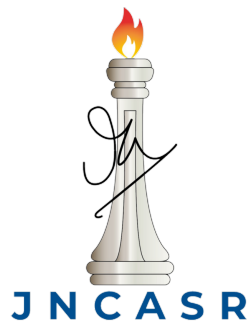


Iterative eigenvalue solutions

Lanczos algorithm for symmetric matrices



Akash Bansal

Engineering Mechanics Unit
Jawaharlal Nehru Centre For Advanced Scientific Research

This dissertation is submitted for the degree of
MASTER OF SCIENCE (ENGINEERING)

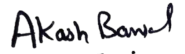
JNCASR, Bengaluru, India

January 2025

To my loving family ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.


Akash Bansal
January 2025

CERTIFICATE

I hereby certify that the matter embodied in this thesis entitled “**Iterative Eigenvalue Solutions - Lanczos Algorithm for Symmetric Matrices**” has been carried out by **Akash Bansal** at the Engineering Mechanics Unit, Jawaharlal Nehru Centre for Advanced Scientific Research, Bangalore, India under my supervision and that it has not been submitted elsewhere for the award of any degree or diploma.



Santosh Ansumali

(Research Supervisor)

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Santosh Ansumali, for his unwavering support, invaluable guidance, and continuous encouragement throughout the course of my research. His profound knowledge and insightful feedback have been instrumental in shaping the direction and quality of this thesis. I am incredibly fortunate to have had the opportunity to learn from him and benefit from his expertise.

I am also grateful to Prof. Srikant Sastry for the insightful discussions during the project. Additionally, I extend my heartfelt thanks to Dr. Diwakar Seyyanur Venkatesan for his invaluable help with coding implementation issues and providing the best possible suggestions that were crucial to the successful completion of this work.

I would like to acknowledge my seniors Akshay, Praveen, Surya, Akhilesh, Soumya, and Sangamesh for their guidance. Their advice has been invaluable in navigating the challenges of this research.

I am thankful to my EMU batchmates Tawde, Jishnu, Ganesh, Guru, and Abhisek for the memorable moments and fun we shared throughout this journey.

Special thanks to Malay from Chennai Mathematical Institute for his help with the formal proofs, which significantly strengthened the rigor of this thesis.

I would also like to express my gratitude to my JNC friends Kamo-Geeta, Mansi, Nil, Debashree, Ayon, Pinka, and, of course, Bru for their support and encouragement.

Finally, my deepest thanks go to my parents and my brother Akshit for their unwavering love and support throughout my academic pursuits.

Abstract

Lanczos algorithm is an effective tool for constructing an approximate tridiagonalization of a symmetric matrix. The basic procedure creates a set of vectors and makes them orthogonal to create an orthonormal basis of the Krylov subspace. While the original formulation failed in finite precision, subsequent modification where one reorthogonalized the set of vectors is quite stable. In its current form, it is known for its efficiency in computing a subset of eigenvalues and eigenvectors for large sparse symmetric matrices via this tridiagonal representation.

This thesis revisits the Lanczos algorithm and its behavior in finite precision. The starting point is a well-known observation that the Lanczos method fails when few of the eigenvalues converges. At that point, the vectors lose orthogonality and the method produces spurious eigenvalues. Furthermore, it is also known that larger the spectral gap, faster such failure occurs. This failure is typically circumvented via reorthogonalization of the Lanczos vectors. In this thesis, it is shown that augmenting the original matrix is an effective way to reduce the spectral gap and thus slow down considerably the emergence of spurious eigenvalues. It is shown that the new approach of augmented matrix leads to a stable scheme and thus provides an alternate way to think about Lanczos and other Krylov subspace-based methods in finite precision.

Table of contents

List of figures	xiii
List of tables	xv
1 Introduction	1
1.1 Lanczos algorithm: State-of-the-art Eigenvalue solver	4
1.2 Approach overview	4
1.3 Thesis organisation	5
2 Floating point numbers	7
2.1 Representation of Numbers on computers	7
2.2 IEEE Floating Point Standard	9
2.3 Error Analysis	13
2.3.1 Error Model for Floating point operations	15
2.3.2 Error in Dot product	17
2.4 Forward and Backward Error Analysis	22
3 Gram-Schmidt Orthogonalization	23
3.1 Introduction	23
3.2 Gram-Schmidt Orthogonalization	24
3.3 Finite Precision Effect	27
3.4 Doing the algorithm twice	29
3.5 Alternate Formulations	31
3.6 Reorthogonalization in Lanczos algorithm	32
4 Computational methods for Eigenvalue problem	33
4.1 Rayleigh quotient	34
4.2 Power Method	37
4.3 Krylov subspace	42

5	Lanczos algorithm	45
5.1	Lanczos method	45
6	Lanczos algorithm: A Formal approach	53
6.1	Lanczos procedure	53
6.2	Lanczos algorithm for degenerate matrices	55
6.3	Orthogonal Projection operator	56
6.3.1	Eigensystem of Projection Matrix	57
6.4	Rounding Error Analysis for Lanczos Algorithm	59
7	Lanczos Algorithm in Finite Precision	67
7.1	Key issues in the Lanczos method	67
7.2	Block Lanczos Method	70
7.3	Observations in the Lanczos Method	72
7.4	Enhancing Lanczos Method Implementation: Insights and Remedies	77
7.4.1	Computing k eigen pairs: Working algorithm	81
8	Applications using Lanczos Algorithm	83
8.1	Principal Component Analysis	83
9	Outlook	87
	References	89

List of figures

2.1	Institute of Electrical and Electronic Engineers (IEEE) floating point standard (IEEE754)	9
4.1	The Rayleigh quotient $R(A, x)$, converges to the dominant eigenvalue $\lambda_1 = 3.0$ of matrix A of size 203×203 for $m = 34$ power iterations. The matrix A is a diagonal matrix so its eigenvalues will be same as the diagonal elements of A . $A_{203 \times 203} = \text{diag}(0, 0.01, 0.02, \dots, 1.99, 2, 2.5, 3.0)$	38
4.2	Error plots for matrix A . $A_{203 \times 203} = \text{diag}(0, 0.01, 0.02, \dots, 1.99, 2, 2.5, 3.0)$	39
4.3	Power iterations on matrix $A_{200 \times 200}$ to compare convergence depending on the eigengap.	40
4.4	Comparison of error in eigenvalue for figures 4.4a ($\frac{ \lambda_2 }{ \lambda_1 } = 0.1$) and 4.4b ($\frac{ \lambda_2 }{ \lambda_1 } = 0.9$).	40
4.5	Comparison for error in eigenvector for figures 4.5a ($\frac{ \lambda_2 }{ \lambda_1 } = 0.1$) and 4.5b ($\frac{ \lambda_2 }{ \lambda_1 } = 0.9$).	41
5.1	Behaviour of top eight Ritz values (θ_i) for $k = 40$ iterations of Lanczos method for matrix A of size 503×503 . At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V_k^T A V_k$ where $V_k = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V_k^T A V_k$ where $V_k = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$	50
5.2	Behaviour of Ritz values for Lanczos method for matrix A of size 503×503 from the bottom end of the eigen spectra. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V_k^T A V_k$ where $V_k = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$	51

- 7.1 Lanczos method without Reorthogonalization indicating multiplicities of Ritz values. The behaviour of top eight Ritz values (θ_i) for $k = 60$ iterations of Lanczos method for matrix A of size 503×503 is shown. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V^T A V$ where $V = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$ 68
- 7.2 Lanczos method with Reorthogonalisation don't have this issue of appearance of multiple Ritz values. The behaviour of top eight Ritz values (θ_i) for $k = 68$ iterations of Lanczos reorthogonalisation method for matrix A of size 503×503 is shown. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V^T A V$ where $V = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$ 69
- 8.1 Using 'PCA' function in Python, first two principal components are obtained and hence, the data is projected to obtain clusters in order to label the data as malignant or benign. 85
- 8.2 Using the first 5 Lanczos vectors i.e. $V_5 = \{v_1, v_2, v_3, v_4, v_5\}$ and computing the tri-diagonal matrix $T_5 = V_5^T A V_5$ where A is 30×30 features matrix. Computing the first two dominant approximate eigenvectors using the eigenvectors of T_5 , one can use them as good approximations to the first two principal components. Thus, all the data points are projected using these approximations to the principal components resulting in separation of data into clusters. 86

List of tables

2.1	Representation in scientific notation	8
2.2	Zero and Infinity representation for 64-bit double in IEEE 754 Standard . .	11
7.1	The Block Lanczos method extends the traditional Lanczos algorithm by reaching a higher number of Lanczos vectors, thus ensuring a greater number of approximate eigenpairs. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$	70
7.2	Block Lanczos Method implemented on A of size 9×9	72
7.3	Lanczos Method implemented on A with $\lambda_i = \{-2.5, -1.5, 1.0, 2.0, 3.0, 100000\}$	73
7.4	Lanczos Method implemented on A_1	74
7.5	Lanczos Method implemented on A with $\lambda_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100, 500, 1000\}$	75
7.6	Lanczos Method implemented on A with $\lambda_i = \{1, 2, \dots, 9, 10, 100, 200, \dots, 900, 1000\}$	75
7.7	Lanczos method implemented on A with $\lambda_i = \{1, 2, 3, \dots, 28, 19, 30, 1000\}$	76
7.8	Lanczos method implemented on A with $\lambda_i = \{1, 2, 3, \dots, 28, 29, 30, 1000, 1000 + 1 \times 10^{-15}\}$	77
7.9	Lanczos Method implemented on $A = \text{diag}(-2.5, -1.5, 1.0, 2.0, 3.0, 100000)$	79
7.10	Lanczos Method implemented on B_{A_1}	80
7.11	Lanczos method implemented on B_A of size $(k+1)n \times (k+1)n$ using $A_{original}$ with $\lambda_i = \{0, 1, 2, 3, 4, \dots, 498, 499, 500, 550, 600\}$ placed along the diagonal along with its k different permutation groups. All the eigen pairs for the matrix $A_{original}$ can be computed.	81

Chapter 1

Introduction

Computational linear algebra is a fundamental building block for scientific computing as well as machine learning [1] and data sciences [2]. Indeed, modern software tools for linear algebra are quite expressive and powerful. A few example of such tools are Matlab [3, 4], TensorFlow [1], and NumPy. In C++, optimized libraries like Blitz++, Boost, UBLAS, GNU Scientific Library (GSL) [5], and Eigen enable efficient execution of diverse linear algebra tasks. These tools are widely employed in areas as diverse as computational physics [6], computational biology [7], computational finance [8], data science [2], and image processing [9]. Indeed, many of the machine learning tools crucially rely on linear algebra tools such as principal component analysis (PCA) [10], linear regression [1], eigenvalue finding or singular value decompositions (SVD) [11]. In scientific computing, it is central to solution methods used to solve partial differential equations where one often reduces the problem at hand to either solving a set of linear equations or to a Eigenvalue problem. In computational linear algebra, the focus lies on seeking efficient numerical techniques in finite precision computing devices. These techniques are often employed for various tasks, such as matrix inversion, solving least squares problems, computing matrix norms, and estimating condition numbers. Any computational tool for a linear algebraic task [11] involves

1. Basic Matrix Operations: The set of atomic tasks in linear algebra involves various matrix algebra operations such as addition, subtraction, matrix-matrix and matrix-vector multiplication, and matrix transposition. This set is essential for building more complex linear algebraic tasks. In order to perform common linear algebra operations efficiently, there is a collection of low level routines known as Basic Linear Algebra Subprograms or BLAS [12],[13]. They are written in Fortran and was first developed in the late 1970s. They are designed to provide highly optimized implementations for different architectures. BLAS is commonly used in scientific computing environments and numerical libraries like NumPy [14] and Matlab [3]. Python is widely used in

basic matrix operations due to its simplicity, versatility, and the availability of powerful libraries such as NumPy and SciPy, which offer efficient implementations of matrix operations in a straightforward manner and computationally efficient for both small and large datasets. In C++ [15], one can use BLAS functionality interface by using libraries such as Eigen [16], Octave [5] or Armadillo [17]. Linear algebra on GPUs and accelerators is foundational to modern day machine learning. For example, TensorFlow [1] uses tensors, i.e., multi-dimensional arrays of uniform type, to construct highly efficient neural networks.

2. System of Linear Equations: One often reduces a scientific problem to a solution of a system of linear equations in the form of

$$Ax = b, \quad (1.1)$$

where A is a $n \times n$ matrix, x is an unknown vector, and b is the constant vector, the task is to find the solution vector x . The field is well-evolved and many state of art algorithms [18] and solvers exist for the purpose. Broadly, these methods can be classified as direct methods and iterative schemes. While a direct solver such as Gaussian elimination, LU decomposition [11] is very exact for a general matrix even the state of art direct method will scale like $O(n^3)$. This might be fine for small size dense matrix, however it is quite wasteful when the matrix is sparse which only has $O(n)$ non-zero entries.

A typical iterative solver for diagonally-dominant matrices such as Gauss-Seidel or Jacobi iterations [19] provides an approximate solution at a much lesser cost. It scales as $O(n)$ for sparse matrices. The Conjugate Gradient method [18] is an iterative technique commonly used to solve systems of linear equations, particularly for sparse matrices, by minimizing the error in a Krylov subspace [20]. It is closely related to the Lanczos algorithm [21], which iteratively constructs an orthogonal basis for the Krylov subspace, and both methods exploit the properties of the symmetric positive-definite matrices, with Lanczos also being used for eigenvalue computations in symmetric eigenvalue problems [21].

3. Matrix Factorization: A common way to solve linear equations which is also useful in extracting other useful information from matrix is to factorize a given matrix into a product of two or more matrices with specific properties. Examples include LU decomposition, QR decomposition [18] and the square root of a symmetric positive definite matrix [11]. These methods typically have a computational cost that scales

as $O(n^3)$. This cubic complexity arises from the need to perform a series of matrix multiplications, which dominate the computational workload. As a result, matrix factorization methods can become prohibitively expensive for large matrices, particularly when n is large.

4. **Eigenvalue and Eigenvector Computations:** Very often one is interested in understanding matrix features via its eigenvalues and eigenvectors. Thus, a fundamental task for computational linear algebra is to solve an eigenvalue problem, $AX = \lambda X$ [11]. Even though, several algorithms and techniques have been developed over decades to efficiently compute eigenvalues, finding algorithms to do so remain an active area of research in numerical linear algebra, especially for large-scale problems.

Similarity transformations are often used as an important ingredient in finding eigenvalues. In this approach, a given matrix A is transformed into a similar matrix $M = P^{-1}AP$, where P is an invertible matrix as the eigenvalues of M are the same as the eigenvalues of A . The advantage lies in the fact that P can be chosen in such a way that the matrix M is in a much more simpler form (e.g., tri-diagonal or diagonal), making it easier to compute the eigenvalues.

This simplification enables the use of more efficient and specialized algorithms for finding eigenvalues. For large but sparse matrices, iterative methods and Krylov subspace [20] based methods such as Lanczos [22] remain the method of choice. In general, the best choice of eigenvalue computation method depends on the specific characteristics such as sparsity, symmetry, and the required precision.

In this thesis, the main focus is on one of the key problems mentioned above, namely the state of the art eigenvalue solver. This part utilizes iterative eigenvalue algorithms specialized for symmetric matrices to efficiently compute their eigenvalues. These methods [23] take far less than $O(n^2)$ storage and $O(n^3)$ flops [20]. Lanczos Algorithm [22] is becoming more and more popular because of its ability to handle large and large matrices. This is because rather than using the full matrix to calculate eigenvalues and eigenvectors, one can use matrix-vector product to converge to the required number of eigenvectors. These algorithms are implemented to find eigenpairs in such a way that one needs to provide only a subroutine to compute matrix-vector product AX rather than storing matrix A explicitly in C++. The implementation of the Lanczos algorithm in a finite arithmetic machine presents several challenges. One major issue arises from the algorithm's tendency to encounter convergence problems, particularly when approaching certain eigenvalues. To delve into these challenges, I'll examine specific examples and explore potential solutions to mitigate

these issues. Ultimately, our aim is to devise a robust algorithm capable of effectively addressing these hurdles.

1.1 Lanczos algorithm: State-of-the-art Eigenvalue solver

Lanczos algorithm [21] is an iterative numerical procedure to approximate eigenvalues and eigenvectors of large sparse symmetric matrices. It is particularly efficient for large sparse matrices since it only requires matrix-vector products with A and operations with vectors of size equal to the number of non-zero elements in a row or column. Orthogonalization of Lanczos vectors is crucial for numerical stability, especially in the presence of round-off errors. The Lanczos process naturally leads to a tridiagonal matrix T_k , which is computationally cheaper to diagonalize compared to the original matrix A . Lanczos algorithm finds applications in various fields such as quantum chemistry[24], structural mechanics[21], and machine learning[1], where eigenvalue problems arise in the analysis of large-scale systems.

1.2 Approach overview

In finite precision, there are round-off errors which will break the orthogonality of Lanczos vectors resulting in emergence of spurious [23] eigenvalues. These spurious eigenvalues are not at all part of the actual physical system under consideration. The key starting point is the observation that Lanczos method behaves much better if the eigenvalues are not well-separated from each other. This is a very different behavior than other iterative methods such as power method. The key idea in the thesis is to augment the matrix and move to a higher dimension where eigengap is destroyed. This transition is done in a controlled manner to achieve a slow but certain convergence of the Lanczos method. For this augmentation, the idea of similarity transform using permutation matrices is used [11]. It is shown through various numerical examples how this issue of spurious eigenvalues can be delayed and hence, more and more eigenpairs can be captured. An approach of very significant importance, one that may seem very counterintuitive i.e., slowing down the Lanczos process by artificially creating very closely spaced clusters around all the eigenvalues for a system is used. In numerical techniques, the typical aim is to enhance the solution by accelerating the numerical process. However, this isn't the case in this scenario.

1.3 Thesis organisation

- Chapter 2: Chapter 2 starts with a discussion on the way numbers are represented in computers. This poses an immediate need to standardize this representation of numbers which is taken into consideration by IEEE. As a consequence of the finite precision in representing numbers, certain errors arise during the execution of floating-point operations. Thus, it is very important to perform the error analysis and deduce error models to understand this issue better. This aspect will be explored in section 2.3. Also, an overview of how forward and backward error analysis works is discussed in section 2.4. This chapter holds significance as it addresses the challenges encountered by all numerical methods when operating within finite precision.
- Chapter 3: This chapter is based on an important tool i.e., Gram-Schmidt orthogonalization which is regularly used to obtain an orthogonal set of basis from an independent set of vectors. There are some stability issues in finite precision which are discussed using a small example. Certain remedies are suggested to counter these stability issues in the subsequent sections of this chapter. Also, this tool is being used in the Lanczos algorithm which is the main focus of this thesis.
- Chapter 4: Chapter 4 starts with an exploration of diverse computational techniques applicable to solving eigenvalue problems. Furthermore, the chapter delves into the foundational aspects of designing iterative eigenvalue solvers, emphasizing similarity transformations [11]. The chapter also provides a concise overview of the Rayleigh quotient [20]. Subsequently, the implementation of the Power Method [18] in C++ is discussed. Concurrently, the challenges associated with the Power Method are briefly examined. Additionally, the concept of the Krylov subspace [20] which serves as the cornerstone for advanced techniques such as the Lanczos method [21] is explained.
- Chapter 5: This chapter discusses the theoretical aspects of the Lanczos algorithm which is one of the most popular method to compute eigenpairs for a large sparse symmetric matrix. Also, it explains in detail, the reasons why the Lanczos algorithm is very efficient computationally. Certain results are also presented which shows the strength of this powerful algorithm in computational linear algebra.
- Chapter 6: Chapter 6 provides a formal mathematical analysis of the Lanczos algorithm, following the approach outlined by Paige [25]. It elucidates the key reasons why a smaller matrix of size k can yield good estimates of the actual eigenpairs of the original matrix of larger size n .

- Chapter 7: Chapter 7 discusses various issues encountered in the Lanczos algorithm. Furthermore, one of the important algorithms to deal with degeneracy i.e., Block version of the Lanczos algorithm is discussed. There are certain observations related to the convergence of the Lanczos algorithm which are in some sense counterintuitive since slowing the convergence of this algorithm helps in achieving better results which is not a very common aspect usually seen in numerical methods. Finally, an algorithm is suggested to capture more eigenpairs in a simplified and more stable manner.
- Chapter 8: This chapter shows a few applications where the Lanczos algorithm can be used to compute eigenpairs for symmetric matrices. An important application in machine learning i.e., Principal component analysis is shown which can also be implemented using the Lanczos algorithm.
- Chapter 9: This chapter provides concluding remarks on the issues related to finite precision arithmetic in the Lanczos algorithm. It also mentions how combining similarity transforms of the original matrix using permutation matrices can be utilized to capture more eigenpairs. Additionally, it outlines future objectives, including a formal analysis of the extended method and a thorough numerical analysis to evaluate the computational advantages of the current approach.

Chapter 2

Floating point numbers

Real numbers are typically represented using a fixed number of bits on computers, which introduces limitations on the precision and range of values that can be accurately represented. This means that real numbers are approximated rather than precisely represented, leading to potential inaccuracies in calculations involving very large or very small numbers or operations that require high precision. Our understanding of the limitations and capabilities of finite precision computing has significantly evolved over time. An important contribution in this regard was error analysis framework as developed by Wilkinson [26]. The purpose of this chapter is to briefly review field of finite precision algebra in context of its importance in computational linear algebra.

This chapter is organized as follows: I commence with a concise overview of the imperative role of the floating-point number system in representing real numbers on digital systems, as detailed in Section 2.1. Subsequently, a brief introduction to the IEEE arithmetic [27] standard is provided in Section 2.2. Following this, the impact of rounding errors on operations will be discussed and also the concept of catastrophic cancellation during the computation of differences will be discussed briefly.

2.1 Representation of Numbers on computers

In almost all computing devices, data is stored using binary digits or bits. It's important to note that because only a limited number of bits can represent each data type, it's necessary to determine the required number of bits for encoding real numbers and establish a protocol for this encoding. For instance, in scientific computing, it's commonly accepted that one should aim for at least approximately 7 – 8 correctly represented significant digits. This implies a need for approximately $7 \times \log_2 10 \approx 23$ bits. Therefore, it's understandable that single precision real numbers are represented using 32 bits (2^5 bits). Essentially it's a trade-

off between precision, accuracy, and speed. When representing real numbers in a digital system with finite precision, there's a delicate balance to be struck. Typically increasing precision requires more bits, which in turn increases memory usage and computational complexity. However, this can lead to improved accuracy in calculations. On the other hand, reducing precision can save memory and speed up computations, but it may sacrifice accuracy, especially in numerical calculations involving small or large numbers, or in operations prone to rounding errors. Therefore, finding the optimal balance depends on the specific requirements and constraints of the problem at hand.

The most suitable method for meeting the aforementioned requirements is scientific notation, which expresses real numbers through a mantissa, base and an exponent. For instance, Table 2.1 demonstrates the representation of the number 2.998×10^8 in scientific notation, emphasizing these three essential components. In normalized floating-point format,

Number (in Scientific Notation)	Mantissa	Base	Exponent
2.998×10^8	2.998	10	8

Table 2.1 Representation in scientific notation

which is prevalent in computing, a variation of scientific notation is employed. Similar to standard scientific notation, 32 bits are divided into three parts: one bit for the sign, a mantissa representing fractions (which begins with a non-zero leading bit), and an exponent indicating the power of base by which the mantissa is multiplied. This format represents a number as $(sign) \times base^{\pm exponent} \times mantissa$.

In general, one can express a number x as follows:

$$x = (-1)^{s_0} \beta^{\text{exponent}} (b_0.b_1 b_2 \dots b_m)_\beta \equiv (-1)^{s_0} \beta^{\text{exponent}} \left(b_0 + \frac{b_1}{\beta} + \frac{b_2}{\beta^2} + \dots + \frac{b_m}{\beta^m} \right) \quad (2.1)$$

Here, β is called the radix, and the point preceding b_1 is known as the radix point. The exponent typically has upper (U) and lower (L) limits, i.e., $L \leq \text{exponent} \leq U$. Additionally, b_0 is termed the most significant digit, while b_m is referred to as the least significant digit.

For 64-bit double-precision [28] floating-point representation, the mantissa is kept at 52 bits to ensure the representation of approximately 15-16 significant decimal digits. The exponent is given an 11-bit representation, with one bit used for the sign. This allows for exponents ranging from -1023 to 1023. To prevent issues with signed zero, a biased exponent convention is employed, where the range (0 to 2047) is divided into negative parts (-1023 to -1) and positive parts (1 to 1023). The exponent 1023 is considered as zero. The first digit

of the mantissa is kept non-zero to provide unique representations of number (note that e.g. $5.4 \times 10^{13} = 0.054 \times 10^{15}$).

Once there is such a representation, numbers can be represented over real line. There are many possible ways to represent real numbers and in initial days of computing many variations were tried. As personal computers became increasingly widespread since the 1980s, manufacturers of digital devices recognized the need for standardization. Consequently, the Institute of Electrical and Electronics Engineers (IEEE) developed and standardized the floating-point number system for computers. In next section, this standardized system, also known as IEEE 754 is outlined.

2.2 IEEE Floating Point Standard

Since 1985, the IEEE Standard 754 has set forth rules governing floating-point computation, covering number representation and defining special values like 0 and infinity (∞). The standard also outlines systematic procedures for handling exceptions such as division by zero and ensures uniformity in rounding numbers. It specifies floating-point formats across various precisions, including examples such as single precision with 32-bit representation and double precision with 64-bit representation. Additionally, in GPU architectures, 16-bit half precision is commonly utilized for speed optimization purposes. All these precision formats are standardized accordingly as shown in figure 2.1.

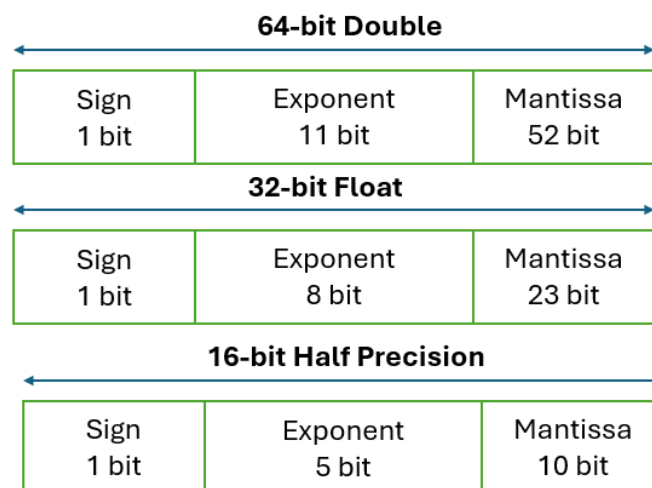


Fig. 2.1 Institute of Electrical and Electronic Engineers (IEEE) floating point standard (IEEE754)

I will only focus on double precision with 64-bit representation. This does not compromise on generality as basic idea remains the same even for other precisions (for ex. single precision 32-bit). This system represents real numbers x in double precision as

$$\begin{aligned} x &= (-1)^s 2^{d_{10}d_9 \dots d_0} \times 1.b_1 b_2 \dots b_{52} \\ &\equiv (-1)^s \left(1 + \frac{b_1}{2} + \frac{b_2}{2^2} + \dots + \frac{b_{52}}{2^{52}} \right) \times 2^{(2^{10} \times d_{10} + \dots + 2^1 \times d_1 + d_0) - 1023} \end{aligned} \quad (2.2)$$

where the implied 1 (not stored) as the most significant bit of the mantissa or significand is explicitly shown for clarity and b_i are binary digit representing mantissa and d_i represent binary digit for exponent. The exponent takes a minimum value [28] of -1022 and a maximum value of 1023 . The exponents -1023 which correspond to $d_i = 0$ and 1024 which correspond to $d_i = 1$ are reserved for representing special numbers. The rationale behind ensuring that the minimum magnitude value $|x_{\min}|$ is less than $|x_{\max}|$ is to prevent overflow when computing $\frac{1}{|x_{\min}|}$. It's important to note that exceeding the largest representable number in the computer during computation results in an overflow. Conversely, computing $\frac{1}{|x_{\max}|}$ will result in underflow, making it smaller than the smallest representable number in the computer. Underflows are generally less troublesome in numerical computing compared to overflows.

Due to the implied extra digit in mantissa, the largest number x_{\lim} which can be represented by 64 bit representation of this type can be seen from the following

$$\begin{aligned} x_{\lim} &= 2^{11111111110} \times \mathbf{1}.\underbrace{11 \dots 1}_{52 \text{ times}} \equiv \left(1 + \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^{52}} \right) \times 2^{2046-1023} \\ &\equiv \frac{1 - \frac{1}{2^{53}}}{1 - \frac{1}{2}} \times 2^{1023} \approx 1.79769313 \times 10^{308}. \end{aligned} \quad (2.3)$$

Similarly, the smallest number ϵ_0 represented by this system would be

$$\begin{aligned} \epsilon_0 &= 2^{00000000001} \times \mathbf{1}.\underbrace{00 \dots 0}_{52 \text{ terms}} \equiv 2^{1-1023} = 2^{-1022} \\ &\approx 2.225073 \times 10^{-308}. \end{aligned} \quad (2.4)$$

one shown in bold here is an implied bit in mantissa and the decimal point is also implied and not stored as information.

It would be useful to also compute the next largest number

$$\begin{aligned}\epsilon_1 &= 2^{00000000001} \times \underbrace{1.00 \dots 1}_{52 \text{ terms}} \equiv 2^{1-1023} \left(1 + 2^{-52}\right) = \epsilon_0 + 2^{-1075} \\ &\approx \epsilon_0 + 2.47 \times 10^{-324}.\end{aligned}\tag{2.5}$$

Similarly, it might be worthwhile at this point to ask: What is the number just smaller and larger than 1? Is it again going to be a gap of 10^{-324} ? To answer this, one can recall that the for double in the IEEE754 format mantissa for 1 is $1.\underbrace{00 \dots 0}_{52 \text{ times } 0}$ and for next number $1 + \epsilon$ is $1.\underbrace{00 \dots 0}_{51 \text{ times } 0} 1$. Thus the gap is $2^{-52} \approx 2.2204 \times 10^{-16}$. Similarly the number y just smaller than one can be estimated from the fact that the mantissa for number $y \approx 0.99999$ which is just smaller than 1 is $1.\underbrace{1 \dots 1}_{52 \text{ times } 1}$ and the exponent is 2^{-1} . When one compute

$$1 - y = 1 - 2^{-1} \left(1 + 2^{-1} + \dots + 2^{-52}\right)$$

the result is $2^{-53} \approx 1.1102 \times 10^{-16}$ which essentially illustrates that in floating point representation, numbers are not equally spaced.

Another important aspect of this number system is treatment of special numbers. Thus, it is instructive to spend some time analyzing these numbers. In this representation, few of the special numbers are mentioned in table 2.2.

Value	Sign	Exponent (11 bits)	Significand (52 bits)
+0	0	00...00	00...00
-0	1	00...00	00...00
$+\infty$	0	11...11	00...00
$-\infty$	1	11...11	00...00

Table 2.2 Zero and Infinity representation for 64-bit double in IEEE 754 Standard

Also, there are some more important special numbers represented as follows:

- **Representation of Non Numbers (NaN):** It is very important to decide what to do when an illegal operation such as the computation of $0/0$ is encountered. One simple solution is to treat these as an unrecoverable error which causes a halt in computation. However, this might not be always a desirable thing. The result of an illegal floating

point operation is termed as NAN in IEEE754 standard and computation is not halted. For example, if zero is divided by zero, the result is NAN. According to the standard IEEE754, when the result of a floating point operation is indeterminate, it is called a quiet NaN (QNaN). Operations like $0/0$ or ∞/∞ or $\sqrt{-1}$ over set of real numbers are example of NAN. Quiet NaNs appears due to undefined floating point operations and they get normally carried over in the computation.

$$(Q)NAN = \underbrace{(0, 1)}_{\text{signed single bit}} \underbrace{11 \dots 11}_{11 \text{ terms of Exponent}} \mathbf{1.} \underbrace{0 \dots 1 \dots 0}_{51 \text{ zero terms of Mantissa}} \quad (2.6)$$

The signature of QNaN is that most significant bit of the mantissa (significand) is zero and there exist at least one 1 in the rest of the mantissa.

IEEE standard has a second type of NAN termed as a Signalling Nan (SNaN) which is used to signal an error. For example, when after a floating point operation an underflow or overflow happens, SNaN is used. Similarly, when an operation is done without a valid value in the variable (undefined) SNaN would be the result.

$$(S)NAN = \underbrace{(0, 1)}_{\text{signed single bit}} \underbrace{11 \dots 11}_{11 \text{ terms of Exponent}} \mathbf{1.} \underbrace{10 \dots 1 \dots 0}_{\text{Mantissa}} \quad (2.7)$$

The signature of SNaN is that most significant bit of the mantissa is one.

- **Representation of Subnormal Numbers:** For small numbers, due to finite representation one need to think through what to do with the condition $x - y = 0 \implies x = y$. This is an issue which can be understood by considering an example of a number system where smallest representable number is around 2×10^{-308} and hence, $x = 5.38 \times 10^{-307}$ and $y = 5.31 \times 10^{-307}$ whose difference is $y = 7.0 \times 10^{-309}$, a number much smaller than the smallest representable number. So due to underflow, one will get a strange relation $x - y = 0$ even though, clearly $x \neq y$. The IEEE standard uses denormalized or subnormal numbers, which guarantee $x - y = 0 \implies x = y$. The leading bit of the significand assumed to be 1 allowing for a more efficient use of the available bits known as normalization sacrifices the ability to represent very small numbers with high precision. Subnormal numbers, also known as gradual underflow, are a special case where the leading bit of the significand is 0. Thus, the subnormal number z are defined as the number in the set given by following binary sequence

$$z = \underbrace{(0, 1)}_{\text{signed single bit}} \underbrace{00 \dots 00}_{11 \text{ terms of Exponent}} \mathbf{0.} \underbrace{b_1 \dots b_{52}}_{\text{Mantissa}} \quad (2.8)$$

This allows for the representation of numbers smaller than the smallest normalized number. Subnormal numbers help extend the range of representable numbers closer to zero and improve numerical precision for very small values.

2.3 Error Analysis

One significant outcome of representing an infinite set of real numbers using a limited number of bits is the necessity for approximations in representation. For any floating point number x represented as

$$x = (-1)^{s_0} \beta^{\text{exponent}} (0.b_1 b_2 \cdots b_t \cdots)_{\beta}, \quad (2.9)$$

its floating point representation is often given as

$$\text{fl}(x) = (-1)^{s_0} \beta^{\text{exponent}} (0.b_1 b_2 \cdots b'_t)_{\beta}, \quad (2.10)$$

where b'_t depends on implementation and decided either by rounding or chopping. At this juncture, it's worth noting that in the IEEE standard, if a floating-point number is represented with a mantissa of m bits, the total number of bits used denoted as t is $m + 1$ because of the inclusion of the implicit significant bit, $b_1 = 1$. For example, for a 64-bit double, $t = 53$ and for a 32-bit float, $t = 24$ respectively.

The notion of “unit in last place” or **ULP**, is sometimes used when describing the accuracy of a floating point result. It is defined as

$$1\text{ULP} = \beta^{\text{exponent}} \underbrace{(0.00 \cdots 1)}_{t \text{ bit}}_{\beta} \equiv \beta^{\text{exponent}-t} \quad (2.11)$$

Thus, it can be seen that one ULP is essentially distance between adjacent floating-point numbers. Floating-point calculations frequently necessitate rounding or truncation to ensure they can be accommodated within their finite representation. The error introduced due to this approximation (rounded or chopped) is termed as “Round-off errors”. For any floating point number x the floating -point representation $\text{fl}(x)$ in two ways:

- **Chopping:** Given m -bit representation, chopping a number implies that only the first m digits of the mantissa are retained, simply chopping off the remainder. This means the representation is

$$\text{fl}(x) = (-1)^{s_0} \beta^{\text{exponent}} (0.b_1 b_2 \cdots b_m)_{\beta}, \quad (2.12)$$

Thus, binary representation chopping introduces an error of

$$\text{fl}(x) - x = (-1)^{s_0} 2^{\text{exponent}} \left(\frac{b_{m+1}}{2^{m+1}} + \frac{b_{m+2}}{2^{m+2}} + \dots \right), \quad (2.13)$$

where $m = 23$ for 32-bit float and $m = 52$ for 64-bit double floating point representations respectively. Thus, one can say that

$$\begin{aligned} \frac{\text{fl}(x) - x}{x} &= \frac{\left(\frac{b_{m+1}}{2^{m+1}} + \frac{b_{m+2}}{2^{m+2}} + \dots \right)}{\left(\frac{b_1}{2^1} + \frac{b_2}{2^2} + \dots + \frac{b_m}{2^m} \right)} \\ \Rightarrow \frac{\text{fl}(x) - x}{x} &\leq \frac{\left(\frac{1}{2^{m+1}} + \frac{1}{2^{m+2}} + \dots \right)}{\left(\frac{1}{2^1} \right)} \\ \Rightarrow \frac{\text{fl}(x) - x}{x} &\leq \frac{1}{2^m} \left(1 + \frac{1}{2^1} + \dots \right) \\ \Rightarrow \frac{\text{fl}(x) - x}{x} &\leq \frac{1}{2^{m-1}}. \end{aligned} \quad (2.14)$$

Notice that $m = 24$ for the single precision numbers. Also notice that from equation 2.13, it can be shown that

$$|\text{fl}(x) - x| = 2^{\text{exponent}-m} \left(\frac{b_{m+1}}{2^1} + \frac{b_{m+2}}{2^2} + \dots \right) \leq 2^{\text{exponent}-m+1}, \quad (2.15)$$

which implies

$$|\text{fl}(x) - x| \leq \text{ULP}, \quad (2.16)$$

where $\text{ULP} = 2^{\text{exponent}-t}$.

- **Rounding:** In this method of rounding a real number, one replaces a given number by the nearest p significant digit number, with a set of rules for breaking ties if there is an ambiguity. This means the representation is

$$\text{fl}(x) = \begin{cases} (-1)^{s_0} \beta^{\text{exponent}} (0.b_1 b_2 \dots b_m)_\beta, & \text{if } 0 \leq b_{m+1} < \frac{\beta}{2} \\ (-1)^{s_0} \beta^{\text{exponent}} \left[(0.b_1 b_2 \dots b_m)_\beta + (0.00 \dots 1)_\beta \right], & \text{if } \frac{\beta}{2} \leq b_{m+1} < \beta \end{cases} \quad (2.17)$$

It is evident that error in this case is

$$-\frac{1}{2^m} \leq \frac{\text{fl}(x) - x}{x} \leq \frac{1}{2^m}, \quad (2.18)$$

In terms of ULP, for rounding

$$|\text{fl}(x) - x| \leq \frac{ULP}{2}. \quad (2.19)$$

Typically floating point error for $x \neq 0$ is quantified in terms of the relative error ε

$$\varepsilon = \frac{\text{fl}(x) - x}{x}. \quad (2.20)$$

from above discussion see that $\varepsilon \leq 2^{1-m}$ for truncation and for rounding one can have the bound as $-2^{-m} \leq \varepsilon \leq 2^{-m}$. Sometimes, it might be better to talk in terms of the absolute error $(\text{fl}(x) - x)$ too. This can also be re-written as

$$\text{fl}(x) = x(1 + \varepsilon), \quad |\varepsilon| \leq u \quad (2.21)$$

with

$$u = \begin{cases} \leq ULP & \text{if chopping,} \\ \leq \frac{ULP}{2} & \text{if rounding.} \end{cases} \quad (2.22)$$

2.3.1 Error Model for Floating point operations

In this section, the primary interest is the quantification of errors due to floating point operations. The fact that numbers are represented approximately only, lead to error in result of all operations. For any operation op between two numbers a and b , it is assumed that $\text{fl}(a \text{ op } b)$ is being computed. This is the basic model of computing. From this it follows,

$$\text{fl}(a \text{ op } b) = (a \text{ op } b)(1 + \varepsilon), \quad |\varepsilon| \leq u. \quad (2.23)$$

This is also sometime written as

$$(a \text{ op } b) = \frac{\text{fl}(a \text{ op } b)}{(1 + \varepsilon)}, \quad |\varepsilon| \leq u. \quad (2.24)$$

The model essentially assumes that the computed value of the floating point operation is the rounded value of actual answer. In other words, the floating-point $\text{sum}(\oplus)$, $\text{difference}(\ominus)$,

multiplication \otimes and division \oplus are

$$\begin{aligned}
 \tilde{x} \oplus \tilde{y} &= (x(1 + \epsilon_x) + y(1 + \epsilon_y))(1 + \epsilon^+) = (x + y) \left(1 + \frac{x\epsilon_x + y\epsilon_y}{x + y}\right) (1 + \epsilon^+) \\
 \tilde{x} \ominus \tilde{y} &= (x(1 + \epsilon_x) - y(1 + \epsilon_y))(1 + \epsilon^-) = (x - y) \left(1 + \frac{x\epsilon_x - y\epsilon_y}{x - y}\right) (1 + \epsilon^-) \\
 \tilde{x} \otimes \tilde{y} &= (x(1 + \epsilon_x) \times y(1 + \epsilon_y))(1 + \epsilon^*) = (x \times y) (1 + \epsilon_x + \epsilon_y + \epsilon_x \times \epsilon_y) (1 + \epsilon^*) \\
 \tilde{x} \oplus \tilde{y} &= (x(1 + \epsilon_x) \div y(1 + \epsilon_y))(1 + \epsilon^\div) \approx (x \div y) (1 + \epsilon_x - \epsilon_y - \epsilon_x \times \epsilon_y) (1 + \epsilon^\div)
 \end{aligned} \tag{2.25}$$

Thus, one can get the correctly rounded real sum/difference of approximate data. One can also compute relative error in these quantities as

$$\begin{aligned}
 \epsilon_\oplus &= \epsilon^+ + \left(\frac{x\epsilon_x + y\epsilon_y}{x + y}\right) (1 + \epsilon^+) \\
 \epsilon_\ominus &= \epsilon^- + \left(\frac{x\epsilon_x - y\epsilon_y}{x - y}\right) (1 + \epsilon^+) \\
 \epsilon_\otimes &= \epsilon^* + (\epsilon_x + \epsilon_y + \epsilon_x \times \epsilon_y) (1 + \epsilon^*) \\
 \epsilon_\oplus &= \epsilon^\div + (\epsilon_x - \epsilon_y - \epsilon_x \times \epsilon_y) (1 + \epsilon^\div)
 \end{aligned} \tag{2.26}$$

Thus, for positive numbers addition is benign operation and subtraction can be catastrophic if $x \sim y$. In other words, one needs to be careful while subtracting two near by number of same sign and adding two near by (in magnitude) numbers of opposite sign. Also, it can be seen that multiplication and division are benign operation and only getting rounded due to approximation involved in the floating point computations.

Corollary 2.3.0.1. *For two vectors, x , y , and scalar α , it can be said that*

$$fl(x - \alpha y) = x - \alpha y - \delta y, \quad \|\delta y\| \leq (\|x\| + 2\|\alpha y\|) \epsilon_1 \tag{2.27}$$

where $\epsilon_1 \approx \epsilon$.

Proof.

$$fl(x - \alpha y) = (x_i - \alpha y_i (1 + \epsilon_i^*)) (1 + \epsilon_i^-) \tag{2.28}$$

Also, $fl(x - \alpha y)$ can be written as $(x - \alpha y - \delta y)$,

$$\text{where } \delta y_i = x_i \epsilon_i^- - \alpha y_i \epsilon_i^- - \alpha y_i \epsilon_i^* - \alpha y_i (\epsilon_i^- \epsilon_i^*) \tag{2.29}$$

Also, $\varepsilon_i^- \approx \varepsilon_1$, $\varepsilon_i^* \approx \varepsilon_1$ and $(\varepsilon_i^- \varepsilon_i^*) \ll \varepsilon_1$.

$$\delta y_i := x_i \varepsilon_1 - \alpha y_i \varepsilon_1 - \alpha y_i \varepsilon_1 = (x_i - 2\alpha y_i) \varepsilon_1 \quad (2.30)$$

Using, $\|a - b\| \leq \|a\| + \|b\|$,

$$\|x - 2\alpha y\| \leq \|x\| + 2\|\alpha y\| \quad (2.31)$$

$$\|(x - 2\alpha y) \varepsilon_1\| \leq (\|x\| + 2\|\alpha y\|) \varepsilon_1. \quad (2.32)$$

$$\text{Hence, } \|\delta y\| \leq (\|x\| + 2\|\alpha y\|) \varepsilon_1. \quad (2.33)$$

□

2.3.2 Error in Dot product

Finally, let us look at error propagation in computing dot product from vectors in floating point numbers. Here, the focus is on error due to operations only. The actual expression for two vectors x and y is $s = x_1 y_1 + x_2 y_2 + x_3 y_3 + \dots$. The error can be computed by noticing that at every stage an error is introduced according to rule $\text{fl}(a \text{ op } b) = (a \text{ op } b) (1 + \varepsilon)$. For example in stage one, it can be written as

$$\tilde{s}_1 = x_1 \otimes y_1 = x_1 y_1 (1 + \varepsilon_1^*). \quad (2.34)$$

where ε_1^* is error due to finite precision multiplication of x_1 and y_1 . In next stage, one have to compute $x_2 y_2$ and add it to \tilde{s}_1 . In terms of addition error ε_2^+ , and error ε_2^* due to finite precision multiplication of x_2 and y_2

$$\tilde{s}_2 = \tilde{s}_1 \oplus x_2 \otimes y_2 = [x_1 y_1 (1 + \varepsilon_1^*) + x_2 y_2 (1 + \varepsilon_2^*)] (1 + \varepsilon_2^+). \quad (2.35)$$

Similarly, in next stage one have

$$\tilde{s}_3 = \tilde{s}_2 \oplus x_3 \otimes y_3 = [[x_1 y_1 (1 + \varepsilon_1^*) + x_2 y_2 (1 + \varepsilon_2^*)] (1 + \varepsilon_2^+) + x_3 y_3 (1 + \varepsilon_3^*)] (1 + \varepsilon_3^+) \quad (2.36)$$

Thus, defining $\varepsilon_1^+ = 0$, one can write the general term in the sequence as

$$\tilde{s}_n := x_1 y_1 (1 + \Theta_1^*) + x_2 y_2 (1 + \Theta_2^*) + x_3 y_3 ((1 + \Theta_3^*) + \dots \quad (2.37)$$

where

$$1 + \Theta_i = (1 + \epsilon_i^*) \prod_{k=i}^n (1 + \epsilon_k^+) \equiv (1 + \epsilon_i^*) \frac{\prod_{k=1}^n (1 + \epsilon_k^+)}{\prod_{k=1}^{i-1} (1 + \epsilon_k^+)} \quad (2.38)$$

A particularly elegant way to simplify this expression is to use the inequality

Theorem 2.3.1.

$$\boxed{|\Theta_i| \leq \frac{nu}{|1 - \frac{nu}{2}|}} \quad (2.39)$$

Proof. Also it is known that in terms of unit roundoff u , one have

$$\left| \prod_{k=1}^n (1 + \epsilon_k^+) - 1 \right| \leq |(1 + u)^n - 1| \leq |\exp(nu) - 1| \quad (2.40)$$

where the fact that for all $x \geq 0$, $1 + x < \exp(x)$ is used. Thus, the previous equation can be written as

$$\begin{aligned} \left| \prod_{k=1}^n (1 + \epsilon_k^+) - 1 \right| &\leq \left| nu + \frac{(nu)^2}{2} + \frac{(nu)^3}{6} + \frac{(nu)^4}{24} + \dots \right| \\ &\leq |nu| \left| 1 + \frac{(nu)}{2} + \frac{(nu)^2}{4} + \frac{(nu)^3}{8} + \dots \right| \\ &\leq \frac{nu}{|1 - \frac{nu}{2}|} \end{aligned}$$

which allow us to write $\boxed{|\Theta_i| \leq \frac{nu}{|1 - \frac{nu}{2}|}}$. □

Thus it can be seen that for two vectors \mathbf{x} and \mathbf{y} , one have

$$|\text{fl}(\mathbf{x} \cdot \mathbf{y}) - (\mathbf{x} \cdot \mathbf{y})| \leq \sum_{i=1}^n |x_i y_i| \frac{nu}{|1 - \frac{nu}{2}|} \leq |\mathbf{x}| |\mathbf{y}| \frac{nu}{|1 - \frac{nu}{2}|} \quad (2.41)$$

In terms of the relative error, one have

$$\frac{|\text{fl}(\mathbf{x} \cdot \mathbf{y}) - (\mathbf{x} \cdot \mathbf{y})|}{|\mathbf{x} \cdot \mathbf{y}|} \leq \frac{|\mathbf{x}| |\mathbf{y}|}{|\mathbf{x} \cdot \mathbf{y}|} \frac{nu}{|1 - \frac{nu}{2}|} \quad (2.42)$$

Thus, it can be seen that if $\mathbf{x} = \mathbf{y}$, one have very high accuracy. However, the error is bounded by $|\mathbf{x}| |\mathbf{y}|$. Thus, for the case, $|\mathbf{x} \cdot \mathbf{y}| \ll |\mathbf{x}| |\mathbf{y}|$, there is very low accuracy.

An alternate way to write the same expression is

Corollary 2.3.1.1.

$$\text{fl}(x^T y) = (x + \delta x)^T y, \quad \boxed{\|\delta x\| \leq \frac{nu}{|1 - \frac{nu}{2}|} \|x\|} \quad (2.43)$$

Corollary 2.3.1.2. *If A is a matrix with element a_{ij} for which there are at most m non zero elements per row of A , and $|A|$ is a matrix with elements $|a_{ij}|$, then we have*

$$\text{fl}(Ay) = (A + \delta A)^T y, \quad \boxed{|\delta A| \leq \frac{mu}{|1 - \frac{mu}{2}|} |A|} \quad (2.44)$$

It will be assumed that

$$\|A\| = \sigma, \quad \| |A| \| = \beta \sigma \quad (2.45)$$

where $\|\cdot\|$ represents the 2-norm, and $|A|$ is the matrix with elements $|\alpha_{ij}|$, α_{ij} being the elements of A .

Corollary 2.3.1.3. *Assume that taking square roots introduces a relative error no greater than u ,*

$$\beta = \text{fl} \left(\sqrt{\text{fl}(w^T w)} \right) = \left[1 + \frac{n+2}{2} u_3 \right] \|w\| \quad (2.46)$$

where $|u_3| < u$.

Proof. : We will be using Bernoulli's inequality

$$\sqrt{1+x} \leq 1 + \frac{x}{2}, \quad x \geq -1 \quad (2.47)$$

Recall that Bernoulli's inequality can be proven using the fact that geometric mean of 1 and $1+x$ is $\sqrt{1+x}$ while arithmetic mean is $1+x/2$. As arithmetic mean is always greater than geometric mean, we have Bernoulli's inequality for square root. This also mean

$$\sqrt{1+x} := 1 + \frac{y}{2}, \quad |y| \leq |x| \quad (2.48)$$

We have

$$\text{fl}(w^T w) = \|w\|^2 + w_1^2 \Theta_1 + w_2^2 \Theta_2 + w_3^2 \Theta_3 + \dots \quad (2.49)$$

with

$$|\Theta_i| \leq \frac{nu}{|1 - nu/2|} \quad (2.50)$$

$$\implies \text{fl}(w^T w) := (1 + nu_1) \|w\|^2. \quad (2.51)$$

where $|u_1| < u$. So, we have

$$\text{fl}\left(\sqrt{\text{fl}(w^T w)}\right) = \sqrt{\text{fl}(w^T w)} (1 + u_2) \quad (2.52)$$

where $|u_2| < u$. So, we have

$$\beta = \text{fl}\left(\sqrt{\text{fl}(w^T w)}\right) = \sqrt{1 + nu_1} (1 + u_2) \|w\| \leq \left(1 + \frac{nu_1}{2}\right) (1 + u_2) \|w\| \quad (2.53)$$

$$\beta \leq \left(1 + \frac{nu_1 + 2u_2 + nu_1 u_2}{2}\right) \|w\| \leq \left(1 + \frac{nu_1}{2} + u_2\right) \|w\| \quad (2.54)$$

which gives

$$\beta = \text{fl}\left(\sqrt{\text{fl}(w^T w)}\right) := \left(1 + \frac{n+2}{2}u_3\right) \|w\| \quad (2.55)$$

Where $|u_3| \leq u$. □

Corollary 2.3.1.4. *Assuming $D(f(\epsilon))$ will be used to represent a diagonal matrix each of whose diagonal elements is bounded by $f(\text{relative precision})$.*

$$v = fl(w/\beta) = D(1 + \epsilon)w/\beta \quad (2.56)$$

Proof. Using $\tilde{x} \oplus \tilde{y} = (x/y)(1 + \epsilon)$ for two numbers, x and y

$$v_i = fl(w/\beta) = \begin{bmatrix} 1 + \epsilon & 0 & 0 & \cdots & 0 \\ 0 & 1 + \epsilon & 0 & \cdots & 0 \\ 0 & 0 & 1 + \epsilon & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 + \epsilon \end{bmatrix} w_i/\beta$$

$$v = fl(w/\beta) = D(1 + \epsilon)w/\beta$$

□

Corollary 2.3.1.5. *The corollary essentially highlights that the floating-point error involved while computing norm of a vector is very small, as it is only n times the machine epsilon ϵ as*

shown. Therefore, the norm of a vector can be computed very accurately.

$$v^T v = (1 + 2\varepsilon) w^T w / \beta^2 := 1 + (n + 4)\varepsilon \quad (2.57)$$

Proof.

$$v^T v = (1 + 2\varepsilon) w^T w / \beta^2 \quad (2.58)$$

Using $\beta := (1 + \frac{n+2}{2} u_3) \|w\|$, it can be shown that

$$(1 + 2\varepsilon) w^T w / \beta^2 := \frac{1 + 2\varepsilon}{(1 + \frac{n+2}{2} u_2)^2} \quad (2.59)$$

$$\text{Using Bernoulli's inequality for square root, } \sqrt{1+x} := 1 + \frac{y}{2}, \quad |y| \leq |x| \quad (2.60)$$

$$\frac{1}{(1 + \frac{x}{2})^2} = \frac{1}{(1 + x + \frac{x^2}{4})} = 1 - \frac{(x + \frac{x^2}{4})}{(1 + x + \frac{x^2}{4})} \quad (2.61)$$

$$\frac{1}{(1 + \frac{x}{2})^2} := 1 - 2y, \text{ where } y \approx \frac{(\frac{x}{2} + \frac{x^2}{8})}{(1 + x + \frac{x^2}{4})} \quad (2.62)$$

$$\text{Therefore, } v^T v := (1 + 2\varepsilon) (1 - (n + 2) u_3). \quad (2.63)$$

Then choosing $u_4 = -u_3$,

$$v^T v := (1 + 2\varepsilon) (1 + (n + 2) u_4) = 1 + (n + 4)\varepsilon \quad (2.64)$$

where $|\varepsilon| < u$.

□

All the error bound [29] quantities, $\varepsilon_1 \approx \varepsilon$, $u \approx \varepsilon$ and $u_3 \approx \varepsilon$. Summarizing all the useful corollaries which will be later used in the rounding analysis are:

1. Vector subtraction

$$\text{fl}(x - \alpha y) = x - \alpha y - \delta w, \quad \|\delta w\| \leq (\|x\| + 2\|\alpha y\|) \varepsilon \quad (2.65)$$

2. Vector inner-product

$$\text{fl}(v^T u) = (v + \delta v)^T u, \quad \|\delta v\| \leq n \varepsilon \|v\| \quad (2.66)$$

3. Matrix-vector multiplication If there are at most m non-zero elements per row of A , then

$$\text{fl}(Au) = (A + \delta A)^T u, \quad |\delta A| \leq m \varepsilon |A|, \quad (2.67)$$

$$\text{and } \|\delta A\| \leq \|\delta A\| \leq m \varepsilon \|A\| = m \beta \varepsilon \sigma. \quad (2.68)$$

4. Normalization

$$\beta = \text{fl} \left(\sqrt{\text{fl}(w^T w)} \right) = \left[1 + \frac{n+2}{2} \varepsilon \right] \|w\| \quad (2.69)$$

$$v = \text{fl}(w/\beta) = D(1 + \varepsilon)w/\beta \quad (2.70)$$

$$v^T v = (1 + 2\varepsilon)w^T w/\beta^2 := 1 + (n+4)\varepsilon \quad (2.71)$$

2.4 Forward and Backward Error Analysis

Once an approximation \hat{f} to $f(x)$ is computed with a given precision, there is a need to comment on the confidence in the answer. In other words, one need to assess the quality of the numerical approximation. If one have such an estimate, the absolute and relative errors from the true quantity is termed as the forward errors and one wish to reduce this error to as small as possible. In ideal case, this can be reduced to unit roundoff u . However, many times it is not feasible. An alternate approach is to assume that discrete answer is true answer but for a different data set. In other words, it can be said that $\hat{f} = f(x + \delta x)$ for some δx . Then one need to ask that what is the value of δx for which $\hat{f} = f(x + \delta x)$. As for an example, for two floating point numbers x and y , it was seen that approximate multiplication is $x \otimes y = x \times y(1 + \varepsilon^*)$ where $|\varepsilon^*| \leq u$. In this case, ε^* is the relative forward error. One can also follow the backward error approach and say that $x \otimes y = x \times \hat{y}$, where $\hat{y} = y(1 + \varepsilon^*)$.

Any floating point operation is called backward stable if, for any input data set it produces a computed answer with a small backward error. However, in general the definition of “small” is context dependent. In general, a given problem has several methods of solution, some of which are backward stable and some not.

Chapter 3

Gram-Schmidt Orthogonalization

3.1 Introduction

Gram-Schmidt orthogonalization [18] is a method in linear algebra that transforms a set of linearly independent vectors into a set of orthogonal vectors. This technique is essential for tasks like finding orthonormal basis, solving linear systems, and performing computations involving inner products or projections. Gram-Schmidt orthogonalization is used routinely in various fields, including physics, engineering, and computer science, where orthogonal basis simplify calculations and analysis. This method also plays a crucial role in Krylov subspace methods for eigenvalue solutions since an orthogonal basis is a preliminary requirement in those methods. Gram-Schmidt orthogonalization will be employed within the Lanczos algorithm, which forms the central focus of this thesis.

This chapter starts with an overview of the procedure involved in Gram-Schmidt orthogonalization along with an example solved in infinite precision in section 3.2 showcasing the strength of this powerful tool. In this example, an orthogonal set of vectors is obtained from an independent set of vectors. Section 3.3 illustrates the numerical instability issue involved in finite precision resulting in the loss of orthogonality for the same set of vectors encountered in the previous section. To achieve numerically stable results, the same algorithm is repeated once again as shown in section 3.4. Also, there is this alternate version of Gram-Schmidt formulation which is discussed in section 3.5. This is the matrix version of Gram-Schmidt orthogonalization and is the most stable one. Yet, this variant holds limited practical significance due to its high computational overhead. Finally, a brief insight on the method of repeating the Gram-Schmidt algorithm twice i.e, Reorthogonalization is provided in the section 3.6 which is essentially one of the computationally expensive remedies to deal with the convergence issues involved in the Lanczos algorithm.

3.2 Gram-Schmidt Orthogonalization

Let \mathcal{V} be a vector space with a given definition of the inner product between two vectors x and y denoted as $\langle x, y \rangle$. For this vector space, given an arbitrary basis $\{x_1, x_2, x_3, \dots, x_N\}$, this method provides a method to create an alternate set of basis vectors $\{v_1, v_2, v_3, \dots, v_N\}$ which form orthonormal basis set. The basic idea is to iterate over vectors one by one and subtract out the projection of all the previous one. The resultant is then orthogonal to all the previous vectors.

$$\begin{aligned}
 v_1 &= \frac{x_1}{\|x_1\|}, \\
 v'_2 &= x_2 - \langle v_1, x_2 \rangle v_1, \quad v_2 = \frac{v'_2}{\|v'_2\|}, \\
 v'_3 &= x_3 - \langle v_1, x_3 \rangle v_1 - \langle v_2, x_3 \rangle v_2, \quad v_3 = \frac{v'_3}{\|v'_3\|}, \\
 &\vdots \\
 v'_k &= x_k - \langle v_1, x_k \rangle v_1 - \langle v_2, x_k \rangle v_2 - \dots - \langle v_{k-1}, x_k \rangle v_{k-1}, \quad v_k = \frac{v'_k}{\|v'_k\|},
 \end{aligned} \tag{3.1}$$

It is evident that v_i forms an orthogonal set. For example,

$$\langle v_2, v_1 \rangle = \frac{1}{\|v'_2\|} (\langle x_2, v_1 \rangle - \langle v_1, x_2 \rangle) = 0. \tag{3.2}$$

and

$$\langle v_3, v_1 \rangle = \frac{1}{\|v'_3\|} (\langle x_3, v_1 \rangle - \langle v_1, x_3 \rangle - \langle v_2, x_3 \rangle \langle v_2, v_1 \rangle) = 0. \tag{3.3}$$

Similarly, for any $j < k$, we have

$$\langle v_k, v_j \rangle = 0. \tag{3.4}$$

This process of obtaining an orthonormal set of vectors is called the Gram-Schmidt orthogonalization [18]. The Gram-Schmidt process involves computing dot products of vectors and each dot product involves $2n - 1$ operations. Also, computing norm of a vector involves $O(n)$ operations. This is because the total number of operations is dominated by the n multiplications and $(n - 1)$ additions, both of which are linear in the size of the vector n . The square root operation for most practical purposes is treated as a constant time operation, $O(1)$.

So, in computing a vector at k th stage, one would need to subtract the previous $k - 1$ vectors after computing the inner products which is $O(kn)$ operations. Thus, for computing a set of k – orthonormal vectors the computational cost involved is typically $O(k^2n)$ operations.

Let us consider an example to illustrate the algorithm. The same example, we shall use in subsequent section to explain the difficulties in numerical implementations in finite precision. Let $\mathcal{V}^{(4)}$ be a four dimensional vector-space equipped with standard Euclidian inner product. In this space, one shall consider four basis vectors

$$x_1 = \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} \quad x_2 = \begin{pmatrix} 1 \\ 0 \\ \varepsilon \\ 0 \end{pmatrix} \quad x_3 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \varepsilon \end{pmatrix} \quad x_4 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (3.5)$$

with ε being some arbitrary number. According to the Gram–Schmidt Orthogonalization procedure, we have

$$\begin{aligned} v_1 &= \begin{pmatrix} \frac{1}{\sqrt{1+\varepsilon^2}} \\ \frac{\varepsilon}{\sqrt{1+\varepsilon^2}} \\ 0 \\ 0 \end{pmatrix}, \\ v'_2 &= \begin{pmatrix} 1 \\ 0 \\ \varepsilon \\ 0 \end{pmatrix} - \frac{1}{\sqrt{1+\varepsilon^2}} \begin{pmatrix} \frac{1}{\sqrt{1+\varepsilon^2}} \\ \frac{\varepsilon}{\sqrt{1+\varepsilon^2}} \\ 0 \\ 0 \end{pmatrix} = \frac{\varepsilon}{1+\varepsilon^2} \begin{pmatrix} \varepsilon \\ -1 \\ 1+\varepsilon^2 \\ 0 \end{pmatrix}, \\ \Rightarrow v_2 &= \frac{1}{\sqrt{1+\varepsilon^2}\sqrt{2+\varepsilon^2}} \begin{pmatrix} \varepsilon \\ -1 \\ 1+\varepsilon^2 \\ 0 \end{pmatrix}, \end{aligned} \quad (3.6)$$

$$\begin{aligned}
v'_3 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ \varepsilon \end{pmatrix} - \begin{pmatrix} \frac{1}{1+\varepsilon^2} \\ \frac{\varepsilon}{1+\varepsilon^2} \\ 0 \\ 0 \end{pmatrix} - \frac{\varepsilon}{(1+\varepsilon^2)(2+\varepsilon^2)} \begin{pmatrix} \varepsilon \\ -1 \\ 1+\varepsilon^2 \\ 0 \end{pmatrix} \\
&= \frac{\varepsilon}{(1+\varepsilon^2)(2+\varepsilon^2)} \left[\begin{pmatrix} \varepsilon(2+\varepsilon^2) \\ -(2+\varepsilon^2) \\ 0 \\ (1+\varepsilon^2)(2+\varepsilon^2) \end{pmatrix} - \begin{pmatrix} \varepsilon \\ -1 \\ 1+\varepsilon^2 \\ 0 \end{pmatrix} \right], \\
&= \frac{\varepsilon}{(2+\varepsilon^2)} \begin{pmatrix} \varepsilon \\ -1 \\ -1 \\ 2+\varepsilon^2 \end{pmatrix}, \\
\Rightarrow v_3 &= \frac{v'_3}{\|v'_3\|} = \frac{1}{\sqrt{2+\varepsilon^2}\sqrt{3+\varepsilon^2}} \begin{pmatrix} \varepsilon \\ -1 \\ -1 \\ 2+\varepsilon^2 \end{pmatrix},
\end{aligned} \tag{3.7}$$

$$\begin{aligned}
v'_4 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \frac{1}{\sqrt{1+\epsilon^2}} \begin{pmatrix} \frac{1}{\sqrt{1+\epsilon^2}} \\ \frac{\epsilon}{\sqrt{1+\epsilon^2}} \\ 0 \\ 0 \end{pmatrix} - \frac{\epsilon}{(1+\epsilon^2)(2+\epsilon^2)} \begin{pmatrix} \epsilon \\ -1 \\ 1+\epsilon^2 \\ 0 \end{pmatrix} - \frac{\epsilon}{(2+\epsilon^2)(3+\epsilon^2)} \begin{pmatrix} \epsilon \\ -1 \\ -1 \\ 2+\epsilon^2 \end{pmatrix} \\
v'_4 &= \frac{\epsilon}{3+\epsilon^2} \begin{pmatrix} \epsilon \\ -1 \\ -1 \\ -1 \end{pmatrix} \quad \text{Again } v_4 = \frac{v'_4}{||v'_4||}, \\
v_4 &= \frac{1}{\sqrt{3+\epsilon^2}} \begin{pmatrix} \epsilon \\ -1 \\ -1 \\ -1 \end{pmatrix}.
\end{aligned} \tag{3.8}$$

Therefore, it can be shown that the inner products $\langle v_2, v_1 \rangle, \langle v_3, v_1 \rangle, \langle v_3, v_2 \rangle, \langle v_4, v_1 \rangle, \langle v_4, v_2 \rangle$ and $\langle v_4, v_3 \rangle$ are all zero. Also the vectors v_1, v_2, v_3 and v_4 are normalized vectors. Hence, one can obtain an orthonormal set of vectors $\{v_1, v_2, v_3, v_4\}$ from an arbitrary set of four vectors i.e., $\{x_1, x_2, x_3, x_4\}$ using Gram-Schmidt orthogonalization in an infinite precision scenario. However, in finite precision arithmetic there are some issues involved which results in numerical instability of the Gram-Schmidt algorithm which will be discussed in next section.

3.3 Finite Precision Effect

In this section, the effect of finite precision computing is shown to illustrate the instability in the basic methodology. In the present context instability implies loss of orthogonality in approximate answer. The model begins with the fact that ϵ is so small that, one have

$1 + \varepsilon^2 = 1$. Thus,

$$\begin{aligned}
 v_1 &\approx \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix}, \\
 v'_2 &= \begin{pmatrix} 1 \\ 0 \\ \varepsilon \\ 0 \end{pmatrix} - 1 \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -\varepsilon \\ \varepsilon \\ 0 \end{pmatrix}, \\
 \Rightarrow v_2 &= \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ -1 \\ 1 \\ 0 \end{pmatrix}, \\
 v'_3 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ \varepsilon \end{pmatrix} - 1 \times \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} - 0 \times \begin{pmatrix} 0 \\ -1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -\varepsilon \\ 0 \\ \varepsilon \end{pmatrix} \\
 \Rightarrow v_3 &= \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ -1 \\ 0 \\ 1 \end{pmatrix}
 \end{aligned} \tag{3.9}$$

Notice that

$$\langle v_3, v_2 \rangle = \frac{1}{2}. \tag{3.10}$$

$$\begin{aligned}
v'_4 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - 1 \times \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} - 0 \times \begin{pmatrix} 0 \\ -1 \\ 1 \\ 0 \end{pmatrix} - 0 \times \begin{pmatrix} 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} \\
\Rightarrow v_4 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{aligned} \tag{3.11}$$

Thus, the algorithm has failed in finite precision since $\langle v_3, v_2 \rangle = 0.5$. It is well known in the literature that the naive version of the algorithm is unstable in finite precision and thus unusable.

3.4 Doing the algorithm twice

After the first attempt, a set of basis vectors as an approximation to the orthogonal set is obtained as

$$y_1 = \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} \quad y_2 = \begin{pmatrix} 0 \\ -1 \\ 1 \\ 0 \end{pmatrix} \quad y_3 = \begin{pmatrix} 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} \quad y_4 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.12}$$

Once again the Gram-Schmidt procedure is followed by computing

$$\begin{aligned}
 v_1 &\approx \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix}, \\
 v'_2 &= \begin{pmatrix} 0 \\ -1 \\ 1 \\ 0 \end{pmatrix} - (-\varepsilon) \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \varepsilon \\ -1 \\ 1 \\ 0 \end{pmatrix}, \\
 \Rightarrow v_2 &= \frac{1}{\sqrt{2}} \begin{pmatrix} \varepsilon \\ -1 \\ 1 \\ 0 \end{pmatrix}, \\
 v'_3 &= \begin{pmatrix} 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} - (-\varepsilon) \times \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} \varepsilon \\ -1 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} \varepsilon \\ -1 \\ -1 \\ 2 \end{pmatrix} \\
 \Rightarrow v_3 &= \frac{1}{\sqrt{6}} \begin{pmatrix} \varepsilon \\ -1 \\ -1 \\ 2 \end{pmatrix},
 \end{aligned} \tag{3.13}$$

Notice that,

$$\langle v_3, y_2 \rangle = 0, \tag{3.14}$$

and

$$\langle v_3, v_2 \rangle = \frac{\epsilon^2}{\sqrt{12}}, \quad (3.15)$$

is a very small quantity which was 0.5 when the Gram-Schmidt process was done only once.

$$\begin{aligned} v'_4 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - 1 \times \begin{pmatrix} 1 \\ \epsilon \\ 0 \\ 0 \end{pmatrix} - 0 \times \begin{pmatrix} \epsilon \\ -1 \\ 1 \\ 0 \end{pmatrix} - 0 \times \begin{pmatrix} \epsilon \\ -1 \\ -1 \\ 2 \end{pmatrix} \\ \Rightarrow v_4 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \end{aligned} \quad (3.16)$$

Thus, using the same Gram-Schmidt procedure twice, a more numerically stable orthogonal set of vectors $\{v_1, v_2, v_3, v_4\}$ is obtained from an arbitrary set of four vectors i.e., $\{x_1, x_2, x_3, x_4\}$ in finite precision. The computational cost involved would be double than the conventional Gram-Schmidt procedure but the gain achieved is obtaining a much more stable algorithm in numerical precision.

3.5 Alternate Formulations

Therefore, in exact arithmetic, the inner products $\langle v_2, v_1 \rangle$, $\langle v_3, v_1 \rangle$ and $\langle v_3, v_2 \rangle$ are all zero, which is always not the case when computing v_1 , v_2 and v_3 in floating point arithmetic. One of the key reason being the truncation errors associated with the inner products computed while forming the Gram-Schmidt vectors using the column version. Another reason for the loss of orthogonality among Gram-Schmidt vectors is the errors associated with the vector operations in the formulation mentioned above. So, all these errors suggest to explore numerical techniques to reduce these errors.

So, one of the more stable way to compute orthogonal vectors is using the matrix-version of Gram-Schmidt orthogonalization. This technique computes vector v_1 in index notation as

$v_1^i = \frac{x_1^i}{\|x_1\|}$ and

$$v_2^i = x_2^i - \frac{x_2^i x_1^j}{\|x_1\|} \frac{x_1^i}{\|x_1\|} \text{ or} \quad (3.17)$$

$$v_2^i = \left(\delta_{ij} - \frac{x_1^i x_1^j}{\|x_1\|} \frac{1}{\|x_1\|} \right) x_2^i \text{ or} \quad (3.18)$$

$$\implies v_2^i = M_{ij} x_2^i. \quad (3.19)$$

where

$$M_{ij} = \delta_{ij} - \frac{x_1^i x_1^j}{\|x_1\|} \frac{1}{\|x_1\|}. \quad (3.20)$$

In this way, one needs to keep on multiplying the vectors x_i s with the updated matrix M in order to obtain an orthonormal set of vectors. This is a more stable way to compute an orthogonal set of vectors [30] but it comes at its own cost of storing $O(n^2)$ elements in the matrix and then, computing matrix-vector operations which is $O(n^2)$. Also, one needs to compute the inner products and norms at every step which comes at its own computational cost as mentioned in section 3.2.

So, one can attempt to find an intermediate pathway in the column version and the matrix version of the Gram-Schmidt orthogonalization where it is possible to get away by incorporating some of the properties of the matrix version itself responsible for the stability in the column version itself. A more stable version is using the column version of Gram-Schmidt orthogonalization twice as it is shown in the previous section.

3.6 Reorthogonalization in Lanczos algorithm

The numerically stable algorithm in finite precision for obtaining an orthonormal set of vectors involves using the Gram-Schmidt orthogonalization procedure two times. Usually, two times is sufficient in order to obtain the requisite orthogonal basis. This part is also utilized in implementing the Lanczos algorithm [20] in order to compute a fully orthogonal set of vectors. Usually there are some issues in the convergence of Lanczos method due to the loss of orthogonality. The prescription provided to achieve a stable method is perform the Lanczos algorithm with full reorthogonalization [18] which is essentially performing the orthogonalization step twice. However, this algorithm comes at its own large computational cost due to reorthogonalization step. The further details of this algorithm will be mentioned when discussing the issues associated with the convergence of conventional Lanczos algorithm in chapter 6.

Chapter 4

Computational methods for Eigenvalue problem

This thesis aims to develop an Eigenvalue solver based on the Lanczos algorithm. Before describing the Lanczos algorithm, a few basic concepts of computational linear algebra relevant to the design of eigenvalue solvers will be reviewed in this chapter. The computational methods to find eigenvalues are often classified as direct or iterative solvers. The set of methods, which directly compute all or a specific subset of eigenvalues of a matrix are known as direct methods and due to their cost and scaling requirements are typically used for smaller matrices or specific types of matrices where exact eigenvalues can be determined efficiently. One of the direct methods is Householder's method [19] which scales like $O(n^3)$ and due to their high computational complexity, this thesis will not be using the direct methods. Among the iterative methods, the power method is the simplest and is often used to find the dominant eigenvalue. However, subspace iteration methods are often used to compute eigenvalues and eigenvectors of large sparse matrices where direct methods may be computationally impractical. These methods are often based on iterating within Krylov subspaces, which are subspaces of the vector space generated by repeated application of the given matrix to a guess vector. Among this class of the methods, the most widely used is the Lanczos method which efficiently computes a tridiagonal approximation of the original matrix which is used to extract eigenvalues and eigenvectors. The choice of a specific solver often depends on factors such as matrix size, sparsity, symmetry, accuracy requirements, and available computational resources. The field is so mature that in practice, one leverages specialized software libraries (e.g., LAPACK, ARPACK [19]) for mature implementation of specific solvers and if needed combines different algorithms there itself to optimize the efficiency and accuracy of eigenvalue computations for various applications.

This chapter starts with a brief discussion on Rayleigh quotient [20] in section 4.1 which is used to approximate the eigenvalue and eigenvector of a symmetric positive definite square matrix. Section 4.2 discusses the conventional way to compute the dominant eigenpairs i.e. the power method [20] for any arbitrary square matrix. Also, it can be used to compute the subsequent eigenvalues which turns out to be computationally expensive as discussed later in this section. There are certain cases where the power method fails will also be discussed briefly in this section. Later on in this chapter, the fundamental basis for writing an eigenvalue algorithm i.e. similarity transformations is discussed in section 4.3. Also, a computationally attractive subspace given by Krylov [20] is mentioned.

4.1 Rayleigh quotient

A matrix A is symmetric positive definite [11] if $A_{ij} = A_{ji}$ for all elements, and all eigenvalues λ_i are positive definite, i.e., $\lambda_i > 0$. In physical applications, one often encounters symmetric positive definite matrices [20]. For any symmetric positive definite matrix A one often defines the Rayleigh quotient [20] as

$$R(A, y) = \frac{(y_i A_{ij} y_j)}{(y_k^2)}. \quad (4.1)$$

This provides a measure of the similarity between a given vector and an eigenvector associated with a matrix or an operator.

Theorem 4.1.1. *Let A be a symmetric positive definite matrix and y be a non-zero vector. Then, the Rayleigh quotient $R(A, y)$ lies between the minimum and maximum eigenvalues of A , denoted as $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$, respectively:*

$$\lambda_{\min}(A) \leq R(A, y) \leq \lambda_{\max}(A)$$

Proof. One starts with defining a convex function, $\phi = y_i A_{ij} y_j$, subjected to the constraint that the magnitude of vector y is unity i.e. $y_i^2 = 1$. Thus, a function ψ given by

$$\psi = y_i A_{ij} y_j - \alpha (y_i^2 - 1). \quad (4.2)$$

Differentiating w.r.t. y_k :

$$\frac{\partial \psi}{\partial y_k} = 0 \implies A_{kj} y_j - \alpha y_k = 0. \quad (4.3)$$

or

$$\boxed{A_{kj} y_j = \alpha y_k.} \quad (4.4)$$

which after taking dot product with respect to y simplifies as

$$\alpha = \frac{y_k A_{kj} y_j}{y_k^2}. \quad (4.5)$$

Using (4.4), one can say that (y, α) must be an eigenpair of A . For any solution $\alpha = \lambda_i$, $y = p_i$ (where p_i is the eigenvector corresponding to the eigenvalue λ_i), the function, $\phi = y_i A_{ij} y_j$, becomes $p_i A_{ij} p_j$ which can further be written as

$$p_i A_{ij} p_j = p_i (\lambda_i p_i) \implies p_i A_{ij} p_j = \lambda_i p_i^2 \quad (4.6)$$

or

$$\lambda_i = \frac{p_i A_{ij} p_j}{p_i^2}. \quad (4.7)$$

Therefore, when the vector y becomes equal to the eigenvector p_1 (corresponding to the largest eigenvalue λ_1 of matrix A), it serves as the global maximizer, providing the absolute maximum value of λ_1 or $\lambda_{\max}(A)$. On the same lines, when the vector y becomes equal to the eigenvector p_n (corresponding to the smallest eigenvalue λ_n of A), it serves as the global minimizer, achieving an absolute minimum value of λ_n or $\lambda_{\min}(A)$.

□

Formally, it can also be written as

$$\lambda_n \|y\|_2^2 \leq y^T A y \leq \lambda_1 \|y\|_2^2, \quad (4.8)$$

$$\lambda_1 = \max_{\substack{y \in \mathbb{R}^n \\ \|y\|=1}} y^T A y, \quad (4.9)$$

$$\lambda_n = \min_{\substack{y \in \mathbb{R}^n \\ \|y\|=1}} y^T A y, \quad (4.10)$$

which is also known as the **Rayleigh-Ritz** [18] theorem. This theorem is a direct consequence of eigendecomposition $A = P \Lambda P^T$ for real symmetric matrices as shown. By using $y = P x$,

$$y^T A y = x^T P^T A P y = x^T \Lambda x \quad (4.11)$$

$$y^T A y = \sum_{i=1}^n \lambda_i |x_i|^2 \quad (4.12)$$

$$\text{Since, } \lambda_n \sum_{i=1}^n |x_i|^2 \leq \sum_{i=1}^n \lambda_i |x_i|^2 \leq \lambda_1 \sum_{i=1}^n |x_i|^2 \quad (4.13)$$

and $\|x\|_2^2 = \|P^T y\|_2^2 = \|y\|_2^2$ for any orthogonal P , one have

$$\lambda_n \|y\|_2^2 \leq y^T A y \leq \lambda_1 \|y\|_2^2. \quad (4.14)$$

It follows that

$y^T A y \leq \lambda_1$, $\|y\|_2 = 1$, and $y^T A y \geq \lambda_n$, $\|y\|_2 = 1$ and it happens when $y = p_1$ and $y = p_n$, respectively. Hence, it is shown that

$$\lambda_1 = \max_{\substack{y \in \mathbb{R}^n \\ \|y\|=1}} y^T A y, \quad (4.15)$$

$$\lambda_n = \min_{\substack{y \in \mathbb{R}^n \\ \|y\|=1}} y^T A y. \quad (4.16)$$

Example: Given a symmetric positive definite matrix:

$$A = \begin{bmatrix} 1 & 3 \\ 3 & 9 \end{bmatrix} \text{ and a vector } y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \text{ then} \quad (4.17)$$

$$\phi = y_i A_{ij} y_j = y_1^2 + 9y_2^2 + 6y_1 y_2. \quad (4.18)$$

$$\phi = (y_1 + 3y_2)^2 \geq 0. \quad (4.19)$$

The value of ϕ is minimum at $y_1 = -3y_2$ subjected to $y_1^2 + y_2^2 = 1$ which will give $\phi = 0$ at $y_1 = \frac{-3}{\sqrt{10}}$ and $y_2 = \frac{1}{\sqrt{10}}$. Substituting $y_1 = \cos \theta$ and $y_2 = \sin \theta$, ϕ becomes $(\cos \theta + 3 \sin \theta)^2$ which can be written as $10 \left(\frac{1}{\sqrt{10}} \cos \theta + \frac{3}{\sqrt{10}} \sin \theta \right)^2 = 10 \sin^2(\theta + \theta_0)$ where $\sin \theta_0 = \frac{1}{\sqrt{10}}$. Thus, the maximum value for $10 \sin^2(\theta + \theta_0)$ is 10 when $\sin^2(\theta + \theta_0) = 1$ which is possible when $\cos \theta = \frac{1}{\sqrt{10}}$ or $\sin \theta = \frac{3}{\sqrt{10}}$. Therefore,

- The maximum of the Rayleigh quotient is 10 (also same as λ_1), achieved when $y = \frac{1}{\sqrt{10}}(1, 3)^T$.
- The minimum is 0 (also same as λ_2), when $y = \frac{1}{\sqrt{10}}(-3, 1)^T$.
- The overall range of Rayleigh quotient $R(A, y) = \frac{y_1^2 + 9y_2^2 + 6y_1 y_2}{y_1^2 + y_2^2}$ is between 0 and 10.

The Rayleigh-Ritz theorem gives an alternate characterization of the smallest and largest eigenvalue of a real symmetric matrix. It is also possible to provide a similar characterization for *any* eigenvalue. One can constrain vector y orthogonal to vector w_1 (where $w_1 \in \mathbb{R}^n$) and considering another problem using equation 4.9 given by

$$\max_{\substack{y^T w_1 = 0 \\ \|y\| = 1}} y^T A y \text{ and} \quad (4.20)$$

using $x = Py$ (where P is set of n eigenvectors of A), one can derive a lower bound

$$\max_{\substack{y^T w_1 = 0 \\ \|y\| = 1}} y^T A y = \max_{\substack{(Px)^T w_1 = 0 \\ \|x\| = 1}} x^T P^T A P x \quad (4.21)$$

$$\max_{\substack{(Px)^T w_1 = 0 \\ \|x\| = 1}} x^T \Lambda x \geq \max_{\substack{(Px)^T w_1 = 0 \\ x_3 = x_4 = \dots = x_n = 0 \\ x_1^2 + x_2^2 = 1}} \lambda_1 x_1^2 + \lambda_2 x_2^2 \geq \lambda_2. \quad (4.22)$$

Combining equation 4.20 to 4.22, one have the following inequality

$$\max_{\substack{y^T w_1 = 0 \\ \|y\| = 1}} y^T A y \geq \lambda_2. \quad (4.23)$$

Moreover, it can be noticed that the equality above is attained if $w = p_1$. Hence, one can have a variational characterization of λ_2 given by

$$\lambda_2 = \min_{w_1 \in \mathbb{R}^n} \max_{\substack{y^T w_1 = 0 \\ \|y\| = 1}} y^T A y. \quad (4.24)$$

The example case showcased above is an instance of a more general variational characterization results known as *Courant- Fischer theorem* [18] given by

$$\lambda_i = \min_{w_1, w_2, \dots, w_{i-1}} \max_{\substack{y^T w_j = 0, \\ j=1, 2, \dots, i-1 \\ \|y\| = 1}} y^T A y. \quad (4.25)$$

4.2 Power Method

The power method [20] provides an intuitive and simple way of finding the dominant eigenvalue of a square matrix A of size n by n . In this method, one relies on the fact that any arbitrary vector b_0 of size n by 1, upon repeated multiplication with the matrix A , tends to

align the vector in the direction of the dominant eigenvalue. Thus, starting with an arbitrary guess vector b_0 one calculates $Ab_0, A^2b_0, A^3b_0 \dots$ with the expectation that the sequence will converge to the dominant eigenvector. To approximate the eigenvalue at any step, there is a scalar value calculated using the matrix A and the vector x obtained at i th power iteration given by $\frac{\langle x, Ax \rangle}{\langle x, x \rangle}$ also known as the Rayleigh quotient (4.1) for matrix A and vector x .

The power algorithm [18] is

$$\begin{aligned} y_{i+1} &= Ax_i, \\ x_{i+1} &= \frac{y_{i+1}}{\|y_{i+1}\|}. \end{aligned} \quad (4.26)$$

This method is such that:

- $y_{k+1} = \frac{Ay_k}{\|Ay_k\|}$ converges to the eigenvector e_1 within the tolerance specified.
- $\lambda_1^k = \frac{\langle y_k, Ay_k \rangle}{\langle y_k, y_k \rangle}$ converges to the dominant eigenvalue λ_1 .

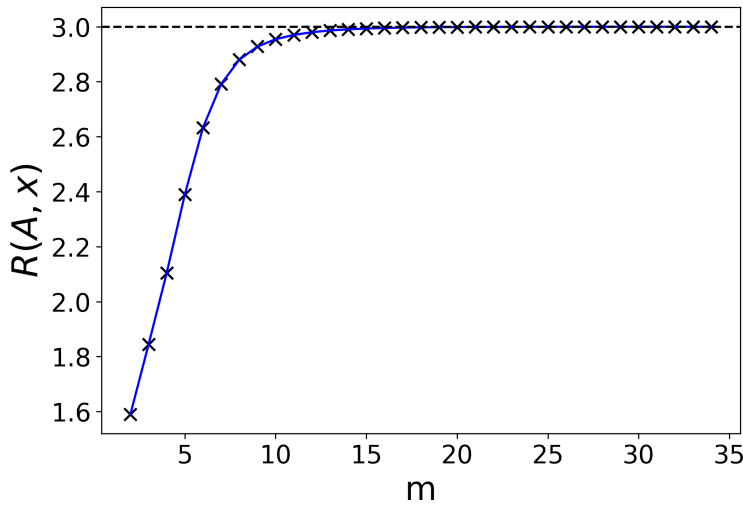
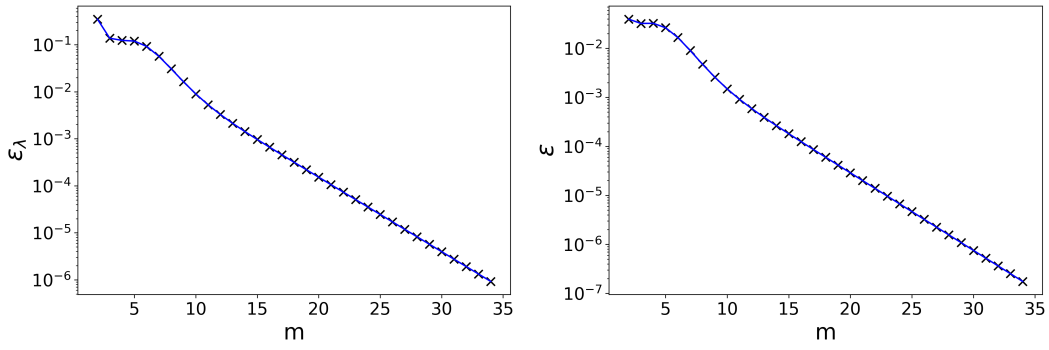


Fig. 4.1 The Rayleigh quotient $R(A, x)$, converges to the dominant eigenvalue $\lambda_1 = 3.0$ of matrix A of size 203×203 for $m = 34$ power iterations. The matrix A is a diagonal matrix so its eigenvalues will be same as the diagonal elements of A . $A_{203 \times 203} = \text{diag}(0, 0.01, 0.02, \dots, 1.99, 2, 2.5, 3.0)$.

As shown in figure 4.1, the convergence of Rayleigh quotient to the dominant eigenvalue λ_1 using power iteration for matrix A can be observed. The matrix A is a diagonal matrix of size 203×203 so its eigenvalues will be same as the diagonal elements of A . Power Method converges to the dominant eigenvalue, $\lambda_1 = 3.0$. In figure 4.2a, the variation of error for computing dominant eigenvalue for A is shown. Also, the variation of error for computing the



(a) The variation of error ϵ_λ , in the dominant eigenvalue for m power iterations. (b) The variation of error in the dominant eigenvector for m power iterations.

Fig. 4.2 Error plots for matrix A .

$$A_{203 \times 203} = \text{diag}(0, 0.01, 0.02, \dots, 1.99, 2, 2.5, 3.0).$$

dominant eigenvector for A is shown in 4.2b. For the eigenvalue and eigenvector to converge within a tolerance of $\delta = 1e - 6$, it takes $m = 34$ power iterations.

This method can be explained by taking into consideration the case of a symmetric matrix, A with no degeneracy in its eigen spectrum. Hence, the random vector b_0 can be expanded in the basis set consisting of eigenvectors as

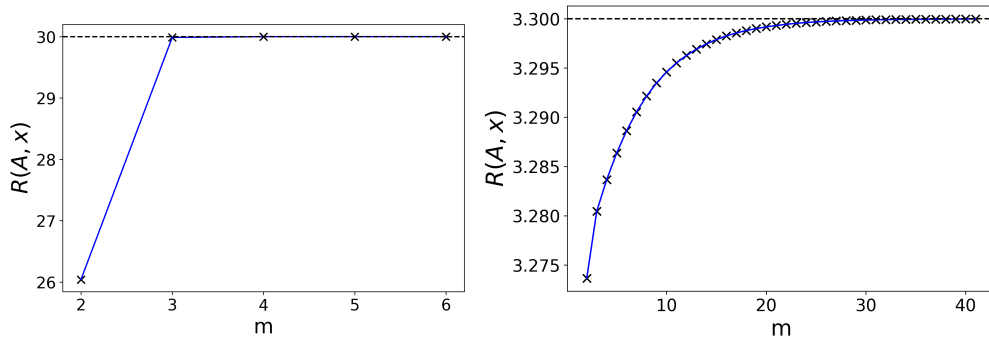
$$b_0 = c_1 e_1 + c_2 e_2 + \dots + c_n e_n, \quad (4.27)$$

where e_i is the i th eigenvector corresponding to the eigenvalue λ_i in the descending order of their magnitude for matrix A . Upon repeated multiplication of A on both sides of (4.27),

$$A^k b_0 = \lambda_1^k \left[c_1 e_1 + c_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k e_2 + \dots + c_n \left(\frac{\lambda_n}{\lambda_1} \right)^k e_n \right]. \quad (4.28)$$

$$\lim_{k \rightarrow \infty} A^k b_0 = \lambda_1^k c_1 e_1. \quad (4.29)$$

Hence, this procedure converges to the dominant eigenvector, e_1 of matrix A . This procedure converges rapidly if $|\lambda_2| \ll |\lambda_1|$, i.e., $|\lambda_2|$ is sufficiently less than $|\lambda_1|$ as shown in figure 4.3a. In the limit of $|\lambda_2| \approx |\lambda_1|$, the convergence is poor as shown in figure 4.3b. Also the variation of error in dominant eigenvalue for eigengap $\frac{|\lambda_2|}{|\lambda_1|} = 0.1$ (figure 4.4a) and $\frac{|\lambda_2|}{|\lambda_1|} = 0.9$ (figure 4.4b) is in accordance with the behavior mentioned before. Similarly, the error in the dominant eigenvector decreases below a tolerance of $\delta = 1e - 6$ in $m = 6$ power iterations (rapidly converges) for eigengap $\frac{|\lambda_2|}{|\lambda_1|} = 0.1$ (figure 4.5a) which is much slower i.e., $m = 41$ power iterations when eigengap $\frac{|\lambda_2|}{|\lambda_1|} = 0.9$ (figure 4.5b).



(a) The Rayleigh quotient i.e., $R(A, x)$ converges to the dominant eigenvalue $\lambda_1 = 30.0$ for a diagonal matrix A of size 200×200 for $m = 6$ power iterations. (b) The Rayleigh quotient i.e., $R(A, x)$ converges to the dominant eigenvalue $\lambda_1 = 3.3$ for a diagonal matrix A of size 200×200 for $m = 41$ power iterations.

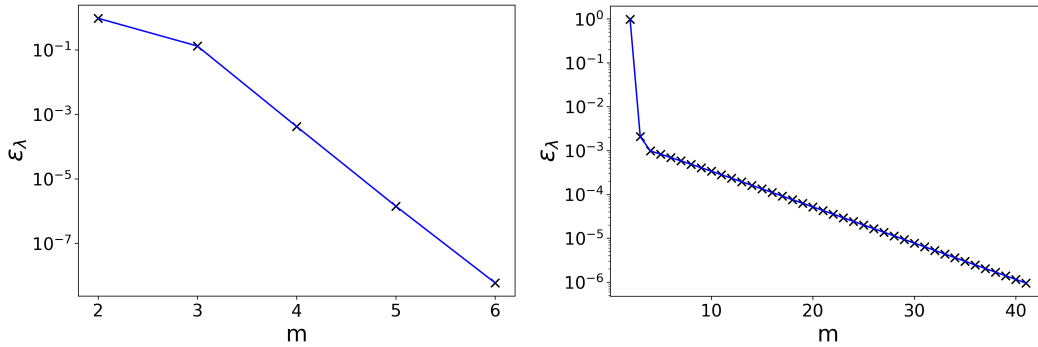
$A_{200 \times 200} = \text{diag}(0, 0.0001, 0.0002, \dots, 0.0196, 0.0197, 3, 30).$

The last two diagonal elements of A have the absolute maximum value of $\lambda_1 = 30.0$ and $\lambda_2 = 3.0$ implying the absolute maximum value of $\lambda_1 = 3.3$ and eigengap $\frac{|\lambda_2|}{|\lambda_1|} = 0.1$

$A_{200 \times 200} = \text{diag}(0, 0.0001, 0.0002, \dots, 0.0196, 0.0197, 3, 3.3).$

The last two diagonal elements of A have the absolute maximum value of $\lambda_1 = 3.3$ and $\lambda_2 = 3.0$ implying the absolute maximum value of $\lambda_1 = 3.3$ and eigengap $\frac{|\lambda_2|}{|\lambda_1|} = 0.9$.

Fig. 4.3 Power iterations on matrix $A_{200 \times 200}$ to compare convergence depending on the eigengap.



(a) The variation of error ϵ_λ , in eigenvalue for $m = 6$ power iterations. (b) The variation of error ϵ_λ , in eigenvalue for $m = 41$ power iterations.

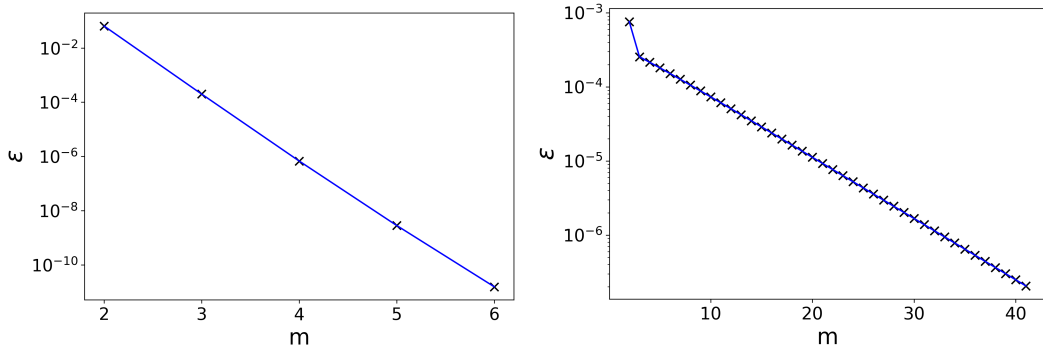
$A_{200 \times 200} = \text{diag}(0, 0.0001, 0.0002, \dots, 0.0196, 0.0197, 3, 30).$

$A_{200 \times 200} = \text{diag}(0, 0.0001, 0.0002, \dots, 0.0196, 0.0197, 3, 3.3).$

Fig. 4.4 Comparison of error in eigenvalue for figures 4.4a ($\frac{|\lambda_2|}{|\lambda_1|} = 0.1$) and 4.4b ($\frac{|\lambda_2|}{|\lambda_1|} = 0.9$).

Where can this method fail:

1. For unsymmetric matrices, this method fails when the dominant eigenvalue is complex in nature. Since complex roots always exist in pairs having the same magnitude, they



(a) The variation of error in eigenvector for $m = 6$ power iterations. (b) The variation of error in eigenvector for $m = 41$ power iterations.

$A_{200 \times 200} = \text{diag}(0, 0.0001, 0.0002, \dots$
 $\dots, 0.0196, 0.0197, 3, 30).$

$A_{200 \times 200} = \text{diag}(0, 0.0001, 0.0002, \dots$
 $\dots, 0.0196, 0.0197, 3, 3.3).$

Fig. 4.5 Comparison for error in eigenvector for figures 4.5a ($\frac{|\lambda_2|}{|\lambda_1|} = 0.1$) and 4.5b ($\frac{|\lambda_2|}{|\lambda_1|} = 0.9$).

will correspond to the top two eigenvectors, e_1 and e_2 . Power method focuses on finding the eigenvalue with the largest magnitude. In this case, this method won't be able to converge to e_1 , since both e_1 and e_2 have the same magnitude of eigenvalues.

2. For symmetric matrices, if there is a repeated pair of eigenvalues for the dominant eigenvalue, this method will not converge to the dominant eigenvector e_1 .
3. If the starting vector is in an invariant subspace in which the dominant eigenvector, e_1 does not lie, the power iterations can not converge to e_1 . The simple reason being that even on multiplying the starting vector with matrix A again and again, it will always stay in the invariant subspace only. Since the dominant eigenvector e_1 is not a part of this subspace, the iterated vector can not converge to the dominant eigenvector, e_1 .

In the section above, the power method is used to approximate the dominant eigenpair of a matrix. This method can be modified through the use of a procedure called *deflation* [20] in order to approximate the subsequent eigenpairs in the decreasing order.

Once the dominant eigenpair $(\hat{\lambda}_1, \hat{e}_1)$ (where \hat{e}_1 is a column vector), is approximated using the power method, subsequent eigenvalues can be targeted in the decreasing order one-by-one in a serial fashion. Rather than implementing the power method on the original matrix A , the same power method will be implemented on the deflated matrix for approximating the eigenpair $(\hat{\lambda}_2, \hat{e}_2)$ given by

$$B = A - \hat{\lambda}_1(\hat{e}_1 \hat{e}_1^t). \quad (4.30)$$

As the dominant eigenpair is removed from B , implementing the power method on B would approximate the second dominant eigenpair of matrix A .

This can be shown by starting with a new random vector, g_0 such that

$$g_0 = c_1 e_2 + c_2 e_3 + \cdots + c_{n-1} e_n, \quad (4.31)$$

where e_2, e_3, \dots, e_n are the eigenvectors corresponding to eigenvalues $\lambda_2, \lambda_3, \dots, \lambda_n$ in the descending order of their magnitude for matrix B . Multiplying by matrix B on both sides of (4.31) again and again, one can obtain,

$$B^k g_0 = \lambda_2^k \left[c_1 e_2 + c_2 \left(\frac{\lambda_3}{\lambda_2} \right)^k + \cdots + c_{n-1} \left(\frac{\lambda_n}{\lambda_2} \right)^k \right]. \quad (4.32)$$

$$\lim_{k \rightarrow \infty} B^k g_0 = \lambda_2^k c_1 e_2. \quad (4.33)$$

Hence, the approximated eigenpair $(\hat{\lambda}_2, \hat{e}_2)$ is obtained on the same lines. Thus, this procedure can be followed sequentially to find all the eigenpairs for matrix A .

4.3 Krylov subspace

Krylov subspace [20] is a mathematical tool which can be used to systematically generalize the idea of power iterations. Krylov pointed out that when a matrix or a linear operator acts repeatedly on a given starting vector, the collection of these vectors forms a subspace. The subspace generated by these vectors is termed as Krylov subspace. For a matrix A of size n by n and a vector b , the Krylov subspace $\mathcal{K}_r(A, b)$ is defined as the linear span of the set of vectors generated via repeated multiplication of A and b . In other words,

$$\mathcal{K}_r(A, b) = \text{span}\{b, Ab, A^2b, \dots, A^{r-1}b\}. \quad (4.34)$$

The dimension of the Krylov subspace $\mathcal{K}_r(A, b)$ is at most equal to n (the size of the matrix A). In practice, the dimension of the Krylov subspace can be much smaller. The fact that one can stop anywhere in the krylov subspace makes it computationally efficient. This subspace has r -independent vectors which can be shown by arranging all these vectors in a matrix and finding its rank. The Krylov subspace is intimately connected to the eigenvalues of the matrix A . The vectors forming the Krylov subspace can serve as a basis for approximating solutions to the linear systems. They can also be used to compute the eigen values of the matrix A . By choosing an appropriate starting vector b , the Krylov subspace can capture most important information about the solution space, allowing for efficient iterative methods. These methods are computationally attractive in the sense that it only require matrix-vector

products, which can be provided as an input subroutine where sparsity pattern of the matrix can be taken into account.

While designing eigensolvers, it is a common practice to take advantage of similarity transformations and make use of similar matrices that preserve the eigenvalues of original matrix A . One obvious choice for forming the similar matrix is to use the vectors obtained from the Krylov subspace and ensure they are orthonormal to each other. The methods that make use of the Krylov subspace are called Krylov subspace methods [20].

One of the common similarity transformation can eventually be achieved by starting from the most formal definition of eigenvalues and eigenvectors given by

$$AX = \lambda X, \quad (4.35)$$

where X corresponds to the eigenvector and λ corresponds to the eigenvalue for a given matrix A of size n by n . Using an orthogonal matrix $B = [b_1, b_2, \dots, b_n]$ of size n by n with columns of B as vectors following the orthonormality relation i.e. $b_i b_j^T = \delta_{ij}$, it can be shown that left multiplying B^T to the equation 4.35 gives

$$B^T A X = \lambda B^T X \implies B^T A I X = \lambda B^T X, \quad (4.36)$$

where I is the identity matrix of size n by n and using $B B^T = I$, gives

$$(B^T A B)(B^T X) = \lambda (B^T X), \quad (4.37)$$

$$M Y = \lambda Y \quad (4.38)$$

where $M = B^T A B$ and $Y = B^T X$, matrix M which is an orthogonal similarity transformation of A and hence, matrix A and M will have identical eigenvalues. The eigenvectors X for matrix A can be achieved from matrix Y by using the transformation given by $X = B Y$.

These ideas of Krylov subspace and orthogonal similarity transformation will further be discussed and implemented in designing the state of the art eigenvalue solver i.e., Lanczos algorithm [22] in the next chapter.

Chapter 5

Lanczos algorithm

The state-of-the-art eigenvalue solver i.e., Lanczos method used for large sparse symmetric matrices is discussed in this chapter. The Lanczos method makes use of the Krylov subspace and is used to compute a few k – eigenmodes for a symmetric matrix. Also, the main reasons behind the computational effectiveness and its ability to handle large sparse symmetric matrices efficiently will be discussed.

5.1 Lanczos method

Lanczos method [22] is often used for approximating eigenvalues and eigenvectors of large sparse symmetric matrices in areas such as quantum mechanics [31], signal processing [21], and machine learning [32]. The method is named after Cornelius Lanczos who introduced it in 1950s. This method is particularly used when working with very large matrices that are difficult to store explicitly, and when only a few eigenvalues and eigenvectors need to be computed. Similar to the power algorithm, it relies heavily on computing matrix-vector products.

Starting from a given vector v , based on these matrix-vector products one iteratively constructs a Krylov subspace $\mathcal{K}_r(A, v)$, with $r \leq n$, spanned by the initial vector and a series of matrix-vector multiplications with the given matrix. The vectors in this subspace are made orthogonal using the Gram-Schmidt process. This subspace also provides an iterative algorithm that efficiently finds a few dominant eigenvalues and their corresponding eigenvectors.

The method is based on the following set of observations:

- Starting from a vector v , repeated matrix-vector products, as done in the power method, form a Krylov subspace $\mathcal{K}_k(A, v) = \{v, Av, \dots, A^{k-1}v\}$.

- If the vectors in this subspace are made orthogonal using the Gram-Schmidt process, they form an orthogonal matrix $T_k = \{v_1, v_2, \dots, v_k\}$ of size $n \times k$, with v_k being the k th orthogonal vector created from the set of orthogonal vectors.
- To compute v_k , one just needs to work with a three-term recursion, i.e.,

$$v_k = Av_{k-1} - \frac{\langle Av_{k-1}, v_{k-1} \rangle}{\langle v_{k-1}, v_{k-1} \rangle} v_{k-1} - \frac{\langle Av_{k-1}, v_{k-2} \rangle}{\langle v_{k-2}, v_{k-2} \rangle} v_{k-2} \quad (5.1)$$

which is proved subsequently. An interesting insight that will be noticed later is forming b_k in such a way using equation (5.1), is that by construction v_k becomes orthogonal to v_{k-1} , v_{k-2} and also the vectors $b \in \{v_1, v_2, \dots, v_{k-4}, v_{k-3}\}$. Therefore, in this algorithm, the cost of a single iteration does not depend on the step of the iteration. In a single step of iteration, one has to execute a matrix-vector multiplication and $7n$ further floating point operations.

- The matrix $T_n = V_n A V_n$ is a similarity transform and thus preserve eigenvalues and eigenvectors.
- Since T_k is tri-diagonal, one only needs to compute $3k$ members of this matrix. The eigenvalues of the tri-diagonal matrix at k th step also called the Ritz values can be computed efficiently by tri-diagonal QR algorithm [19] in $O(k^2)$ flops. The cost of computing eigenvalues is negligible in compared to the cost of forming vector Av_{k-1} using matrix-vector multiplication.

These properties essentially capture the essence of the method.

The vectors in the Lanczos procedure [21] are obtained in a sequential manner starting from a randomly selected vector v_1 , the new vector v_2 is obtained by finding the component of Av_1 orthogonal to initial vector v_1 using the Gram-Schmidt orthogonalization process,

$$v_2 = Av_1 - \frac{\langle Av_1, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1, \quad (5.2)$$

$$\text{which gives } \langle v_2, v_1 \rangle = 0. \quad (5.3)$$

Notice that from (5.2), vector Av_1 can be written in terms of v_2 and v_1 ,

$$Av_1 = v_2 + \frac{\langle Av_1, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1$$

or

$$Av_1 = c_2 v_2 + c_1 v_1.$$

Using Av_1 , for a vector d orthogonal to v_2 and v_1 i.e.,

$\langle d, v_2 \rangle$ and $\langle d, v_1 \rangle$ are zero, one can evaluate the $\langle d, Av_1 \rangle$ as

(5.4)

$$\langle d, Av_1 \rangle = \langle d, c_2 v_2 + c_1 v_1 \rangle$$

or

$$\langle d, Av_1 \rangle = c_2 \langle d, v_2 \rangle + c_1 \langle d, v_1 \rangle.$$

Since d is orthogonal to vectors v_2 and v_1 respectively,

$$\langle d, Av_1 \rangle = c_2 \cancel{\langle d, v_2 \rangle}^0 + c_1 \cancel{\langle d, v_1 \rangle}^0.$$

$$\therefore \langle d, Av_1 \rangle = 0.$$

The next vector v_3 is obtained using Av_2 and making it orthogonal to the previous vectors v_2 and v_1 respectively using the Gram-Schmidt process,

$$v_3 = Av_2 - \frac{\langle Av_2, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2 - \frac{\langle Av_2, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1, \quad (5.5)$$

$$\text{which gives } \langle v_3, v_2 \rangle = 0 \text{ and } \langle v_3, v_1 \rangle = 0. \quad (5.6)$$

Notice that from (5.5), vector Av_2 can be written in terms of v_3 , v_2 and v_1 ,

$$Av_2 = v_3 + \frac{\langle Av_2, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2 + \frac{\langle Av_2, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1$$

or

$$Av_2 = c_3 v_3 + c_2 v_2 + c_1 v_1.$$

Using Av_2 , for a vector g orthogonal to v_3 , v_2 and v_1 i.e.,

$\langle g, v_3 \rangle$, $\langle g, v_2 \rangle$ and $\langle g, v_1 \rangle$ are zero, one can evaluate the $\langle g, Av_2 \rangle$ as (5.7)

$$\langle g, Av_2 \rangle = \langle g, c_3 v_3 + c_2 v_2 + c_1 v_1 \rangle$$

or

$$\langle g, Av_2 \rangle = c_3 \langle g, v_3 \rangle + c_2 \langle g, v_2 \rangle + c_1 \langle g, v_1 \rangle.$$

Since g is orthogonal to vectors v_3 , v_2 and v_1 respectively,

$$\begin{aligned} \langle g, Av_2 \rangle &= c_3 \underbrace{\langle g, v_3 \rangle}_{=0} + c_2 \underbrace{\langle g, v_2 \rangle}_{=0} + c_1 \underbrace{\langle g, v_1 \rangle}_{=0} \\ &\therefore \langle g, Av_2 \rangle = 0. \end{aligned}$$

Once more continuing the process v_4 can be obtained as,

$$v_4 = Av_3 - \frac{\langle Av_3, v_3 \rangle}{\langle v_3, v_3 \rangle} v_3 - \frac{\langle Av_3, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2 - \frac{\langle Av_3, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1, \quad (5.8)$$

$$\text{which gives } \langle v_4, v_3 \rangle = 0, \langle v_4, v_2 \rangle = 0 \text{ and } \langle v_4, v_1 \rangle = 0. \quad (5.9)$$

However, the coefficient of v_1 in (5.8), i.e., $\langle Av_3, v_1 \rangle$ is zero. This can be shown by considering the fact [20] that, for a symmetric matrix A and two vectors x and y ,

$$\langle Ax, y \rangle = \langle x, Ay \rangle, \quad (5.10)$$

which implies that

$$\langle Av_3, v_1 \rangle = \langle v_3, Av_1 \rangle. \quad (5.11)$$

The vector d in (5.4) becomes v_3 since v_3 is orthogonal to v_2 and v_1 ,

$$\langle Av_3, v_1 \rangle = 0. \quad (5.12)$$

Hence, v_4 from equation (5.8), can be calculated using three terms,

$$v_4 = Av_3 - \frac{\langle Av_3, v_3 \rangle}{\langle v_3, v_3 \rangle} v_3 - \frac{\langle Av_3, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2. \quad (5.13)$$

The next vector v_5 is obtained on the similar lines as,

$$v_5 = Av_4 - \frac{\langle Av_4, v_4 \rangle}{\langle v_4, v_4 \rangle} v_4 - \frac{\langle Av_4, v_3 \rangle}{\langle v_3, v_3 \rangle} v_3 - \frac{\langle Av_4, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2 - \frac{\langle Av_4, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1. \quad (5.14)$$

Using (5.7), the vector g becomes v_4 and since v_4 is orthogonal to v_3 , v_2 and v_1 ,

$$\therefore \langle v_4, Av_2 \rangle = 0 \text{ or } \langle Av_4, v_2 \rangle = 0. \quad (5.15)$$

Again using (5.4), the vector d becomes v_4 and since v_4 is orthogonal to v_2 and v_1 ,

$$\therefore \langle v_4, Av_1 \rangle = 0 \text{ or } \langle Av_4, v_1 \rangle = 0. \quad (5.16)$$

Hence, v_5 from equation (5.14) can be calculated again by using only 3 terms,

$$v_5 = Av_4 - \frac{\langle Av_4, v_4 \rangle}{\langle v_4, v_4 \rangle} v_4 - \frac{\langle Av_4, v_3 \rangle}{\langle v_3, v_3 \rangle} v_3. \quad (5.17)$$

Therefore, for a vector v_m , if Av_m can be written as a linear combination of a set of all vectors $v \in \{v_0, v_1, \dots, v_m, v_{m+1}\}$ and v_m being orthogonal to all these vectors b , then $\langle v_{k-1}, Av_m \rangle$ or $\langle Av_{k-1}, v_m \rangle$ will be zero where m ranges from zero to $k-3$, i.e.,

if $Av_m = c_1 v_1 + c_2 v_2 + \dots + c_m v_m + c_{m+1} v_{m+1}$,
 $m \in \{0, 1, 2, \dots, k-1\}$
 and
 $\langle v_{k-3}, b \rangle = 0 \forall b \in \{v_1, v_2, \dots, v_{m-1}, v_m\}$,
 then $\langle v_{k-3}, Av_m \rangle = 0$ or $\langle Av_{k-3}, v_m \rangle = 0$.

(5.18)

Hence, v_k can be computed by using only 3 terms:

$$v_k = Av_{k-1} - \frac{\langle Av_{k-1}, v_{k-1} \rangle}{\langle v_{k-1}, v_{k-1} \rangle} v_{k-1} - \frac{\langle Av_{k-1}, v_{k-2} \rangle}{\langle v_{k-2}, v_{k-2} \rangle} v_{k-2}. \quad (5.19)$$

Therefore, in the Lanczos method, at every step, only three terms are calculated. Thus, one just needs to work with the three-term recursion.

Using these Lanczos vectors, one can form a matrix $V_k = \{v_1, v_2, \dots, v_k\}$ and compute the matrix $T_k = V_k^T A V_k$. Due to the three-term recursion in this method, T_k becomes a tridiagonal

matrix, so one has to calculate only $3k$ terms to form this matrix T_k . The matrix T_k will be

$$T_k = \begin{pmatrix} \langle v_1, Av_1 \rangle & \langle v_2, Av_1 \rangle & 0 & \cdots & 0 \\ \langle v_2, Av_1 \rangle & \langle v_2, Av_2 \rangle & \langle v_2, Av_3 \rangle & 0 & 0 \\ 0 & \langle v_3, Av_2 \rangle & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \langle v_{k-1}, Av_k \rangle \\ 0 & \cdots & 0 & \langle v_k, Av_{k-1} \rangle & \langle v_k, Av_k \rangle \end{pmatrix}, \quad (5.20)$$

or

$$T_k = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & 0 \\ 0 & \beta_3 & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \beta_k \\ 0 & \cdots & 0 & \beta_k & \alpha_k \end{pmatrix}. \quad (5.21)$$

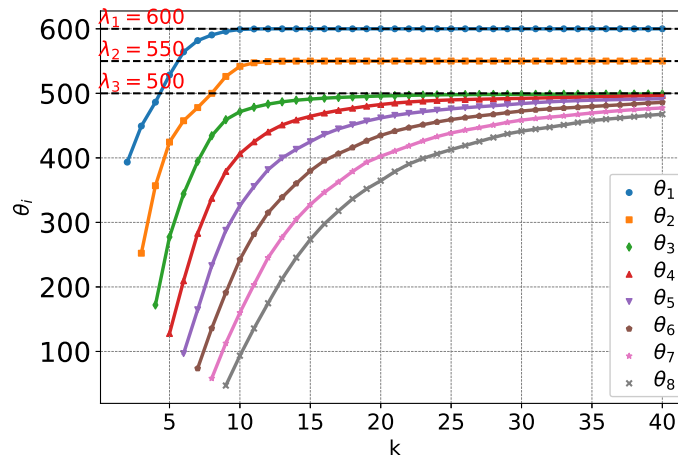


Fig. 5.1 Behaviour of top eight Ritz values (θ_i) for $k = 40$ iterations of Lanczos method for matrix A of size 503×503 . At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V_k^T A V_k$ where $V_k = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V_k^T A V_k$ where $V_k = \{v_1, v_2, \dots, v_{k-1}, v_k\}$.
 $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$.

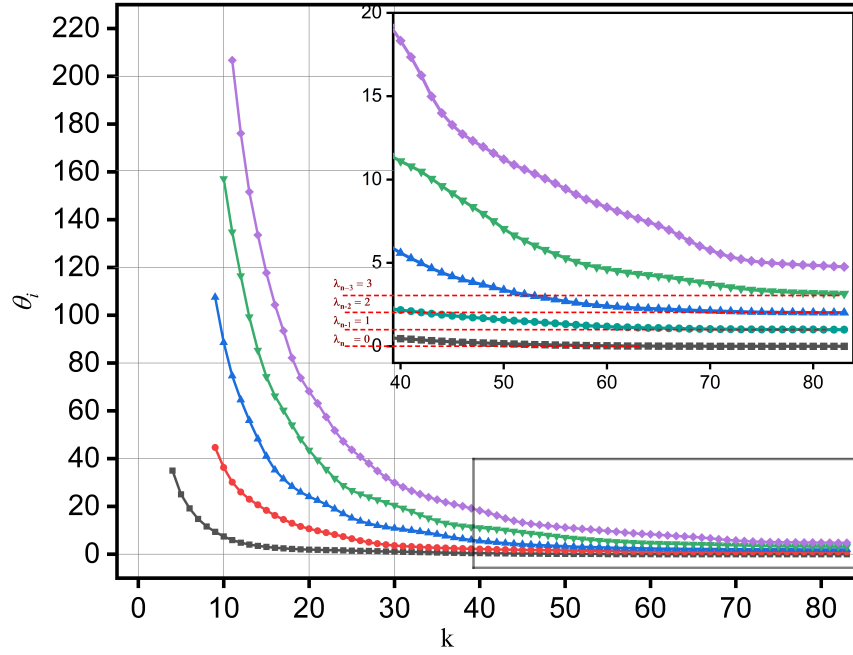


Fig. 5.2 Behaviour of Ritz values for Lanczos method for matrix A of size 503×503 from the bottom end of the eigen spectra. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V_k^T A V_k$ where $V_k = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$.

Now, at every step, one can obtain the matrix T_k and find the eigenvalues θ_i and eigenvectors g_i . These eigenvalues, also known as Ritz values [20], are a good approximation to the dominant eigenvalues of A . The approximations to the eigenvectors y_i (also called Ritz vectors) [20] of A can be computed using $y_i = V_k g_i$. In figure 5.1, the behavior of top eight Ritz values (θ_i) for $k = 40$ iterations of the Lanczos method for matrix A of size 503×503 is shown. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V^T A V$ where $V_k = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. The eigenvalues of matrix A are given by $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$. As shown in figure 5.1, the top three Ritz values can be seen converging to the dominant three eigenvalues of matrix A within the first fifty Lanczos steps. Also, it can be observed from figure 5.2 where the Lanczos algorithm is shown converging for the smallest magnitude eigenvalues.

Chapter 6

Lanczos algorithm: A Formal approach

The Lanczos algorithm computes a smaller tridiagonal matrix T_k , whose eigenvalues provide good approximations to the smallest and largest eigenvalues of a larger symmetric matrix A . In this chapter, I will formally present the Lanczos process as discussed by C.C. Paige [25]. It will be demonstrated that the eigenvalues of the tridiagonal matrix (also known as Ritz values) provide lower bounds for the k greatest eigenvalues and upper bounds for the k smallest eigenvalues of the original matrix A . Paige explains this by utilizing the orthogonal projection operator [25]. Additionally, I will discuss the failure of the Lanczos algorithm due to the loss of orthogonality among Lanczos vectors in finite precision, as revealed by Paige's rounding error analysis [29].

6.1 Lanczos procedure

The Lanczos process [33] formally can be written as a set of following steps:

- Let v_1 be given with $\|v_1\| = 1$,
- Choose $\hat{v}_2 = A v_1 - \alpha_1 v_1$,
- $v_2 = \frac{\hat{v}_2}{\beta_2}$; where $\beta_2 = \|\hat{v}_2\|$.
- $\hat{v}_3 = A v_2 - \alpha_2 v_2 - \beta_2 v_1$,
- $v_3 = \frac{\hat{v}_3}{\beta_3}$; where $\beta_3 = \|\hat{v}_3\|$.
- \vdots

Theorem 6.1.1. *In matrix notation, the Lanczos algorithm can be expressed as follows:*

$$A V_k = V_k T_k + E_k, \quad E_k = \beta_{k+1} v_{k+1} (e^k)^T \quad (6.1)$$

where e^k is the k th column of the identity matrix, $V_k = \{v_1, v_2, \dots, v_k\}$ and

$$T_k = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & 0 \\ 0 & \beta_3 & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \beta_k \\ 0 & \cdots & 0 & \beta_k & \alpha_k \end{pmatrix}. \quad (6.2)$$

Proof. To show this, it is known from chapter 5 that the Lanczos vectors can be written in three-term recursion as follows:

$$\begin{aligned} \hat{v}_2 &= Av_1 - \alpha_1 v_1, \\ v_2 &= \hat{v}_2 / \beta_2, \quad \text{where } \beta_2 = \|\hat{v}_2\|, \\ \beta_2 v_2 &= Av_1 - \alpha_1 v_1, \\ \implies Av_1 &= \beta_2 v_2 + \alpha_1 v_1. \end{aligned} \quad (6.3)$$

One start with $\beta_1 = 0$

$$\begin{aligned} Av_1 &= \beta_2 v_2 + \alpha_1 v_1 \\ Av_2 &= \beta_3 v_3 + \alpha_2 v_2 + \beta_2 v_1 \\ Av_3 &= \beta_4 v_4 + \alpha_3 v_3 + \beta_3 v_2 \\ &\vdots \end{aligned} \quad (6.4)$$

As an example consider T_2 and T_3 . Then

$$T_2 = \begin{pmatrix} \alpha_1 & \beta_2 \\ \beta_2 & \alpha_2 \end{pmatrix}, \quad T_3 = \begin{pmatrix} \alpha_1 & \beta_2 & 0 \\ \beta_2 & \alpha_2 & \beta_3 \\ 0 & \beta_3 & \alpha_3 \end{pmatrix} \quad (6.5)$$

So,

$$V_2 T_2 = \begin{pmatrix} \alpha_1 v_1 + \beta_2 v_2 & \beta_2 v_1 + \alpha_2 v_2 \end{pmatrix}, \quad E_2 = \begin{pmatrix} 0 & \beta_3 v_3 \end{pmatrix} \quad (6.6)$$

$$V_3 T_3 = \begin{pmatrix} \alpha_1 v_1 + \beta_2 v_2 & \beta_2 v_1 + \alpha_2 v_2 + \beta_3 v_3 & \beta_3 v_2 + \alpha_3 v_3 \end{pmatrix}, \quad E_3 = \begin{pmatrix} 0 & 0 & \beta_4 v_4 \end{pmatrix} \quad (6.7)$$

$$\text{Hence, } \boxed{A V_k = V_k T_k + E_k, \quad E_k = \beta_{k+1} v_{k+1} (e^k)^T} \quad (6.8)$$

□

Using equation 6.1, it also means

$$V_k^T A V_k = V_k^T V_k T_k \implies \boxed{T_k = (V_k^T V_k)^{-1} V_k^T A V_k} \quad (6.9)$$

Notice that

$$V_k^T E_k = 0. \quad (6.10)$$

Also, the matrix T_k is given by

- The diagonal element α_i is just the Rayleigh quotient for matrix A . So, if λ_i with $i = 1, 2, \dots, N$ are ordered values of Eigenvalues, hence,

$$\lambda_1(A) \leq \alpha_i \leq \lambda_N(A) \quad (6.11)$$

6.2 Lanczos algorithm for degenerate matrices

If for the original matrix λ_i with $i = 1, 2, \dots, N$ are ordered values of distinct eigenvalues and x_i as corresponding eigenvectors, one knows that

$$A x_i = \lambda_i x_i, \quad x_i^T x_j = \delta_{ij}. \quad (6.12)$$

Then v_j can be expressed as

$$v_j = \sum_i a_i^{(j)} x_i. \quad (6.13)$$

Here it should be noted that if $\lambda_1 = \lambda_2$ are two-fold degenerate eigenvalues then

$$A^r v_1 = \left(a_1^{(1)} x_1 + a_2^{(1)} x_2 \right) \lambda_1^r + \sum_{i=3} a_i^{(1)} x_i \lambda_i^r. \quad (6.14)$$

so, it is evident that Krylov subspace is spanned by $n - 1$ vectors $\left(a_i^{(1)} x_1 + a_i^{(2)} x_2\right), x_3, x_4, \dots, x_N$. Thus, degeneracy reduces the dimension of the Krylov subspace. Similarly, if v_i is orthogonal to a certain eigenvector x_k , the dimension of the Krylov subspaces reduces by one. Thus a Krylov subspace based method can only find the components of the eigensystem which lie in this linear subspace and can not find x_k or in the degenerate case considered above can not find x_1 and x_2 individually.

6.3 Orthogonal Projection operator

Denoting the orthonormal basis vectors v_j for $j = 1, \dots, N$, then they are j th column of the matrix $V_k = \{v_1, v_2, \dots, v_k\}$. These vectors v_i form a basis for a linear space here onwards denoted as M_k . Then any vector $x \in \mathcal{R}^n$ can be represented as $x = y + W u$ where $y \in M_k$ (Krylov subspace) is $V_k z$ with z being a size k vector and u being a size $n - k$ vector. Here, consider $W = \{w_{k+1}, w_{k+2}, \dots, w_N\}$ whose linearly independent columns are orthogonal to V_k . The symmetric matrix P_V of size $N \times N$ defined as

$$P_V = V_k (V_k^T V_k)^{-1} V_k^T \quad (6.15)$$

is the orthogonal projection operator onto M_k . This can be seen from

$$P_V x = V_k (V_k^T V_k)^{-1} V_k^T V_k z + V_k (V_k^T V_k)^{-1} \cancel{V_k^T W} u = V_k (V_k^T V_k)^{-1} V_k^T V_k z = V_k z. \quad (6.16)$$

which also means for $y \in M_k$ which implies $y \equiv V_k z$ hence,

$$P_V y = V_k (V_k^T V_k)^{-1} V_k^T V_k z = V_k z = y. \quad (6.17)$$

In other words, this projection operator does nothing to a vector lying in M_k . Also, note that

$$(x - P_V x)^T P_V x = (W u)^T V_k z = u^T \cancel{W^T V_k} z = 0. \quad (6.18)$$

This shows that the vector $P_V x$ is orthogonal to the complement of M_k i.e., $W u$.

Thus, by definition of orthogonal projection operator, it is evident that P_V is a projection operator onto M_k .

6.3.1 Eigensystem of Projection Matrix

This section aims to analyse the properties of $k \times k$ matrix T_k . To do so, one can connect its properties with matrix $P_V A P_V$ and show that the two are closely related. Let μ_i with $i = 1, 2, \dots, N$ be the eigenvalues of $P_V A P_V$ with corresponding eigenvectors as y_i . Similarly, let $\theta_i^{(k)}$ with $i = 1, 2, \dots, k$ be the ordered values of Eigenvalues of matrix T_k with the corresponding normalized eigenvectors as $z_j^{(k)}$.

$$z_j^{(k)} = \left\{ \zeta_{1,j}^{(k)}, \zeta_{2,j}^{(k)} \cdots \zeta_{k,j}^{(k)} \right\}^T, \quad j = 1, 2, \dots, k. \quad (6.19)$$

So, $Z^{(k)} = \{z_1^{(k)}, z_2^{(k)}, \dots, z_k^{(k)}\}$ is the set of eigenvectors for T_k .

Lemma 6.3.1. $P_V A P_V$ has k eigenvalues common with T_k . The remaining $N - k$ eigenvalues of $P_V A P_V$ are zero.

Proof.

$$P_V A P_V w_i = P_V A \left[V_k (V_k^T V_k)^{-1} \cancel{V_k^T w_i} \rightarrow 0 \right] = 0, \quad i = k+1, \dots, N \quad (6.20)$$

This implies $N - k$ eigenpairs of $P_V A P_V$ are

$$y_i = w_i, \quad \mu_i = 0, \quad i = k+1, k+2, \dots, N. \quad (6.21)$$

$W = \{w_{k+1}, \dots, w_N\}$ is orthogonal to M_k . Thus, the remaining k eigenvectors which must be orthogonal to W then has to necessarily lie in the linear space M_k . Thus, there must exist a set of vectors $z_i^{(k)}$ such that

$$y_i = V_k z_i^{(k)}, \text{ such that } P_V A P_V y_i = \mu_i y_i \quad i = 1 \cdots k. \quad (6.22)$$

However, using the fact that for $y_i \in M_k$ with $y = V_k z_i^{(k)}$, one must have

$$P_V y_i = y_i \equiv V_k z_i^{(k)} \quad i = 1 \cdots k. \quad (6.23)$$

One can write for $i = 1 \cdots k$, and hence,

$$P_V A P_V y_i = P_V A V_k z_i^{(k)} = V_k \left[(V_k^T V_k)^{-1} V_k^T A V_k \right] z_i^{(k)} \equiv V_k T_k z_i^{(k)}. \quad (6.24)$$

Thus,

$$V_k T_k z_i^{(k)} = \mu_i V_k z_i^{(k)} \quad i = 1 \cdots k. \quad (6.25)$$

One can annihilate V from left hand side by multiplying the equation by $(V_k^T V_k)^{-1} V_k^T$ to obtain

$$T_k z_i^{(k)} = \mu_i z_i^{(k)} \quad i = 1 \cdots k. \quad (6.26)$$

□

To conclude, the k eigenvectors of T_k are connected with k eigenvectors of $P_V A P_V$, and the corresponding eigenvalues are the same for both matrices. The larger matrix $P_V A P_V$ however has $n - k$ eigenvalues as zero.

Notice that Rayleigh quotient of the matrix $P_V A P_V$ for $y_i \in M_k$ (which implies $P_V y = y$) are

$$R(P_V A P_V, y) = \frac{y^T P_V A P_V y}{y^T y} = \frac{y^T A y}{y^T y} = R(A, y) \quad (6.27)$$

So Ritz pair $\{\theta_j^{(k)}, x_{(k)}^j \equiv V_k z_{(k)}^j\}$ are approximation to Eigenvalue and eigenvectors of the matrix A .

The Courant-Fischer theorem can be used very effectively to obtain inequalities involving the two sets of eigenvalues θ_i and λ_i respectively. C.C. Paige [25] compared the eigenvalues of matrix A with symmetric matrix $P_V A P_V$ which seems an easier approach than is usually given in the literature.

By definition, using the Courant-Fisher theorem on matrix A it follows that

$$\lambda_i = \min_{w_1, w_2, \dots, w_{i-1}} \max_{\substack{y^T w_j = 0, \\ j=1, 2, \dots, i-1}} y^T A y \quad (6.28)$$

where there are i constraints on the vector y . Similarly, implementing the same theorem on matrix $P_V A P_V$ it gives

$$\theta_i = \min_{w_1, w_2, \dots, w_{i-1}} \max_{\substack{y^T w_j = 0, \\ j=1, 2, \dots, i-1; k+1, \dots, n}} y^T (P_V A P_V) y \quad (6.29)$$

where w_1, \dots, w_{i-1} are arbitrary vectors in \mathbb{R}^n and w_{k+1}, \dots, w_n are linearly independent vectors orthogonal to T_k . Since the quantity in equation 6.29 is a much more constrained quantity as vector y is subjected to $(n - k + 1)$ constraints. Therefore,

$$\lambda_i \geq \theta_i. \quad (6.30)$$

It can also be shown [25] that

$$\theta_i \geq \lambda_{i+n-k}. \quad (6.31)$$

Hence, it is shown that

$$\lambda_i \geq \theta_i \geq \lambda_{i+n-k}, \quad i = 1, 2, \dots, k. \quad (6.32)$$

Thus finding the eigenvalues θ_i of T_k gives lower bounds on the k greatest and upper bounds on the k least, eigenvalues of A . This essentially becomes the key reason that Lanczos algorithm can be used to very good estimates of the smallest and largest eigenvalues of matrix A .

6.4 Rounding Error Analysis for Lanczos Algorithm

The following corollaries derived in chapter 2 was used by C.C.Paige [29] to perform the Rounding error analysis for the Lanczos algorithm.

1. Vector subtraction

$$\text{fl}(x - \alpha y) = x - \alpha y - \delta y, \quad \|\delta y\| \leq (\|x\| + 2\|\alpha y\|) \varepsilon \quad (6.33)$$

2. Vector inner-product

$$\text{fl}(v^T u) = (v + \delta v)^T u, \quad \|\delta v\| \leq n \varepsilon \|v\| \quad (6.34)$$

3. Matrix-vector multiplication

If there are at most m non-zero elements per row of A , then

$$\text{fl}(Au) = (A + \delta A)^T u, \quad \|\delta A\| \leq m \varepsilon \|A\|, \quad (6.35)$$

$$\text{and } \|\delta A\| \leq \|\|\delta A\|\| \leq m \varepsilon \|A\| = m \beta \varepsilon \sigma. \quad (6.36)$$

4. Normalization

$$\beta = \text{fl}\left(\sqrt{\text{fl}(w^T w)}\right) = \left[1 + \frac{n+2}{2} \varepsilon\right] \|w\| \quad (6.37)$$

$$v = \text{fl}(w/\beta) = D(1 + \varepsilon)w/\beta \quad (6.38)$$

$$v^T v = (1 + 2\varepsilon)w^T w/\beta^2 = 1 + (n+4)\varepsilon \quad (6.39)$$

The computational variant of the Lanczos algorithm that will be analyzed was called A1 [29] and in absence of rounding errors can be described as follows.

Let v_1 be given with $v_1^T v_1 = 1$, then

$$u_1 := Av_1 \quad (6.40)$$

and for $j = 1, 2, 3, \dots$ do steps (6.41) to (6.45)

$$\alpha_j := v_j^T u_j \quad (6.41)$$

$$w_j := u_j - \alpha_j v_j \quad (6.42)$$

$$\beta_{j+1} := +\sqrt{(w_j^T w_j)}, \quad \text{if } \beta_{j+1} = 0 \quad \text{then STOP} \quad (6.43)$$

$$v_{j+1} := w_j / \beta_{j+1} \quad (6.44)$$

$$u_{j+1} := Av_{j+1} - \beta_{j+1} v_j. \quad (6.45)$$

This essentially describes a way to produce vectors v_j and the coefficients α_j, β_{j+1} of the symmetric tridiagonal matrix.

Theorem 6.4.1. *Let A be an $n \times n$ real symmetric matrix with at most m non-zero elements in any row, and such that $\|A\| = \sigma$, $\|A\| = \beta\sigma$. If the variant of the Lanczos algorithm described by equations (6.40) to (6.45) is implemented on a floating point digital computer with relative precision ε and applied for k steps to A starting with a normalized initial vector v_1 , then $\alpha_j, \beta_{j+1}, v_{j+1}$ will be computed for $j = 1, 2, \dots, k$ such that*

$$AV_K = V_K T_k + \beta_{k+1} v_{k+1} e_k^T + \delta V_k \quad (6.46)$$

$$\text{where } V_k \equiv [v_1, \dots, v_k], \quad T_k \equiv \begin{pmatrix} \alpha_1 & \beta_2 & 0 & \dots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & 0 \\ 0 & \beta_3 & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \beta_k \\ 0 & \dots & 0 & \beta_k & \alpha_k \end{pmatrix}, \quad \delta V_k \equiv [\delta v_1, \dots, \delta v_k]$$

and e_k is the k th column of the unit matrix, and for $j = 1, 2, \dots, k$,

$$|v_{j+1}^T v_{j+1} - 1| \leq \varepsilon_0 \quad (6.47)$$

$$\|\delta v_j\| \leq \sigma \varepsilon_1 \quad (6.48)$$

$$\beta_{j+1} |v_j^T v_{j+1}| \leq 2\sigma \varepsilon_1 \quad (6.49)$$

and

$$\varepsilon_0 \equiv (n+4)\varepsilon, \quad \varepsilon_1 \equiv (7+m\beta)\varepsilon. \quad (6.50)$$

Corollary 6.4.1.1.

$$\|v\| = 1 + \varepsilon(n+4)/2 \quad (6.51)$$

Proof. $v = ((1+\varepsilon)w/\beta)$.

Now, $\beta = (1 + \varepsilon(n+2)/2) \|w\|$, so this becomes

$$\begin{aligned} \frac{1+\varepsilon}{1 + \varepsilon(n+2)/2} &:= (1+\varepsilon)(1 + \varepsilon(n+2)/2) \\ &:= 1 + \varepsilon(n+4)/2. \end{aligned}$$

□

Corollary 6.4.1.2. *Using equation 6.35,*

$$u_1 = A v_1 - \delta u_1, \quad \|\delta u_1\| = \|\delta A v_1\| \leq m\beta \varepsilon \sigma, \quad (6.52)$$

$$\|u_1\| \leq [1 + \varepsilon(n+2m\beta+4)/2] \sigma \quad (6.53)$$

where it is assumed that v_1 was found by normalizing a given vector as in equation 6.39.

Proof. In this step, one can evaluate

$$u_1 = \text{fl}(\text{fl}(A) \text{fl}(v_1)) \quad (6.54)$$

But it is just showed

$$1 - \frac{(n+4)}{2} \varepsilon \leq \|\text{fl}(v_1)\| \leq 1 + \frac{(n+4)}{2} \varepsilon \quad (6.55)$$

Also one knows

$$u_1 = (A + \delta A) \text{fl}(v_1), \quad |\delta A| \leq m\varepsilon |A|, \quad (6.56)$$

using it gives

$$\|u_1\| \leq \sigma \|\text{fl}(v_1)\| + m\varepsilon \beta \sigma. \quad (6.57)$$

so,

$$\|u_1\| \leq \sigma \left(1 + \frac{(n+4)}{2} \varepsilon + m\varepsilon \beta \right). \quad (6.58)$$

□

Corollary 6.4.1.3.

$$\alpha_j = \text{fl}(\text{fl}(v_j^T) \text{fl}(u_j)), \quad (6.59)$$

$$\|\alpha_j\| \leq \left(1 + \frac{3n+4}{2}\varepsilon\right) \|u_j\| \quad (6.60)$$

where it is assumed that v_1 was found by normalizing a given vector as in equation 6.39.

Proof.

$$\alpha_j = \text{fl}(v_{j1})u_{j1}(1 + \varepsilon_1) + \text{fl}(v_{j2})u_{j2}(1 + \varepsilon_2) + \dots \quad (6.61)$$

$$\implies \|\alpha_j\| \leq \|u_j\| \|\text{fl}(v_j)\| (1 + n\varepsilon). \quad (6.62)$$

But $\|\text{fl}(v_j)\| \leq 1 + \frac{(n+4)}{2}\varepsilon$ So, one have

$$\implies \|\alpha_j\| \leq \|u_j\| \left(1 + \frac{(n+4)}{2}\varepsilon\right) (1 + n\varepsilon). \quad (6.63)$$

One term extra coming

$$\implies \|\alpha_j\| \leq \|u_j\| \left(1 + \frac{(3n+4)}{2}\varepsilon\right). \quad (6.64)$$

Alternatively, $\alpha_j = \text{fl}(v_j^T u_j) = (v_j + \delta v_j)^T u_j = v_j^T u_j + (\delta v_j)^T u_j$ where $|\delta v_j| \leq n\varepsilon \|v_j\|$.

So $|\alpha_j| \leq |v_j^T u_j| + |(\delta v_j)^T u_j| \leq (1 + \varepsilon(n+4)/2)\|u_j\| + n\varepsilon(1 + \varepsilon(n+4)/2)\|u_j\|$, the last inequality using $\|v_j\| = 1 + \varepsilon(n+4)/2$ and Cauchy-Schwartz. This simplifies to $(1 + \varepsilon(3n+4)/2)\|u_j\|$.

So

$$|\alpha_j| \leq \left(1 + \frac{3n+4}{2}\varepsilon\right) \|u_j\|.$$

And from the error analysis of the dot product, one have $\delta\alpha_j \leq n\|u_j\|\varepsilon$.

□

Corollary 6.4.1.4.

$$w_j = u_j - \alpha_j v_j - \delta w_j, \quad \|\delta w_j\| \leq 3\varepsilon \|u_j\|. \quad (6.65)$$

Proof. By the error equation for vector subtraction, $\|\delta w_j\| \leq \varepsilon(\|u_j\| + 2\alpha_j \|v_j\|)$.

But $\|v_j\| = 1 + \varepsilon(n+4)/2$ and $\alpha_j = \left(1 + \frac{3n+4}{2}\varepsilon\right) \|u_j\|$. So the above expression simplifies to $\varepsilon(3\|u_j\| + 4(n+2)\varepsilon\|u_j\|) = 3\varepsilon\|u_j\|$.

So $\|\delta w_j\| \leq 3\varepsilon\|u_j\|$.

□

Corollary 6.4.1.5.

$$\|w_j\|^2 = \|u_j\|^2 + \alpha_j^2(\|v_j\|^2 - 2) - 2\alpha_j\delta\alpha_j - 2\delta w_j^T(u_j - \alpha_j v_j) + \|\delta w_j\|^2. \quad (6.66)$$

Proof.

$$\begin{aligned}
\|w_j\|^2 &= w^T w \\
&= (u_j^T - \alpha_j v_j^T - \delta w_j^T)(u_j - \alpha_j v_j - \delta w_j) \\
&= \|u_j\|^2 - \alpha_j u_j^T v_j - u_j^T \delta w_j - \alpha_j v_j^T u_j + \alpha_j \|v_j\|^2 + \alpha_j v_j^T \delta w_j - \alpha_j w_j^T u_j + \alpha_j \delta w_j^T v_j + \|\delta w_j\|^2 \\
&= \|u_j\|^2 - 2\alpha_j v_j^T u_j - 2\delta w_j^T u_j + 2\alpha_j \delta w_j^T v_j + \alpha_j \|v_j\|^2 + \|\delta w_j\|^2 \\
&= \|u_j\|^2 - 2\alpha_j(\alpha_j + \delta \alpha_j) - 2\delta w_j^T u_j + 2\alpha_j \delta w_j^T v_j + \alpha_j \|v_j\|^2 + \|\delta w_j\|^2 \\
&= \|u_j\|^2 + \alpha_j^2(\|v_j\|^2 - 2) - 2\alpha_j \delta \alpha_j - 2\delta w_j^T(u_j - \alpha_j v_j) + \|\delta w_j\|^2.
\end{aligned}$$

□

Corollary 6.4.1.6.

$$\|w_j\|^2 + \alpha_j^2 - \|u_j\|^2 \leq (3n + 10)\|u_j\|^2 \varepsilon. \quad (6.67)$$

Proof. Rearranging the previous corollary, one gets

$$\|w_j\|^2 + \alpha_j^2 - \|u_j\|^2 = \alpha_j^2(\|v_j\|^2 - 1) - 2\alpha_j \delta \alpha_j - 2\delta w_j^T(u_j - \alpha_j v_j) + \|\delta w_j\|^2.$$

Then, using the bounds above,

$$\begin{aligned}
\|w_j\|^2 + \alpha_j^2 - \|u_j\|^2 &= \alpha_j^2(\|v_j\|^2 - 1) - 2\alpha_j \delta \alpha_j - 2\delta w_j^T(u_j - \alpha_j v_j) + \|\delta w_j\|^2 \\
&\leq |\alpha_j^2(\|v_j\|^2 - 1)| + |2\alpha_j \delta \alpha_j| + |2\delta w_j^T(u_j - \alpha_j v_j)| + \|\delta w_j\|^2 \\
&\leq \alpha_j^2(n + 4)\varepsilon + 2\|\alpha_j\|n\|u_j\|\varepsilon + 6\varepsilon\|u_j\|\|u_j - \alpha_j v_j\| + 9\varepsilon^2\|u_j\|^2 \\
&\leq (1 + (3n + 4)\varepsilon)\|u_j\|^2(n + 4)\varepsilon + 2\left(1 + \frac{3n + 4}{2}\varepsilon\right)\|u_j\|^2n\varepsilon + 6\varepsilon\|u_j\|\|u_j - \alpha_j v_j\| \\
&\leq (n + 4)\varepsilon\|u_j\|^2 + 2n\varepsilon\|u_j\|^2 + 6\varepsilon\|u_j\|\|u_j\| \\
&= (3n + 10)\varepsilon\|u_j\|^2
\end{aligned}$$

The last inequality using the fact that $\|u_j - \alpha_j v_j\| \leq \|u_j\|$, because $\alpha_j v_j \perp (u_j - \alpha_j v_j)$. □

Corollary 6.4.1.7.

$$\beta_{j+1} = \left[1 + \varepsilon \frac{(n+2)}{2}\right] \|w_j\| \leq [1 + (2n + 6)\varepsilon] \|u_j\| \quad (6.68)$$

Proof. The equality has been proved before. The inequality comes from the previous corollary:

$$\begin{aligned}
\|w_j\|^2 + \alpha_j^2 - \|u_j\|^2 &\leq (3n+10)\|u_j\|^2\epsilon \\
\|w_j\|^2 &\leq \|u_j\|^2 - \alpha_j^2 + (3n+10)\|u_j\|^2\epsilon \\
\|w_j\|^2 &\leq \|u_j\|^2 + (3n+10)\|u_j\|^2\epsilon \\
\|w_j\|^2 &\leq (1 + (3n+10)\epsilon)\|u_j\|^2 \\
\|w_j\| &\leq (1 + (2n+6)\epsilon)\|u_j\|
\end{aligned}$$

□

Corollary 6.4.1.8.

$$\beta_{j+1}v_{j+1} = w_j + \delta w'_j, \quad \|\delta w'_j\| \leq \|w_j\|\epsilon. \quad (6.69)$$

Proof. $\beta v = (\beta I)v$. This, by the error analysis for matrix-vector multiplication, gives $w = \text{fl}(\beta v) = (\beta I + \delta \beta I)v = \beta v + \delta \beta I v$, where $\|\delta \beta I\| \leq \epsilon \beta$ and $\delta \beta I v = -\delta w$. Now $\|v\| = 1 + \epsilon(n+4)/2$, so $\|\delta \beta I v\| \leq \epsilon \beta$. From above, $\beta \leq [1 + (2n+6)\epsilon]\|u\|$, so $\|\delta w\| \leq \epsilon\|u\|$. □

Corollary 6.4.1.9.

$$u_j = Av_j - \beta_j v_{j-1} - \delta u_j, \quad \|\delta u_j\| \leq (1 + m\beta)\sigma\epsilon + 2\|u_{j-1}\|\epsilon \quad (6.70)$$

Proof. By the error bound for vector subtraction, $\delta u_j \leq \|Av_j\|\epsilon + 2\|\beta_j v_{j-1}\|\epsilon$. Now $\|\beta_j v_{j-1}\|\epsilon = \|(1 + (2n+6)\epsilon)\|u_{j-1}\| \cdot (1 + \epsilon(n+4)/2)\epsilon \leq \|u_{j-1}\|\epsilon$.

Now, for the matrix vector multiplication Av , one have $\text{fl}(Av) = (A + \delta A)v$ where $\|\delta A\| \leq m\beta\epsilon\sigma$. So $\|\text{fl}(Av)\| \leq \sigma(1 + \epsilon(n+4)/2) + m\beta\epsilon\sigma$.

So $\|Av\|\epsilon \leq \sigma\epsilon$.

Additionally, the matrix vector product itself adds an error of $m\beta\sigma\epsilon$, so the total error is less than

$$\sigma\epsilon + \|u_{j-1}\|\epsilon + m\beta\sigma\epsilon$$

.

□

Corollary 6.4.1.10.

$$\begin{aligned}
\beta_{j+1}v_{j+1} &= Av_j - \alpha_j v_j - \beta_j v_{j-1} - \delta v_j \\
\|\delta v_j\| &\leq (1 + m\beta)\sigma\epsilon + (4\|u_j\| + 2\|u_{j-1}\|)\epsilon.
\end{aligned}$$

Proof. The equation comes from the algorithm.

one can use the previous corollaries to get $\|\delta v_j\| \leq \|\delta w'_j\| + \|\delta w_j\| + \|\delta u_j\| \leq (1 + m\beta)\sigma\varepsilon + (4\|u_j\| + 2\|u_{j-1}\|)\varepsilon$. \square

Corollary 6.4.1.11.

$$\begin{aligned}\beta_{j+1}v_j^T v_{j+1} &= v_j^T u_j - \alpha_j v_j^T v_j + v_j^T (\delta w'_j - \delta w_j) \\ &= \delta\alpha_j - \alpha_j(v_j^T v_j - 1) + v_j^T (\delta w'_j - \delta w_j)\end{aligned}$$

Proof. Both the equalities come from substituting values found in previous corollaries. \square

Corollary 6.4.1.12.

$$\beta_{j+1}|v_j^T v_{j+1}| \leq 2(n+4)\|u_j\|\varepsilon. \quad (6.71)$$

Proof. One can use the fact that $|v_{j+1}^T v_{j+1} - 1| \leq (n+4)\varepsilon$. Using this and the previous corollary,

$$\begin{aligned}\beta_{j+1}|v_j^T v_{j+1}| &= \delta\alpha_j - \alpha_j(v_j^T v_j - 1) + v_j^T (\delta w'_j - \delta w_j) \\ &\leq n\|u_j\|\varepsilon + (1 + \varepsilon(3n+4)/2)\|u_j\|(n+4)\varepsilon + (1 + \varepsilon(n+4)/2)4\|u_j\|\varepsilon \\ &\leq 2(n+4)\|u_j\|\varepsilon.\end{aligned}$$

\square

Hence, the scalar β_{k+1} which is the norm of the vector v_k becomes an important parameter to be considered. As shown in [29] and discussed in this chapter,

$$|v_k^T v_{k+1}| \leq \frac{2\sigma\varepsilon_0}{\beta_{k+1}} \quad (6.72)$$

where σ represents the 2-norm of A and $\varepsilon_0 \equiv (n+4)\varepsilon$ respectively. It is clear that as the scalar β_{k+1} goes close to zero, the inner product of vectors v_k and v_{k+1} is a finite value and hence, the Lanczos vectors are no more orthogonal. This is the part that becomes one of the key reasons for the Lanczos algorithm to fail in the finite precision. This aspect will be further discussed in the next chapter.

Chapter 7

Lanczos Algorithm in Finite Precision

Finite precision arithmetic introduces loss of orthogonality in Lanczos vectors, leading to multiple approximations of the original eigenvalues and potentially yielding false positive results in the numerical process. Therefore, this Lanczos process can not be continued further. This chapter explains the above-mentioned problem in detail. The Lanczos algorithm's convergence behavior exhibits high sensitivity to the proximity of eigenvalues. This aspect is examined in detail in Section 7.3, where discussions focus on leveraging this behavior for beneficial outcomes. Finally, an attempt to capture all the eigenmodes using permutation groups is shown in section 7.4.

7.1 Key issues in the Lanczos method

It is observed as one can keep on computing more and more Lanczos vector, the Lanczos vectors are no more orthogonal in finite precision due to the accumulation of rounding errors. Due to this loss of orthogonality, instead of matrix T_k having one eigenvalue approximately equal to $\lambda_i(A)$, it will have more than one eigenvalue nearly equal to $\lambda_i(A)$ known as spurious eigenvalue [23] which is clear from figure 7.1.

Then, to avoid these subsequent vectors losing orthogonality, one has to make use of the expensive implementation of the Lanczos algorithm called *Lanczos with full re-orthogonalization*. It can be observed from figure 7.2 in which the Lanczos method with full re-orthogonalization is implemented that the Ritz values are not repeating and also the vectors obtained are orthogonal to each other. In a full re-orthogonalization algorithm, the Gram-Schmidt orthogonalization process is implemented at each step for finding b_k , i.e.,

$$v_k = Av_{k-1} - \sum_{i=0}^{k-1} \langle Av_{k-1}, v_i \rangle v_i. \quad (7.1)$$

So, the cost of the algorithm increases considerably. The k th step of the algorithm requires a matrix-vector multiplication and $(2k + O(1))n$ floating point operations.

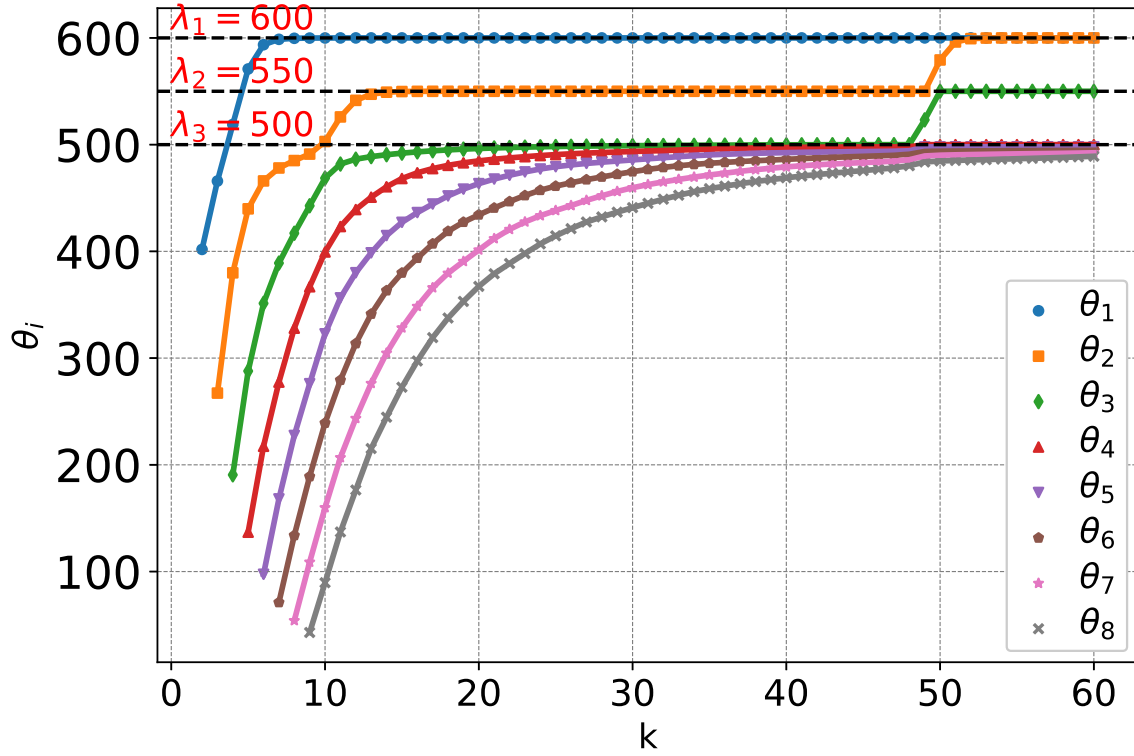


Fig. 7.1 Lanczos method without Reorthogonalization indicating multiplicities of Ritz values. The behaviour of top eight Ritz values (θ_i) for $k = 60$ iterations of Lanczos method for matrix A of size 503×503 is shown. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V^T A V$ where $V = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$.

Another defect of the method in finite precision is the emergence of multiple copies of converged Ritz values as discussed in [33] shows that the Lanczos vector v_k gains large components in the directions of converged Ritz vectors which leads to loss of orthogonality. This systematic loss is explained by Paige's theorem [20] which states that

$$\langle y_k, v_{k+1} \rangle \propto \frac{1}{\beta_{k+1} |g_i(k)|}, \quad (7.2)$$

where $y_i = V_k g_i$ i.e. the Ritz vector corresponding to the Ritz value θ_i , v_{k+1} is the computed Lanczos vector in the next step and $g_i(k)$ is the k th bottom entry of g_i (where g_i is the eigenvector corresponding to the eigenvalue or Ritz value, θ_i for matrix T_k).

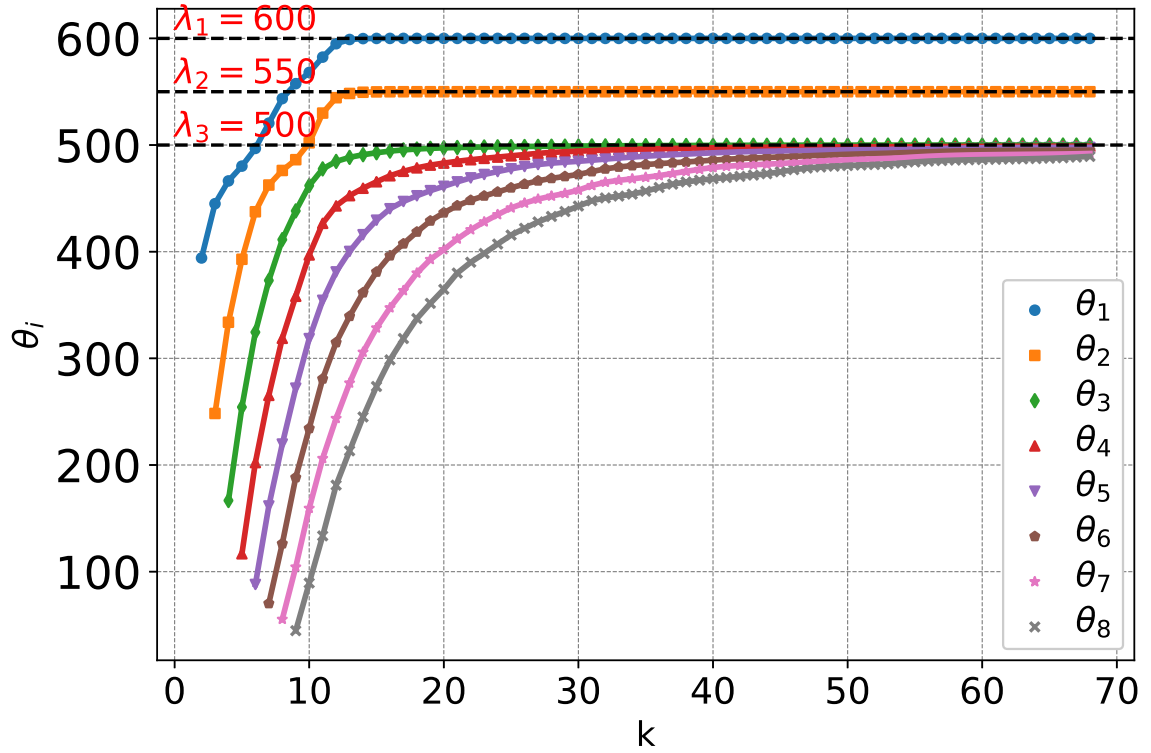


Fig. 7.2 Lanczos method with Reorthogonalisation don't have this issue of appearance of multiple Ritz values. The behaviour of top eight Ritz values (θ_i) for $k = 68$ iterations of Lanczos reorthogonalisation method for matrix A of size 503×503 is shown. At each Lanczos step k , the eigenvalues (Ritz values) are computed for the matrix $T_k = V^T A V$ where $V = \{v_1, v_2, \dots, v_{k-1}, v_k\}$. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$.

This quantity $\beta_{k+1}|g_i(k)|$ represents the error bound on the corresponding Ritz value θ_i . As the Ritz value converges and its error bound approaches zero, the Lanczos vector v_{k+1} gains a large component in the direction of the Ritz vector $y_{k,i}$. By monitoring the computed error bounds, one can conservatively predict which v_k will have significant components aligned with each Ritz vector. Consequently, one can then selectively orthogonalize [20] the Lanczos vectors against those Ritz vectors with substantial components. This ensures that the Lanczos vectors remain nearly orthogonal, allowing the process to be extended further. Nevertheless, executing this approach can be computationally expensive and not accurate always.

One more computationally cheaper way to handle the issue of multiple Ritz values is by starting the algorithm with a block of random vectors [18] which will be further discussed in the next section.

7.2 Block Lanczos Method

Block Lanczos algorithm [34] can be very helpful in order to handle the issue of loss of orthogonality of Lanczos vectors involved in the conventional Lanczos method. This method starts with a block of random vectors and as shown in table 7.1 the emergence of spurious eigenvalues can be delayed as one keep on using more random vectors in the starting block. Since the algorithm can be used for obtaining more number of Lanczos vectors. Hence, more eigenpairs can be captured.

Random vectors in starting block, p	Number of Lanczos vectors when Spurious eigenvalue appears, k
2	96
3	123
4	152
5	185

Table 7.1 The Block Lanczos method extends the traditional Lanczos algorithm by reaching a higher number of Lanczos vectors, thus ensuring a greater number of approximate eigenpairs. $\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$.

Algorithm for Block Lanczos method [34]:

- Initialize with a random ortho-normal matrix X of size n by p , where n is the dimension of the matrix A and p is the number of random vectors to start with. Let $X_1 = X$ and compute $M_1 = X_1^t A X_1$.

- Compute:

$$Z_{i+1} = \begin{cases} AX_1 - X_1 M_1 & \text{if } i = 1, \text{ or} \\ AX_i - X_i M_i - X_{i-1} R_i^t & \text{if } i > 1. \end{cases} \quad (7.3)$$

- Compute X_{i+1} and R_{i+1} such that X_{i+1} is ortho-normal, R_{i+1} is upper triangular [34] and

$$Z_{i+1} = X_{i+1} R_{i+1}. \quad (7.4)$$

- Compute AX_{i+1} and

$$M_{i+1} = X_{i+1}^t A X_{i+1}. \quad (7.5)$$

Hence, one can compute the symmetric block tridiagonal matrix, \bar{M}_s where $1 \leq s \leq n/p$ as

$$\bar{M}_s = \begin{pmatrix} M_1 & R_2^t & 0 & \cdots & 0 \\ R_2 & M_2 & R_3^t & 0 & 0 \\ 0 & R_3 & \ddots & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & R_s^t \\ 0 & \cdots & 0 & R_s & M_s \end{pmatrix} \text{ and} \quad (7.6)$$

$$\bar{X}_s = (X_1, X_2, \dots, X_s). \quad (7.7)$$

Thus, one can solve for the eigenvalues of this block tridiagonal matrix, \bar{M}_s which will be approximations to the eigenvalues of A and similarly the eigenvectors g_i of matrix \bar{M}_s will be used to approximate the eigenvectors, y_i of A given by $y_i = \bar{X}_s g_i$. As one keeps on using more and more random vectors in the starting block, the sparsity pattern for the tridiagonal block matrix keeps on decreasing, causing a substantial increase in the computations. For example, when one start with a block of two random vectors, the matrix \bar{M}_s has a pentadiagonal structure. However, in the Lanczos algorithm, it is very important to start with a block of p random vectors in order to handle the degeneracy of at most p eigenpairs [21]. This can be shown by considering a symmetric matrix

$$A = \begin{bmatrix} -64 & 16 & 0 & 16 & 0 & 0 & 0 & 0 & 0 \\ 16 & -64 & 16 & 0 & 16 & 0 & 0 & 0 & 0 \\ 0 & 16 & -64 & 16 & 0 & 16 & 0 & 0 & 0 \\ 16 & 0 & 16 & -64 & 16 & 0 & 16 & 0 & 0 \\ 0 & 16 & 0 & 16 & -64 & 16 & 0 & 16 & 0 \\ 0 & 0 & 16 & 0 & 16 & -64 & 16 & 0 & 16 \\ 0 & 0 & 0 & 16 & 0 & 16 & -64 & 16 & 0 \\ 0 & 0 & 0 & 0 & 16 & 0 & 16 & -64 & 16 \\ 0 & 0 & 0 & 0 & 0 & 16 & 0 & 16 & -64 \end{bmatrix} \quad (7.8)$$

of size nine by nine with a degeneracy of three and the eigenvalues

$$\lambda_i = \{-109.255, -86.6274, -86.6274, -64, -64, -64, -41.3726, -41.3726, -18.7452\}. \quad (7.9)$$

The block Lanczos algorithm using a starting block of three random vectors can be used to capture all the nine eigenvalues with a degeneracy of three as shown in table 7.2.

Block Lanczos steps	Vectors in \tilde{X}_s	Ritz values obtained θ_i
2	6	-107.338 -83.0802 -67.7687 -63.3241 -46.3532 -19.9837
3	9	-109.255 -86.6274 -86.6274 -64 -64 -64 -41.3726 -41.3726 -18.7452

Table 7.2 Block Lanczos Method implemented on A of size 9×9 .

But the degeneracy of a matrix A cannot be predetermined. Therefore, this algorithm operates more as an iterative procedure, where one can continually increase the number of random vectors in the initial block by doubling them.

7.3 Observations in the Lanczos Method

The significant observation lies in the distinctive pattern evident when the Lanczos method fails i.e., as the eigenvalues approach their approximate values, multiple Ritz values become apparent [33] and the method fails. As shown in table 7.3 when the Lanczos method is implemented on a diagonal matrix A of size 6 by 6 given by $\lambda_i = \{-2.5, -1.5, 1.0, 2.0, 3.0, 100000\}$, it is observed that as the five eigenvalues near convergence in the sixth Lanczos step, multiple eigenvalues emerge, one of which matches the dominant eigenvalue 100000. The similar

behaviour can be seen for another symmetric matrix,

$$\mathbf{A}_1 = \begin{bmatrix} 1956.96 & 5398.64 & -7266.68 & -715.352 & -5252.04 & -9029.24 \\ 5398.64 & 14916.4 & -20077 & -1974.99 & -14509 & -24946.8 \\ -7266.68 & -20077 & 27028 & 2657.45 & 19530.1 & 33581.5 \\ -715.352 & -1974.99 & 2657.45 & 264.13 & 1919.41 & 3302.53 \\ -5252.04 & -14509 & 19530.1 & 1919.41 & 14118 & 24267.7 \\ -9029.24 & -24946.8 & 33581.5 & 3302.53 & 24267.7 & 41731.5 \end{bmatrix}, \quad (7.10)$$

with eigenvalues $\lambda_i = \{100000, 5.0, 4.0, 3.0, 2.0, 1.0\}$ as shown in table 7.4. In the final row of Table 7.4, it becomes apparent that as the Ritz values approach the actual eigenvalues, a new Ritz value 81035.9 emerges, likely representing a duplicate of the eigenvalue 100000.

Lanczos steps	$\lambda_1 =$ 100000	$\lambda_2 =$ 3.0	$\lambda_3 =$ -2.5	$\lambda_4 =$ 2.0	$\lambda_5 =$ -1.5	$\lambda_6 =$ 1.0
2	100000					1.319
3	100000	2.582			-1.834	
4	100000	2.795		2.117		1.065
5	100000	3.069	-2.357	2.136	-1.179	
6	100000	2.999	-2.499	1.998	-1.498	100000

Table 7.3 Lanczos Method implemented on A with $\lambda_i = \{-2.5, -1.5, 1.0, 2.0, 3.0, 100000\}$.

A remarkable but ironic consequence of the Paige (1971) error analysis [25] also suggests that loss of orthogonality goes hand-in-hand with convergence of a few Ritz pairs,

$$\|A\hat{y}_i - \theta_i \hat{y}_i\|_2 \approx |\beta_{k+1}| |\hat{z}_{ki}| \quad (7.11)$$

where $\hat{\theta}_i$ are the computed Ritz values for the floating point version of the symmetric tridiagonal matrix, \hat{T}_k and \hat{z}_i is the corresponding floating point version of the eigenvector. The vector \hat{y}_i is the floating point approximation of eigenvector of matrix A given by $\hat{y}_i = V_k \hat{z}_i$. It can be observed that as the Eigenpair converges i.e., $A\hat{y}_i$ is equal to the $\theta_i \hat{y}_i$ within the tolerance specified, the scalar β_{k+1} goes to zero. It was further shown that β_{k+1} is the norm

Lanczos steps	$\lambda_1 =$ 100000	$\lambda_2 =$ 5.0	$\lambda_3 =$ 4.0	$\lambda_4 =$ 3.0	$\lambda_5 =$ 2.0	$\lambda_6 =$ 1.0
2	100000				2.246	
3	100000		4.034		1.997	
4	100000	4.59854		2.57198	1.8279	
5	100000	4.907	3.524		2.04077	1.15705
6	100000	4.90776	3.52462	81035.9	2.04077	1.15702

Table 7.4 Lanczos Method implemented on A_1

of the vector v_k . Also, as shown in [29],

$$|v_k^T v_{k+1}| \leq \frac{2\sigma \varepsilon_0}{\beta_{k+1}} \quad (7.12)$$

where σ represents the 2-norm of A and $\varepsilon_0 \equiv (n+4)\varepsilon$ respectively. It is clear that as the scalar β_{k+1} goes close to zero, the inner product of vectors v_k and v_{k+1} is a finite value and hence, the Lanczos vectors process are no more orthogonal. Hence, the Lanczos process can not be continued further and as a numerical indication, it has already been shown that there is an emergence of spurious eigenvalues as soon as few of the Eigenpairs converge or the vectors lose orthogonality.

Also, the convergence of eigenvalues in Lanczos iterations depends upon the spectral gap [20] i.e. the difference in absolute magnitudes of the eigenvalues. If the difference is large, then they converge in the very few initial Lanczos steps as seen in table 7.3, and the eigenvalue 100000 converges in the first Lanczos step itself. Furthermore, in table 7.5 considering the diagonal matrix $A = \text{diag}(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100, 500, 1000)$, there is a significant contrast in the eigenvalue differences among top three eigenvalues 1000, 500 and 100 compared to the spectral gap for top three eigenvalues 1000, 900 and 800 of another diagonal matrix $A = \text{diag}(1, 2, 3, \dots, 9, 10, 100, 200, 300, \dots, 800, 900, 1000)$ in table 7.6. In the former case, the Lanczos algorithm converges within the first four steps exploiting the significant spectral gap while in the latter case, it requires eleven Lanczos steps for convergence. This is following the work done by Paige [25] where ε_j is the error norm of the approximation of vectors y_j to some eigenvectors x_j of matrix A given by

$$\varepsilon_j^2 \leq \frac{\lambda_j - \theta_j + \sum_{i=1}^{j-1} (\lambda_i - \lambda_{j+1}) \varepsilon_i^2 / (1 - \varepsilon^2)}{(\lambda_j - \lambda_{j+1})}. \quad (7.13)$$

As the gap i.e., $(\lambda_j - \lambda_{j+1})$ is large, the error norm ϵ_j^2 is small, and hence, one of the eigenpair converges.

Lanczos steps	$\lambda_1 =$ 1000	$\lambda_2 =$ 500	$\lambda_3 =$ 100
2	915.584	38.287	
3	997.417	469.209	12.623
4	1000	499.989	99.352

Table 7.5 Lanczos Method implemented on A with $\lambda_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100, 500, 1000\}$.

Lanczos steps	$\lambda_1 =$ 1000	$\lambda_2 =$ 900	$\lambda_3 =$ 800
2	731.223	67.186	
3	864.141	388.61	23.864
4	923.186	658.903	271.879
5	973.037	792.111	458.753
6	988.694	823.133	558.142
7	995.965	857.677	746.449
8	999.179	888.535	787.157
9	999.926	898.449	797.153
10	999.989	899.655	798.917
11	1000	899.992	799.949

Table 7.6 Lanczos Method implemented on A with $\lambda_i = \{1, 2, \dots, 9, 10, 100, 200, \dots, 900, 1000\}$.

It can be shown that the Lanczos method behaves much better if the eigenvalues are not well-separated i.e. the emergence of multiple Ritz values comes at a much later stage. It is evident when the Lanczos algorithm is implemented for the case of a diagonal matrix given by $A = \text{diag}(1, 2, 3, \dots, 28, 19, 30, 1000)$ of size 31×31 in comparison to the case where there is one more extra eigenvalue of $\lambda_{32} = 1000 + 1 \times 10^{-15}$ also included in the matrix

with

$$\lambda_i = \{1, 2, 3, \dots, 28, 29, 30, 1000, 1000 + 1 \times 10^{-15}\}, \quad (7.14)$$

of size 32 by 32 as shown in the tables 7.7 and 7.8. The multiple eigenvalue i.e., 919.755 emerges in the eleventh Lanczos step in table 7.7 when ten approximate eigenvalues were computed and hence, the conventional Lanczos method can be used to approximate these ten eigenpairs only. However, from table 7.8, it is significant that when one is including one more eigenvalue i.e., $\lambda_{32} = 1000 + 1 \times 10^{-15}$ in order to create closely spaced eigenvalues in the spectra of A , the Lanczos method can work till twenty Lanczos steps and can compute eighteen better approximates of eigenpairs.

So, this fact can be well-established from the above observations that one can actually use the conventional Lanczos method for a much more longer number of iterations if the eigenvalues are closely spaced. Even though the eigenvalues converges faster when the spectral gap is large but that would also suggest the emergence of multiple Ritz values known as spurious eigenvalues [23] which is not desirable and is not at all the part of the actual physical system. The method fails due to the loss of orthogonality of Lanczos vectors which is observed numerically when there are spurious eigenvalues being observed. Since Lanczos method is used to compute few k eigenpairs for a symmetric matrix rather than focussing only on computing one eigenvalue as in case of the power method, it becomes important for this method to run for more number of iterations in order to get more approximate eigenpairs. This can be established by slowing down the convergence of the extreme eigenvalues which can be made possible by reducing the spectral gaps among them.

Lanczos Steps	Ritz values				
11	1000	919.755(Multiple)	29.941	27.885	24.713
	21.105	15.518	11.392	6.358	3.179
	1.484				

Table 7.7 Lanczos method implemented on A with $\lambda_i = \{1, 2, 3, \dots, 28, 19, 30, 1000\}$

Lanczos Steps	Ritz Values				
20	1000	1000	999.93 (Multiple)	29.99	28.98
	27.953	26.37	25.01	22.81	21.06
	18.208	15.26	12.81	11.11	8.81
	5.929	4.27	3.03	2.01	1.00

Table 7.8 Lanczos method implemented on A with $\lambda_i = \{1, 2, 3, \dots, 28, 29, 30, 1000, 1000 + 1 \times 10^{-15}\}$.

7.4 Enhancing Lanczos Method Implementation: Insights and Remedies

To enhance the Lanczos method, one needs to find a way to delay the spurious eigenvalues as mentioned in the previous section as late as possible in order to obtain the maximum number of eigenpairs from both the trailing and leading edge of the eigen spectrum. This can be achieved by ensuring closely spaced eigen pairs which would result in delayed convergence for the Lanczos algorithm. One can achieve this by perturbing the eigenvalues in the eigen spectra for the system under consideration.

Numerically, it can be shown that the Lanczos algorithm behaves very differently on a different permutation for the same matrix. One can define a similarity transformation using a permutation matrix for a diagonal matrix $A = \text{diag}(-2.5, -1.5, 1.0, 2.0, 3.0, 100000)$ such that $A_{perm1} = P_1 A P_1$ where

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (7.15)$$

$$A_{perm1} = \begin{bmatrix} -2.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.5 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{100000} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 2.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1.0} \end{bmatrix}. \quad (7.16)$$

The Lanczos algorithm implemented for $A = \text{diag}(-2.5, -1.5, 1.0, 2.0, 3.0, 100000)$ and the tridiagonal matrix

$$M = \begin{bmatrix} 15642.1 & 36323.9 & 0 & 0 & 0 & 0 \\ 36323.9 & 84359.2 & 5.04644 & 0 & 0 & 0 \\ 0 & 5.04644 & -0.571758 & 1.09691 & 0 & 0 \\ 0 & 0 & 1.09691 & 0.996597 & 1.93608 & 0 \\ 0 & 0 & 0 & 1.93608 & -0.074908 & 258.635 \\ 0 & 0 & 0 & 0 & 258.635 & 99999.3 \end{bmatrix} \quad (7.17)$$

obtained is very different from

$$M_{perm1} = \begin{bmatrix} 41.8451 & 2013.91 & 0 & 0 & 0 & 0 \\ 2013.91 & 99959.3 & 91.2015 & 0 & 0 & 0 \\ 0 & 91.2015 & -0.383081 & 1.40726 & 0 & 0 \\ 0 & 1.40726 & 0.908207 & 1.50662 & 0 & 0 \\ 0 & 0 & 0 & 1.50662 & -0.677639 & 1.26616 \\ 0 & 0 & 0 & 0 & 1.26616 & 108.019 \end{bmatrix} \quad (7.18)$$

obtained for A_{perm1} in equation 7.16.

Lanczos steps	$\lambda_1 =$ 100000	$\lambda_2 =$ 3.0	$\lambda_3 =$ -2.5	$\lambda_4 =$ 2.0	$\lambda_5 =$ -1.5	$\lambda_6 =$ 1.0
2	99999.9					1.269
3	100000		2.433		-1.630	
4	100000	2.795		-2.117		1.033
5	100000	2.947	-2.391	1.625	-1.147	
6	100000	2.946	-2.393	1.622	-1.156	108.034

Table 7.9 Lanczos Method implemented on $A = \text{diag}(-2.5, -1.5, 1.0, 2.0, 3.0, 100000)$.

Also, as shown in table 7.9, the Ritz values obtained while implementing the Lanczos algorithm on A_{perm1} are very different from the original matrix as shown in table 7.3.

This technique of using permutations of original matrix along with the original matrix can be used to create closely-spaced eigen gaps which can delay the convergence of the Lanczos algorithm and hence one can achieve more approximate eigenpairs. It can be shown that the symmetric matrix A_1 in equation 7.10 with eigenvalues $\lambda_i = \{100000, 5.0, 4.0, 3.0, 2, 0, 1.0\}$ along with the permutation group $A_{1_{perm2}} = P_2^T A_1 P_2$ computed using

$$P_2 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (7.19)$$

can be used to construct a bigger matrix

$$B_{A_1} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_2 \mathbf{A}_1 \mathbf{P}_2 \end{bmatrix} \quad (7.20)$$

as shown in table 7.10 can be used to compute all the eigenvalues for the original matrix A_1 . The last Ritz value 100000 which emerges in the seventh Lanczos step is the result of using

one more permutation group along with the original matrix A_1 when the Lanczos algorithm is implemented on the larger matrix B_{A_1} .

Steps	$\lambda_1 =$ 100000	$\lambda_2 =$ 3.0	$\lambda_3 =$ -2.5	$\lambda_4 =$ 2.0	$\lambda_5 =$ -1.5	$\lambda_6 =$ 1.0	
2	100000	2.03				1.31	
3	100000			3.48	1.63		
4	100000	4.458			2.236	1.17	
5	100000		4.43		2.219	1.15	24.16
6	100000	4.88	3.579		2.03	1.01	100000
7	100000	5.0	4.0	3.0	2.0	1.0	100000

Table 7.10 Lanczos Method implemented on B_{A_1}

Utilizing similarity transformation by using distinct permutations alongside the original matrix within the diagonal blocks presents a beneficial approach for computing all eigenpairs of the original matrix using the Lanczos algorithm. The spurious eigenvalues can be pushed further by implementing the Lanczos algorithm on the larger matrix. Upon detection of a spurious eigenvalue, enlarging the size of the larger matrix can be achieved by incorporating additional permutations of original matrix. Based on the observation from Figure 7.1, with a matrix $A_{original} = diag(0, 1, 2, 3, 4, 498, 499, 500, 550, 600)$ and rows $n = 503$ the Lanczos method encounters failure after approximately forty-five iterations due to the emergence of spurious eigenvalues. Consequently, it becomes inadequate for computing the remaining eigen pairs of the matrix $A_{original}$. However, if we continue to augment our original matrix with additional permutation groups and integrate them into a larger matrix,

$$B_A = \begin{bmatrix} A_{original} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & P_1 A_{original} P_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & P_2 A_{original} P_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & P_k A_{original} P_k \end{bmatrix} \quad (7.21)$$

of size $n(k+1)$ by $n(k+1)$ one can actually compute all the eigenvalues as shown in the table 7.11.

No. of permutation groups clubbed, k	Lanczos iterations	Eigenvalues computed
16	720	$\lambda_i = \{0, 1, 2, \dots, 499, 500, 550, 600\}$

Table 7.11 Lanczos method implemented on B_A of size $(k+1)n \times (k+1)n$ using $A_{original}$ with $\lambda_i = \{0, 1, 2, 3, 4, \dots, 498, 499, 500, 550, 600\}$ placed along the diagonal along with its k different permutation groups. All the eigen pairs for the matrix $A_{original}$ can be computed.

7.4.1 Computing k eigen pairs: Working algorithm

The various steps to implement this algorithm are:

1. Start with including one permutation of the original matrix along with the original matrix in the diagonal block of the bigger matrix of size $2n_{original}$ by $2n_{original}$.
2. Run the Lanczos method using the bigger matrix and compute the eigenpairs.
3. There is a criteria given by [23], [24] to actually discard the spurious eigenvalues. One has to compute the tridiagonal matrix \tilde{T}_k from the tridiagonal matrix T_k obtained in step 2 by removing the first row and column. This is done to remove the effects of the random vector from the matrix. Then, the eigenvalues obtained in the matrix \tilde{T}_k which are equal to the eigenvalues obtained in the matrix T_k are the ones spurious in nature and thus, are discarded.
4. Use the criteria mentioned above and see if the desired k number of eigen pairs are obtained.
5. If more eigenpairs are required, then start including more permutations of the original matrix into the bigger matrix. One can actually start doubling the size of the bigger matrix at every step by including more permutation groups of the original matrix at every step.

Chapter 8

Applications using Lanczos Algorithm

8.1 Principal Component Analysis

Principal Component Analysis (PCA) [10] is a widely used statistical technique in the field of data analysis and dimensionality reduction. It's particularly effective for reducing the dimensionality of large datasets while preserving the essential information. PCA accomplishes this by transforming the original variables into a new set of variables, called principal components, which are linear combinations of the original variables. These principal components are orthogonal to each other and capture the maximum variance in the data.

1. **Understanding the Data:** PCA begins with a dataset consisting of n observations and p variables/features. Each observation represents a data point, and each variable represents a different aspect or attribute of the data.
2. **Standardization:** Before applying PCA, it's essential to standardize the data by subtracting the mean and scaling to unit variance. Standardization ensures that all variables contribute equally to the analysis, preventing variables with larger scales from dominating the computation of principal components.
3. **Covariance Matrix:** PCA operates on the covariance matrix of the standardized data. The covariance matrix provides information about how each variable varies with every other variable. It's calculated by taking the dot product of the transpose of the data matrix and the data matrix itself, divided by the number of observations minus one.
4. **Eigenvalue Decomposition:** The next step involves computing the eigenvalues and eigenvectors of the covariance matrix. Eigenvectors are vectors that don't change direction when a linear transformation is applied to them, only their magnitude changes.

Eigenvalues represent the amount of variance explained by each eigenvector. These eigenvectors and eigenvalues form the basis of the principal components.

5. **Selection of Principal Components:** The eigenvectors are ranked in descending order of their corresponding eigenvalues. The eigenvector with the highest eigenvalue is the first principal component, the second highest eigenvalue corresponds to the second principal component, and so on. Typically, we choose the top k eigenvectors that capture the most variance in the data.
6. **Projection:** Once the principal components are identified, the original data is projected onto these components. This is done by taking the dot product of the standardized data matrix and the matrix composed of the selected eigenvectors.
7. **Dimensionality Reduction:** PCA allows us to reduce the dimensionality of the dataset by retaining only the top k principal components. These components represent a compressed representation of the original data while preserving as much of the variance as possible. The reduced dataset can then be used for further analysis or visualization.

By following these steps, PCA provides a powerful tool for analyzing high-dimensional data and extracting meaningful insights.

Lanczos method can also be used to perform PCA in a computationally efficient manner. Rather than first computing all the eigenvectors and arranging them in a descending order, it is possible to use the first few k —dominant eigenvector approximations obtained after a few steps of Lanczos iterations. To illustrate this, PCA will be performed and first two principal components will be obtained for a well-known dataset frequently used in machine learning and statistical analysis i.e., the "Breast Cancer Wisconsin" [35] dataset which originated from a study conducted at the University of Wisconsin. Each data point in the dataset is labeled as either benign (357 data points) or malignant (212 data points), indicating whether the corresponding breast mass is benign (non-cancerous) or malignant (cancerous). In total, there are 569 instances based on 30 attributes such as radius, texture, perimeter, area, smoothness, compactness, concavity, symmetry etc. which is essentially a 30 dimensional feature set. PCA was applied to a dataset containing 569 data points, projecting them onto the first two principal components derived from a 30 by 30 covariance matrix. The results are shown in figure 8.1 which clearly shows formation of cluster in some sense i.e., yellow for the malignant breast masses, thus separating the two labels. This part is performed using the 'PCA' function provided by the 'scikit-learn' [36] library in Python.

In figure 8.2, the principal components are taken to be in the first two eigen directions obtained using the Lanczos algorithm by considering the matrix $V = \{v_1, v_2, v_3, v_4, v_5\}$ using

first five Lanczos vectors [32] and computing the tri-diagonal matrix T_k of size five by five. It is fairly clear that the same results are captured using first five Lanczos vectors which is computationally much more cheaper compared to computing the eigenpairs for a matrix of size thirty by thirty and then rearranging them in a descending order and finally picking up the dominant principal components.

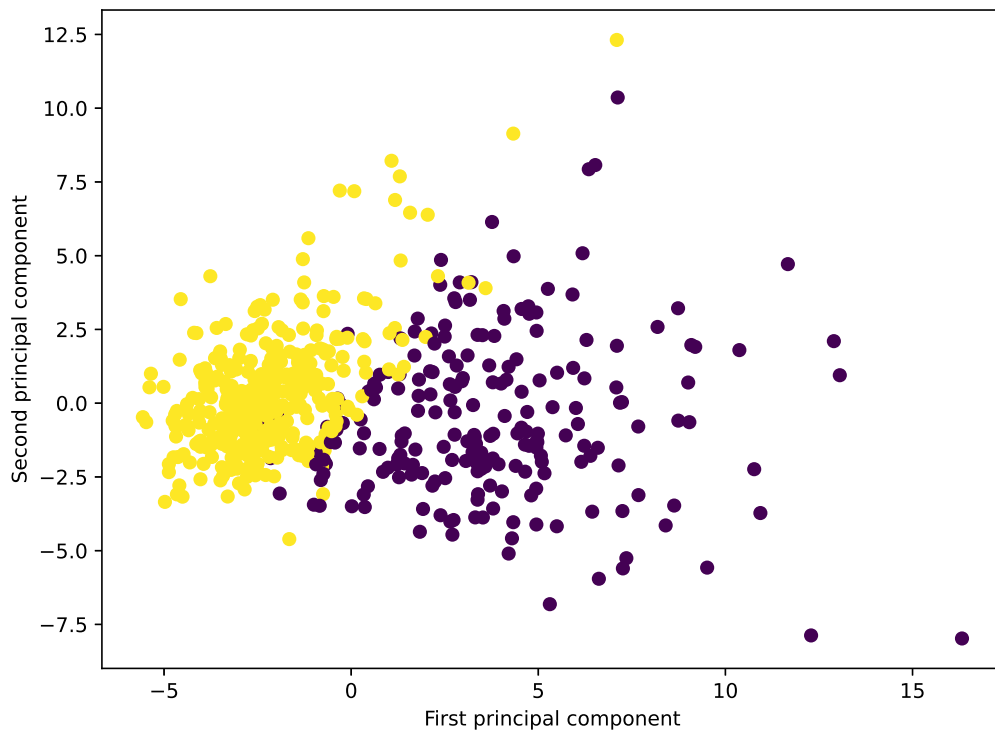


Fig. 8.1 Using 'PCA' function in Python, first two principal components are obtained and hence, the data is projected to obtain clusters in order to label the data as malignant or benign.

The result demonstrated in this chapter highlights that the Lanczos algorithm not only retains the accuracy required for effective PCA but also enhances scalability, making it feasible to apply PCA to high-dimensional data prevalent in modern applications.

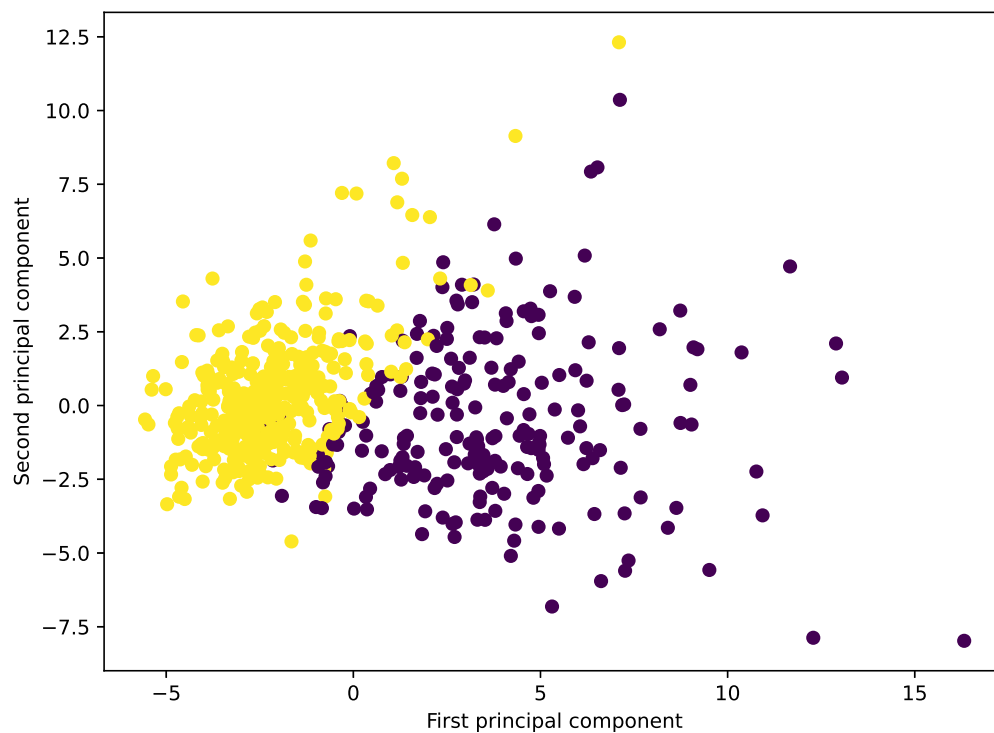


Fig. 8.2 Using the first 5 Lanczos vectors i.e. $V_5 = \{v_1, v_2, v_3, v_4, v_5\}$ and computing the tri-diagonal matrix $T_5 = V_5^T A V_5$ where A is 30×30 features matrix. Computing the first two dominant approximate eigenvectors using the eigenvectors of T_5 , one can use them as good approximations to the first two principal components. Thus, all the data points are projected using these approximations to the principal components resulting in separation of data into clusters.

Chapter 9

Outlook

This thesis has focused on the iterative eigenvalue solution techniques using Krylov subspaces with an emphasis on the Lanczos algorithm for the computation of eigenvalues for large sparse symmetric matrices. One of the popular applications in machine learning i.e., Principal component analysis is implemented in C++ using the Lanczos vectors, and the results are compared with the in-built functions in Python's library Sci-kit Learn. The focus was on the behavior of the method when finite precision arithmetic is used. The loss of orthogonality of Lanczos vectors due to finite precision arithmetic was demonstrated through various examples. It was shown that as predicted by the Paige analysis [33] the method leads to spurious eigenvalues whenever the Ritz values converge to a particular eigenvalues.

A new observation in this thesis was that permutations of the same matrix have a very different convergence behavior of eigenvalues for the Lanczos algorithm in finite precision. This became the key motivation to club various similarity transformations of the original matrix using different permutations together to slow down the Lanczos process and capture more eigenvalues. When going into a higher dimension and computing the larger matrix

$$B_A = \begin{bmatrix} A_{\text{original}} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & P_1 A_{\text{original}} P_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & P_2 A_{\text{original}} P_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & P_k A_{\text{original}} P_k \end{bmatrix} \quad (9.1)$$

of size $n(k+1)$ by $n(k+1)$. Unlike the original matrix A , the augmented matrix B_A has a very bad spectral gap in finite precision and would lead to degenerate Eigenvalues in infinite

precision. A seemingly counter-intuitive result in the thesis is that a bad spectral gap which slows down the convergence for this algorithm helps in better stability behaviour for the method. It is shown numerically that an appropriately augmented matrix indeed can evaluate all eigenvalues.

A formal analysis of the extended method is left as future work. Only such an analysis can shed light on the observation that a badly conditioned matrix leads to better convergence of the Lanczos method. Such theoretical analysis will put the variant proposed in this thesis on sounder footing. Similarly, a careful numerical analysis is needed to understand the computational merit of this approach.

References

- [1] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, third edition, 2022.
- [2] Thomas Nield. *Essential Math for Data Science: Take Control of Your Data with Fundamental Linear Algebra, Probability, and Statistics*. O'Reilly, 2022.
- [3] Stephen J. Chapman. *MATLAB Programming for Engineers*. Cengage Learning, 6th edition, 2020.
- [4] The MathWorks Inc. Matlab version: 9.13.0 (r2022b), 2022. <https://www.mathworks.com>.
- [5] Jesper Schmidt Hansen. *GNU Octave: Beginner's Guide: Become a proficient octave user by learning this high-level scientific numerical tool from the ground up*. Packt Publishing Ltd, 2011.
- [6] T. Pang. *An Introduction to Computational Physics*. Cambridge, second edition.
- [7] Shawn T.O' Neil. *A Primer for Computational Biology*. Oregon State University.
- [8] Edward Tsang and Serafin Martinez-Jaramillo. Computational finance. *IEEE computational intelligence society newsletter*, 3(8):8–13, 2004.
- [9] Mark J. Burge Wilhelm Burger. *Principles of Digital Image Processing: Core Algorithms*. Springer, 2009.
- [10] J.E. Jackson. *A User's Guide to Principal Components*. Wiley, 1991.
- [11] Gilbert Strang. *Linear Algebra and its applications*. Cengage, 2006.
- [12] D. Kincaid C. Lawson, R. Hanson and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5:5308–323, 1979.
- [13] Jack J Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J Hanson. An extended set of fortran basic linear algebra subprograms: Model implementation and test programs. Technical report, Argonne National Lab., IL (USA), 1987.

- [14] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [15] Bruce Eckel. *Thinking in C++*. Pearson Education, second edition.
- [16] Gael Guennebaud, Benoit Jacob, et al. Eigen: a c++ linear algebra library. *URL <http://eigen.tuxfamily.org>*, Accessed, 22, 2014.
- [17] Conrad Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. 2010.
- [18] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Hindustan Book Agency, third edition, 2007.
- [19] William T. Vetterling William H. Press, Saul A. Teukolski and Brian P. Flannery. *Numerical Recipes in Fortran*. Cambridge University Press, second edition, 1986.
- [20] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Society of Industrial and Applied Mathematics (SIAM) Philadelphia, 1998.
- [21] Louis Komzsik. *The Lanczos Method Evolution and Application*. Society of Industrial and Applied Mathematics (SIAM) Philadelphia, 2003.
- [22] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. 1950.
- [23] Jane Cullum and Ralph A. Willoughby. Computing eigenvalues of very large symmetric matrices — an implementation of a lanczos algorithm with no reorthogonalization. *Journal of Computational Physics* 44.2 (1981): 329-358.
- [24] Prakash Dayal. Numerical simulations of condensed matter: From eigensolvers to quantum spin glass, 2006. Diss. ETH No. 16588.
- [25] Christopher Conway Paige. The computation of eigenvalues and eigenvectors of very large sparse matrices, 1971.
- [26] J.H. Wilkinson. *Rounding Errors in Algebraic Processes; Notes on Applied Science No. 32*. HMSO, 1963.
- [27] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*, 23(1):5–48, 1991.
- [28] V. Rajaraman. Ieee standard for floating point numbers. *Resonance*, 21:11–30, 2016.
- [29] C. C. PAIGE. Error analysis of the lanczos algorithm for tridiagonalizing a symmetric matrix. *IMA Journal of Applied Mathematics*, 18:341 – 349, December 1976.

-
- [30] Å. Björck. Numerics of gram-schmidt orthogonalization. *Linear Algebra and its Applications*, 197–198:297–316, 1994.
 - [31] Hua-Gen Yu. A complex guided spectral transform lanczos method for studying quantum resonance states. *The Journal of Chemical Physics*, 141(24), 2014.
 - [32] Daniel Povey and Ariya Rastrow. Computational issues in principal components analysis. 2010.
 - [33] Chris C. Paige. Accuracy and effectiveness of the lanczos algorithm for the symmetric eigenproblem. *Linear algebra and its applications* 34 (1980): 235-258.
 - [34] G.H. Golub and R. Underwood. The block lanczos method for computing eigenvalues. pages 361–377, 1977.
 - [35] University of Wisconsin Hospitals, Madison. Breast cancer wisconsin (diagnostic) dataset. UCI Machine Learning Repository, 1995. [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)).
 - [36] Andrew Ng. *Machine Learning Yearning*. Deeplearning.ai, 2018.

