacian matrices of this model and real PPI networks and also the parameters such as clustering coefficient, maximum degree etc. Detailed analysis of algorithm and comparisons can be found in [8, 10].

More information about construction graphs of PPI networks can be found in [71, 72, 73] and for more references and examples about structural properties of PPI networks the reader is referred to [10, 12, 16, 17].

CHAPTER 3

COMPUTATION OF EIGENVALUES OF LARGE SPARSE MATRICES

The connectivity matrices of graphs described in Chapter 2 do not have a regular structure and in general they are in sparse form. Most of the entries of the Laplacian matrices are zero. In contrast to the dense matrices, in case of sparse matrices only the nonzero entries are stored. This property is especially important for very large matrices in terms of storage restrictions when numerical methods are applied to determine the eigenvalues and eigenvectors.

Numerical methods can be divided into two classes: direct methods and iterative methods. Direct methods produce the solution in a finite number of steps whereas iterative methods produce a sequence of approximations that converges to the true solution. Theoretically, an iterative method requires infinite number of steps, but in applications when the desired accuracy is reached, the iterations are terminated. Almost all eigenvalue algorithms are iterative [74].

The most widely used iterative method for solving eigenvalue problems is the *QR* algorithm. The practical idea of this algorithm depends on writing the matrix A as a product of an orthogonal matrix Q and an upper triangular matrix R. Each step of the algorithm performs a unitary similarity transformation to the matrix A (see for example Chapters 3 and 5 [74], Chapters 1 and 3 [45], Chapters 2 and 3 [69]). Although, the similarity transformations preserve the basic structures of the matrices such as symmetry and real form, the sparsity structure of the matrix A is not preserved. After a few steps of the *QR* algorithm, due to fill-in, the sparse matrix A becomes dense. Therefore, when the eigenvalues of sparse matrices are sought, the algorithms preserving the sparse structure should be used [74]. Excluding the QR algorithm, the most widely used iterative methods for the computation of the eigenvalues and eigenvectors

of sparse matrices are the projection methods, which exploit matrix vector multiplications, so that the sparse structure is preserved. The basic idea of a projection method is to find eigen pairs by extracting the approximate eigenvector from a specified low dimensional subspace. First, the eigenvalue problem is reduced to an approximate lower dimensional problem, then lower dimensional problem is solved by generally *QR* method or some similar algorithms. In other words, the method consists of approximating the exact eigenvector u by a vector $\tilde{u}$ belonging to some projection subspace $\mathcal{K}$ [6]. The uniqueness of the eigenvalues and eigenvectors is guaranteed by imposing the Petrov–Galerkin condition ([74], Chapter 4 [69], Chapter 3 [6]). This condition assures that the residual $(A\tilde{\lambda} - \tilde{\lambda}\tilde{u})$ of the approximate eigenvector $\tilde{u}$ must be orthogonal to some subspace $\mathcal{L}$:

$$v^*(A\tilde{\lambda} - \tilde{\lambda}\tilde{u}) = 0 \quad \forall v \in \mathcal{L},$$

where $\tilde{u} \in \mathcal{K}$. If the matrix V with the column vectors $v_1, \ldots, v_n$ is a basis for $\mathcal{L}$, left subspace, and the matrix W with the column vectors $w_i, \ldots, w_n$ is a basis for $\mathcal{K}$, right subspace, the Petrov–Galerkin condition reduces the eigenvalue problem into a lower dimensional subspace:

$$B_m y = \tilde{\lambda} y, \qquad B_m = V^* A W. \tag{3.1}$$

Projection methods are classified as oblique projection methods and orthogonal projection methods with respect to the choice of left subspace, $\mathcal{L}$. If $\mathcal{L}$ is chosen to be different from $\mathcal{K}$, the method becomes an oblique projection method [6]. In an oblique projection method, two different bases for $\mathcal{K}$ and $\mathcal{L}$ are constructed. These basis must be bi-orthogonal to each other. This may seem expensive but there are cheaper oblique projection methods than other projection methods [59]. A disadvantage of this method is that the approximated eigenvalue problem 3.1 becomes more ill-conditioned than orthogonal projection methods. Despite this fact, oblique methods are efficient in approximating right eigenvectors [6]. If $\mathcal{L}$ is chosen to be equal to $\mathcal{K}$, it is then called an orthogonal projection method [6]. There are two important classes of orthogonal projection methods: Krylov subspace methods and Jacobi-Davidson type algorithms. Both methods are based on the multiplication of the matrix A, with a starting vector q, and approximating the eigenvectors by constructing the set $K = \{q, Aq, A^2q, \ldots\}$. However, as the powers of A increase, the vectors, Aq, in the set K become linearly dependent. Instead of this, both methods try to create an orthogonal basis for approximating the eigen space. The Krylov subspace methods insists on keeping a Krylov structure on the set K whereas Jacobi-Davidson type algorithms use a correction equation [67] and Chapter 6 [69].

Projection methods compute the external (largest and smallest) and well separated eigenvalues very efficiently. However they are not very efficient in computing the clustered eigenvalues or the eigenvalues lying in the interior of the spectrum directly. An alternative way for increasing the performance of methods is applying spectral transformations to the matrices. These transformations basically rely on the idea that if $p(x)$ is a polynomial of degree k, $p(x) = b_0 + b_1 x + \ldots + b_k x^k$, and λ is an eigenvalue of matrix A with eigenvector x then $p(\lambda)$ would be an eigenvalue of $p(A)$ with the same corresponding eigenvector x. In the transformed matrix, the desired eigenvalues should be well separated, and the spectrum should be computed with a reasonable effort. Therefore, to apply such a transformation, it remains only to detect a polynomial satisfying these requirements. The most popular spectral transformation is shift and invert method. It starts by choosing a shift, say σ close to the desired eigenvalue. Then the matrix A is shifted with this value, $(A - \sigma I)$, and the eigen problem is applied to the inverse of the shifted matrix:

$$(A - \sigma I)^{-1} x = \tilde{\lambda} x.$$

The desired eigenvalues of the transformed matrix would be largest in magnitude and they can be computed easily by methods like the power method. Then eigenvalues of A can be found by the following formula $\lambda_j = \sigma + \frac{1}{\tilde{\lambda}_j}$. The most important drawback of shift and invert method is that it brings the problem of inverting a matrix. This is handled by transforming the problem to a linear system and then solving this linear system (forward and backward substitution). In many applications, the linear system is solved by applying LU decomposition. Experiments showed that for large sparse matrices, computing LU is an expensive approach. An alternative way to avoid this expensive decomposition is using harmonic Ritz vectors which will be explained in Section 3.4.

There are excellent books about the large sparse eigenvalue problems [68], Chapters 5 and 6 [69], Chapter 6 [74], Chapters 8 and 9 [75], Chapters 4, 5, 7 and 8 [6]. See also the review article of Sorensen [67] and the article of Golub, van der Vorst [35] about the history of the eigenvalue computation in the last century. In the rest of this Chapter, we will outline the algorithms for the Krylov subspace methods like the Arnoldi method, the Lanczos method, the Krylov-Schur method, the inverse free preconditioned Krylov method (EIGIFP), Jacobi-Davidson type methods, (JDQR).

3.1 Krylov Subspace Methods

This section is based on Chapter 4 [6], Chapters 1 and 2 [24], Chapter 13 [56] and Chapter 6 [74] and on the articles [47, 57, 65, 66, 67].

The basic and simplest iterative method for solving eigenvalue problem

$$Ax = \lambda x \tag{3.2}$$

is the power method. It starts with a random vector q and goes over multiplying the vector with higher powers of the matrix A. Krylov subspace methods follow the path of power method. The difference is that methods depending on Krylov subspace use all the information from the previously computed vectors whereas power method uses only the recent information that comes from last computed vector.

We will now give some definitions which are needed throughout this chapter.

Definition 3.1 *Let $A \in \mathbb{R}^{n\times n}$ be a matrix and $q \in \mathbb{R}^{n\times 1}$ be a vector. The space spanned by the set $\{q, Aq, A^2q, \ldots, A^{j-1}q\}$ is called the j–th Krylov space associated by A and q. It is denoted by $K_j(A, q)$.*

Definition 3.2 *A matrix A is called upper Hessenberg if $a_{ij} = 0$ for $i > j + 1$ and lower Hessenberg if $a_{ij} = 0$ for $i < j - 1$.*

The basic idea of the Krylov subspace methods originates from the famous Schur theorem. According to Schur theorem every matrix has a basic Schur decomposition, or can be translated to a Jordan form. The methods exploit this idea and translate the matrices into a simpler and lower dimensional form so that it would be easy to compute the eigenvalues of the transformed matrices.

Krylov methods try to find an approximate eigenpair $\{\tilde{\lambda}, \tilde{q}\}$ to the eigenproblem (3.2) by projecting the matrix A onto the associated Krylov subspace. This is done by imposing the Galerkin condition $(A\tilde{q} - \tilde{\lambda}\tilde{q})\perp\mathcal{K}$:

$$v^*(A\tilde{q} - \tilde{\lambda}\tilde{q}) = 0 \quad \forall v \in \mathcal{K}. \tag{3.3}$$

Let $\{q_1, q_2, \ldots, q_m\}$ be an orthogonal basis for the Krylov subspace and $Q = [q_1\, q_2\, \ldots q_m]$ with columns consisting of these basis vectors. The condition (3.3) is equivalent to

$$q_j^*(AQy - \tilde{\lambda}Qy) = 0,\ \ j = 1, \ldots, m,$$

where $\tilde{q} = Qy$. Therefore $(\tilde{\lambda}, y)$ is an eigen pair of the matrix Q^*AQ i.e. the projection of the matrix A onto the Krylov subspace. Here $\tilde{\lambda}$ is called Ritz value and the corresponding eigenvector $\tilde{q}$ is called the Ritz vector.

The construction scheme of basis set is the key point of Krylov subspace algorithm. There exist different types of method depending on different approaches in this construction scheme. For the rest of this section, we will present the main different versions of Krylov subspace methods.

3.1.1 Arnoldi Method

Arnoldi process can be seen as an improved version of power method. Basically, it starts with a random vector q, continues to compute $Aq, A^2q, \ldots$ and tries to construct the Krylov space with these vectors. The major difference from power method is that Arnoldi process uses the information gained from all computed vectors.

Starting with a random vector q and computing $\{q, Aq, A^2q, \ldots, A^jq\}$ results in an ill-conditioned set of vectors. Because as j increases, the vectors A^jq points to the dominant eigenvector of A more and more, i.e. they point the same direction. Therefore they will not constitute a good basis set and should be replaced with an orthonormal set. This can be achieved by applying various orthogonalization techniques to the Krylov subspace. The most common orthogonalization scheme is the Gram−Schmidt process. In literature, Arnoldi method is sometimes interpreted as a modified Gram−Schmidt process for building an orthonormal basis for Krylov subspace, $K_m(A, q)$. However, different implementations also exist. For example, in [32] Householder transformations are applied in Arnoldi method for constructing the orthogonal basis. It is reported that Householder algorithm is numerically reliable but it is more expensive than Gram Schmidt [6]. We describe here the standard Arnoldi method with Gram–Schmidt orthogonalization process.

Gram–Schmidt orthogonalization starts with normalizing the random initial vector q

$$q_1 = \frac{q}{\|q\|_2}.$$

Then Aq_1 is calculated and orthogonalized against q_1:

$$\tilde{q_2} = Aq_1 - < Aq_1, q_1 > .$$

Then $\tilde{q_2}$ is normalized and so on. At the k-th step, q_k is computed by

$$\tilde{q}_k = Aq_k - \sum_{j=1}^{k} g_j h_{jk}, \text{ where } h_{jk} =< Aq_k, q_j >, \tag{3.4}$$

$$q_{k+1} = \frac{\tilde{q}_{k+1}}{h_{k+1,k}} \text{ where } h_{k+1,k} = \|\tilde{q}_{k+1}\|_2. \tag{3.5}$$

From Equations 3.4 and 3.5, we obtain:

$$Aq_k = \sum_{j=1}^{k+1} q_j h_{jk}, \text{ k} = 1, 2, 3, \ldots, \tag{3.6}$$

where the values h_{jk} are as given above.

Algorithm 1 shows the basic steps of Arnoldi method with modified Gram-Schmidt re-orthogalization. Here, the computation of eigenvalues are not included. It can be noticed that matrix vector product is done only in one step, 3–rd step. The steps 5 to 8 are orthogonalization steps of q_{k+1} against already computed vectors. Re-orthogonalization is done in the steps between 9 and 14. These steps are irrelevant in exact arithmetic because Arnoldi procedure without reorthogonalization would produce orthogonal vectors. However, in floating point arithmetic, orthogonality is generally lost due to rounding errors. Therefore a second re-orthogonalization procedure is required. At 15–th step, the algorithm stops if the condition, $h_{k+1,k} = 0$, is satisfied. Because if $h_{k+1,k} = 0$, by Equation (3.5) $\tilde{q}_{k+1} = 0$. This means that the vectors $q, Aq, A^2q, \ldots, A^kq$ are linearly independent, i.e. Krylov subspace is invariant under A. The computed Ritz values will be exact eigenvalues of A. However, in floating point arithmetic $h_{k+1,k}$ will never be zero. But its quantity gives an idea for determining the convergence of approximations.

The equations (3.6) can be represented in matrix form: Let $Q_m = [q_1, \ldots, q_m] \in \mathbb{C}^{n \times m}$ be an

Algorithm 1 The basic Arnoldi process with re-orthogonalization

1: Give the matrix A and the initial vector q
2: $q_1 = \frac{q}{\|q\|_2}$
3: **for** $k = 1, \ldots, m-1$ **do**
4: $\quad q_{k+1} \leftarrow Aq_k$
5: $\quad$ **for** j= $1, \ldots, k$ **do**
6: $\quad\quad h_{jk} \leftarrow q_j^* q_{k+1}$ $\quad\rightarrow$ Orthogonalizing the vector computed in step 3
7: $\quad\quad q_{k+1} \leftarrow q_{k+1} - q_j h_{jk}$
8: $\quad$ **end for**
9: $\quad$ **for** j=$1, \ldots, k$ **do**
10: $\quad\quad \alpha \leftarrow q_j^* q_{k+1}$ $\quad\rightarrow$ Re-orthogonalization
11: $\quad\quad q_{k+1} \leftarrow q_{k+1} - q_j \alpha$
12: $\quad\quad h_{jk} \leftarrow h_{jk} + \alpha$
13: $\quad\quad h_{k+1,k} \leftarrow \|q_{k+1}\|_2$
14: $\quad$ **end for**
15: $\quad$ **if** $h_{k+1,k} = 0$ **then**
16: $\quad\quad$ Stop $\quad\rightarrow$ An invariant subspace is detected and the algorithm is stopped
17: $\quad$ **end if**
18: $\quad q_{k+1} \leftarrow \frac{q_{k+1}}{h_{k+1,k}}$ $\quad\rightarrow$ Orthonormalization
19: **end for**

orthogonal matrix and the Hessenberg matrix be

$$H_{m+1,m} = \begin{pmatrix} h_{11} & h_{12} & \cdots & h_{1,m-1} & h_{1m} \\ h_{21} & h_{22} & \cdots & h_{2,m-1} & h_{2m} \\ 0 & h_{32} & \cdots & h_{3,m-1} & h_{3m} \\ 0 & \ddots & & \vdots & \vdots \\ 0 & & & h_{m,m-1} & h_{mm} \\ 0 & 0 & \dots & 0 & h_{m+1,m} \end{pmatrix}.$$

Then the Arnoldi process is described by Equation (3.6) can be rewritten as:

$$AQ_m = Q_{m+1}H_{m+1,m}. \tag{3.7}$$

If $Q_{m+1}H_{m+1,m} = Q_mH_m + q_{m+1}h_{m+1,m}e_m^\top$ where e_m equals to the m-th column of the identity matrix, the process can be summarized as:

$$AQ_m = Q_mH_m + q_{m+1}h_{m+1,m}e_m^\top. \tag{3.8}$$

The columns of Q_m are referred as Arnoldi vectors and H_m is referred as Arnoldi matrix.

If $h_{m+1,m} = 0$, Equation (3.8) becomes

$$AQ_m = Q_mH_m \longrightarrow Q_m^*AQ_m = H_m.$$

This means that the projection of A onto the m dimensional Krylov subspace is the matrix H_m. Since $m \ll n$, eigenvalues of H_m can easily be computed by QR algorithm and they will be exactly the m eigenvalues of A restricted to that subspace. However, in applications it is very rare to have $h_{m+1,m} = 0$ and in the Arnoldi method this idea is used to find approximations to the eigenvalues of A.

Even if $|h_{m,m+1}| \neq 0$, the eigenvalues of A can still be approximated by the eigenvalues of H_m. Let μ be the eigenvalue of H_m with associated eigenvector x and $v = Q_mx$. The pair (μ, v) is called Ritz pair. If it was an exact eigen pair of A, the residual norm, $||Av - \mu v||$, would equal to 0. In addition, if the residual was small, the pair would still be a good approximation. Therefore, the residual can be used as a parameter for checking the convergence degree of Ritz values. Fortunately, the norm of the residual is not needed to be calculated by direct

computations. Instead, the following formulation exists [74]:

$$\begin{aligned}
\|Av - \mu v\|_2 &= \|AQ_m x - \mu Q_m x\|_2 \\
&= \|Q_m H_m x + q_{m+1} h_{m+1,m} e_m^\top x - \mu Q_m x\|_2 \\
&= \|Q_m \mu x + q_{m+1} h_{m+1,m} e_m^\top x - \mu Q_m x\|_2 \\
&= \|q_{m+1} h_{m+1,m} e_m^\top x\|_2 \\
&= \|q_{m+1}\| |h_{m+1,m}| \|e_m^\top x\|_2 \\
&= |h_{m+1,m}||x_m|
\end{aligned}$$

where x_m is the last component of the vector x.

From this result, it can be seen that whenever the eigenvalues are well conditioned, if $h_{m+1,m}$ and the last component of eigenvector of H_m are small, the approximations to the eigenvalues of A will be good.

The performance of the Arnoldi algorithm depends strongly on the initial vector q. The algorithm behaves different with respect to q. For example, if one wants to find the largest eigenvalue of the matrix A and if the initial starting vector is the related eigenvector then the process terminates in one step. If the initial eigenvector is a linear combination of j eigenvectors, the algorithm terminates in j steps. Unfortunately in real applications, such an information for choosing a good initial vector q is not available. However, the information obtained after a few Arnoldi steps can be used to develop the initial vector, q. This is done by applying restarting. Restarting means replacing the starting vector q with an improved vector q^+ and continuing the factorization with this new vector.

There are two restarting methods applied to Arnoldi algorithm. One restarting scheme is applied explicitly. It starts with a random vector and implements m steps of Arnoldi process to obtain an approximate eigenvector and eigenvalue. If the desired accuracy is not reached, this new eigenvector is then used as an initial vector and the algorithm is started from the beginning. This is called explicitly restarting Arnoldi algorithm.

The other restarting approach is called implicit restarting which was introduced by Sorensen [65]. This approach is more reliable and saves computer storage and time. Therefore, it is a common choice used in many applications. The implicitly restarting procedure basically combines the implicitly shifted QR algorithm and Arnoldi method. The idea originates from the fact that if q is a linear combination of k desired eigenvectors, the algorithm ends in

k steps. Implicitly restarted Arnoldi (IRA) algorithm tries to build the initial vector with approximations of wanted eigenvectors by the help of polynomial filters (see for example [74]). An iteration of implicitly restarted Arnoldi process will be described below.

If k eigenvalues of the matrix A are sought, then $m = k + j$ steps of Arnoldi process is performed (generally j is chosen equal to k) and Equation (3.8) is obtained: Then the m eigenvalues, $\mu_1, \ldots, \mu_m$, of H_m are computed by QR iteration and the spectrum of H_m is divided into two sets. One set contains the wanted eigenvalues, $\mu_1, \ldots, \mu_k$, and the other set contains the unwanted eigenvalues, $\mu_{k+1}, \ldots, \mu_m$. After this point, the shifts related with the unwanted portion of the spectrum are chosen and QR algorithm is applied to H_m with shifts $\nu_1, \ldots, \nu_j$.

$$(H_m - \nu_1 I) \ldots (H_m - \nu_j) = V_m R_m, \tag{3.9}$$

where V_m is an orthogonal matrix and R_m is an upper triangular matrix. Let $\widehat{H}_m = V_m^{-1} H_m V_m$, $\widehat{Q}_m = Q_m V_m$ and $\hat{q}_1$ be the first column of $\widehat{Q}_m$. The next step would be to restart the Arnoldi process from the beginning with $\hat{q}$ (the initial vector). But now, there is no need to start it from the beginning, because multiplying (3.8) by V_m yields:

$$\begin{aligned} AQ_m V_m &= Q_m H_m V_m + q_{m+1} h_{m+1,m} e_m^\top V_m, \\ A\widehat{Q}_m &= Q_m V_m \widehat{H}_m + q_{m+1} h_{m+1,m} e_m^\top V_m, \\ A\widehat{Q}_m &= \widehat{Q}_m \widehat{H}_m + q_{m+1} h_{m+1,m} e_m^\top V_m. \end{aligned} \tag{3.10}$$

The vector $e_m^\top V_m$ has exactly $m - j - 1$ leading zeros so if last j entries are dropped, βe_k would be obtained where β is a scalar. If the last j columns of equation are dropped:

$$A\widehat{Q}_k = \widehat{Q}_k \widehat{H}_k + \breve{q}_{k+1} \breve{h}_{k+1,k} e_k^\top + q_{m+1} h_{m+1,m} \beta e_k^\top,$$

where $\breve{q}_{k+1}$ is (k+1)st columns of $\widehat{Q}_k$ and $\breve{h}_{k+1,k}$ is (k+1,k) entry of $\widehat{H}_k$.

Finally, we obtain

$$A\widehat{Q}_k = \widehat{Q}_k \widehat{H}_k + \hat{q}_{k+1} \hat{h}_{k+1,k} e_k^\top, \tag{3.11}$$

where $\hat{q}_{k+1} = \gamma(\breve{q}_{k+1} \breve{h}_{k+1,k} + q_{m+1} h_{m+1,m} \beta$, γ is chosen so that $\|\hat{q}_{k+1}\|_2 = 1$ and $\hat{h}_{k+1,k} = \frac{1}{\gamma}$

It should be noticed that Equation (3.11) is the same as the Equation (3.8) except the indices (In Equation (3.11), the index numbers, k, represent a lower number than the index numbers in Equation 3.8), m.) Therefore it can be concluded that IRA does not need to start from

the beginning with $\hat{q}_1$, instead the algorithm can be designed to extract $\widehat{Q}_k$ and $\widehat{H}_k$ to start at step k, so that $k-1$ steps of Arnold algorithm are saved [74]. IRA is analyzed in detail in [66] and a comparison of explicit and implicit restarting with an analysis of using exact shifts can be found in [50]. The method can be interpreted as applying a polynomial to the matrix A and then multiplying the starting vector with this polynomial to improve the accuracy of approximate eigenvalues and eigenvectors

$$\hat{q} = p(A)q,$$

where $p(x)$ is a polynomial designed to suppress the unwanted portion of the spectrum. The key point here is the choice of the right polynomial. Determining polynomial equals to finding the appropriate shifts in the algorithm. These shifts are chosen according to the wanted part of the spectrum which has to be suppressed. The most reliable shifts are the exact unwanted eigenvalues of H_m. But in the applications alternative strategies such as Chebsyhev polynomial [57] or Leja points are also used [67].

IRA is implemented in the popular software ARPACK [48] which is known to be one of the most reliable software package for eigenvalue and eigenvector computation. However, due to the instability problems of QR algorithm, the method may fail to detect the eigenvalues. Moreover, the IRA is not very efficient for detecting clustered or multiple eigenvalues. For this kind of eigenvalue problems, block versions of the method are used [2]. The difference of a block method from one step Arnoldi algorithm is that it starts with an $n \times r$ matrix instead of a vector and the Krylov space would be of the form:

$$K_{mr}(A, X) = \text{span}\{X, AX, A^2X, \ldots, A^{m-1}X\}, \tag{3.12}$$

where X is a $n \times r$ matrix of starting vectors. The block version of Arnoldi was developed in [2], which called as augmented block Householder Arnoldi method (ABHA). It exploits as an orthogonalization technique, the Householder method to set up the Krylov basis instead of Gram-Schmidt orthogonalization. The method implements a compact WY representation. In this representation the Q matrix in Householder QR decomposition ($A = QR$) is replaced with $Q = I + YTY^\top$, where Y is lower trapezoidal and T is square upper triangular matrix. More information about this application can be found in [2]. After applying these transformations to the subspace $K_{mr}(A, X)$, the following Arnoldi equation is obtained:

$$AV_{mr} = V_{mr+r}H_{mr+r} = V_{mr}H_{mr} + V_{m+1}H_{m+1,m}E_r^\top, \tag{3.13}$$

whereas V_{mr} is $n \times mr$ orthogonal matrix and H_{mr} is $mr \times mr$ upper block Hessenberg matrix. During computations, some block diagonal matrices $H_{j+1,j}$ may become singular. This means the set of vectors in (3.12) became linearly dependent to previously computed vectors. But unlike the single vector case, this linear dependency does not imply the invariance of the subspace under A. The subspace, K_{mr}, will be invariant under A if the block diagonal matrix is totally zero.

After m steps of Arnoldi method, a compact Schur form of H_{mr} is computed to find the eigenvalues:

$$H_{mr} Q_{mr}^{H_{mr}} = Q_{mr}^{H_{mr}} U_{mr}^{H_{mr}},$$

where $U_{mr}^{H_{mr}}$ is a quasi triangular matrix with eigenvalues of H_{mr} are in 1×1 or 2×2 blocks. Then the eigenvalues are sorted with respect to the desired ones and undesired ones like the case in IRA. The general scheme of the method is similar to IRA but details are different. In [2], a block version of IRA is presented for computing the restarting vectors and then implicitly starting the method. The idea in this algorithm is mainly inspired from IRA. The difference is using Householder methods instead of Gram-Schmidt orthogonalization schemes and a starting matrix is used instead of the starting vector.

3.1.2 Lanczos Method

Lanczos method is a special case of Arnoldi method. In the simplest sense, it can be seen as an abbreviated version of Arnoldi algorithm for symmetric matrices. Despite this fact, historically Lanczos method was developed earlier than Arnoldi method [35].

When A is symmetric, the Hessenberg matrix H_m in Equation (3.8) becomes tridiagonal.

$$H_m = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \dots & 0 \\ 0 & \beta_2 & \dots & & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & & & & \beta_{m-1} \\ 0 & & & \beta_{m-1} & \alpha_m \end{pmatrix}.$$

This structure of H_m simplifies most of the coefficients in Equations (3.4) and (3.5). The

Equation (3.4) becomes a recurrence:

$$\tilde{q}_{k+1} = Aq_k - q_k\alpha_k - q_{k-1}\beta_{k-1} \text{ where } \alpha_k =< Aq_k, q_k >, \tag{3.14}$$

$$q_{k+1} = \frac{\tilde{q}_{k+1}}{\beta_k} \text{ where } \beta_k = \|\tilde{q}_{k+1}\|_2. \tag{3.15}$$

The basic steps of Lanczos method without re-orthogonalization is given in Algorithm 2 (The algorithm is taken from [74]).

Algorithm 2 Lanczos method without re-orthogonalization

Require: Initial nonzero vector $q \in \mathbf{R}^n$ and symmetric matrix $A \in \mathbf{R}^{n\times n}$

1: $q_1 = \frac{q}{\|q\|_2}$
2: **for** k= $1, \ldots, m-1$ **do**
3: $q_{k+1} \leftarrow Aq_k$
4: $\alpha_k \leftarrow q_k^\top q_{k+1}$ → Computation of Lanczos coefficient
5: $q_{k+1} \leftarrow q_{k+1} - q_k\alpha_k$
6: **if** $k > 1$ **then**
7: $q_{k+1} \leftarrow q_{k+1} - q_{k-1}\beta_{k-1}$
8: **end if**
9: $\beta_k \leftarrow \|q_{k+1}\|_2$ → Computation of other Lanczos coefficient
10: **if** $\beta_k = 0$ **then**
11: Stop → The algorithm breaks down
12: **end if**
13: **end for**
14: $q_{k+1} \leftarrow \frac{q_{k+1}}{\beta_{k+1}}$

Like Arnoldi method, the only matrix vector product occurs at third step. Then the related coefficients and vectors are computed between the steps 4 to 9. At step 10 when the condition, $\beta_k = 0$, is fulfilled, the algorithm terminates. Because $\beta_k = 0$ implies that the vector q_{k+1} is zero and that means the space span$\{q_1, \ldots, q_k\}$ is invariant under A. Moreover it is easy to see that at each step the new Lanczos vector is calculated using only two previously computed vectors, which is advantageous for saving computer storage.

At each iteration step k the new Lanczos vector q_{k+1} is calculated by orthogonalizing Aq_k against two previously computed vectors q_k and q_{k-1}. In exact arithmetic the orthogonality of space spanned by $\{q_1, \ldots, q_m\}$ is being kept for all m. The following theorem and corollary in

[24] highlights this result:

Theorem 3.3 *Let A be $n \times n$, real, symmetric matrix with n distinct eigenvalues. Let v_1 be a unit starting vector with a nonzero projection on every eigenvector of A. Using the basic single vector Lanczos recursions defined by (3.16) and (3.15) to generate Lanczos matrices H_j and Lanczos vectors $Q_j = [q_1, \ldots, q_j]$. Then for any $j \leq n$, we have that*

$$Q_j^\top Q_j = I_j.$$

Furthermore for each such j

$$H_j = Q_j^\top A Q_j,$$

is the orthogonal projection of A onto the subspace spanned by the Q_j. In other words, H_j represents the operator A restricted to span$\{Q_j\}$.

Corrollary 3.4 *Theorem (3.3) implies that for each $2 \leq j \leq n$ the basic single Lanczos recursion given by the equations (3.16) and (3.15) generates an orthonormal basis for each of the Krylov subspaces, $\mathcal{K}^j = span\{q_1, Aq_1, \ldots, A^{j-1}q\}$. Furthermore for each such j the Lanczos matrix H_j is the representation of the restriction of the matrix A to that Krylov space. Therefore the eigenvalues of T_j are the eigenvalues of A restricted to $\mathcal{K}$.*

Unfortunately the results of the theorem is true only in exact arithmetic. Generally, in floating point arithmetic, the orthogonality of Lanczos vectors is being lost after a few steps. This is an important problem because loss of orthogonality results in ill conditioned sets and this affects approximations. At the early stages of the development of the Lanczos algorithm, it was thought that the loss of orthogonality was due to only the rounding errors and therefore they were inevitable. This lead to a decrease in the interest for the method. But then, in [53] and [54] Paige found that the losses of orthogonality in Lanczos algorithm is not only due to computational errors. Instead, they are a combination of round-off errors in floating point arithmetic and the convergence of one or more of the eigenvalues of the Lanczos matrices H_j to the eigenvalues of A as j has increased. Paige examined Lanczos algorithm without reorthogonalization and with reorthogonalization. He found that the first one behaves like the latter one until the convergence of the eigenvalues of Lanczos matrices. In other words, the significant losses in the orthogonality of Lanczos vectors appears if the algorithm has been carried out far enough that eigenvalues of Lanczos matrix have converged to desired ones.

Moreover it was shown in [54] that the loss of orthogonality occurs along the directions of the corresponding converged Ritz vectors. In summary, the lost of orthogonality among Lanczos vectors is due to the interaction of round off errors with convergence of eigenvalues [24].

In the floating point arithmetic, the Equation (3.16) can be rewritten as:

$$\beta_{j+1}q_{j+1} = Aq_j - \alpha_j q_j - \beta_j q_{j-1} - f_j, \tag{3.16}$$

where f_j is a vector corresponding to the rounding errors at step j. The rounding errors are important in the sense that they show the degree of loss of orthogonality between the Lanczos vectors. It was proved in [60] that this loss follows a certain rule.

Theorem 3.5 *Let $w_{ik} = q_i^* q_k$. Then the w_{ik} satisfy the following recurrence*

$$\begin{aligned} w_{kk} &= 1 \quad for\ k = 1, \ldots, j, \\ w_{kk-1} &= q_k^* q_{k-1} \quad for\ k = 2, \ldots, j, \\ \beta_{j+1}w_{j+1k} &= \beta_{k+1}w_{jk+1} + (\alpha_k - \alpha_j)w_{jk} + \beta_k w_{jk-1} - \beta_j w_{j-1k} + q_j^* f_k - q_k^* f_j, \end{aligned} \tag{3.17}$$

for $1 \leq k < j$, and $w_{jk+1} = w_{k+1j}$. Here $w_{k0} = 0$.

The theorem implies that the loss of orthogonality depends somehow on the coefficients in the Lanczos procedure. The loss of orthogonality has two important consequences on the approximate eigenvalues. The first of impact is about the multiplicity of eigenvalues: simple eigenvalues of the matrix A may appear as multiple eigenvalues of H_j. The second one is about the appearance of ghost eigenvalues: some Ritz values of H_j may not belong to the spectrum of A.

One way to prevent the loss of orthogonality is using orthogonalization schemes i.e. re-orthogonalize each Lanczos vector against all or some previously computed ones. The other way is to accept the loss of orthogonality and then try to deal with outcomes of this loss [24].

Lanczos, himself, also proposed re-orthogonalization to maintain orthogonality. Two kinds of re-orthogonalization methods are suggested for Lanczos algorithm. One is using full re-orthogonalization, i.e. re-orthogonalize the new computed vector against all previously computed ones. To do this, all vectors are needed to be saved. This may result in loss of the advantages of Lanczos method in saving storage. The other re-orthogonalization method is limited reorthogonalization. There are two different types of this technique. The one is local

re-orthogonalization. It is done over a few computed vectors. Generally, these vectors are chosen as the recently computed Ritz vectors. The other limited re-orthogonalization scheme is called selective re-orthogonalization. In this approach, orthogonality is kept with respect to some selected eigenvectors. Both types of re-orthogonalization methods can overcome the problems occurring by full re-orthogonalization. They also produce more accurate results. More information about these methods is given in [24].

Methods using no re-orthogonalization aim to solve the problem over computed Lanczos vectors. If the Ritz values have appeared over and over as an eigenvalue of the Lanczos matrix H_j, it is assigned as an eigenvalue. Otherwise it is not accepted as an eigenvalue. This method is advantageous and efficient in calculating the spectrum of very large matrices.

We will now outline the Lanczos partial reorthogonalization and implicitly restarted block Lanczos method, which are efficient versions of the Lanczos method for computing eigenvalues of large sparse matrices.

The first method is the Lanczos partial reorthogonalization (LPRO) developed by Horst Simon in [60]. The idea in LPRO originates from using the Equations (3.17) for estimates of the level of orthogonality. The level of orthogonality, K_j, among Lanczos vectors at step j is defined as

$$K_j = \max_{1 \le k \le j-1} |q_j^* q_k|.$$

But, computational experiments showed that assigning $K_j = \sqrt{\epsilon}$, i.e., maintaining semi-orthogonality between Lanczos vectors, is sufficient to compute well approximated eigenvalues. When the equations (3.17) were used, the calculating f_j's became problem (due to the Equation 3.16). Because of this, the following alternative formula is suggested in [60]:

$$\begin{aligned} w_{kk} &= 1 \quad \text{for } k = 1, \ldots, j \\ w_{kk-1} &= \psi_k \quad \text{for } k = 2, \ldots, j \\ w_{j+1k} &= \frac{1}{\beta_{j+1}}[\beta_{k+1} w_{jk+1} + (\alpha_k - \alpha_j) w_{jk} + \beta_k w_{jk-1} - \beta_j w_{j-1k}] + \varphi_{jk} \end{aligned} \tag{3.18}$$

for $1 \le k < j$, and $w_{jk+1} = w_{k+1j}$ where $w_{k0} = 0$ and φ_{jk} and ψ_k are certain random numbers.

Computational experiments in [60] showed that this new formulation (3.18) can detect quite well the steps at which the orthogonalization has been lost. Moreover, depending on the computational experiments for different values of φ_{jk} and ψ_k, using random numbers is suggested

in [60]:

$$\begin{aligned} \varphi_{jk} &= \epsilon(\beta_{k+1} + \beta_{j+1})\Theta \\ \psi_k &= \epsilon_n \frac{\beta_2}{\beta_{j+1}}\Psi \end{aligned}$$

where $\Theta \in N(0, 0.3)$ represents the numbers distributed normally with mean 0 and standard deviation 0.3 and $\Psi \in N(0, 0.6)$

The information obtained from the Equation (3.18) is then used to decide which Lanczos vectors should be orthogonalized against the current vectors. When the orthogonality is lost, it was first suggested to group the vectors into batches and then orthogonalize the current vector against the ones from the batch. Each batch is constructed according to the following scheme: when $w_{j+1k} > \sqrt{\epsilon}$ the values w_{j+1k} are checked until $w_{j+1k-s} < \epsilon^{3/4}$ and $w_{j+1k+l} < \epsilon^{3/4}$ is satisfied so that an index set is obtained. Then q_{j+1} is orthogonalized against all the vectors belonging the index set. The value $\epsilon^{3/4}$ is the optimal value determined by computational experiments.

The second example for different implementations of Lanczos algorithm is the implicitly restarted block Lanczos method (IRBL). Implicit restarts can also be applied to Lanczos method as in the case for Arnoldi method. A combination of both implicit restarts and block vectors is developed by Calvetti, Reichel and Baglama in [4]. The algorithm starts with a set of orthonormal vectors, $V_r = [v_1, \ldots v_r]$, with block size r. Applying m steps of Lanczos method with the initial matrix V_r results in the following equation

$$AV_{mr} = V_{mr}H_{mr} + F_rE_r^*, \tag{3.19}$$

where $V_{mr} \in \mathbb{R}^{n\times mr}$, $V_{mr}I_{mr\times r}$, $V^*_{mr\times mr}V_{mr\times mr}$ and $F_r \in \mathbb{R}^{n\times r}$ with $V^*_{mr}F_r = 0$. I_{mr} is $mr \times mr$ identity matrix, $I_{mr\times r}$ is the matrix consisting of first r columns of I_{mr} and E_r is the $mr \times r$ matrix consisting last r columns of I_{mr}. Notice that this equation is similar to (3.13).

T_{mr} is $mr \times mr$ symmetric block tridiagonal matrix of the form

$$T_{mr} = \begin{pmatrix} D_1 & B_1^\top & 0 & & \\ B_1 & D_2 & B_2^\top & & \\ & B_2^\top & D_3 & & \\ & \ddots & \ddots & & \\ & & & B_{m-1}^\top & \\ & & B_{m-1} & D_m & \end{pmatrix},$$

where D_j's are $r \times r$ Hermitian diagonal blocks and B_j's are $r \times r$ nonsingular upper triangular subdiagonal blocks. If $\{\theta, y\}$ are the eigen pairs of T_{mr}, $\{\theta, x = V_{mr}y\}$ will be the Ritz pair of A. The residual norm for this Ritz pair will be

$$\begin{aligned} \|Ax - \theta x\|_2 &= \|AV_{mr} - \theta V_{mr}y\|_2, & (3.20) \\ &= \|F_r E_r^* y\|_2. & (3.21) \end{aligned}$$

The residual norm for IRBL can also be computed directly without calculating the Ritz pairs as in Lanczos algorithm.

The idea behind implicit restarts in block Lanczos method is the same as the idea behind IRA and IRL, i.e. it aims to represent the initial starting matrix with a new improved matrix in the direction of the wanted eigenvectors. After applying m steps of block Lanczos algorithm, if the residual error is larger than the desired value, implicit restart is applied. The matrix U_r is defined by $U_r := p_m(A)V_r$, where p_m is a polynomial of degree m and important for the efficiency of restarts. The polynomials are determined with respect to unwanted portion of the spectrum. The roots are referred as shifts. In the implementation of method in [3], Leja points are used for roots of polynomials. The columns of U_r are orthogonalized:

$$U_r = V_r^+ R_r^+,$$

where V_r^+ is $n \times r$ orthonormal matrix and R_r^+ is upper triangular. V_r is replaced by V_r^+ and then an m step of block Lanczos algorithm is performed again. This process is continued until residual norm is below the desired value.

3.2 Krylov-Schur Methods

This section is based on Chapter 5 [69], and on the articles [39, 58, 70]. Krylov-Schur method is developed for avoiding the numerical instabilities of IRA. It was proposed by Stewart in [70]. The method can be viewed as a simplification of the IRA when exact shifts are used in (3.9). Before we explain the details of the method, we will give some useful facts about the Krylov subspaces.

Definition 3.6 *Let $u_1, \ldots, u_{k+1}$ be linearly independent and let $U_k = (u_1 \ldots u_k)$. A Krylov decomposition of order k is a relation of the form*

$$AU_k = U_k B_k + u_{k+1} b_{k+1}^H. \quad (3.22)$$

Equivalently, the equation can be written

$$AU_{k+1} = U_{k+1}\widehat{B}_k, \tag{3.23}$$

where $U_{k+1} = (U_k\ u_{k+1})$ *and* $\widehat{B}_k = \begin{pmatrix} B_k \\ \hat{b}_{k+1} \end{pmatrix}$.

If U_{k+1} is orthonormal, then the Krylov decomposition is called orthonormal decomposition. $R(U_{k+1})$ is the space spanned by this decomposition and U_{k+1} constitutes a basis for this decomposition. Thus, when the Equations (3.8) and (3.22) are compared, it can be seen that Arnoldi Equation (3.8) is a specialized version of Krylov decomposition (3.22).

An important property of Krylov decompositions is that they are closed under the similarity and translation transformations. If Q is a nonsingular matrix, post multiplying Equation (3.22) by Q

$$AU_kQ = U_kQ(Q^{-1}B_kQ) + u_{k+1}b_{k+1}^HQ$$

results in a Krylov decomposition similar to the previous one. Let $\gamma\tilde{u}_{k+1} = u_{k+1} - u_ka$, then putting this into the Equation (3.22) yields

$$AU_k = U_k(B_k + ab_{k+1}^H) + \tilde{u}_{k+1}(\gamma b_{k+1}^H)$$

and this decomposition is again equivalent to the decomposition in (3.22) since the range of basis are the same. This translation is called transition. These two properties of decomposition yield the following important result:

Theorem 3.7 *Every Krylov decomposition is equivalent to (a possibly reduced) Arnoldi decomposition.*

This theorem can be proved by applying the transformations described above on the Equation (3.22). The proof in [69] yields the general scheme of the algorithm for transforming a Krylov decomposition to an Arnoldi decomposition.

Krylov–Schur method consists of two parts: expansion and extraction. The expansion phase starts with multiplying the initial vector with the matrix A and proceeds as the usual Arnoldi algorithm. After m steps of Krylov–Schur method, the following equation is obtained:

$$AU_m = U_mS_m + u_{k+1}b_{k+1}^H$$

, where S_m is a quasi-triangular matrix in real Schur form (with 1×1 or 2×2 blocks on the diagonal entries). In general, this stage is accomplished by first obtaining an Arnoldi decomposition of degree m. Then this decomposition is transformed to a Krylov-Schur form by unitary similarities.

The extraction phase of Krylov–Schur method is simpler. Since the matrix S_m is of quasi-triangular form, the Ritz values are on the diagonals. They are either on 1×1 or 2×2 blocks. These values are divided into two sets: Ω_w (containing wanted portion of the spectrum) and Ω_u (containing unwanted portion of the spectrum). Then the desired Ritz values of the matrix are moved into the leading submatrix of S_m with orthogonal transformations. So that a reordered Krylov–Schur form is obtained:

$$A\tilde{U}_m = (\tilde{U}_m \; u_{m+1}) \begin{pmatrix} S_w & \star \\ 0 & S_u \\ b_w^* & \star \end{pmatrix},$$

where $\lambda(S_w) = \Omega_w$ and $\lambda(S_u) = \Omega_u$ and b_w^* is of length p and the leading subvector.

At the end, this decomposition is truncated to order p and then extended to a Krylov decomposition of order m. The truncated decomposition is of the form:

$$A\tilde{U}_p = (\tilde{U}_p \; \tilde{u}_{p+1}) \begin{pmatrix} S_w \\ b_w^* \end{pmatrix},$$

where $\tilde{U_p}$ is the first p columns of $\tilde{U}_m$ and $u_{m+1} = \tilde{u}_{p+1}$ [39].

The key point in the last step originates from the fact that Krylov–Schur decomposition can be truncated at any point, i.e. if the truncation is in the form

$$A(U_1 \; U_2) = (U_1 \; U_2) \begin{pmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{pmatrix} + U(b_1^H \; b_2^H),$$

then $AU_1 = U_1 S_{11} + ub_1^H$ is still a Krylov–Schur decomposition.

The basic steps of the method are illustrated in algorithm 3 (This algorithm is adapted from [69]). At step 3, the decomposition is expanded to order m by the Arnoldi method. At step 7, the transformations are determined and applied to the Hessenberg matrix to obtain the Schur form. Then spectrum is divided into two parts at step 8, this is done by appropriate transformations. In the rest of the algorithm, the truncation procedures is implemented. If the

Algorithm 3 The Krylov–Schur cycle

Require: A Krylov–Schur decomposition $AU = \hat{U}\hat{S} = US + u_{k+1}b_{k+1}^{\top}$

1: $\hat{T} = \hat{S}$

2: **for** j= $k + 1, \ldots, m$ **do**

3: Apply Arnoldi algorithm to expand the decomposition to order m

4: **end for**

5: $T = \hat{T}[1 : m, 1 : m]$

6: $B = \hat{T}[m + 1, m]$

7: Transform $S = Q^H T Q$ so that it is triangular

8: Select $m - k$ Ritz values and move them to the end of S

9: $S = S[1 : k, 1 : k]$

10: $b^H = BQ[m, 1 : k]$

11: $U = UQ[:, 1 : k]$

work counts in implicitly restarted Arnoldi (IRA) and Krylov–Schur methods were compared, the total number of floating point additions and multiplications in expansion phase would be $2n(m^2 - k^2)$. However, for contraction phase these values are different. The accumulation of U is done in $m \times m$ Q matrix and new U_k is obtained from $U_m Q[:, 1 : k]$ for both algorithms. Then, the total number of works for IRA would be $nmk - \frac{1}{2}k^2$ and for Krylov–Schur method would be nmk. If m is taken to be equal to $2k$, the operation counts for Krylov–Schur is $7nk^2$ whereas for IRA, it is $3nk^2$. Hence IRA is superior to Krylov–Schur in marginal operation cost [70].

Implicitly restarted Arnoldi method with exact shifts and Krylov–Schur method with the same shifts have the same effect which was proved in [70]. But each have their pros and cons. Different polynomials can be used instead of shifts in IRA but this is not possible in Krylov–Schur method. On the other hand when exact shifts are used Krylov–Schur method is more preferable due to the reliable process for exchanging the eigenvalues. The residual of the approximate eigen pair (M,Z), $r = AZ - ZM$, provides a good criteria for determining the convergence. In the Frobenius norm, it is equivalent to

$$||AZ - ZM||_r^2 = ||BW - WM||_F^2 + ||b_W^H||_F^2,$$

where $Z = UW$. More information about the residual and stability of method can be found in [70].

Like Arnoldi method, Krylov–Schur can also be adapted to symmetric case. It is realized that when this happens, Krylov–Schur method becomes equivalent to the thick restarted Lanczos method [78]. The significant difference arises from the structure of the Krylov–Schur matrix, S_m, which consists of a diagonal and tridiagonal part. So, obtaining S_m from thick restarted form equals to diagonalization in the Lanczos procedure. However, this requires a special treatment in order to benefit from the symmetry [39].

Harmonic extraction for finding the interior eigenvalue can be implemented to Krylov–Schur method instead of usual shift and invert methods. Basic information about this extraction theory is given in Section 3.4. We give here only the general procedure for applying them to Krylov–Schur method.. The linear subspace W_k in Definition 3.8 given as

$$W_k = V := (A - \tau I)U_k,$$

where τ is he target value around which the eigenvalues are sought. Then from the Krylov–Schur decomposition, this subspace is equivalent to

$$V = U(B + \tau I) + ub^*.$$

Here V is not orthonormal but by some extra work it can be orthonormalized. After some calculations, one obtains

$$U = V(B - \tau I)^{-1} - ug^*, \tag{3.24}$$

where $g := (B - \tau I)^{-*}b$. Equation (3.24) is the Krylov decomposition of the shifted and inverted operator. In [58], it is shown that the eigenvalues of the matrix $(B + gb^*)$ are the harmonic Ritz values of A with corresponding harmonic Ritz vectors, Vz:

$$(B + gb^*)z = (\tilde{\theta}^{-1} + \tau)z.$$

This idea is implemented to the Krylov–Schur method by applying a translation to the original Krylov–Schur decomposition in order to obtain the Equation 3.24. More details about this implementation can be found in [58].

The block version of Krylov–Schur method for symmetric matrices is proposed by Saad and Zhou in [79]. This block implementation starts with an $n \times b$ matrix, where b is the size of blocks, instead of an initial vector like other block algorithms ($ABHA, IRBL$) defined in previous Sections. However, the rest of the implementation includes some important alterations.

The first one is the internal increase in the size of starting vector, k_s. That is, algorithm augments the block Krylov decomposition of size k_s to size k_f by a special block Lanczos pseudo code given in [79]. After, invoking this pseudo code, the following equation is obtained:

$$HV_{k_f} = V_{k_f} T_{k_f} + FB^\top \tag{3.25}$$

with $V_{k_f}^\top V_{k_f}^\top = I_{k_f}$ and $F^\top V_{k_f} = 0$. After this enlargement of space the algorithm goes on as in the one vector case with appropriate modifications for the blocks. The other important alteration is about handling the rank deficiency. The rank deficiency of matrix F implies that the augmented basis is not orthogonal any more. This situation appears when the computed vectors are not linearly independent. The authors in [79] dealt with this problem by suggesting an adaptive block size b. They proposed to adjust the block size with respect to the rank deficiency of matrix F. The details of this implementation and numerical comparisons with other block methods can be found in same article.

3.3 Inverse Free Preconditioned Krylov Subspace Method

Inverse free Krylov subspace method is developed by Golub and Ye in [36]. This method is a combination Krylov subspace methods with preconditioning. It is an extension of the inexact inverse iteration scheme and Rayleigh-Ritz projection methods. It is designed for solving the generalized eigenvalue problems

$$Ax = \lambda Bx$$

but it can also be applied to standard eigenvalue problems by choosing $B = I$. We will describe here the application of the method for generalized eigenvalue problems.

The algorithm starts with a normalized initial vector, x_0, then the The Rayleigh-Ritz quotient with respect to initial vector x_0 is computed

$$\rho_0 = \frac{x_0^\top A x_0}{x_0^\top B x_0} \tag{3.26}$$

Then a basis, V_m, for the shifted Krylov subspace

$$K_m = \text{span}\{x_0, (A - \rho_0 B)x_0, \ldots, (A - \rho_0 B)^m x_0\}$$

is calculated. Here m denotes a user specified value and it determines the dimension of the

shifted Krylov space. The new matrices are

$$A_m = V_m^\top (A - \rho_0 B) V_m, \tag{3.27}$$

$$B_m = V_m^\top B V_m. \tag{3.28}$$

calculated. The smallest eigen pair (μ_1, q_1) of (A_m, B_m) is found and new approximations are computed:

$$\rho_1 = \rho_0 + \mu_1, \tag{3.29}$$

$$x_1 = V_m q_1. \tag{3.30}$$

This procedure is iterated until with k until the desired accuracy is reached and is called outer iteration. The inner iteration is the construction of the basis, V_m, for Krylov subspace. Three different ways to construct the basis is proposed in [36]: by the Lanczos algorithm, a B-orthonormal basis by Arnoldi algorithm and C_k-orthogonal basis by a variation of Lanczos algorithm. The method was tested using all these three basis and the first two bases performed similarly, and the last basis showed poorer performance than the others [36].

Using Lanczos algorithm for inner iteration is just an application of the algorithm 2 to the matrix $C_k = A - \rho_k B$. The loss of orthogonality with increasing m is also observed in this application. However, this loss is not very important until V_m becomes seriously ill-conditioned. In that case partial reorthogonalization is suggested to apply in order to resolve the problem. The idea of constructing a B-orthonormal basis by Arnoldi algorithm is nearly the same as algorithm 1 with some small modifications. The convergence analysis of the procedure is described in [36]. They proved the following theorem which indicates that ρ_k converges to an eigenvalue and x_k converges in the direction of an eigenvector [49].

Theorem 3.8 *Let $\lambda_1 < \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of (A,B) and (ρ_{k+1}, x_k) be the approximate eigen pair obtained from $(\rho_k, x_k)$by one step of inverse free Krylov subspace method. Let $\sigma_1 < \sigma_2 \leq \ldots \leq \sigma_n$ be the eigenvalues of $A - \rho_k B$. If $\lambda_1 < \rho_k < \lambda_2$ then*

$$\rho_{k+1} - \lambda_1 \leq (\rho_k - \lambda_1)\epsilon_1^2 + O((\rho_k - \lambda_1)^{3/2}), \tag{3.31}$$

where

$$\epsilon_m = \min_{\rho \in P_M, \rho(\sigma_1)=1} \max |\rho(\sigma_i)| \leq 2\left(\frac{1 - \sqrt{\psi}}{1 + \sqrt{\psi}}\right)^m.$$

Here, P_m denotes the set of all polynomials of degree greater than m and $\phi = \frac{\sigma_2 - \sigma_1}{\sigma_n - \sigma_1}$. This theorem implies also that the convergence of algorithm depends on the spectral distribution of C_k, which leads to apply appropriate preconditioning on C_k that leaves the eigenvalues of (A, B) same.

The idea of preconditioning originates from above theorem, that aims to transform the pencil (A, B) to a new pencil $(\widehat{A}, \widehat{B})$ with the same eigenvalues but $\widehat{C}_k = \widehat{A} - \rho_k \widehat{B}$ has more appropriate spectrum distribution. Here the notion of appropriate means having a suitable spectrum distribution for applying the bounds on theorem.

Let L_k be some matrix used for preconditioning:

$$(\widehat{A}, \widehat{B}) = (L_k^{-1} A L_k^{-\top}, L_k^{-1} B L_k^{-\top}). \tag{3.32}$$

Applying the method for the transformed matrices requires the calculation of

$$\widehat{C} = \widehat{A} - \rho_k \widehat{B} = L_k^{-1}(A - \rho_k B) L_k^{-\top}. \tag{3.33}$$

So if L_k is chosen so that C_k has a good spectral distribution, the algorithm will accelerate. For example, if a complete $LDL^\top$ factorization of $(A - \rho_k B)$ was found then assigning $L_K = L$ would yield

$$C_k = L_k^{-1}(A - \rho_k B) L_k^{-\top} = L_k^{-1} L D L^\top L_k^{-\top} = D,$$

where D is a diagonal matrix and can be accepted to have an appropriate diagonal entries. An implicit implementation of preconditioning for the method is also developed in [35]. The algorithm starts the same way as non–preconditioned method. One step of new algorithm is as follows:

First an appropriate preconditioner L_k is constructed. Then the basis $V_m = [v_0, \ldots, v_m]$ for $L_k^{-\top} \widehat{K}_m$ is constructed. The new matrices $\widehat{A}$ and $\widehat{B}$ are formed in the usual way:

$$\begin{aligned} A_m &= V_m^\top (A - \rho_k B) V_m, \\ B_m &= V_m^\top B V_m. \end{aligned}$$

Then smallest eigenvalues, μ_1, and eigenvectors, q, of A_m and B_m are found and new approximations are:

$$\begin{aligned} \rho_{k+1} &= \rho_k + \mu_1, \\ x_{k+1} &= V_m q. \end{aligned}$$

The only difference from non–preconditioned case is in the inner iteration. The basis is constructed for the space $L_k^{-\top}\widehat{K}_m$ instead of K_m. As in the previous case there are two alternatives for inner iterations: constructing with Arnoldi method or constructing with Lanczos method. However the numerical studies have revealed that using Arnoldi method for preconditioned algorithm is much more efficient than using Lanczos method for preconditioned algorithm [36, 49].

3.4 Jacobi-Davidson Type Algorithms

This section is based on Chapter 6 [74], Chapter 4 [6] and on the papers [62, 63]. Jacobi-Davidson method for solving eigenvalue problems originates from the idea of projecting the matrix onto a subspace like the Krylov subspace algorithms. The difference to the methods like Arnoldi and Krylov–Schur depending on preserving Krylov space, is that Jacobi-Davidson algorithm does not insist on keeping a Krylov structure in the projected subspace.

The general principle of the algorithm depends on approximating the eigenvalues and eigenvectors by expanding the previously obtained subspace. That is, if $\{q_1, q_2, \ldots, q_k\}$ spans the subspace that A is projected onto, then the algorithm tries to find a q_{k+1} to obtain a better approximation for the desired eigenvector. The origin of the algorithm is based on the approaches by Jacobi and Davidson. Therefore first, a brief summary of the Davidson's and Jacobi's method will be given:

Davidson in his article [25] claimed to expand the subspace by orthogonalizing the correction against the residual. That is, assume A is projected over some subspace $\mathcal{K}$ and $\{q_1, \ldots, q_k\}$ forms an orthonormal basis for that subspace. Let θ_k be the Ritz value and v_k be the corresponding Ritz vector of A. The residual is $r_k = Av_k - \theta_k v_k$. Davidson suggested to compute the correction t for v_k from the equation:

$$(D_A - \theta_k I)t = r_k, \tag{3.34}$$

where D_A is the matrix consisting of the diagonal entries of A. Then the correction t is orthogonalized against the basis vectors. Finally, the orthogonalized correction is assigned as q_{k+1} and the subspace is expanded by one dimension. This method worked very well with the diagonally dominant matrices.

Jacobi in his paper [42], offered two new iterative methods. Today, one is very well known as the Jacobi method, the other was called Jacobi's orthogonal component correction (JOCC) by Gerard Sleijpen and Henk Van der Vorst in [62]. Jacobi developed the first approach to increase the efficiency of his second method. It is about transforming a matrix into a new form with plane rotations. The second idea, JOCC, starts with assuming the matrix is diagonally dominant and partitioning the matrix as

$$A = \begin{pmatrix} \mu & c^\top \\ b & F \end{pmatrix},$$

where μ is the largest diagonal elements, c and b are the vectors in the related size and F is a square matrix. μ can be interpreted as an approximation to the largest eigenvalue of A with corresponding eigenvector q and e_1 be the corresponding approximation to the eigenvector q. Let

$$q = \begin{pmatrix} 1 \\ z \end{pmatrix},$$

where z is the other component of the vector q orthogonal to e_1. Then, the eigenvalue problem turns into

$$\begin{pmatrix} \mu & c^\top \\ b & F \end{pmatrix} \begin{pmatrix} 1 \\ z \end{pmatrix} = \lambda \begin{pmatrix} 1 \\ z \end{pmatrix}.$$

Equivalently, it becomes a linear system of the form:

$$\begin{aligned} \mu + c^\top z &= \lambda, \\ (F - \lambda I)z &= -b. \end{aligned} \tag{3.35}$$

A rearrangement of the system (3.35) will yield:

$$\begin{aligned} \lambda &= \mu + c^\top z, \\ (D - \lambda I)z &= (D - F)z - b. \end{aligned} \tag{3.36}$$

where D is the diagonal of F. To solve the system (3.36), Jacobi suggested an iterative method.

$$\begin{aligned} \theta_k &= \mu + c^\top z_k, \\ (D - \theta_k I)z_{k+1} &= (D - F)z_k - b. \end{aligned}$$

by setting $z_1 = 0$ with some appropriate error bound until desired convergence is reached.

Combining the ideas of Davidson and Jacobi, Gerard Sleijpen and Henk Van der Vorst suggested a new method, the Jacobi-Davidson method [62]. We will give the basics of this method in the rest of this section.

Let $\{q_1, \ldots, q_k\}$ span the k−th dimensional subspace, $\mathcal{K}$ which A is projected onto. The algorithm tries to expand the subspace by a correction on q_k. This correction is being searched in the orthogonal complement of q_k, $q_k^\top$. Suppose the largest eigenvalue, λ, and corresponding eigenvector, x, are sought. If q_k is an approximation for x, then the correction would satisfy:

$$A(q_k + t) = \lambda(q_k + t) \tag{3.37}$$

with the condition $t \perp q_k$.

The orthogonality condition implies that A can be restricted to the complement of q_k (The orthogonal projection of a matrix A to the complement of a vector u is $B = (I - u\,u^*)A(I - u\,u^*)$). With this restriction the Equation 3.37 becomes

$$(I - q_k q_k^*)(A - \lambda I)(I - q_k q_k^*)t = -(A - \theta_k)q_k \tag{3.38}$$

with $t \perp q_k$ where θ_j is the corresponding Ritz value. In practice, λ is unknown, so it can be replaced by

$$(I - q_k q_k^*)(A - \theta_k I)(I - q_k q_k^*)t = -r_k. \tag{3.39}$$

Equation (3.39) is called the Jacobi's correction equation. Various iterative solvers can be used to solve this equation. It is reported in [62] that the method does not require to solve 3.39 with a high accuracy. On the other hand numerical experiments showed that the convergence of whole algorithm increases with more accurate solutions of Equation (3.39).

From Equation (3.39), it can be deduced that

$$(A - \theta_k I)t - \epsilon q_k = -r_k. \tag{3.40}$$

If there is a suitable preconditioner M, then t can be approximated by

$$\tilde{t} = M^{-1}\epsilon q_k - M^{-1} r_k, \tag{3.41}$$

where

$$\epsilon = \frac{q_k^* M^{-1} r_k}{q_k^* M^{-1} q_k},$$

as a result of the condition $\tilde{t} \perp u_k$.

In the article [62], it is reported that solving Equation (3.39) with an iterative solver is easier than solving Equation (3.41). Because when θ_k becomes very close to λ, the ill-conditioning of the matrix $(A - \theta_k I)$ increases and finding a good preconditioner becomes a big problem. It is also claimed that iteratively solving (3.39) yields more accurate and stable results.

A basic implementation of the algorithm is given in algorithm 4 (This algorithm is adapted from [6]). The steps $3 - 5$ represent the orthogonalization of computed correction against the

Algorithm 4 The basic algorithm for Jacobi-Davidson Method

Require: The starting vector v_0 and the error bound ϵ is determined by the user.

1: Start with $t = v_0$
2: **for** m= $1, 2, \ldots$ **do**
3: **for** j= $1, \ldots, m-1$ **do**
4: $t = t - (q_i^* t) q_i$ → Orthogonalize the correction
5: **end for**
6: $q_m = t/||t||_2, q_m^A = A q_m$
7: **for** i=$1, \ldots, m$ **do**
8: $M_{i,m} = q_i^* q_m^A$ → Compute the projection matrix of A
9: **end for**
10: Compute the largest eigen pair (θ, s) of M
11: u=Qs
12: $u^A = q^A s$
13: $r = u^A - \theta u$ → The residual is computed
14: **if** $||r||_2 \leq \epsilon$ $\tilde{\lambda} = \theta, \tilde{x} = u$ **then**
15: stop → The residual is checked
16: **end if**
17: Solve approximately a $t \perp u$ from
18: $(I - qq^*)(A - \theta I)(I - qq^*)t = -r$ → The corrector equation
19: **end for**

basis vectors and in steps $7 - 9$ the projection matrix of A is computed. At step 13, the residual is calculated. If residual is below some user prescribed ϵ value, then Ritz values are accepted as the approximated eigenvalues. If convergence is not reached, at 18-th step Jacobi-Davidson

correction equation is solved by an iterative solver. In other words, a new update is calculated.

As m increases the storage and computational cost also increases like the situation in Arnoldi's method. An efficient way to struggle this difficulty is to restart the algorithm with the most recent Ritz vector obtained. However, using only one eigenvector would cause a lot of loss of information and sometimes results in stagnation. In order to prevent this problem, some set of vectors should be taken. This set should contain more information about the wanted eigen pair. For example, some Ritz vectors close enough to desired eigenvalue would be an appropriate choice [6, 62].

All the process described up to this part were for computing the largest eigenvalue with its corresponding eigenvector of the matrix (generally for largest eigenvalues). But it is also possible to compute more than one eigen pair and also other parts of the spectrum with JD method. A common procedure, deflation, is used in order to calculate other eigenvalues. Let $\{v_1, \ldots, v_k\}$ be the accepted eigenvector approximations for the matrix A and $V_k = (v_1, \ldots, v_k)$ be the matrix with columns consisting of these approximate eigenvectors. Then to compute the (k+1)th eigenvector, the Algorithm 4 is applied to the deflated matrix

$$B = (I - V_k V_k^*)A(I - V_k V_k^*).$$

Then Equation (3.38) becomes

$$P_m(I - V_k V_k^*)(A - \theta_m I)(I - V_k V_k^*)P_m t_j = -r_j,$$

where $P_m = (I - q_j q_j^*)$. In order to solve this system, a slightly adapted version of the procedure described before should be applied.

The theory developed until here is only applicable when the external eigenvalues of the matrix are sought. Computation of interior eigenvalues needs a much more detailed analysis. This is due to the unclear behavior of the Ritz values for interior parts of the spectrum, where the convergence behavior of Ritz values for interior eigenvalues are not regular. For handling with this difficulty, it was suggested in [62] to use harmonic Ritz values instead of Ritz values for finding interior eigenvalues.

Definition 3.9 *If $\mathcal{V}_k$ is a linear subspace of $\mathbb{C}^n$ then $\tilde{\theta}_k$ is called a harmonic Ritz value of A with respect to some linear subspace W_k if $\tilde{\theta}_k^{-1}$ is a Ritz value of A^{-1} with respect to the subspace W_k [20].*

With harmonic Ritz values more regular convergence behavior is expected for the eigenvalues closest to origin in case of normal matrices. From the above definition, the requirement for calculating A^{-1} seems to be a disadvantage for using harmonic Ritz values. Fortunately, the following theorem shows this computation is no longer needed.

Theorem 3.10 *[62] Let V_k be some k dimensional subspace with basis vectors $\{v_1, \ldots, v_k\}$. A value $\tilde{\theta}_k$ is a harmonic Ritz value of A with respect to the subspace $W_k = AV_k$ if and only if*

$$(A\tilde{u}_k - \tilde{\theta}_k \tilde{u}_k) \perp AV_k \tag{3.42}$$

for some $\tilde{u}_k \in V_k$, $\tilde{v}_k \neq 0$. If $\{w_1, \ldots, w_k\}$ span AV_k. Let $V_k := [v_1|\ldots|v_k]$ and $W_k = [w_1|\ldots|w_k]$ and $\tilde{H}_k := (W_k^* V_k)^{-1} W_k^* A V_k$. Condition (3.42) is equivalent to $\tilde{H}_k s = \tilde{\theta}_k s$ for some s.

The above theorem implies that the harmonic Ritz values of a matrix can be computed without calculating the inverse of the matrix. Therefore it is enough to calculate the matrix $\tilde{H} := (W_k^* V_k)^{-1} W_k^* A V_k$ and find its eigenvalues. In the context of Jacobi-Davidson method $\tilde{H}_k$ can be obtained by either constructing an orthogonal basis for AV_k or constructing a bi-orthogonal basis for V_k and AV_k [62]. Constructing an orthogonal basis is more preferable due to the stability reasons. The construction schemes of orthogonal and biorthogonal bases are explained in [62] with a report that using an orthogonal basis in constructing $\tilde{H}$ is more advantageous in terms of stability.

Equation 3.39 implies that, the Jacobi–Davidson method requires to solve a linear system of equations. This part of the algorithm is referred as the inner system and the other part is referred as outer system. It was reported in [62] that the solution of the linear system do not need to be very accurate. Notay, in his article [52], performed an analysis of the connection between the accuracy of the solution of the inner and outer systems and obtained similar results with [62]. He analyzed the case for positive definite matrices and used the conjugate gradient method *CG* to solve the inner system. Whenever A is positive definite the matrix $(I - q_k q_k^*)(A - \theta_k I)(I - q_k q_k^*)$ in Equation 3.39 and its preconditioner, $(I - q_k q_k^*)K(I - q_k q_k^*)$, are proved to be positive definite. He also found a relation between the residual, r_k, in Equation 3.39 and inner iteration. This result lets the user to have an idea about the convergence of outer iterations while inner iterations are still being running. More information about this procedure can be found in [52].

Jacobi–Davidson method is an example of the accelerated Newton scheme [30, 61]. Consider $\tilde{u}$ is a vector with nontrivial component in the direction of desired eigenvector q and u_k is the approximation of q such that $(u_k, \tilde{u}) = (q, \tilde{u}) = 1$. Here, θ_k is chosen so that the residual is orthogonal to another vector w. Then the map defined by

$$F(q) = 0 \quad \text{where} \quad F(u) = Au - Qu \quad \text{with} \quad \theta = \theta(u) = \frac{w^* A u_k}{w^* u}$$

is the nonlinear representation of eigenvalue problem.

If u_k approximates the eigenvector q, the next Newton approximate u_{k+1} would be calculated by

$$u_{k+1} = u_k + t \quad \text{where} \quad t \perp \tilde{u} \quad \text{and} \quad \left(\left. \frac{\partial F}{\partial u} \right|_{u_k} \right) t = -r_k$$

and the Jacobian of F is equal to

$$\left(\left. \frac{\partial F}{\partial u} \right|_{u_k} \right) = \left(I - \frac{u_k w^*}{w^* u_k} \right) (A - \theta_k).$$

Hence, the correction equation of Newton step would be

$$t \perp \tilde{u} \quad \text{and} \quad \left(I - \frac{u_k w^*}{w^* u_k} \right) (A - \theta_k) = -r_k.$$

If $\tilde{u} = u_k$ and $w = u_k$, then Newton correction equation will be the same as Jacobi–Davidson method [61]. For more details about accelerated Newton schemes and relations with other methods, the reader is referred to [30].

CHAPTER 4

SPARSE EIGENVALUE PACKAGES

In this Chapter, we will give description of programs and software packages of the eigenvalue solvers mentioned in Chapter 3. Because the solution of linear systems of equations and eigenvalue problems have wide range of applications in science and engineering, there exists many efficient solvers. Computation of eigenvalues is more difficult than solving linear systems of equations. Most of the methods for solving sparse eigenvalue problems described in Chapter 3 were developed with the last fifteen years. Parallel to the development of new methods, several eigenvalues packages were developed, which can be used in applications. A detailed survey of current sparse eigenvalue freely available packages on the Internet can be found in [40]. Most of the programs are in MATLAB, C, C++ and FORTRAN. The programs are written in MATLAB are SPEIG, AHBEIGS, IRBLEIGS, EIGIFP, LANEIG, JDQR, JDCG and in C++, the SLEPc library. MATLAB, which become a standard language in scientific computing, is a numerical computing environment used both in industry and academia. On the other hand, C++, which is very popular among both industry and academia, is a general purpose object oriented programming language. All eigenvalue solvers implemented in MATLAB are designed for serial computation but, SLEPc library can also be used for parallel computations.

ARPACK is one of the most popular eigen solvers [48], due to its efficiency and robustness. It is written in FORTRAN but a C++ interface, ARPACK++, is also available. ARPACK implements the implicitly restarted Arnoldi(IRA) method (Section 3.2) for both symmetric and non-symmetric problems. In the symmetric case, Lanczos algorithm is used with full re-orthogonalization is used instead of Arnoldi (Section 3.3). In addition it can be used for both standard and generalized problems in both real and complex arithmetic .

In most MATLAB implementations, the user can pass a MATLAB matrix as well as to specify a MATLAB function for the matrix-vector product. Packages like ARPACK are implementing reverse communication, which is independent of the matrix representation and very flexible.

4.1 SPEIG

SPEIG is the MATLAB implementation of the implicitly restarted Arnoldi(IRA) algorithm described in Section 3.1.1. The solver SPEIG is developed by Sorensen and Radke in [57] as an alternative for the *eig* command of MATLAB. When the *eig* command of MATLAB is called for a sparse matrix, *eig* internally calls SPEIG to calculate the eigenvalues.

The basic syntax of SPEIG is $>>\ d\ =\ speig(A)$ where d is a vector with entries consisting of desired eigenvalues of the matrix. If eigenvalues are also needed the syntax is $>>\ [D, V]\ =\ speig(A)$ so D is the diagonal matrix with eigenvalues on diagonal entries and V is the matrix with columns as eigenvectors. SPEIG lets the user to change the default value of the number of eigenvalues calculated by $>>\ d\ =\ speig(A, k)$ where k represents the desired number of eigenvalues. SPEIG is designed for calculating largest eigenvalues with default parameters. But, the user also has chance to calculate other parts of the spectrum with $>>\ d\ =\ speig(A, k, \sigma)$, where σ is the value around which the user wants to calculate the eigenvalues. Moreover, σ has special characters for some parts of the spectrum:

Table 4.1: Character strings for exterior parts of spectrum

'LM'	Largest magnitude
'SM'	Smallest magnitude
'LR'	Largest real part
'SR'	Smallest real part
'BE'	Both ends

The other parameters of SPEIG can be changed by using two functions: *speigget* and *speigset*.

speigset is designed to change the default values of parameters:

$$>>\ opts\ =\ speigset(\text{'}name1\text{'}, \text{'}value1\text{'}, \text{'}name2\text{'}, \text{'}value2\text{'}, \ldots)$$

where name1 is the name of the parameter and value1 is the value desired to assign. If only *speigset* is used, the variable names with their types are displayed.

speigget is designed to extract the parameter values that are created by *speigset*:

$$>> \ p = speigget(opts, `name1')$$

gives the value of parameter name1. If only *speigget* is used, the variable names and their values are displayed.

The parameters and their default values used in SPEIG are given in the Table 4.2. If the

Table 4.2: Parameters of SPEIG

parameter	description	default value
n	dimension of the problem	none
p	dimension of Arnoldi basis	$2k$
tol	Tolerance for convergence of $\frac{\|AV-VD\|}{\|A\|}$	A sym 10^{-10}, A nonsym 10^{-6}
maxit	maximum number of Arnoldi iterations	300
issym	positive if A is symmetric 0 otherwise	0
dopoly	Positive if it specifies a matrix vector product, σ is LR, SR or numeric and polynomial interpolation is to be used to convergence acceleration	0
gui		0
v_0	starting vector	rand(n,1)-0.5

convergence is not reached for the given number of iterations, SPEIG provides the computed result for the eigenvalues and eigenvector with a warning message that maximum number of iterations are exceeded. The parameter *dopoly* indicates the activation or deactivation of polynomial acceleration. If the parameter *gui* set to a negative number, a convergence report is displayed, containing the residual norm at each step, the current iteration number, the elapsed time and the eigenvalues are plotted. Additionally, the general properties of the problems such as size of matrix, number of requested eigenvalues, dimension of the Arnoldi basis, tolerance and maximum number of iterations are displayed.

4.2 AHBEIGS

AHBEIGS is the MATLAB implementation of augmented block Householder Arnoldi method [2] described in Section 3.1.1, with some modifications on the number of the restarting vectors. The codes are available at $http://www.math.uri.edu/\ jbaglama/\#Software$. In theory

the number of restarting vectors are taken to be j and in the package they are implemented as $j + 3$. The number of computed eigenvalues are also determined internally by the algorithm, which can also be larger than the desired values.

The basic syntax of AHBEIGS >> $d = ahbeigs(A)$ returns a vector d with the entries consisting of largest eigenvalues of the matrix. >> $[X, D] = ahbeigs(A)$ where X is a matrix with columns eigenvectors of A and D is the diagonal matrix with eigenvalues in the diagonal entries. The other parameters of the algorithm can be changed by using options structure: >> $options.parametername = parametervalue$.

In Table 4.3 the parameters used in the algorithm are described. *blsz* corresponds to the

Table 4.3: Parameters of AHBEIGS

parameter	description	default value
blsz	block size of Hessenberg matrix	3
nbls	number of blocks in Hessenberg matrix	6
k	number of eigenvalues	6
sigma	desired portion of spectrum	LM
tol	tolerance for convergence	1d-6
V_0	starting matrix	randn
maxit	maximum number of restarts	100
dispr	sets history	0

number of columns of starting matrix (i.e. it is the r value in the formula 3.13). *nbls* is the m value in the formulas (example 3.13), it is increased automatically by the solver if required. The value of *sigma* determines which portion of the spectrum is being calculated, which can take a numerical value or a character. When it is a numerical value, eigenvalues around that value are calculated. The exterior eigenvalues on the spectrum can be calculated with the same special characters in SPEIG in Table 4.1. The value of *tol* determines the convergence. Convergence is controlled by the inequality $\|A_\lambda - \lambda x\|_2 \leq tol\, max(abs(Ritz))$ where Ritz represent the Ritz value of upper block Hessenberg matrix. When the parameter *disp* is set to a positive number, the k Ritz values and related residuals are displayed.

4.3 IRBLEIGS

The package is an implementation of implicitly restarted block Lanczos algorithm described in section 3.2. It was developed in [5] as an extension of the implicitly restarted Lanczos algo-

rithm described in Section 3.2. A detailed description of the method package can be found in [3, 4]. The MATLAB code is available at $http://www.math.uri.edu/\ jbaglama/\#Software$

The basic syntax of IRBLEIGS is >> $d = irbleigs(A)$ where d is a vector with entries consisting of eigenvalues. If eigenvectors are also needed >> $[D, V] = irbleigs(A)$ has to be used, where V is a matrix consisting of eigenvectors and D is a diagonal matrix with eigenvalues on diagonal entries is used. The package calculates largest eigenvalues but it is possible to change this choice. This can be done by >> $[D, V] = irbleigs(A, opts)$ where *opts* is a structure that performs the desired changes in parameters. One can also change the parameter values from the m–file of IRBLEIGS. But this method is not very practical. The syntax for using *opts* is: >> $opts.parametername = parametervalue$ or
$opts = struct('parametername','parametervalue')$. The parameter used in IRBLEIGS are given in Table 4.4.

The solver IRBLEIGS uses Leja points as shifts for the implicit restarts. The letters ML and WL are used for *zertyp* and they refer to the types of the zeros of the Leja polynomials. ML refers to the mapped Leja points and WL refers to the fast Leja points. It should be noted that choosing ML for numerical values of σ is not applicable in the code. The errors for each iteration can also be displayed by setting the parameter *dispr* greater than zero. *Irbleigs* also provides computation history with >> $[D, V, Exitinfo] = irbleigs(A, opts)$. Here, *exitinfo* is a 2-dimensional vector. The first entry of that vector contains the information about accuracy of computed eigenvalues. If exitinfo(1)=0, the desired accuracy is achieved. If exitinfo(1)=1, desired accuracy is not reached in the given number of iterations. When this happens, one choice is to increase the iteration number or use the computed matrix V containing the approximate eigenvectors as an initial matrix and restart the code. Second array of *exitinfo* contains the information about total matrix vector product.

When the code can not determine the k-eigenvalues for given number of iterations for the desired accuracy, the outputs D and V are empty arrays.

4.4 LANEIG

LANEIG is the eigenvalue solver of the PROPACK package [46] which was developed for solving SVD and eigenvalue problems for large sparse matrices. Each routine is implemented

Table 4.4: Parameters of IRBLEIGS

parameter	description	default value
k	number of desired eigenvalues	3
nbls	number of steps of the block-Lanczos method between restarts at the begin of the computation	3
tol	tolerance	10^{-6}
maxit	maximum number of restarts of the block-Lanczos method	100
sigma	type of the desired eigenvalues	LE
blsz	block size of block-Lanczos method	3
	σ is LR, SR or numeric	
	polynomial interpolation to be used to accelerate the convergence	0
eigvec	matrix which the user wants the eigenvectors converge	[,]
v_0	matrix of blsz orthonormal starting vectors	randn(n,blsz)
maxdpol	maximum degree of acceleration polynomial	n if σ is numerical 200 otherwise
zertyp	determines how the zeros of acceleration polynomial are chosen	ML if σ =SE or LE WL if σ is numerical
endpt	determines the strategy for choosing the endpoints of interval K.	FLT if σ is numerical MON if σ is SE or LE
sizint	the interval size	1

separately, so that they can be used alone. The routines in the package depends mainly on Lanczos methods. For example, the SVD solver implements Lanczos bi-orthogonalization with partial re-orthogonalization. LANEIG [60] is the implementation of Lanczos algorithm with partial re-orthogonalization (LPRO) described Section 3.3. The MATLAB codes can be obtained from the following link $http://soi.stanford.edu/\ rmunk/PROPACK/$

The basic syntax of LANEIG is >> $d = laneig(A)$ where d is the vector consisting of smallest eigenvalues of the matrix. If the eigenvectors are also required, one has to use >> $[X, D] = laneig(A)$ where X is the matrix with eigenvectors are columns and D is the diagonal matrix with eigenvalues on diagonal entries. It is also possible to change the number of eigenvalues and also the portion of spectrum desired to compute by using >> $d = laneig(A, k, sigma)$, where k represents the number of eigenvalues and *sigma* represents the portion of spectrum desired to compute. It can be either a number or character string: The parameters of the algo-

Table 4.5: Character strings for computing exterior eigenvalues with laneig

'AL'	algebraically largest
'AS'	algebraically smallest
'LM'	largest in magnitude
'BE'	at both ends

rithm can be changed with the option >> *options.parametername = parametervalue*. These changes are assigned to solver by the command line: >> $[X, D] = laneig(A, k, sigma, options)$

In Table 4.6 the values of the parameters are given.

Table 4.6: Parameters of LANEIG

Parameter	Description	Default value
k	Number of eigenvalues	5
tol	Convergence tolerance	16*eps
lanmax	Maximum dim. Lanczos subspace	
v_0	Starting vector	rand(n,1)-0.5
delta	Level of orthogonality among Lanczos vectors	sqrt(eps/k)
eta	Level of orthogonality after reorthogonalization	$10 * eps^{3/4}$

4.5 EIGIFP

EIGIFP [49] is the implementation of the inverse free preconditioned Krylov subspace method [36]. It can be found from the web page: $http://www.ms.uky.edu/\ qye/software.html$.

The basic syntax of EIGIFP is >> $d = eigifp(A, B)$ Here, d is the smallest eigenvalue of the matrix and the corresponding eigenvector can be computed by >> $[D, V] = eigifp(A, B)$ where V is a $n \times 1$ vector. It is also possible to compute more than one eigenvalues with corresponding eigenvectors using the command: >> $[D, V] = eigifp(A, B, k)$ where D is a diagonal matrix containing the eigenvalues and V is the matrix with columns as corresponding eigenvectors. EIGIFP is designed to compute the smallest eigenvalues of the matrix. But it is also possible to compute the largest eigenvalue by assigning $-A$ instead of A. Some of the parameters of the algorithm can also be changed to improve the efficiency of the package. The syntax for changing the default values of parameters is >> *opts.parametername = parametervalue* and it is implemented in the code as >> $[D, V] = eigifp(A, B, opts)$.

Table 4.7 gives a description of parameter values. The parameter *inneriteration* determines

Table 4.7: Parameters of EIGIFP

Parameter	description	default value
size	dimension of matrix A	none
tolerance	termination threshold	
maxiterations	maximal number of outer iterations	500
inneriteration	sets number of inner iteration	
useprecon	allows preconditioning or not	
iluthresh	threshold value for the incomplete $LDL^\top$	10^{-3}
preconditioner	matrix given by the user for preconditioning	

the dimension of the Krylov subspace constructed by inner iterations. Setting *inneriteration* to a large number sometimes causes out of memory warnings. Setting *useprecon* to NO disables the preconditioning. The parameter *iluthresh* determines the degree of factorization in LU decompositions. Setting it to 0 results in a full(exact) LU factorization and setting it to 1 results in incomplete LU factorization.

4.6 JDQR

The Jacobi-Davidson method in Section 3.5 is implemented in the solver JDQR. The MATLAB codes can be obtained from $http://www.math.uu.nl/people/sleijpen/$.

The basic syntax of JDQR is $>>\ d = jdqr(A)$, where d is a vector with entries equal to five largest eigenvalues of the matrix A. $>>\ [X, J] = jdqr(A)$ returns the normalized eigenvectors of A in columns of X and the Jordan form of A is given by the matrix J. The eigenvalues are diagonal entries of J. It is also possible to visualize the Schur form of A with JDQR: $>>\ [X, J, Q, S] = jdqr(A)$. Q is $n\times5$ orthonormal matrix and S is 5×5 upper triangular matrix. The diagonal of S contains the eigenvalues of A and $X = Q+Y$, where Y is the eigenvectors of the pair S: $S\ Y = Y\ J$. The other parts of the spectrum can also be calculated. This can be done with the command line: $>>\ d = jdqr(A, k, \sigma)$, where k represents the number of eigenvalues and σ determines the part of the spectrum desired to compute. The exterior eigenvalues can be computed with the special characters defined on Table 4.1. Moreover, some parameters of the algorithm can also be changed very practically by using different options. These changes are generally done to increase the efficiency of algorithm. There are two ways to change by

options: >> $options = struct('parametername','parametervalue')$ or
$options.parametername = parametervalue$. Then they are assigned to the code by
>> $d = jdqr(A, options)$

The parameters that can be changed by optional structure are given in Table 4.8. The parame-

Table 4.8: Parameters of JDQR

Parameter	description	default value
jmin	minimum dimension of V	$k+5$
jmax	maximum dimension of V	jmin+k
maxit	maximum number of matrix vector products	5000
tol	convergence tolerance for	10^{-8}
maxit	maximum number of restarts of the block-Lanczos method	100
v_0	starting vector	
disp	provides information about the procedure	0
testspace	defines the test subspace W.	standard
LSolver	linear solver	GMRES
LS-MaxIt	maximum iteration number carried in linear solver	5
precond	preconditioner	identity matrix
Schur	gives Schur decomposition	no
Disp	information about convergence history	0
AvoidStag	precaution to prevent stagnation	no

ter *testspace* determines the type of the Ritz values used during calculation. If it is assigned to "standard", standard Ritz values are chosen and if it is assigned to "harmonic", Harmonic Ritz values are chosen. Interior eigenvalues are more accurately computed if *testspace* assigned to harmonic. *LSolver* gives the opportunity to choose the linear solver used for correction Equation (refjacdav2). Linear solvers that can be used are FGMRES, CG, MINRES, SYMMLQ. Among them, CG, MINRES, SYMMLQ requires positive definite preconditioners. The default value for *disp* is "no". But when it is changed to "1" or "yes", the convergence history is plotted and residual size at each step is displayed. Moreover if it is set to "2", approximate eigenvalues are plotted at each iteration step. Sometimes, the angle between the search space and recently computed eigenvectors may decrease. This means these new vectors are close to the search space. After some iteration, this will lead to stagnation. The parameter *Avoidstag* is designed to prevent this problem. If it is set to "yes", JDQR projects on whole space in correction equation.

4.7 JDCG

The solver JDCG [52] implements the Jacobi-Davidson method with conjugate gradient algorithm described in Section 3.6. The MATLAB algorithm can be found at
$http://homepages.ulb.ac.be/\ ynotay/$.

The basic syntax of JDCG is $>> d = jdcg(A)$ where d is a 1×5 row vector consisting of smallest eigenvalues of the matrix. $>> [X, D] = jdcg(A)$ returns two matrices X and D where X is the matrix with eigenvectors are columns and D is the diagonal matrix with eigenvalues diagonal entries. It is also possible to change the other input parameters. The values of parameters are changed with options $>> options.parametername = parametervalue$. These changes are assigned to solver by the command line: $>> [X, D] = jdcg(A, k, sigma, options)$ where k represents the number of eigenvalues and *sigma* represents the scalar shift taken as a lower bound for the smallest eigenvalue.

Table 4.9 shows the parameters with default values. The solver defines no preconditioner

Table 4.9: Parameters of JDCG

parameter	description	default value
tol	tolerance	1e-8
jmin	minimum dim. of the search subspace	7
jmax	maximum dim of the search subspace	14
maxit	maximum iteration number	5000
V_0	starting vector	ones
disp	visualizes history	0
precond	preconditioner	No

internally. In other words, the preconditioner matrix must be defined by the user. When the parameter *disp* is set to 1, the input parameters, residual size for each step are displayed with a plot of convergence history. It is also possible to see the details of the computation procedure directly by the command line: $>> [X, D, History] = jdcg(A)$. The last argument "history" has 5 columns. First column is the index of the eigen pair, second column is the number of completed JD iterations, third column is the total multiplications with the matrix A, fourth is the residual and the last column keeps the information about approximations of the searched eigenvalue.

4.8 SLEPc

SLEPc is a package developed for solving large scaled sparse eigenvalue problems. The name stands for Scalable Library for Eigenvalue Problem Computation. It is built on PETSc(Portable, Extensible Toolkit for Scientific Computing). Based on PETSc, SLEPc avoids software complications such as matrix definitions. In other words, it is able to interoperate with existing softwares and can handle different storage formats (depending on PETSc). The software is written in C but a Fortran interface is also available. It can be used to solve the standard and generalized eigenvalue problems both in complex and real arithmetic. In addition, some routines for singular value decomposition problems are also available [38]. SLEPc is designed to deal with large sparse problems. Therefore its emphasis is on projection methods such as subspace iteration, Arnoldi, Lanczos and Krylov–Schur methods. Moreover, access to external libraries is also possible. The software also gives the user the opportunity to apply different spectral transformations. Some properties will be explained here and more information can be found in [40, 37]. The software can be downloaded from $http://www.grycap.upv.es/slepc/$. Other documents and information for installing the package also can be found from this web site.

SLEPc has two main objects EPS and ST. EPS stands for eigenvalue problem solver. It is used to specify the problem and provides the access to eigen problem solvers. In addition, it manages the modifications in default parameters such as the number of eigenvalues(nev) or the value of tolerance. ST stands for spectral transformations. The package gives the user the chance to apply the methods with or without spectral transformations (accelerations). Different types of problems such as (generalized) Hermitian or (generalized) non Hermitian matrices are supported. The default problem type is nonhermitian. If a Hermitian matrix is used, this should be specified because the software exploits the advantages of symmetry while applying the methods.

Available methods on SLEPc are power method with deflation, subspace iteration, Arnoldi method with explicit restarts, Lanczos algorithm and Krylov-Schur method. Moreover external libraries such as LAPACK, ARPACK, TRLAN can also be used. The default method is Krylov-Schur. All solvers are designed to find the largest eigenvalue. But it is possible to change this option and find other exterior values or even interior eigenvalues. For determining interior eigenvalues, the software has two different options: harmonic extraction and spectral

transformations. Harmonic extraction uses a target value κ around which the user wants to find the eigenvalues. Spectral transformations is used for two reasons: to compute the interior eigenvalues or to accelerate the convergence. The available transformations are shift of origin, spectrum folding, shift and invert and Cayley transformations. Shift of origin means shifting the matrix with the given σ value. In spectrum folding, the matrix is shifted and then its square is taken. Cayley transformations applies a shift and an anti shift to the problem. The shift and the anti-shift are taken equal for Cayley transformation by default.

The methods are designed to find only one eigenvalue. Although it is possible to increase this number, one should be careful while applying this option. Because changing nev affects the dimension of the working subspace(ncv). The relation between these two values (nev and ncv) are valid for all methods and ncv must equal to max{2nev,nev+15}. This application is reasonable if small portions of spectrum are being searched. When large number of eigenvalues are required, changing the ncv value with respect to the relation results becomes meaningless. If this is done, then the program would have to store a large dimensional dense matrix. This brings a lot of problems in terms of storage and a high computational cost. In the latest version of SLEPc, a parameter, mpd, is introduced to solve this problem. The parameter mpd stands for maximum projected dimension and provides bounds for the size of the problem [38]. But there is no specified relation between mpd and any other parameters. The values for mpd should be assigned intuitively.

The errors are controlled by the residual, $r = A\tilde{x} - \tilde{\lambda}\tilde{x}$. For the Hermitian case, the 2-norm of the residual is used as a bound for the eigenvalue problem. The default value for the error estimate bound, tol, is $10e-7$. This bound is used to decide whether the approximate eigenvalues are converged or not and it can be changed depending on the problem.

It was mentioned above that SLEPc has objects, EPS and ST, to make the necessary changes on default values. These objects contain functions to apply the changes in the prescribed commands. But it is also possible to apply the changes during run time from the command line. Some example commands with descriptions for this type of calling sequence are given below:

./ex4 -file *matrix* **-eps_nev** 50 **-eps_ncv** 100 **-eps_hermitian**

-eps_type lanczos -eps_tol $10e-5$ **-st_type cayley -st_shift** 1

This command tells to the package to find 50 eigenvalues around 1 of the Hermitian matrix

with the tolerance $10e-5$ using the Lanczos method with Cayley transformation. The command at the beginning $./ex4$ presents one of the examples in SLEPc. This example takes a matrix from an exterior address and then finds its eigenvalues. When the matrix of interest is already available, it is advantageous to use the following command:

./ex4 -file *matrix* **-eps_nev** 40 **-eps_ncv** 80 **-eps_hermitian**
-eps_type -eps_krylovschur -eps_harmonic -eps_target 1

This command tells to the package to find 40 eigenvalues around 1 of the Hermitian matrix by Krylov–Schur method using harmonic extraction.

./ex4 -file *matrix* **-eps_nev** 5000 **-eps_mpd** 450 **-eps_hermitian**
-eps_type -eps_arnoldi -eps_smallest_magnitude

This tells to the package to find 5000 smallest eigenvalues of the hermitian matrix by Arnoldi method. It should be noticed that the parameter *mpd* is used instead of *ncv* in this command. The reason is that the number of requested eigenvalues is quite high.

4.9 ANAZASI

ANASAZI is a package within the Trilinos project, a parallel object-oriented software framework containing many packages. Trilinos consists of different packages each of which are designed to solve different numerical analysis problems. The latest version of software can be downloaded from $http://trilinos.sandia.gov/download/trilinos-10.2.html$.

ANASAZI being a part of TRILINOS is designed to solve the generalized eigenvalue problem $Ax = \lambda Mx$ for both Hermitian and non Hermitian cases (by simply taking $M = I$, it can be used to solve the eigenvalue problem $Ax = \lambda x$). It implements the block extension of Krylov–Schur method defined in [70], block Davidson method described in [55] and an implementation of LOBPCG as described in [41]. It is implemented in ANSI C++ language. The package consists of 6 basic classes. Each class is dedicated to perform a specific job for computing eigenvalues. For example, the solver class *Eigenproblem* provides to access to the components of the problems such as matrices, vectors, eigenvalue approximations, preconditioners. The other class *Eigensolver* is designed to keep the basic iterations of the implemented algorithms. However, it does not include the intelligence part to determine where to stop the

iteration, what the eigenvalues of interest are, which output to send and to where or how to orthogonalize the basis for subspaces [7]. Instead, there are four classes to perform these tasks: *StatusTest, SortManager, OrthoManager* and *OutputManager*. This property provides an advantage for users and developers to organize the problems in their desired prescribed way. For instance, one can select the stopping criteria at runtime. More information about ANASAZI can be found in [7, 59].

CHAPTER 5

NUMERICAL RESULTS

5.1 Performance of Eigensolvers

In this section, examples of complex networks are provided in order to illustrate the performance of eigensolvers described in Chapter 4. The solvers are compared in terms of accuracy and computing time. The eigensolvers SPEIG, AHBEIGS, IRBLEIGS, EIGIFP, LANEIG, JDQR and JDCG are written in MATLAB whereas SLEPc is in $C++$. The focus of this study is on symmetric matrices but it should be kept in mind that the codes SPEIG, AHBEIGS, JDQR and the library SLEPc can be applied to the unsymmetric matrices. In addition, the solvers EIGIFP, JDQR and JDCG are developed and designed to handle problems with preconditioning and they are quite effective when a good preconditioner is available [49, 52, 62]. However, we use them here without preconditioning. Moreover it has already been mentioned in previous Chapter that parallel implementation is available in SLEPc but this option is not used in this study.

All eigensolvers investigated in this study are designed to compute k interior or exterior eigenvalues of matrices where k is the number of desired eigenvalues specified by the user. Our purpose in this study is to compute the whole spectrum, so we have assigned n, the size of the matrix, as the value of k. However, some solvers can not compute the whole spectrum. For instance, SPEIG is designed to compute at most $n-3$ eigenvalues. Moreover, we have to adjust some certain parameters in order to compute the whole spectrum with solvers IRBLEIGS and AHBEIGS. The maximum number of eigenvalues computed by them depends on the size of the matrix and some parameters. With all the other solvers, we were able to compute the whole spectrum with the default parameters.

5.1.1 Spectra of Paley Graphs

We will present now the numerical results about the spectra of normalized Laplacian matrices of Paley graphs and some real networks. The Paley graphs are constructed by the *matgraph* toolbox of MATLAB (available at http://www.ams.jhu.edu/ ers/matgraph/) and most real networks are taken from the *Pajek* group (http://www.cise.ufl.edu/research/sparse/matrices/). During the calculations we h used the version 2007a of MATLAB on Intel Core 2 Duo CPU with 2.66 GHz and 3.37 GB RAM, Windows XP operating system and Fedora Core 12.

For testing the accuracy of eigensolvers we have computed the normalized Laplacian matrix of Paley graphs from Chapter 2. These matrices are dense in the sense that the number of nonzero elements is about the half of the whole matrix. For a example, for a Paley graph of size 109, the number of nonzero elements is 5995. The normalized Laplacian matrix of Paley graphs have three eigenvalues computed by (2.10) (0 eigenvalue with multiplicity 1 and two other eigenvalues with multiplicity $\frac{(n-1)}{2}$). In Figures 5.1–5.8, on the y-axis the absolute values of errors between exact and approximate eigenvalues and on the x-axis the eigenvalue numbers are given. The eigenvalues are numbered starting from the largest eigenvalue to the smallest in all figures except for EIGIFP.

In Figures 5.1–5.8, the distribution of errors between exact and approximate eigenvalues of Paley graphs of size $n = 109$ and $n = 2089$ are plotted. As mentioned above, we could compute the whole spectrum with all solvers except SPEIG, AHBEIGS and IRBLEIGS. We have computed $n - 3$ eigenvalues with SPEIG. For smaller matrix, we were able to compute 97 eigenvalues with AHBEIGS and IRBLEIGS with parameters *nbls*= 53 for AHBEIGS, *nbls*= 51 for IRBLEIGS and *blsz*= 2 whereas for larger one, AHBEIGS and IRBLEIGS have calculated 2070 eigenvalues with initial parameters $nbls = 220$ and $blsz = 6$.

The eigenvalues of the normalized Laplacian matrices of Paley graphs are computed by all solvers with a high accuracy. The errors vary between 10^{-14} and 10^{-15}. However, the distributions of errors differ from solver to solver. Especially the block versions of the solvers have different error distributions than the non–block versions (see Figures 5.3 and 5.4 for block versions of algorithms).

The CPU times for calculating eigenvalues of normalized Laplacian matrices are given in Table 5.1. With SLEPc, Krylov–Schur method, it took less than one minute to compute the

Table 5.1: CPU times of Packages in MATLAB

	CPU Times in Seconds	
	n=109	n=2089
SPEIG	0.34	856.64
AHBEIGS	0.30	1309.94
IRBLEIGS	0.58	4765.94
LANEIG	0.36	290.89
EIGIFP	0.19	363.44
JDQR	1.09	6860.38
JDCG	0.22	-

whole spectrum for smaller Paley graph whereas it took 4416 seconds for larger matrix. For smaller matrices the CPU times are quite close for all packages. But as the size of matrices increase, the differences between the CPU times for solvers are much more visible (see Table 5.1).

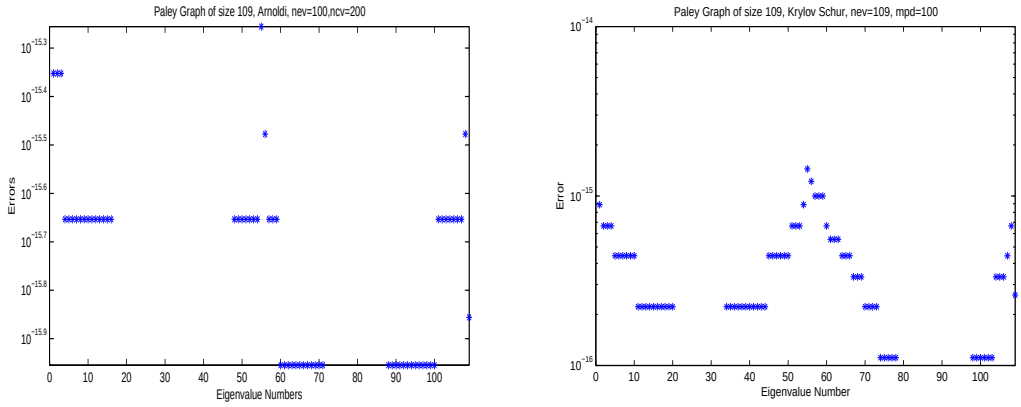


Figure 5.1: Paley graph of size 109, left: SLEPc, Arnoldi method, right: SLEPc, Krylov–Schur method

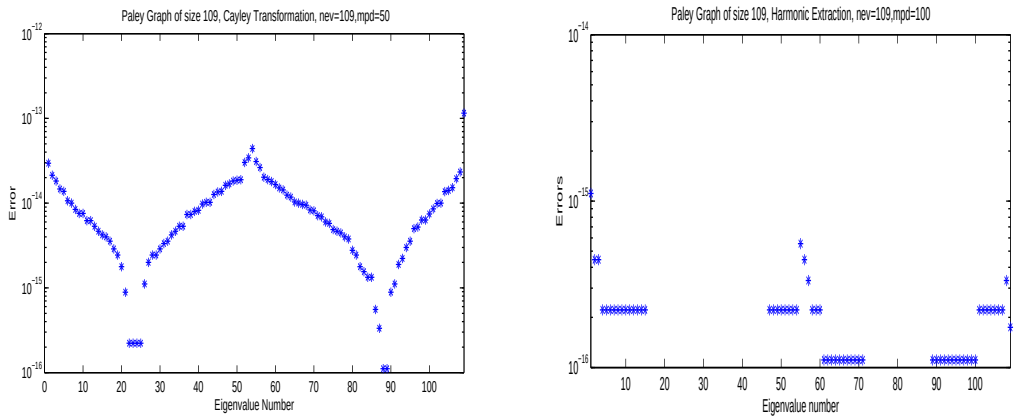


Figure 5.2: Paley graph of size 109, left: SLEPc, Krylov–Schur method right with Cayley transformation: SLEPc, Krylov–Schur method with harmonic extraction

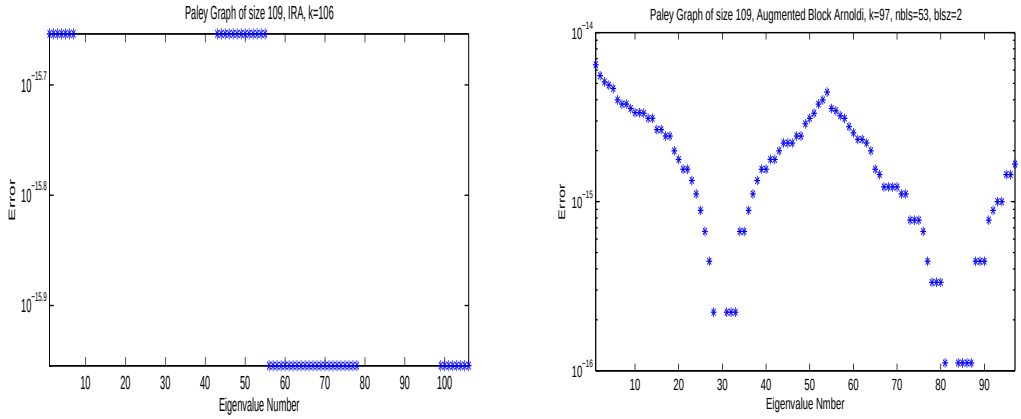


Figure 5.3: Paley graph of size 109, left: SPEIG, right: AHBEIGS

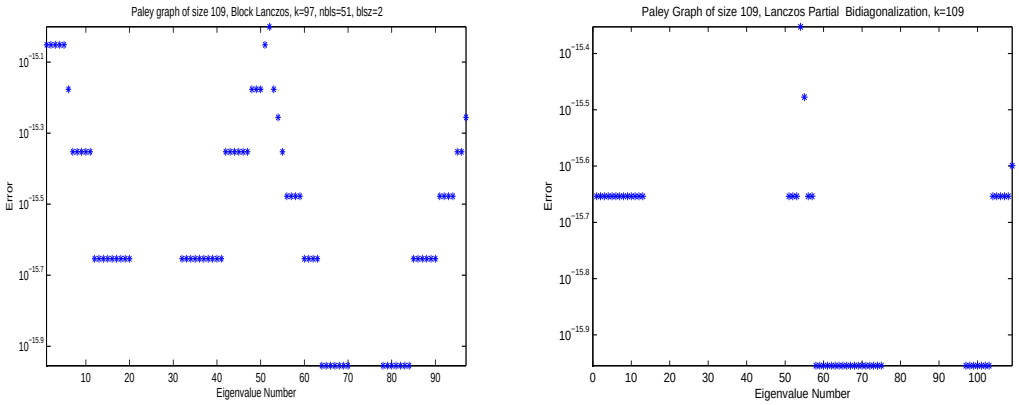


Figure 5.4: Paley graph of size 109, left: IRBLEIGS, right: LANEIG

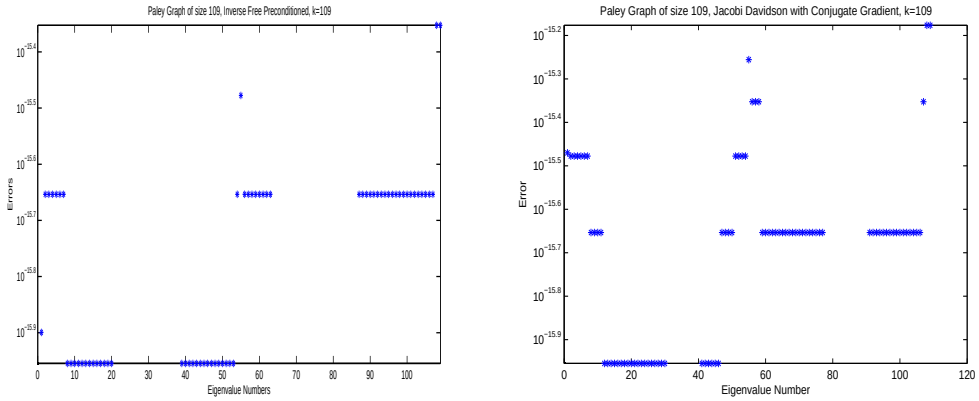


Figure 5.5: Paley graph of size 109, left: EIGIFP, right: JDCG

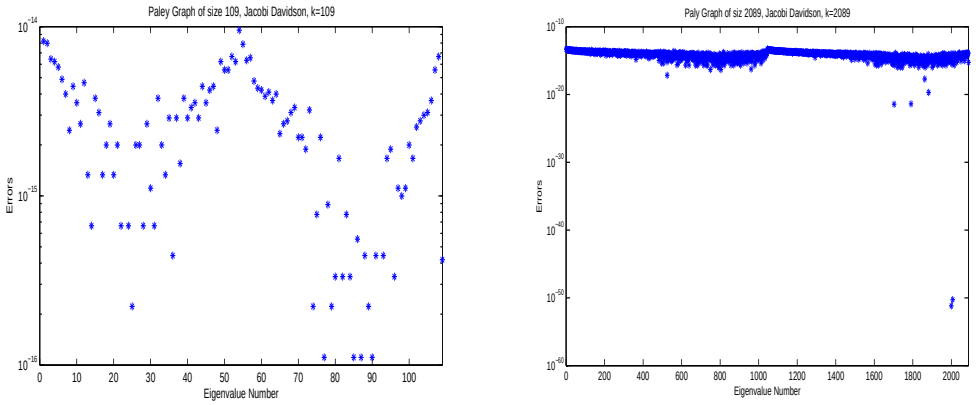


Figure 5.6: JDQR, left: Paley graph of size 109, right: Paley graph of size 2089

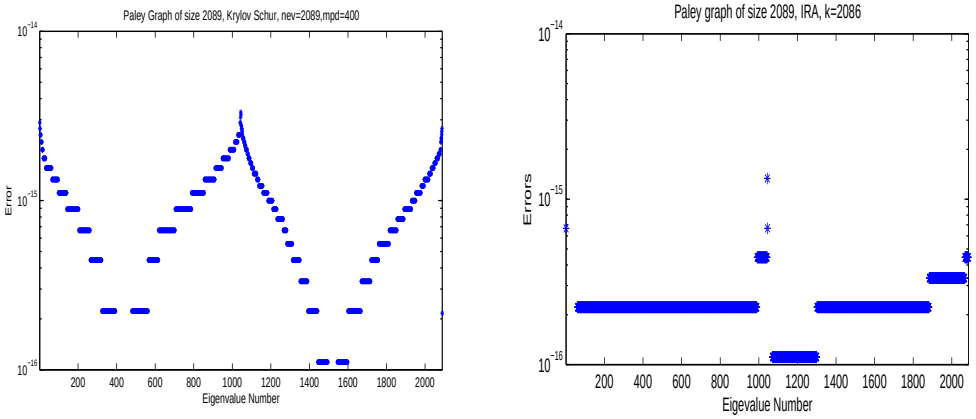


Figure 5.7: Paley graph of size 2089, left: SLEPc, Krylov–Schur method, right: SPEIG

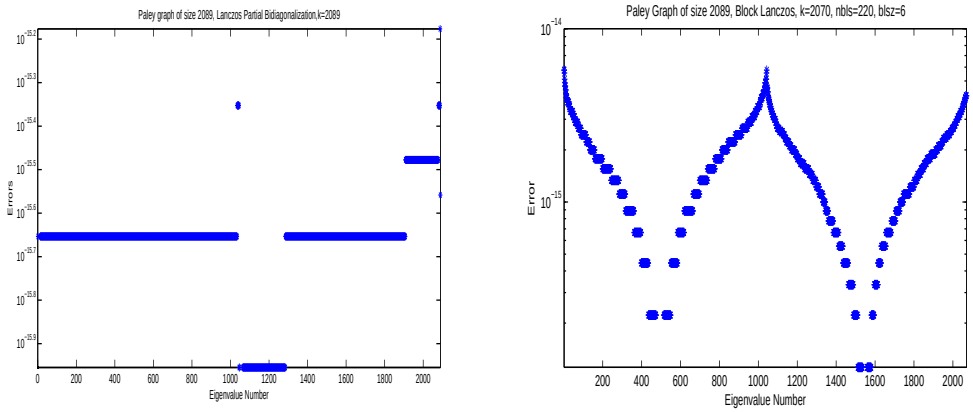


Figure 5.8: Paley graph of size 2089, left: LANEIG, right: IRBLEIGS

5.1.2 Spectra of Empirical Networks

In Figures 5.9–5.11, we have computed the eigenvalues of normalized Laplacian matrices of protein protein interaction networks (PPI). First network represents the interactions of proteins in the microorganism yeast of size 2361 and second one is the interactions in D. melanogaster of size 6900. The first network is taken from Pajek group and the latter is downloaded from Protein Protein Data Set (PPD) ($http://dip.doe-mbi.ucla.edu/dip/Download.cgi?SM=2$). The normalized Laplacian matrices of these networks are unstructured and highly sparse. The density of nonzero elements of smaller network's matrix is %0.2793 and larger networks' is %0.1013.

Since the exact eigenvalues are not known, we have compared the solvers by relative residuals. SLEPc internally calculates the relative residuals with respect to the following formulas:

$$r_\lambda = \frac{||Ax||_2}{||x||_2}, \quad \text{if } \lambda = 0 \tag{5.1}$$

$$r_\lambda = \frac{||Ax - \lambda x||_2}{||\lambda x||_2}, \quad \text{else} \tag{5.2}$$

where λ is the eigenvalue and x is the related eigenvector. In order to be consistent with the results in SLEPc, we have also used Formula 5.2 for calculating errors for other eigensolvers.

Table 5.2: The CPU times

Krylov–Schur	n=6900
without any transformation	6518 s
with Cayley transformation	4567 s
with Harmonic extraction	19459 s

For the smaller network of size 2361, we could compute 2350 eigenvalues with AHBEIGS and IRBLEIGS and 2358 eigenvalues with SPEIG. On the other hand JDQR failed to compute the whole spectrum. When k number of eigenvalues are required to compute, the package has computed less than k eigenvalues. For instance, when k is assigned as 100, JDQR computed only 33 eigenvalues. It was reported in [25, 42, 61] that JDQR are efficient for diagonally dominant matrices. The normalized Laplacian matrices of Paley graphs are diagonally dominant but those from PPI networks are not.

In Figures 5.9–5.11, the relative residuals are plotted for PPI networks of two different sizes. In the figures, the y–axis represents the relative residuals and the x–axis represents the eigenvalue numbers in decreasing order from left to right except for EIGIFP. The residuals for Krylov-Schur method has clusters around 10^{-8} and 10^{-16} for interior eigenvalues and they fluctuate between these numbers for exterior eigenvalues. On the other hand, the relative residuals for results computed by SPEIG and LANEIG are around 10^{-15} and by IRBLEIGS have clusters around 10^{-15} and 10^{-10}. The high peak in the relative errors of SPEIG, LANEIG and IRBLEIGS are due to the 0 eigenvalue or eigenvalues very close to 0. They reflected as peaks in the figures because these packages do not use the relative error formula for smaller eigenvalues.

In Figure 5.11, the residual plots belong to the calculations by SLEPc-Krylov Schur method with transformations applied to the larger matrix (of size 6900). The transformations are done with Cayley method and harmonic extraction and around 0.9999. The CPU times are given on Table 5.2.

The distribution of relative residuals in Figure 5.11 consists of two sets: exterior (eigenvalues between 1-2500th and 4500-6900th arrays) and interior (eigenvalues between 2500th and 4500th arrays) eigenvalues. In this figure, we can see that interior eigenvalues are computed more accurately (with errors around 10^{-15}) than exterior eigenvalues (with errors around 10^{-8})

when the transformations are applied. All the other solvers failed to compute the whole spectrum for this matrix.

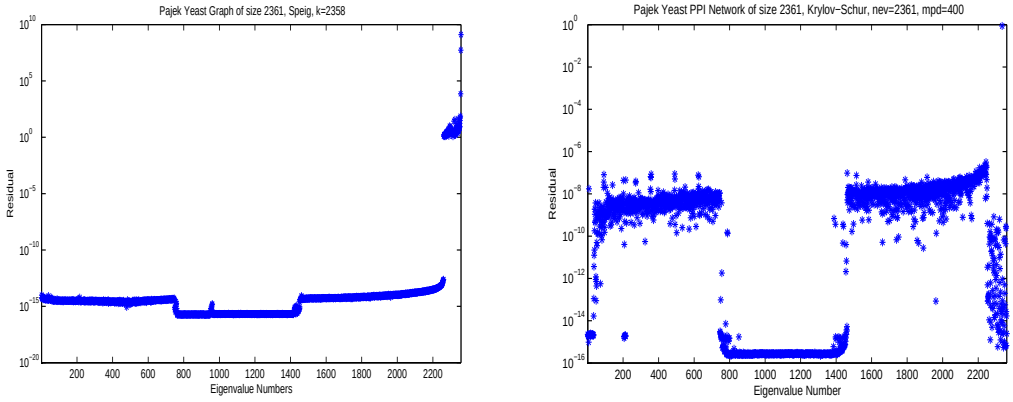


Figure 5.9: Protein–Protein interaction network of yeast of size 2361, left: SPEIG, right: SLEPc, Krylov–Schur

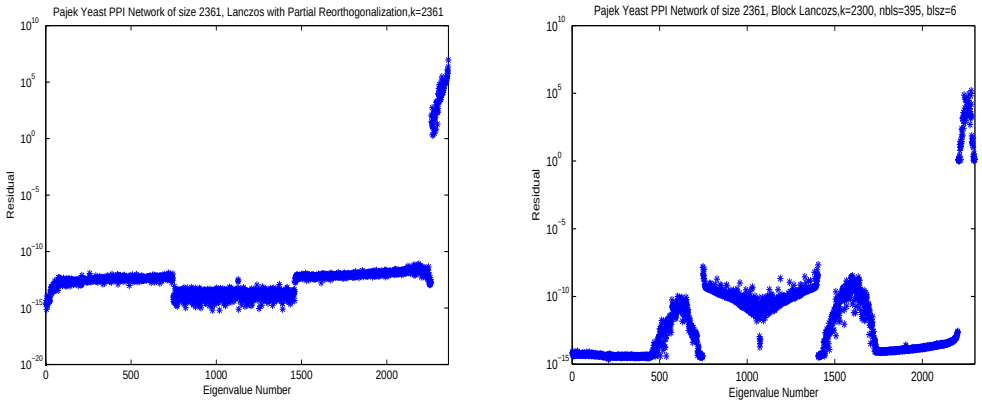


Figure 5.10: Protein–Protein interaction network of yeast of size 2361, left: LANEIG, right: IRBLEIGS

Table 5.3: *mpd* and the CPU times for Real Networks

Matrix size	*mpd*	Time
6027	600	3390 s
7343	1000	7015 s
13332	4000	47333s

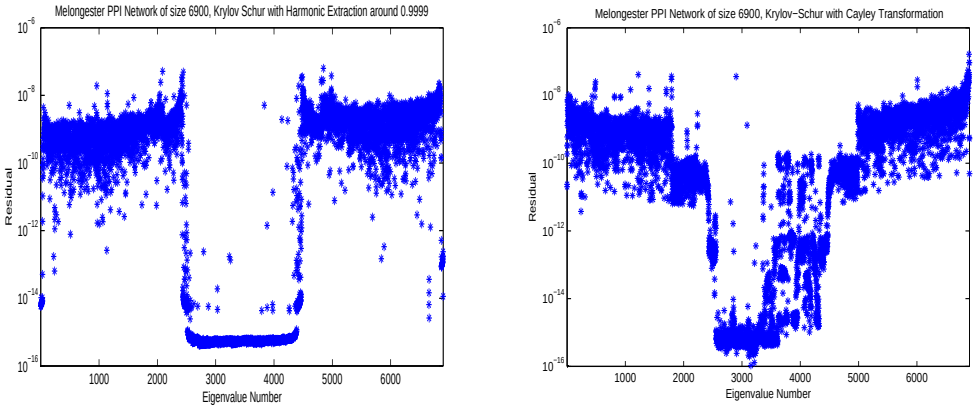


Figure 5.11: Protein–Protein interaction network of melongester of size 6900, left: SLEPc,Krylov–Schur with harmonic extraction around 0.9999, right: SLEPc,Krylov–Schur Cayley transformation around 0.9999

We have also computed the eigenvalues of normalized Laplacian matrices of some real networks. The data is taken from the University of Florida Matrix Collection [26]. The first data describes an Erdös collaboration network and of size 6027, second one is geometric collaboration network and of size 7343 and the final one is a word network and of size 13332.

Figures 5.12 and 5.13 show the relative residual plots for SLEPc–Krylov Schur method. On Table 5.3, the *mpd* values and CPU times for each matrix are given. The parameter *mpd*(maximum projected dimension) restricts the dimension of the space that the matrix is projected onto and if the value of this parameter is not big enough the number of iterations to compute eigenvalues will be large. Other solvers failed to detect the whole spectrum also for these matrices.

The residuals in Figure 5.12 left and right varies between 10^{-6} and 10^{-20}. Especially, in 5.12 left, they are around 10^{-20} for smaller eigenvalues but increase up to 10^{-5} for some interior eigenvalues. In Figure 5.13, the residuals vary between 10^{0} and 10^{-20}.

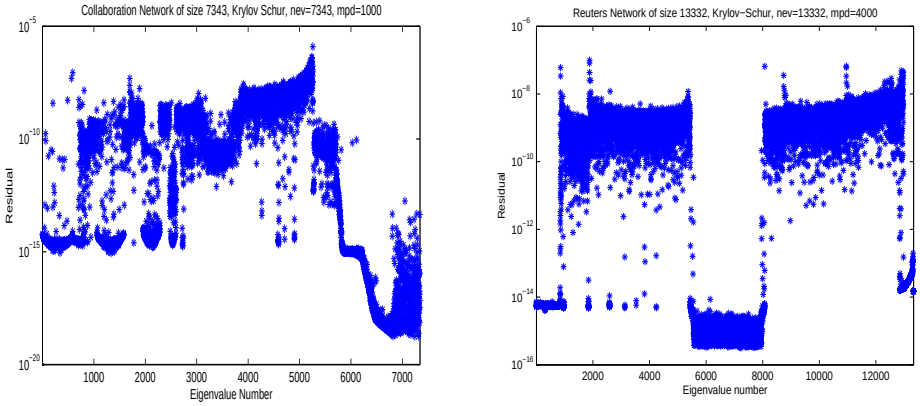


Figure 5.12: SLEPc,Krylov–Schur method, left: Collaboration network of size 7343, right: Word network of size 13332

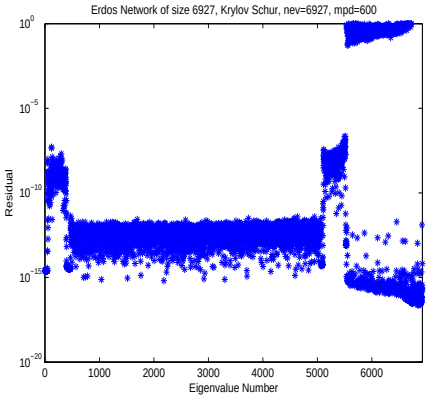


Figure 5.13: SLEPc,Krylov–Schur method, Erdös collaboration network of size 6027

5.2 Spectral Density Plots

In this section, we give the spectral density plots of the normalized Laplacian matrices of networks from previous Section. It was mentioned in Chapter 2 that spectrum of this matrix carries important information about the structural properties of network. We used the density functions resulted by convolving the Dirac delta function, $\delta(\lambda, \lambda_k)$, with the Gaussian and the Lorenz kernels [8]:

$$f(x) = \int g(x, \lambda) \sum_k \delta(\lambda, \lambda_k) d\lambda = \sum_k g(x, \lambda_k) \tag{5.3}$$

where $g(x, \lambda) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\lambda_x)^2}{2\sigma^2})$, the Gaussian kernel and $g(x, \lambda) = \frac{1}{\pi} \frac{\gamma}{(x-m)^2+\gamma^2}$, the Lorenz kernel. When the parameter values σ and γ are decreased, the finer details of the spectrum can be seen more clearly.

In Figure 5.14, the spectral density plot of yeast PPI network of size 2361 is shown both with Gaussian and Lorenz distributions. The PPI networks represent the interactions of proteins in the organisms. The nodes are proteins and there is an edge between two nodes if they interact with each other. In Figure 5.15, the spectral distribution of another PPI network of size 6900 is shown. The original network was of size 40028 with many unconnected components. Here we investigated the largest connected component of the network.

The general pattern observed in Figures 5.14 and 5.15 are in correspondence with the results reported in [9] about PPI networks. Two important features of distributions should be noticed in these figures: the sharp peak around 1 and the symmetry with respect to 1. According to the results in Section 2.2.2, this structure in distribution of eigenvalues indicates a process of node duplications during the evolutionary process of network and this is a general structure observed in PPI networks [8].

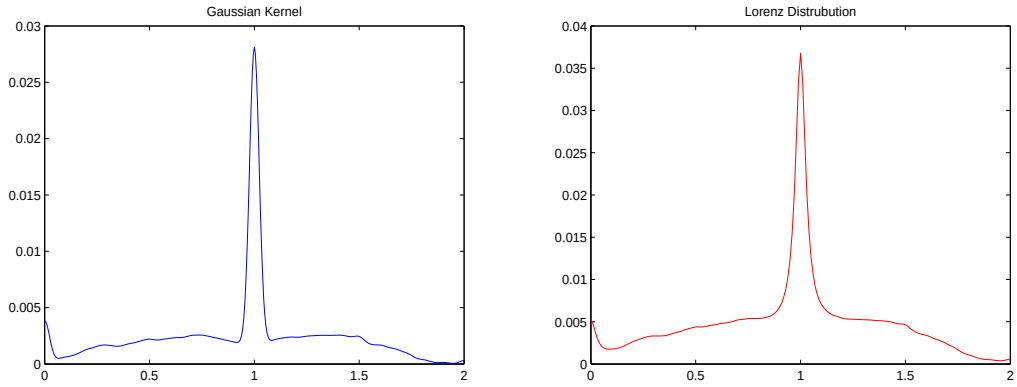


Figure 5.14: The spectral distribution of a protein protein network in size 2361. First picture is plotted with Gaussian kernel. Second picture is plotted with Lorenz distribution.

Figure 5.17, the eigenvalue distribution of collaboration in computational geometry network is shown. Originally the network was weighted. The weights represents the number of common papers between authors. However, in this study all weights are taken 1, so that the number of each work between two authors are assumed to be same. In the Figure 5.17, the peak at 0 signals the presence of small eigenvalues. This implies that the network may consist of many components which are weakly connected to each other. There are smaller peaks at 1 and 1.5 signalling a node and vertex duplications procedures in the evolution of network.

In Figure 5.16 the plot on the left is the eigenvalue distribution of collaboration network constructed by Erdös. In this Figure, although there is a high peak around 1, its shape is

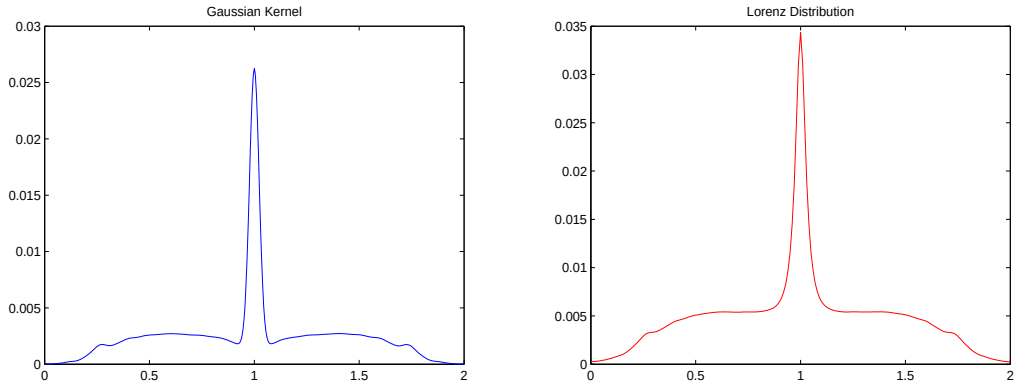


Figure 5.15: The spectral distribution in left belongs to a protein protein interaction network of size 6900

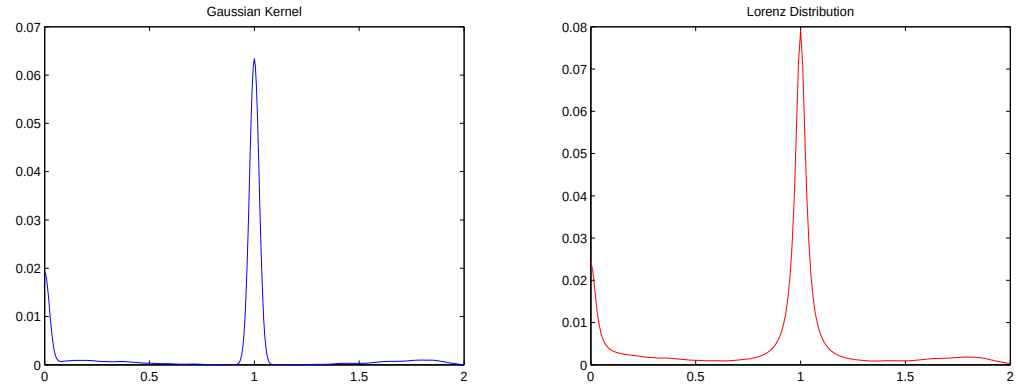


Figure 5.16: The spectral distribution belongs to collaboration network by Erdös, of size 6027

different than the PPI networks. The figure is not symmetric around 1 so we can not expect any bipartite structure and also it consists of a very large connected component.

Figure 5.18 the left plot is the eigenvalue distribution of a Reuters word network. The Reuters terror news network is based on all stories released during 66 consecutive days by the news agency Reuters concerning the September 11 attack on the U.S. The data set in Pajek's format is obtained from the CRA networks and produced by Steve Corman and Kevin Dooley at Arizona State University. The vertices of a network are words (terms); there is an edge between two words if and only if they appear in the same text unit (sentence). Originally the network was weighted and the weights represent the frequency of the edge. But we have taken all weights equal to 1.

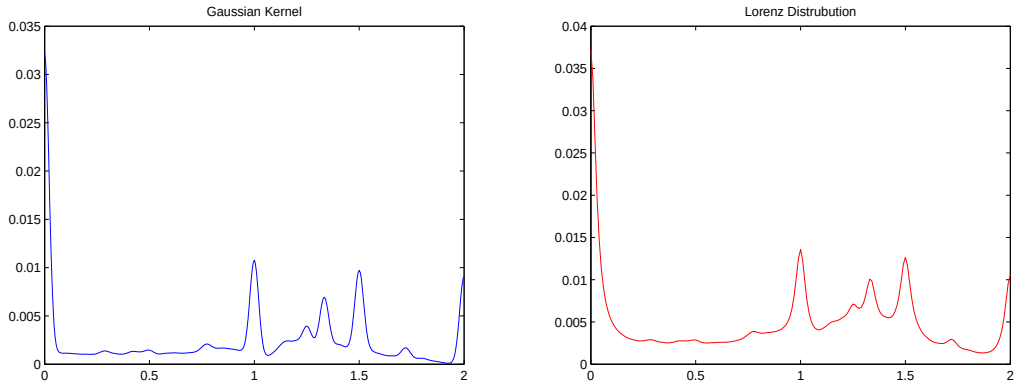


Figure 5.17: The spectral distribution belongs to a collaboration network in computational geometry of size 7343

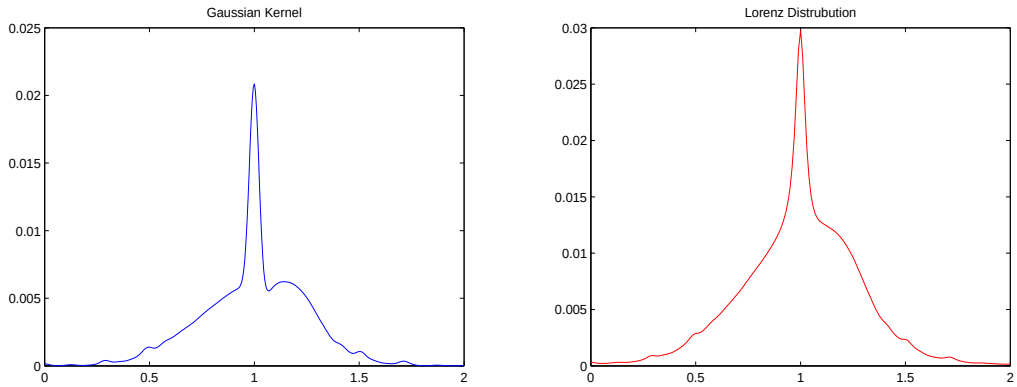


Figure 5.18: The spectral distribution belongs to a word network of size 13332

5.3 Conclusions

In this thesis, the performance and accuracy of different sparse eigensolvers are compared for computing the spectrum of the normalized Laplacian matrices of undirected graphs, which arise in large empirical networks. Usually, the eigensolvers are designed to compute a few eigenvalues of matrices but we have computed the whole spectrum by appropriate adjustments of the parameters. The eigensolvers EIGIFP, LANEIG, JDQR and JDCG have calculated the whole spectrum without adjustments of parameters. On the other hand, for SPEIG, AHBEIGS, IRBLEIGS and SLEPc some parameters must be adjusted.

The computational experiments showed that, the solvers behave similarly and have almost the same performance and accuracy for Paley graphs and empirical networks of size $100 - 2000$.

The CPU times of the solvers start to differ as the size of the normalized Laplacian matrices became large. The eigensolvers SPEIG, LANEIG and the Krylov-Schur method of library SLEPc required less computing time than the others. Moreover JDQR and JDCG failed to compute the eigenvalues for large empirical networks. In addition, we could not compute the whole spectrum with the eigen solvers SPEIG, AHBEIGS, IRBLEIGS, LANEIG, EIGIFP, JDQR and JDCG for matrices of size larger than 3000 due to the memory limitations of MATLAB. On the other hand, the eigensolvers of SLEPc worked without any problem for larger matrices up to size 13000 and the numerical results were quite satisfactory in terms of accuracy and computing time. The spectral density plots of protein protein networks, collaboration networks were in correspondence with those in literature (especially in [8, 9, 10]).

In a future study, we aim to use the block Krylov-Schur and block Jacobi–Davidson algorithms of the ANASAZI package and compare it with the eigensolvers of the SLEPc library. In order to perform spectral analysis of very large networks, parallel implementations of these packages should be used.

REFERENCES

[1] A. Maritan A. Vazquez, A. Flammini and A. Vespignani. Modeling of protein interaction networks. *ComPlexUs*, 1:38–44, 2003.

[2] J. Baglama. Augmented block Householder Arnoldi method. *Linear Algebra and its Applications*, 429:2315 – 2334, 2008.

[3] J. Baglama, D. Calvetti, and L. Reichel. Algorithm 827: irbleigs: a MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix. *ACM Trans. Math. Software*, 29:337–348, 2003.

[4] J. Baglama, D. Calvetti, and L. Reichel. IRBL: an implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems. *SIAM J. Sci. Comput.*, 24:1650–1677, 2003.

[5] J. Baglama, D. Calvetti, L. Reichel, and A. Ruttan. Computation of a few small eigenvalues of a large matrix with application to liquid crystal modeling. *J. Comput. Phys.*, 146:203–226, 1998.

[6] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems*. SIAM, 2000.

[7] C.G. Baker, U.L. Hetmaniuk, R.B. Lehoucq, and H.K. Thornquist. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Trans. Math. Softw.*, 36:1–23, 2009.

[8] A. Banerjee. *The Spectrum of the Graph Laplacian as a Tool for Analyzing Structure and Evolution of Networks*. PhD thesis, Fakultät für Mathematik und Informatik der Universität Leipzig, 2008.

[9] A. Banerjee and J. Jost. Spectral plots and the representation and interpretation of biological data. *Theory in Biosciences*, 126:15– 21, 2007.

[10] A. Banerjee and J. Jost. Laplacian spectrum and protein-protein interaction networks. Technical report, 2008. Available at http://www.citebase.org/abstract?id=oai:arXiv.org:0705.3373.

[11] A. Banerjee and J. Jost. On the spectrum of the normalized graph Laplacian. *Linear Algebra Appl.*, 428:3015–3022, 2008.

[12] A. Banerjee and J. Jost. Spectral plot properties:towards a qualitative classification of networks. *Networks and Heterogenous Media*, 3:395–411, 2008.

[13] A. Banerjee and J. Jost. Graph spectra as a systematic tool in computational biology. *Discrete Appl. Math.*, 157:2425–2431, 2009.

[14] A. Banerjee and J. Jost. Spectral characterization of network structures and dynamics. In N. Ganguly, A. Deutsch, and A. Mukherjee, editors, *Dynamics On and Of Complex Networks*, pages 117–132. Birkhäuser Boston, 2009.

[15] A.L. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[16] A.L. Barabasi and Z.N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5:101–113, 2004.

[17] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424:175–308, 2006.

[18] B. Bollobas. *Modern Graph Theory*. Springer, 1998.

[19] U. Brandes and T. Erlebach. *Network Analysis: Methodological Foundations (Lecture Notes in Computer Science / Theoretical Computer Science and General Issues)*. Springer, 2005.

[20] B.N. Parlett C.C. Paige and H.A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numerical Linear Algebra with Applications*, 2:115–133, 1995.

[21] F.R.K. Chung. *Spectral Graph Theory (Cbms Regional Conference Series in Mathematics)*. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[22] F.R.K. Chung and L. Lu. *Complex Graphs and Networks*. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 2006.

[23] F.R.K. Chung, L. Lu, and V. Vu. Spectra of random graphs with given expected degrees. *Proc Natl Acad Sci U S A*, 100:6313–6318, 2003.

[24] J.K. Cullum and R.A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1*. SIAM, 2002.

[25] E.R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real symmetric matrices. *Journal of Computational Physics*, 17:87–94, 1975.

[26] T.A. Davis. University of Florida sparse matrix collection. http://www.cise.ufl.edu/research/sparse. Technical report, 1997.

[27] P. Erdös and A. Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[28] P. Erdös and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.

[29] P. Erdös and A. Rényi. On the strength of connectedness of a random graph. *Acta Math. Acad. Sci.*, 12:261–267, 1961.

[30] D.R. Fokkema, G.L.G. Sleijpen, and H. Van der Vorst. Accelerated inexact newton schemes for large systems of nonlinear equations. *SIAM J. Sci. Comput.*, 19:657–674, 1998.

[31] Z. Füredi and J. Komlós. The eigenvalues of random symmetric matrices. *Combinatorica*, 1:233–241, 1981.

[32] Homer F.W. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9:152–163, 1988.

[33] C.D. Godsil and G. Royle. *Algebraic Graph Theory*. Springer, New York, 2001.

[34] K.I. Goh, B. Kahng, and D. Kim. Spectra and eigenvectors of scale-free networks. *Physical Review E*, 64:051903, 2001.

[35] G.H. Golub and H.A. van der Vorst. Eigenvalue computation in the 20th century. *J. Comput. Appl. Math.*, 123:35–65, 2000.

[36] G.H. Golub and Q. Ye. An inverse free preconditioned Krylov subspace method for symmetric generalized eigenvalue problems. *SIAM J. Sci. Comput.*, 24:312–334, 2002.

[37] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: Scalable library for eigenvalue problem computations. *Lecture Notes in Computer Science*, 2565:377–391, 2003.

[38] V. Hernandez, J.E. Roman, E. Romero, A. Tomas, and V. Vidal. SLEPc Users Manual. Technical Report DSIC-II/24/02 - Revision 3.0.0, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2009. Available at http://www.grycap.upv.es/slepc.

[39] V. Hernandez, J.E. Roman, A. Tomas, and V. Vidal. Krylov–Schur method in slepc. Technical Report STR-7, Universidad Politécnica de Valencia, 2007.

[40] V. Hernandez, J.E. Roman, A. Tomas, and V. Vidal. A survey of software for sparse eigenvalue problems. Technical Report STR-6, Universidad Politécnica de Valencia, 2007. Available at http://www.grycap.upv.es/slepc.

[41] U. Hetmaniuk and R. Lehoucq. Basis selection in LOBPCG. *J. Comput. Phys.*, 218:324–332, 2006.

[42] C.G.J. Jacobi. Über eine leichtes verfahren, die in der Theorie der säcularstörungen vorkommenden Gleichungen numerisch aufzulösen. *Journal Reine Angew. Math*, pages 51–94, 1846.

[43] Y. Jay and L.G. Jonathan. *Handbook of Graph Theory (Discrete Mathematics and Its Applications)*. CRC, 2003.

[44] J. Jost and M.P. Joy. Spectral properties and synchronization in coupled map lattices. *Rev. E*, 65:16–21, 2001.

[45] D. Kressner. *Numerical Methods for General and Structured Eigenvalue Problems*. Springer, 2005.

[46] R.M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. Technical report, 1998.

[47] R.B. Lehoucq and D.C. Sorensen. Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. Appl*, 17:789–821, 1996.

[48] R.B. Lehoucq, D.C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods.* SIAM, 1998.

[49] J.H. Money and Q. Ye. Algorithm 845: EIGIFP: a MATLAB program for solving large symmetric generalized eigenvalue problems. *ACM Trans. Math. Softw.*, 31:270–279, 2005.

[50] R.B. Morgan. On restarting the Arnoldi method for large nonsymmetric eigenvalue problems. *Math. Comput.*, 65:1213–1230, 1996.

[51] M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[52] Y. Notay. Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem. *Numerical Linear Algebra with Applications*, 9:1070–5325, 2002.

[53] C.C. Paige. Error analysis of the Lanczos algorithms for tridiagonalizing a symmetric matrix. *J. Inst. Math. Appl.*, 18:341–349, 1976.

[54] C.C. Paige. Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Algebra Appl*, 34:235–258, 1980.

[55] P.Arbenz, U.L.Hetmaniuk, R.B.Lehoucq, and R.S.Tuminaro. A comparison of eigensolvers for large-scale 3d modal analysis using amg-preconditioned iterative methods. *Int. J. Numer. Meth. Engng*, 64:204–236, 2005.

[56] B.N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, 1980.

[57] R.J. Radke. *A Matlab Implementation of the Implicitly Restarted Arnoldi Method for Solving Large Scale Eigenvalue Problems*. PhD thesis, Rice University, 1996.

[58] J.E. Roman. Practical implementation of harmoic Krylov Schur. Technical Report STR-9, Universidad Politécnica de Valencia, 2009. Available at http://www.grycap.upv.es/slepc.

[59] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, second edition, 2003.

[60] H.D. Simon. The Lanczos algorithm with partial reorthogonalization. *Mathematics of Computation*, 42:115–142, 1984.

[61] G.L.G. Sleijpen and H. Van Der Vorst. The Jacobi-Davidson method for eigenvalue problems and its relation with accelerated inexact Newton schemes. In *in Iterative Methods in Linear Algebra II*, pages 17–20. Publishing Co. Inc, 1996.

[62] G.L.G. Sleijpen and H.A. Van Der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM Rev*, 42:267–293, 2000.

[63] G.L.G. Sleijpen, H.A. Van Der Vorst, and D.R. Fokkema. Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM J.Sci.Comput*, 20:94–125, 1998.

[64] R. Solomonoff and A. Rapoport. Connectivity of random nets. *Bulletin of Mathematical Biophysics*, 13:107–117, 1951.

[65] D.C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13:357–385, 1992.

[66] D.C. Sorensen. Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations. In *Parallel Numerical Algorithms(Hampton,VA,1994)*, volume 4, pages 119–165. Kluwer, 1997.

[67] D.C. Sorensen. Numerical methods for large eigenvalue problems. *Acta Numerica*, 11:519–584, 2002.

[68] G.W. Stewart. *Matrix Algorithms, Vol I*. SIAM, 1998.

[69] G.W. Stewart. *Matrix Algorithms, Vol II*. SIAM, 2001.

[70] G.W. Stewart. A Krylov–Schur algorithm for large eigenproblems. *SIAM J. Matrix Anal. Appl.*, 24:599–601, 2002.

[71] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Global protein function prediction in protein-protein interaction networks. *ArXiv Condensed Matter e-prints*, 2003.

[72] S. Vishveshwara, K.V. Brinda, and N. Kannan. Protein structure: Insights from graph theory. *Journal of Theoretical Computational Chemistry*, 1:187–211, 2002.

[73] C. von Mering, R. Krause, B. Snel, M. Cornell, S.G. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417:399–403, 2002.

[74] D.S. Watkins. *Fundamentals of Matrix Computations*. Wiley, second edition, 2002.

[75] D.S. Watkins. *The Matrix Eigenvalue Problem*. SIAM, 2007.

[76] D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[77] E.P. Wigner. Characteristic vectors of bordered matrices with infinite dimensions. *The Annals of Mathematics*, 62:548–564, 1955.

[78] K. Wu and H. Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 22:602–616, 2000.

[79] Y. Zhou and Y. Saad. Block Krylov-Schur method for large symmetric eigenvalue problems. *Numerical Algorithms*, 47:341–359, 2008.