Matrix multiplication using the circulant decomposition

A Final Thesis submitted for the completion of requirements for the degree of

Master of Technology (Course Work)

by

Jinka Naga Sai Gowri
Post-graduate Programme
Indian Institute of Science

Under the supervision of

Prof. Murugesan Venkatapathi
Department of Computational and Data Science

Acknowledgement

I express my sincere gratitude to my guide, Professor **Murugesan Venkatapathi**, Department of Computational and Data Science, Indian Institute of Science, for his dedicated guidance, generous help, and the precious time he gave in supervising this dissertation report. I also would like to extend my sincere thanks to other professors who assist and support all the students. I would also like to thank my fellow batch mates who helped me with their valuable suggestions throughout the thesis work.

Date: JUNE 2024 Jinka Naga Sai Gowri

Place: IISc Bangalore 21669

Candidate's Declaration

I hereby declare that the work carried out in this dissertation report entitled "**Matrix Multiplication Using The Circulant Decomposition**" is being submitted in partial fulfilment of the requirements for the award of the degree of **"Master of Technology"** in **"Computational and Data Science"** submitted to the Department of computational and Data Science, Indian Institute of Science, Bengaluru, under the supervision of Professor **Murugesan Venkatapathi**, Computational and Data Science Department, IISc, Bengaluru.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Date: JUNE 2024	Jinka Naga Sai Gowri
Place: IISc Bengaluru	21669

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

(Murugesan Venkatapathi)
Professor
Department of Computational & Data Sciences (CDS),
Indian Institute of Science,
Bangalore - 560012.

Abstract

This work explores the application of the recently proposed circulant decomposition of square matrices. It is known that dense matrices with some periodicity of entries exhibit dominance of some of its 'n' circulant components. First, we analyze the distribution of the magnitudes of circulant components across various matrices with periodicity, such as Toeplitz, Hankel, and Symmetric matrices. This analysis reveals the fraction of circulant components required to approximate such matrices reasonably. Such approximated matrices were earlier used for fast eigenvalue approximations and preconditioning linear solvers. Here, we propose an iterative multiplication of any two matrices in $\mathcal{O}(n^2 \log n^2)$ operations, with well-bound relative errors $< 1\%$, unlike the other methods of approximation. We employ only a few circulant components of the residue matrices in the iterative multiplication of two matrices to quadratically converge to low relative errors. We validate our methodology, the iterations required across a diverse set of matrix types, and the corresponding arithmetic complexity of the algorithm. For reference, we also discuss two other approximate methods applicable to restricted matrix types, i.e., the mean approximation suited for fully uncorrelated entries in the two matrices and the low-rank approximation suited for matrices with a low effective rank.

As an addendum, we also present an approach to solve Toeplitz linear systems in $\mathcal{O}(n * \log n)$ arithmetic operations using the circulant decomposition, along with the $\mathcal{O}(n^2)$ Trench algorithm for exact solutions and another $\mathcal{O}(n * \log n)$ approximate method proposed elsewhere by others.

Contents

Chapter 1

Introduction

Computation with large matrices presents a significant challenge today as they become ubiquitous in use. Matrix decomposition, low-rank approximations, and projections on low-complexity classes are some of the approaches useful in reducing the computation incurred with dense matrices [**1**]. Decomposing a given $n \times n$ matrix into a sum of n circulant matrices with periodic relaxations on the unit circle is one potential method [**2**]. This circulant decomposition can substantially reduce arithmetic operations of computing for certain matrix evaluations with a modest accuracy. It is useful in approximate sparse similarity transformations, eigenvalue approximations, and preconditioning linear solvers.

Matrix multiplication, a fundamental operation in numerous applications, can be particularly time-consuming for large matrices with its $\mathcal{O}(n^3)$ arithmetic complexity for $n \times n$ matrices. The circulant decomposition allows for approximating a dense matrix using its dominant circulant components, and further, multiplying a circulant and any other matrix can be accomplished in $\mathcal{O}(n^2 \log n)$ arithmetic operations. Iterative application of this approach to the residue matrices allows us to reduce the arithmetic complexity for matrix multiplication with a reasonable accuracy in general.

We first recall that any matrix can be decomposed into n cycles that generate its 2n-1 diagonals. Later, we show that the decomposition of a matrix into n circulant matrices with periodic relaxations on the unit circle is equivalent to the decomposition of a similar matrix into such cycles[**2**]. By including only the dominant cycles of a similar matrix in such cycles, we include the dominant circulant components of the given matrix. The approximated matrix is used for matrix multiplication using the fact that multiplication involving circulant matrices can be done in much less arithmetic complexity.

1.1 Circulant decomposition of Matrix

Cycles of Matrix: Let $\mathbf{I}_n$ be the identity matrix of dimension n, and $\mathbf{C}$ be the permutation matrix corresponding to a full cycle.

$$C = \begin{bmatrix} 0 & 1 \\ \mathbf{I}_{n-1} & 0 \end{bmatrix}_{nxn} \tag{1.1}$$

Any matrix A can be decomposed into n cycles given by a power series in C such that $A = \sum_{k=0}^{n-1} \Lambda_k C^k$, where the Hadamard product $A \odot C^k = \Lambda_k C^k$, and Λ_k are diagonal matrices. Entries supported on C^k, i.e., diagonal entries of Λ_k in the above decomposition, are referred to as k^{th} cycle of the matrix A. [**2**]

Example 1.1 The decomposition of a matrix into cycles, $A = \sum_{j=0}^{n-1} \Lambda_j C^j$ for an order 3 magic square.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} C^0 + \begin{bmatrix} d & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & c \end{bmatrix} C^1 + \begin{bmatrix} b & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & g \end{bmatrix} C^2 \tag{1.2}$$

Let the permutation matrix given by a flipped identity matrix be

$$J = \begin{bmatrix} 0 & \dots & 0 & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 1 & 0 & 0 \\ \vdots & \ddots & 0 & 0 & 0 \\ 1 & \dots & 0 & 0 & 0 \end{bmatrix}_{n\times n} \tag{1.3}$$

Remark 1.1 $W^2 = CJ$, and any circulant matrix has an eigen decomposition $R = W\Lambda W^{\dagger}$. R is also given by $R = W^{\dagger}\widetilde{\Lambda}W$ where $\widetilde{\Lambda} = CJ\Lambda J^T C^T$. Thus for $1 \le k \le n-1$, we have $\widetilde{\Lambda}(k,k) = \Lambda(n-k, n-k)$ and $\Lambda(0,0) = \widetilde{\Lambda}(0,0)$.[**2**]

Lemma 1.2: Given diagonal matrices D_k with $D_k(q,q) = e^{i\frac{2\pi kq}{n}}$ (for $0 \le q \le n-1$) and any cirulant matrix R with eigenvalues given by a diagonal matrix $\widetilde{\Lambda}$, the matrices RD_k and $\widetilde{\Lambda}C^k$ are similar.[**2**]

Proof: Using W for a linear transformation of RD_k,

$$\begin{aligned} WRD_kW^\dagger &= WW^\dagger\widetilde{\Lambda}WD_kW^\dagger && (1.4)\\ &= \widetilde{\Lambda}WD_kW^\dagger && (1.5)\\ &= \widetilde{\Lambda}C^k. && (1.6)\end{aligned}$$

The substitution $WD_kW^\dagger = C^k$, can be deduced by evaluating its $(p,l)^{th}$ entry,

$$\begin{aligned} (p,l)^{th} \text{ entry of } WD_kW^\dagger &= \frac{1}{n}\sum_{q=0}^{n-1} e^{-i\frac{2\pi pq}{n}} e^{i\frac{2\pi kq}{n}} e^{i\frac{2\pi ql}{n}} && (1.7)\\ &= \frac{1}{n}\sum_{q=0}^{n-1} e^{-i\frac{2\pi(-p+k+l)q}{n}} && (1.8)\\ &= 1 \text{ when } p = l+k \mod n, && (1.9)\\ &\quad 0 \text{ otherwise} && (1.10)\end{aligned}$$

The matrix $\widetilde{\Lambda}C^k$ is sparse and represents a single cycle, while the similar matrix RD_k is dense. Remark recalls that any matrix decomposes into such cycles. The cyclic decomposition of a matrix A is equivalent to a circulant decomposition of a transformed similar matrix $WAW^\dagger$. Conversely, a circulant decomposition of the given matrix A is equivalent to a cycle decomposition of the transformed similar matrix $WAW^\dagger$; the same is proved in the theorem below.[**2**]

Remark 1.3 $W^2 = CJ$, and any circulant matrix has an eigen decomposition $R = W\Lambda W^\dagger$. R is also given by $R = W^\dagger\widetilde{\Lambda}W$ where $\widetilde{\Lambda} = CJ\Lambda J^TC^T$. Thus, for $1 \le k \le n-1$, we have $\widetilde{\Lambda}(k,k) = \Lambda(n-k,n-k)$ and $\Lambda(0,0) = \widetilde{\Lambda}(0,0)$.

proof Consider a matrix $B = WAW^\dagger$. Recalling its decomposition into cycles, and using the substituion $C^k = WD_kW^\dagger$ from proof of Lemma 1.3, we have $B = \sum_{k=0}^{n-1}\widetilde{\Lambda}C^k = \sum_{k=0}^{n-1}\widetilde{\Lambda}_kWD_kW^\dagger$, where $\widetilde{\Lambda}_k$ are diagonal matrices. Thus, the original matrix A is given by:

$$A = W^\dagger BW = \sum_{k=0}^{n-1} W^\dagger\widetilde{\Lambda}_kWD_k = \sum_{k=0}^{n-1} R_kD_k \tag{1.11}$$

1.1.1 Circulant Component of Matrix:

The n circulant components of any matrix A can be found by using two methods:

- Recursion based on orthogonality
- Using Similarity transformation.

Recursive method for circulant components: The following procedure gives the individual circulant components for a given matrix A.[**2**]

- Initialize $A_0 = A$.
- for $k = 0, 1, 2, \dots, n-2$
  - $R_k(0, j) = \frac{1}{n}(\vec{1}^T A_k \odot C^j \vec{1})$ (i.e., To discover the n unknown entries of circulant R_k, average the entries of A_k in the appropriate cycles for $j = 0, 1, \dots n-1$, and $\vec{1}$, an n-vector with all ones as entries.)
  - $A_{k+1} = (A_k - R_k)D_{-1}$.
- $R_{n-1} = A_{n-1}$

The arithmetic complexity for finding each circulant component is $\mathcal{O}(n^2)$; to find all circulant components, it requires $\mathcal{O}(n^3)$ operations, and a particular circulant component cannot be found directly as this process is sequential.

Using similarity transformation: The circulant matrix component R_k of A can also be evaluated using an inverse transformation of a cycle of $WAW^\dagger$, given by $W^\dagger(WAW^\dagger \odot C^k)W$. Note that R_0 minimizes $\|A - R\|_F$ for any circulant matrix. We know that the cycles of matrix A are similar to the circulant components of matrix B, and vice-versa, where matrix B can be written as $B = WAW^\dagger$. From the *remark 1.3*, any matrix A can be decomposed into cycles $A = \sum_{j=0}^{n-1} \Lambda_j C^j$. Let us consider the circulant decomposition of matrix B as $B = \sum_{i=0}^{n-1} R_i D_i$ where R_i is the $i^t h$ circulant component of matrix B from similarity transformation the i^{th} cycle of matrix A is similar to the i^{th} circulant component of B i.e. The circulant component R_i can be written as inverse transformation of cycle Λ_i. Hence the i^{th} circulant component of B can be written as $W^\dagger(\Lambda_i)W$, which can be further written as $W^\dagger(WAW^\dagger \odot C^k)W$

$$R_k = W^\dagger(WAW^\dagger \odot C^k)W \text{ (k-th circulant component of matrix A)} \tag{1.12}$$

Computing $WAW^\dagger$: The (p, q) entry of the matrix $B = WAW^\dagger$ is given by

$$B(p, q) = \frac{1}{n} \sum_{k=1}^{n} \sum_{j=1}^{n} e^{-i\frac{2\pi pk}{n}} A(k, j) e^{i\frac{2\pi jq}{n}} \tag{1.13}$$

Relation with fast Fourier transform: If $F(A)$ represents DFT of the columns of matrix A, we have

$$B = \frac{1}{n}F(F(A)^{\dagger})^{\dagger} \tag{1.14}$$

If we denote the two-dimensional DFT of the matrix A by $F^2(A)$, then we have $B = \frac{F^2(A)}{n}$. Thus, $WAW^{\dagger}$ is computed in $2n^2 \log n$ time. Similarly, the inverse transformation $W^{\dagger}AW$ can be done in $2n^2 \log n$ operations.

Remark 1.4: Any k^{th} circulant component of a matrix A given by $W^{\dagger}(WAW^{\dagger} \odot C^k)W$ can be found in $3n^2 \log n + n \log n + n$ arithmetic operations.

- $WAW^{\dagger}$ can be evaluated in $2n^2 \log n$ arithmetic operations using Fast-Fourier Transform (FFT).
- $\tilde{B} = WAW^{\dagger} \odot C^k$ can be done in n operations, where $\tilde{B}$ has only n non-zero entries corresponding to the k^{th} cycle.
- $W^{\dagger}\tilde{B}W$ can be done in $n^2 \log n + n \log n$ operations i.e. an inverse transformation.

Hence finding R_k using equation 1.12 takes $3n^2 \log n + n \log n + n$ operations. Additional circulant components do not require a re-evaluation of $WAW^{\dagger}$, and hence can be attained in $n^2 \log n + n \log n + n$ arithmetic operations.

1.2 Weights for circulant components:

The weight of a circulant component in this decomposition provides us its significance in approximating the matrix.

Weight of a cycle -w_i: Let $B = \sum_{i=0}^{n-1} R_i D_i$. The relative weight of circulant component R_i in B is $w_i = \frac{\|R_i\|^2{}_F}{\sum_i \|R_i\|^2{}_F}$. When $B = WAW^{\dagger}$, note that w_i also represents the relative weight of the cycle i in A, and $\sum_i w_i = 1, 0 \leq w_i \leq 1$.

If one circulant component holds more weight in a matrix, we can approximate the matrix effectively using that specific component. Weight distribution of circulant components of a matrix guide us in selecting the right circulant components for the approximation. All the matrices explained in Appendix 2 study how the weight distribution of circulant components depends on the structure of the matrix. Figures 1.1 and 1.2 shows the weight distribution of randomly generated matrices, 1.3 shows the distribution of weights of circulant matrices of buses, 1.4 shows the weight distribution of images and 1.5 illustrate the weight distribution of circulant components of the dft matrix and the dense matrix. To ensure fair comparison across matrices explained in appendix-2, all matrices are generated with a size of 250x250 using the numpy and scipy libraries

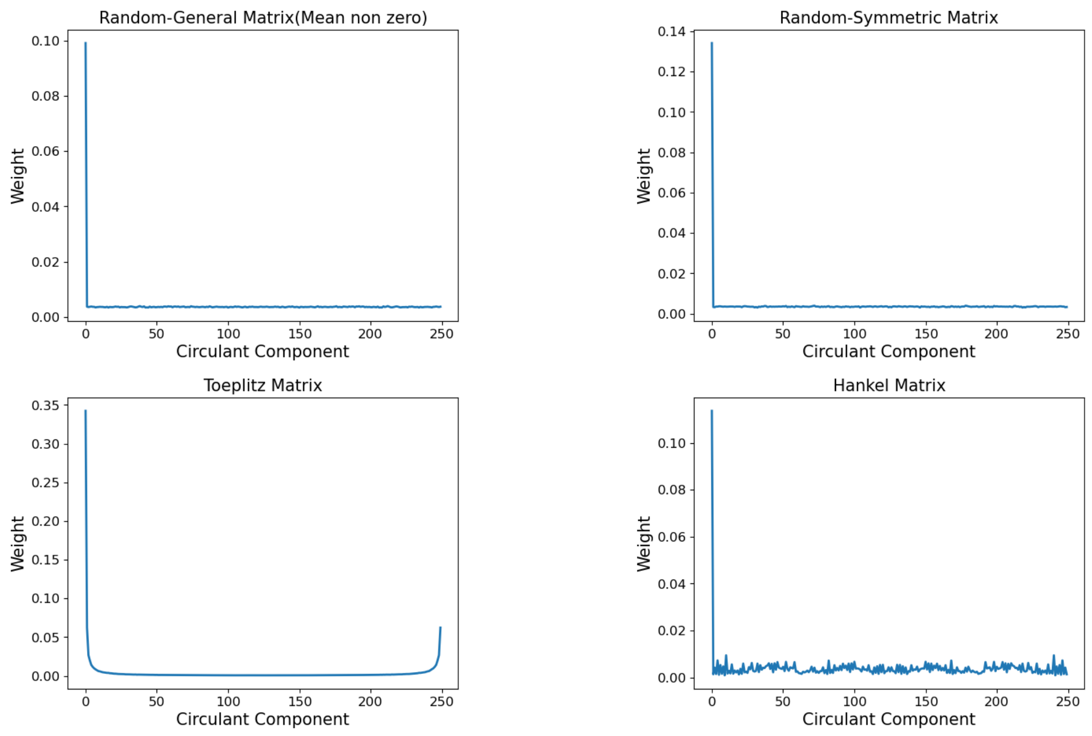


Figure 1.1: Weight Distribution of cycles for matrices generated from a uniform distribution for size 250

1.3 Approximating a Matrix

The decomposition of square matrices into circulant components offers a significant insight any $n \times n$ square matrix A can be expressed as the sum of n circulant matrices R_i along with their associated diagonal matrices D_i, representing the adjustments on the unit circle. This foundational result implies that the complexity of any matrix can be approximated using a reduced set of circulant matrices. The representation $A_{n\times n} = \sum_{i=1}^{n} R_i D_i$ illustrates the decomposition of a matrix A into its constituent circulant components. To approximate A, we can employ a reduced number k of these circulant components. i.e., A can be approximated as

$$A \approx \sum_{i=1}^{k} R_i D_i \tag{1.15}$$

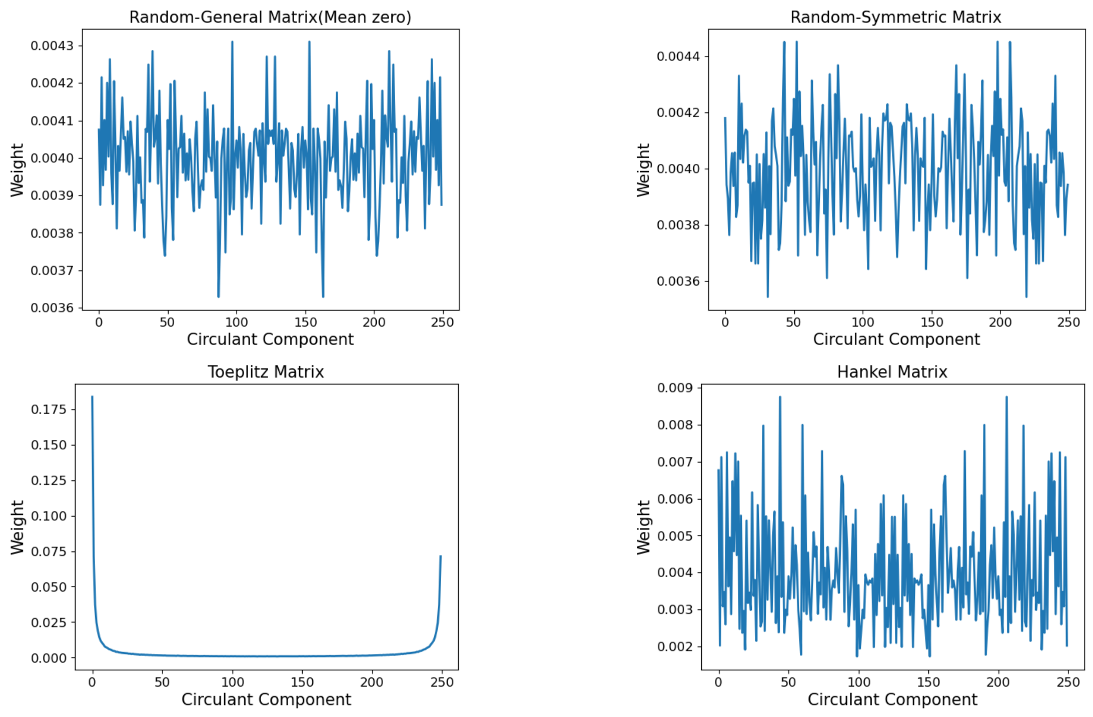


Figure 1.2: Weight Distribution of cycles for matrices generated from a mean zero distribution for size 250

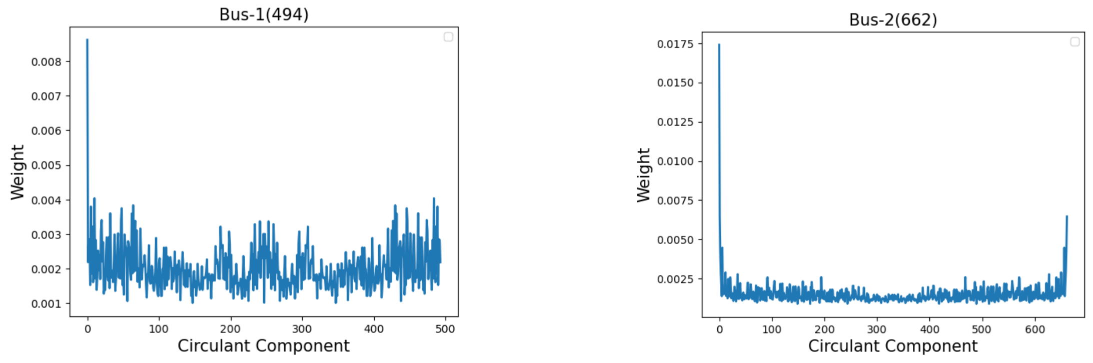


Figure 1.3: Weight Distribution of cycles for Bus matrices taken from Matrix-Market

The selection of the appropriate k is pivotal. Utilizing k circulant components instead of the complete set of n reduces computational complexity significantly while introducing some error compared to the full representation. The equation quantifies the relative error in the matrix approximation

$$\text{Relative error} = \frac{\|A - A_k\|}{\|A\|}$$

where A_k represents the matrix approximation obtained using k circulant matrices. Determining the optimal k necessitates a trade-off between reduced computational complexity and acceptable approximation error. Experimentation with varying values of k,

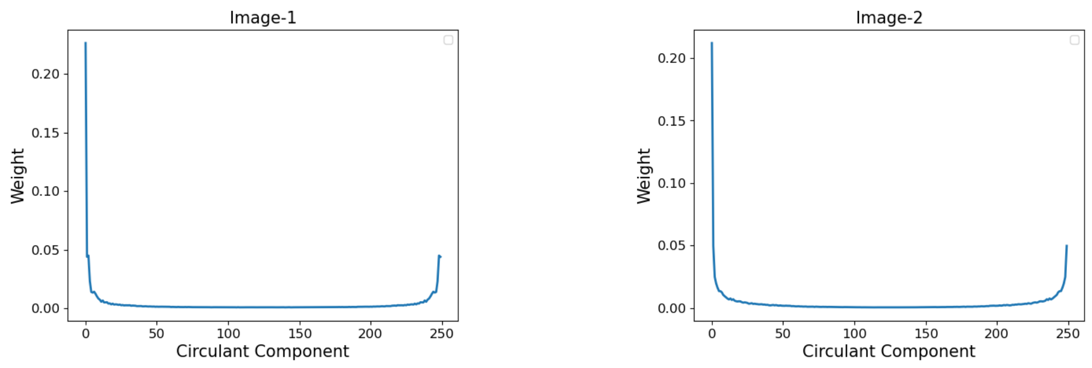


Figure 1.4: Weight Distribution of cycles for Images taken from Dog-Vs-Cat classification dataset

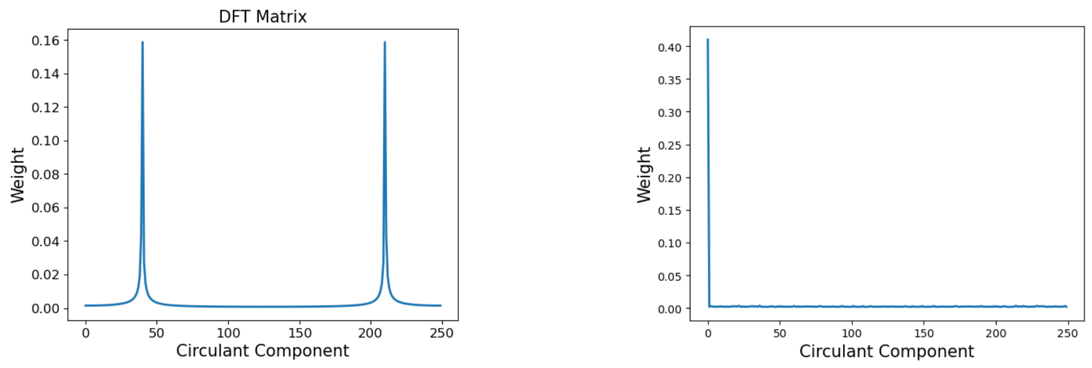


Figure 1.5: Weight Distribution of cycles for a synthetic periodic DFT matrix generated for size 250 on the left, and the dense symmetric DFT matrix based on physics on the right. See 6

typically including $k = 1$ and $k = \log n$, is often conducted to assess the performance of the approximation. In selecting k circulant components for the approximation, priority is accorded to those with greater weight, as determined by the underlying mathematical formulation.

Fraction of Weights Considered for approximation: The Weight of the circulant component is explained in section 1.2. The fraction of weight considered for approximation is given by

$$fraction = \frac{\sum_{i=1}^{k} w_i}{\sum_{i=1}^{n} w_i} \tag{1.16}$$

This fraction of weights, while considering the k-circulant components, depicts the better approximation of the matrices. As this approximation approaches one, the error in the approximation reduces to zero. There should be a trade-off between the value k and the fraction for weights we get by considering the k circulant components. Even if the k value is less, but the fraction of weights is more, the error in the approximation can be reduced. Plots 7.1,7.3, 7.4, 7.5 in Appendix 3 explain how the error in approximation changes with the number of circulant components considered and the matrix's size.

1.3.1 Analysis:

- The relative distribution of weights of the circulant components depends on the periodicity of elements in the matrix, and this periodicity can be non-trivial.
- From figures 1.1 and 1.2 the random matrices generated from a distribution with a non-zero mean have the first circulant component very dominant when compared to the matrices generated from a mean-zero distribution.
- From figures 1.1 and 1.2, it is also evident that in the matrices having periodicity in the elements, very few circulant components are dominant. Toeplitz has two dominant peaks in the distribution of magnitudes of circulant components.
- Figure 1.3 shows the weight distribution of circulant components of the bus matrices taken from the matrix market; from these figures, it can concluded that all the circulant components of the bus matrices have almost the same weights, so they cannot be approximated easily with a few circulant components.
- Figure 1.4 shows the weight distribution of circulant components of the images; from these, it is clear that these images can be approximated with few circulant components, especially dominated by the zero frequency circulant component.
- Figure 1.5
  - Image on the left shows the synthetic DFT matrix weight distribution; it shows that this DFT matrix can be approximated using a few circulant compnents.
  - The Image on the right shows the circulant weight distribution for a dense DFT matrix arising in a physical system.
- All these conclusions can be inferred from the matrix approximation plots shown in Appendix 3.

Chapter 2

Multiplication

Matrix multiplication is a fundamental operation in linear algebra with numerous applications in computer science and beyond. Its algorithmic complexity has been extensively studied for decades. In 1969, Strassen made a breakthrough by showing that $n \times n$ matrices can be multiplied faster than the naive cubic time algorithm [**3**]. This sparked significant progress in finding lower bounds on the exponent ω, which is defined as the smallest constant such that for all $\epsilon > 0$, $n \times n$ matrices can be multiplied using $\mathcal{O}(n\omega + \epsilon)$ arithmetic operations. Recently, it has been shown that $\omega < 2.373$ [**4**], [**5**], [**6**], and a new result by Duan, Wu, and Zhou further improves this to $\omega < 2.3719$ [**7**] The ideal bound would be $\omega = 2$, suggesting a near-linear time algorithm for matrix multiplication. However, a series of studies [**7**, **8**, **9**, **10**] has demonstrated that current techniques cannot achieve $\omega = 2$. Since 1986, all advancements in matrix multiplication have relied on various forms of the laser method [**6**, **11**, **7**]. The strongest limitation of the laser method and its variants is that they cannot show $\omega < 2.3078$[**7**]. To overcome this challenge, researchers have explored strategies like matrix decompositions, low-rank approximations, and projections onto low-complexity classes, which help reduce the computational complexity associated with dense matrices. In this work, two algorithms for matrix multiplication are presented and compared for their potential advantages and disadvantages against the circulant decomposition method[**2**].

- Mean Approximated Matrix Multiplication
- Low-rank approximated matrix multiplication
- Matrix multiplication using Circulant decomposition

2.1 Mean Approximation method:

In matrix multiplication, each element in the resulting matrix arises from the dot product of corresponding columns and rows of the input matrices. For instance, in the matrix multiplication $\mathbf{M} = \mathbf{AB}$, the ij^{th} element of the resulting matrix $\mathbf{M}$ is computed as the dot product of the i^{th} column of matrix $\mathbf{A}$ (denoted as a_i) and the j^{th} row of matrix $\mathbf{B}$ (denoted as b^j).

$$m_{ij} = a_i \cdot b^j \tag{2.1}$$
$$\approx \mathbf{E}[a_i \cdot b^j] \tag{2.2}$$
$$\approx n * \mathbf{E}[a_i] * \mathbf{E}[b^j] \tag{2.3}$$

The mean approximation method approximates the result m_{ij} with the mean value of the dot product $a_i \cdot b^j$. It relies on the assumption of uncorrelated entries rows of matrix $\mathbf{A}$ and the columns of matrix $\mathbf{B}$. However, it's important to note that while this assumption always holds for random matrices, it is not generally applicable.

Algorithm Performance and Considerations: The effectiveness of this algorithm is contingent upon the fact that the rows in $\mathbf{A}$ and the columns in $\mathbf{B}$ are not correlated. Deviations from this assumption may lead to inaccuracies in the computed results. The plots presented in Figure 2.4 illustrate the algorithm's performance on randomly generated matrices and provide insights into its limitations, mainly when applied to network matrices from the matrix market and Non-Randomly Generated matrices(images).

2.2 Low-Rank Approximation

The low-rank approximation technique is applied to matrix multiplication $C = AB$. The method involves expressing the product as a sum of outer products of selected columns from matrix A and corresponding rows from matrix B[**12**]

$$C \approx \sum_{i=0}^{k} a^i \cdot b_j{}^T \tag{2.4}$$

Selecting c columns(rows) for low-rank approximation is as follows. The matrix's k^{th} column(row) is selected using rejection sampling with probability p_k. The complete algorithm is as follows.

- **Initialization:**
  - Define c as the number of samples.
  - p_k as the probability mass function for each $k \in \{1, 2, \ldots, n\}$.
  - Define $A^{(k)}$ and $B_{(k)}$ for each k.
  - Initialize M as the zero matrix for storing the approximation of AB
  - Initialize $t = 0$ for counting the accepted samples.
- **Random Sampling:**
  - Sample an index k uniformly from $\{1, 2, \ldots, n\}$
  - Generate a uniform random number U from $[0, 1]$.
  - accept k if max $p_k U < p_k$.
  - If k is accepted, increment t by 1.
- **Sum Rank-One Matrices:**
  - For each accepted k, update the approximation M:

$$M \leftarrow M + \frac{A^{(k)} B_{(k)}}{c p_k}. \tag{2.5}$$

  - Continue sampling until $t = c$.
- **Error Evaluation:**
  - Compute the relative error in the Frobenius norm:

$$E = \frac{\|M - AB\|_F}{\|AB\|_F} \tag{2.6}$$

2.2.1 Experiments:

Matrices are generated using the singular value decomposition $U\Sigma V^T$. The singular values given by Σ_{kk} are fixed as follows

- **Class-1:** $e^{\frac{-(k-1)}{10}}$
- **Class-2:** $\frac{n-k+1}{n}$
- **Class-3:** $\frac{\log n - k + 1}{\log n}$

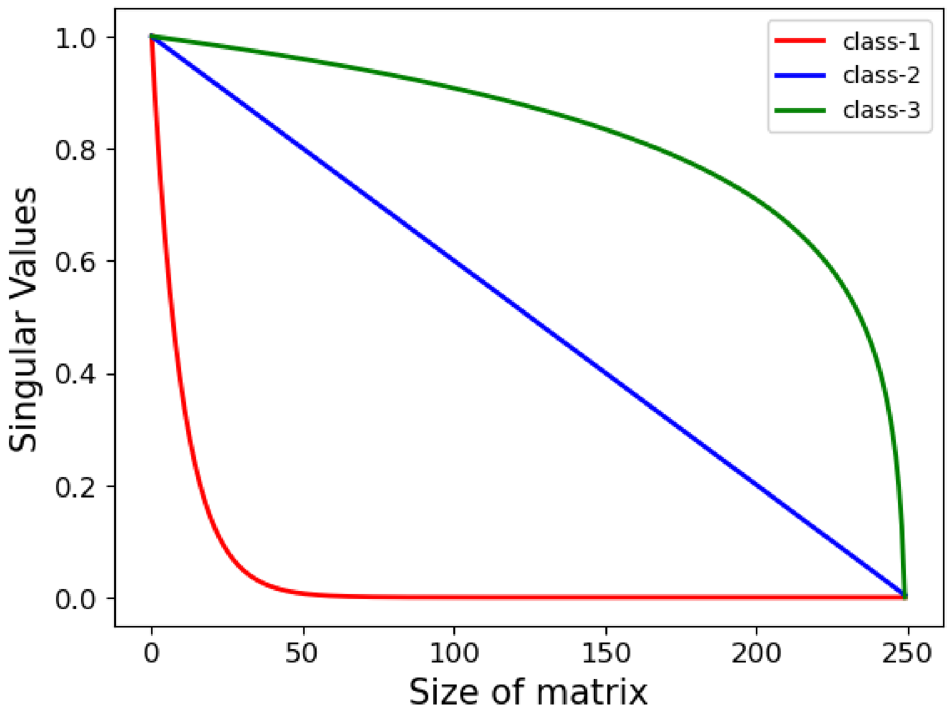


Figure 2.1: Singular Values of the three classes

Figure 2.1 shows the variations of the singular values for size 250. From the figure, it is clear that the matrices generated with eigenvalues belonging to class-1 are low rank compared to others. Using these singular values, two types of trial matrices are generated. Two types of matrices are generated with the three classes of singular values, and the relative error in the multiplication is examined, varying with size for the same number of samples ($c = 0.2 * n$). The relative errors in the multiplication for two types of matrices generated from 3 classes of singular values are shown in the figure 2.2

Building Trial Matrices:

- Type-1:
  - *Initialize:* $M_1 \leftarrow rand[n, n]$ and $M_2 \leftarrow rand[n, n]$
  - *Generate singular vectors:* $Q_1R_1 \leftarrow M_1$ and $Q_2R_2 \leftarrow M_2$; QR factorization of the two matrices.
  - *trial Matrices:* $A \leftarrow Q_1 \Sigma Q_2^T$; generate test matrix for given singular value distribution.
- Type-2:
  - *Initialize:* $M_1 \leftarrow rand[n, n]$ and $M_2 \leftarrow rand[n, n]$ and $M_3 \leftarrow rand[n, n]$
  - *Generate singular vectors:* $Q_1R_1 \leftarrow M_1$ and $Q_2R_2 \leftarrow M_2$ and $Q_3R_3 \leftarrow M_3$;

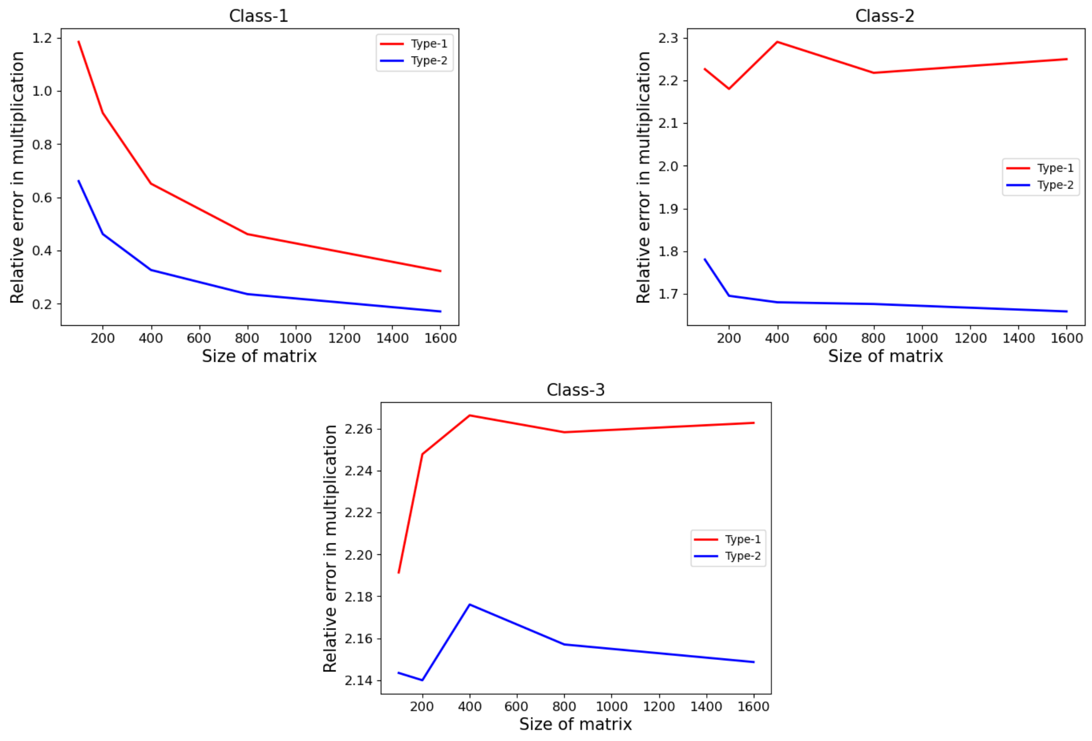


Figure 2.2: Figure shows the variation of relative error with size for each class of singular values and two types of matrices

QR factorization of the three matrices.

– *trial Matrices:* $A \leftarrow Q_1 \Sigma Q_2^T$ and $B \leftarrow Q_2 \Sigma Q_3^T$; generate test matrix for given singular value distribution.

Conclusions:

- Matrices generated from type-2 are effectively low rank, so the relative error in matrix multiplication is less than type-1 for all three classes.
- Class 1 matrices feature an efficient low rank that remains constant regardless of matrix size. This effective low rank is smaller than the number of rows and columns sampled in a large matrix, resulting in reasonable approximations of the product AB as n increases.
- This method is advisable only for low-rank matrices.

2.3 Circulant Decomposition

The circulant decomposition of the matrix explained in Chapter 1 is used for matrix multiplication. The two matrices involved in the multiplication are decomposed into prominent circulant components; these circulant components are involved in the multiplication. From *Remark 1.4*, the indices of circulant components of a matrix in the order of prominence can be found in $\mathcal{O}(n^2 \log n)$. The algorithm 1 explains the complete matrix multiplication using the circulant components after the decomposition of the matrices. The residues are used iteratively to reduce the error in multiplication. cycles_A and cycles_B are lists representing the indices of the circulant components of matrices A and B, respectively, in the order of their dominance. Consider the matrices A and B. These matrices can be written as $A = A_i + \Delta A$ and $B = B_j + \Delta B$, where A_i contains the top k circulant components of A and B_j contains the top k circulant components of B.

$$A_i = \sum_k R_{ai} D_i \quad B_j = \sum_k R_{bj} D_j \tag{2.7}$$

$$\Delta A = A - A_i \quad \text{and} \quad \Delta B = B - B_j \tag{2.8}$$

where

- R_{ai} is the i^{th} circulant component of A
- R_{bj} is the j^{th} circulant component of B

Now, matrix multiplication $M = AB$ can be written as follows.

$$\begin{aligned} M &= A \times B && (2.9)\\ &= (A_i + \Delta A) \times (B_j + \Delta B) && (2.10)\\ &= (A_i B_j) + (A_i \Delta B) + (\Delta A B_j) + (\Delta A \Delta B) && (2.11) \end{aligned}$$

This multiplication is approximated with the first three terms by removing the residues, i.e., M_{app} can be written as

$$M_{app} = (A_i B_j) + (A_i \Delta B) + (\Delta A B_j) \tag{2.12}$$

In practice, the first two terms can be evaluated together as $A_i B$. The relative error in the multiplication is given as $\frac{\|M - M_{app}\|}{\|M\|}$. This relative error in the multiplication can be reduced by iteratively using ΔA and ΔB again as A and B respectively, ΔA and ΔB contain the remaining circulant components other than that of circulant components considered in A_i and B_j.

2.3.1 Arithmetic complexity:

Algorithm 1 Matrix Multiplication with Iterative Refinement

```
0: procedure MATRIX MULTIPLICATION(A, B, cycles_A, cycles_B, iterations)
0:     $M \leftarrow A \times B$
0:     for $it \leftarrow 1$ to iterations do
0:         selected_cycles_A ← cycles_A[1 : num]
0:         selected_cycles_B ← cycles_B[1 : num]
0:         $Ai \leftarrow \sum_{i=0}^{\text{num}-1} R_{ai} \cdot D_{ai}$
0:         $Bj \leftarrow \sum_{i=0}^{\text{num}-1} R_{bi} \cdot D_{bi}$
0:         $\Delta A \leftarrow A - Ai$
0:         $\Delta B \leftarrow B - Bj$
0:         $M \leftarrow \left(\sum_{i=0}^{\text{num}-1} R_{ai} \cdot D_{ai}\right) \times \left(\sum_{i=0}^{\text{num}-1} R_{bi} \cdot D_{bi}\right) + \left(\sum_{i=0}^{\text{num}-1} R_{ai} \cdot D_{ai}\right) \times \Delta B + \Delta A \times \left(\sum_{i=0}^{\text{num}-1} R_{bi} \cdot D_{bi}\right)$
0:         cycles_A ← cycles_A[num + 1 :]
0:         cycles_B ← cycles_B[num + 1 :]
0:         $A \leftarrow \Delta A$
0:         $B \leftarrow \Delta B$
0:     end for
0:     err ← $\frac{\|AB-M\|}{\|AB\|}$
0: end procedure=0
```

Each iteration in the above multiplication algorithm can be decomposed as follows

Indices: It is a list of indices of circulant components of matrices A and B in the order of their energies/magnitudes. Similarity transformation of each matrix using FFT in $n^2 \log n + n^2$ operations, and energy of all n circulant components requires n^2 operations. Hence arithmetic complexity to find $Indices_A$ and $Indices_B$ is $\mathcal{O}(n^2 \log n)$.

Finding top k circulant components: After getting the indices of top k circulant components, each circulant component of a matrix can be found in $3n^2 \log n + n \log n + n$ arithmetic operations or less i.e. $\mathcal{O}(n^2 \log n)$.

Calculating A_i and B_j: We have $A_i = \sum_k R_{ai} D_i$ and $B_j = \sum_k R_{bj} D_j$. These two can be done in $2k(n + n^2)$ operations or in $\mathcal{O}(kn^2)$ complexity.

Calculating ΔA and ΔB: This can be done in $\mathcal{O}(n^2)$ arithmetic operations.

Multiplication: Total multiplication contains the following terms.

$$T_1 = \left(\sum_k R_{ai} D_{ai}\right) \times \left(\sum_k R_{bi} D_{bi}\right) = \left(\sum_k R_{ai} D_{ai}\right) \times B_j \quad = \left(\sum_k R_{ai} D_{ai} \times B_j\right) \tag{2.13}$$

- Each term $D_{ai} \times B_j$ can be done in $\mathcal{O}(n)$

- Each term $R_{ai}D_{ai} \times B_j$ can be done in $\mathcal{O}(n^2 \log n)$. By considering k terms the total arithmetic complexity for this term will be $\mathcal{O}(kn^2 \log n)$

Similarly, the three terms in the multiplication equation 2.11 can be done in $3k(n^2 \log n + n)$ or less. One iteration of the total algorithm can be done in $(5n + 3n^2 + 7n^2 \log n)k + (2n^2 \log n)$operations. By selecting $\log n$ circulant components in each iteration the arithmetic complexity scales to $(5n \log n + 5n^2 \log n + 7n^2 \log n^2)$. Hence, the theoretical arithmetic complexity can be said to be $\mathcal{O}(n^2 \log n^2)$ which is $\widetilde{\mathcal{O}}(n^2)$. This theoretical arithmetic complexity can be used to check the cross-over with size. From the figure 2.3, the crossover size for one iteration is around 700.

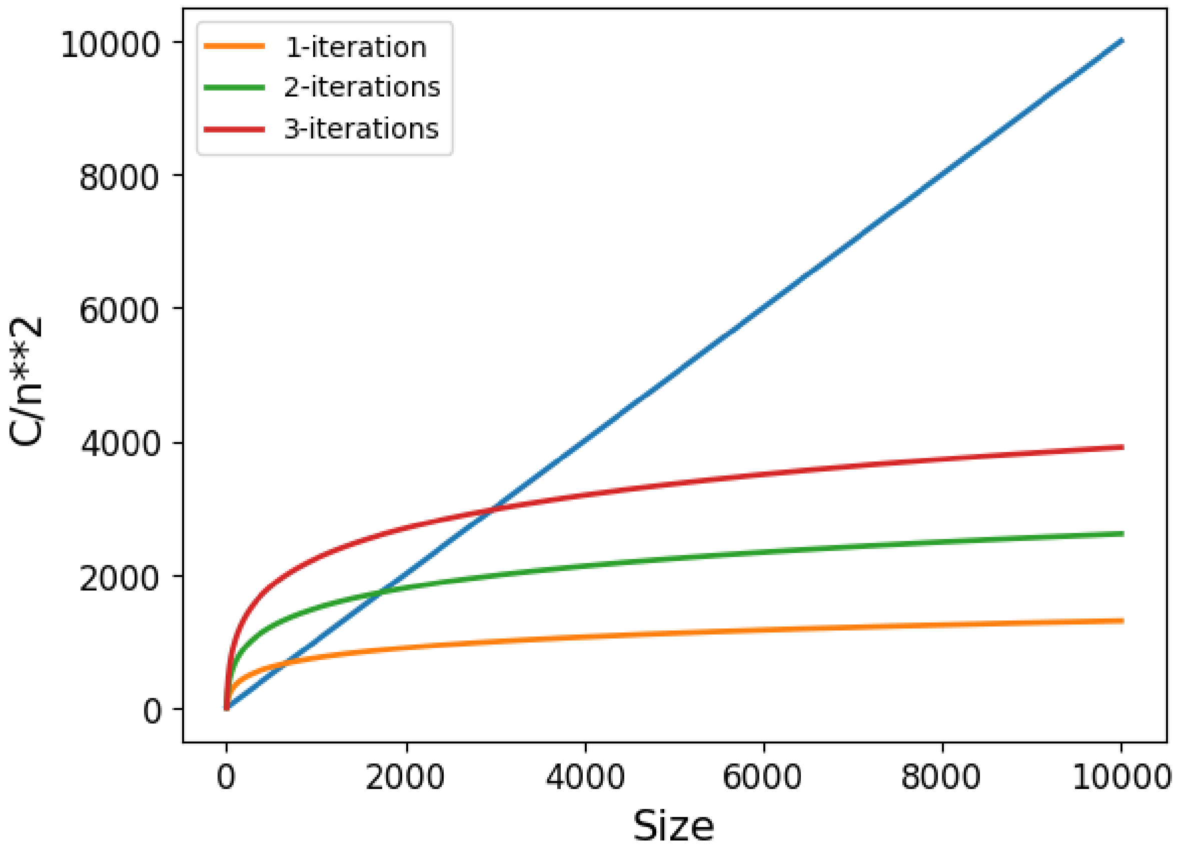


Figure 2.3: Theoretical crossover for the arithmetic complexity is compared against the typical arithmetic complexity n^3 (For easy understanding, values are scaled with n^2)

2.4 Experiments and Analysis

In this section, we investigate the performance of matrix multiplication across various sizes, ranging from 100 to 300, with a step size of 25. The evaluation compares two methods: Mean approximation and circulant decomposition.

2.4.1 Experimental Setup:

Experiments are conducted using numpy and scipy libraries of python. Figures 2.4 compare Mean approximation and circulant decomposition methods. For the circulant decomposition method, circulant components having a normalized weight of more than 0.5 are considered for multiplication, and the algorithm is iterated once.

2.4.2 Analysis:

- Mean approximation works better on matrices where the entries are independent
- Low-rank approximation method works best when the matrices considered for multiplication are effectively low rank.
- Incorporating ΔA and ΔB minimizes the relative error in multiplication in the circulant decomposition approach. Figure 2.4 shows that the relative error in matrix multiplication approaches a precision of 10^{-2} with increasing size.
- Circulant Decomposition works better in cases where any random and periodic matrices are involved.

2.5 Robustness of the circulant decomposition algorithm

From the conclusions drawn from the figure 2.4, it is clear that the errors can be reduced more by incorporating the terms ΔA and ΔB in the multiplication iteratively. Algorithm 1, is employed on the matrices explained in Appendix 2 for $\log n$ iterations; the results in the multiplication errors varying with size are shown in the plot 7.6. From the plots 2.4 and 7.6, it is evident that the relative error in multiplication reduces as the size increases for the circulant decomposition method.

2.6 Conclusions

- **Error in Multiplication:**

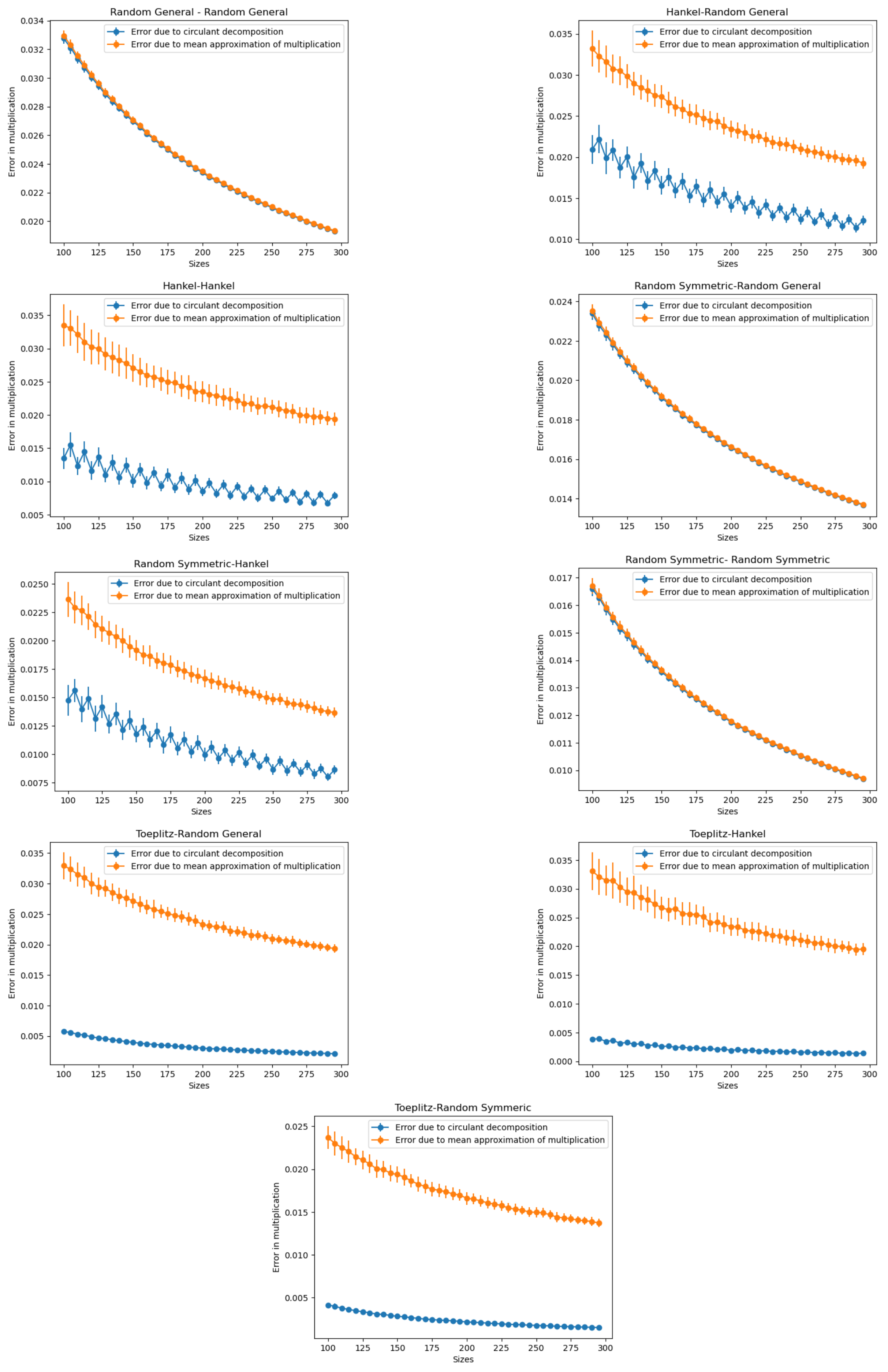


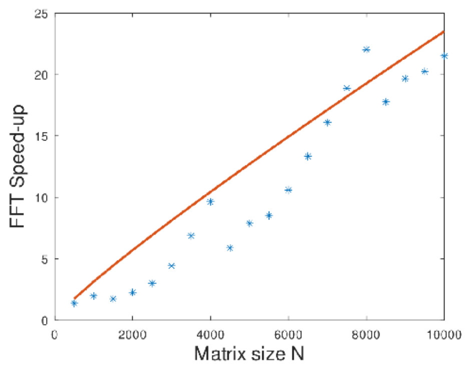
Figure 2.4: Mean and standard deviation of errors in multiplication for both Mean approximation method and circulant decomposition method

Matrix	Iterations-Taken	Cross-over size n
Toeplitz-Toeplitz	1(0.22%)	700
Random Symmetric-Toeplitz	1(0.35%)	700
Toeplitz-Hankel	1 (0.42%)	700
Random General-Toeplitz	1(0.51%)	700
Random Symmetric-Random Symmetric	1(1.01%)	700
Images	2(0.64%)	1,400
Random Symmetric-Hankel	6(1.04%)	4,200
Hankel-Hankel	8(1.02%)	5,600
Random General-Random Symmetric	10(1.03%)	7,000
Random General-Hankel	11(1.04%)	7,700
DFT	16(1.01%)	11,200
Random General-Random General*	17(1.04%)	11,900
Dense Matrix	17(1.05%)	11,900
Bus matrices**	34(0.72%)	23,800

Table 2.1: Table showing the feasible dimension of the matrices considered in the multiplication to get RMSE $\approx$ 1%. $*$Only these random matrices have a non-zero mean. $**$ Bus matrices considered are not dense matrices that were nevertheless used for testing the algorithm's limitations. Cross-over size n for matrices calculated are approximate, it differs from the given values as the arithmetic complexity is logarithmic

  - From the figure 7.6 in appendix 3, it is clear that the relative error in the multiplication decreases as the size increases.
  - In the experiments done, it is clear that for the same number of iterations, the relative error in multiplication is less for special structured matrices(e.g., Toeplitz).
  - The Same algorithm can be diversified and used for any matrix, but different matrices take a different number of iterations for a given relative error, i.e., the smallest matrices required for a gainful application of this algorithm vary from 700-20,000 depending on the matrix type.
  - From the figure 1.2, it is clear that the matrices generated from the mean zero distribution do not have dominating circulant components; we may require many more iterations to converge to the required tolerance in relative error for the case of multiplying two mean-zero random matrices. Also, because the product of such matrices converges towards the zero matrix, the relative error may not be the most appropriate metric to estimate the performance of this algorithm in this case.

- **Arithmetic Complexity:**



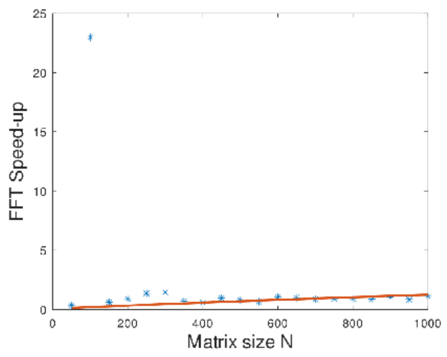


Figure 2.5: The figures (a) and (b) show the hardware dependencies and the effect of compiler optimizations (such as auto-vectorization) on the FFT gains versus the exact multiplication of real matrices of different sizes in terms of the wall-times for execution in a typical workstation i.e., ratio of wall-times for evaluating WA naively and FFT(A), where A is a matrix. Note that this reduces to the ratio of the arithmetic complexity of the two evaluations, i.e., N/(4log(n)), only in a naive serial execution of single instructions. The linear fit lines represent a $c = 20$ for smaller values of N shown in (a) and $c = 8$ for larger N shown in (b) in the gain ratios $\frac{N}{c*4\log n}$. One would like to have $c = 1$ using better optimization of the executions

- The figure 2.3 shows the theoretical cross-over of size for better arithmetic complexity (neglecting the front constant).
- Table 2.1 shows the number of iterations taken by different combinations of the matrices to get the RMS error $< 1\%$ in multiplication. By referring to the figure 2.3, the feasible size of the matrices for this algorithm is calculated and is shown in the table.
- Taking figure 2.3 as reference the crossover size of different combinations of matrices for RMSE error $< 1\%$.
- Our findings in Figure 2.5 suggest that further optimization of the Fast Fourier Transform (FFT) algorithm specific to the available hardware could significantly improve the performance of WA product calculations. Additionally, exploring alternative optimization strategies tailored to evaluate $WAW^\dagger$ within the proposed circulant multiplication algorithm might be beneficial. Implementing such hardware-aware optimizations for parallel computations is crucial to achieving the crossover sizes depicted in Figure 2.3 on a particular computing platform.

• Even though we have restricted our discussion to the multiplication of square matrices, this approach can indeed be used in efficient matrix-vector multiplications provided the number of vectors multiplying a large matrix is also sufficiently large, i.e., $\mathcal{O}(\log n)$. It can also be trivially extended to the multiplication of large rectangular matrices using the circulant decomposition of the largest square blocks of such matrices, and it is attractive when at least one of the dimensions of the product matrix is sufficiently large. This dimension required for the matrix in a gainful application of the algorithm can be inferred from the figure 2.3.

Chapter 3

Toeplitz Linear Solver

3.1 Introduction

Toeplitz linear systems arise in various fields, including the solution of linear ordinary differential equations, delay differential equations, time series analysis, and orthogonal polynomials. For large matrices, standard methods like Gaussian elimination become computationally expensive, scaling as $O(n^3)$. To address this, both direct and iterative methods are used to solve Toeplitz linear systems Tx=bTx=b. Direct solvers are deemed efficient if they operate in $\mathcal{O}(n^2)$ time; examples include Schur-type methods [**13**], Levinson-type methods [**14**], among others [**15**]. Notably, a significant category of direct solvers is based on the displacement equation, employing Gaussian elimination techniques, such as the Heinig method [**16**] and the GKO method [**17**]. However, many of these fast and superfast methods tend to be numerically unstable [**13**, **14**]. The unique structure of Toeplitz matrices offers advantages for various algorithms, with the most commonly used methods for solving Toeplitz linear systems being highlighted here.

3.1.1 Trench Algorithm:

The inversion algorithm proposed by W.F.Trench is applicable only for *Strongly nonsingular Matrix.* An arbitrary matrix is said to be strongly nonsingular when, in addition to being nonsingular itself, all its principal submatrices are nonsingular as well[**18**]. Equivalently, all its principal minors are nonzero. Let us consider a sample Toeplitz matrix of size $n+1 \times n+1$ as T_{n+1}, and denote the inverse of this matrix as B_{n+1} i.e., $B_{n+1} = T_{n+1}^{-1}$.

The detailed B_{n+1} structure is explained in [18]. Using this Trench algorithm, the inversion of the Toeplitz matrix can be found in time complexity of $\mathcal{O}(n^2)$.[18]
But by using circulant approximation of the Toeplitz matrix the inverse of the Toeplitz matrix can be done in $\mathcal{O}(n \log n)$

3.1.2 Using Circulant Decomposition:

Theorem 3.1: Let $T = (a_{p-q})_{p,q=1}{}^{n}$ be a Toeplitz matrix. If each of the systems of equations $Tx = f$, $Ty = e_1$ is solvable, $x = (x_1, x_2, \dots, x_n)^T$, $y = (y_1, y_2, \dots, y_n)^T$, then[20]

- T is invertible
- $T^{-1} = T_1U_1 + T_2U_2$, where

$$T1 = \begin{bmatrix} y_1 & y_n & \dots & y_2 \\ y_2 & y_1 & \ddots & \\ \vdots & \ddots & \ddots & y_n \\ y_n & \dots & y_2 & y_1 \end{bmatrix} U_1 = \begin{bmatrix} 1 & -x_n & \dots & -x_2 \\ & 1 & \ddots & \vdots \\ & & \ddots & -x_n \\ & & & 1 \end{bmatrix} \tag{3.1}$$

$$T_2 = \begin{bmatrix} x_1 & x_n & \dots & x_2 \\ x_2 & x_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & x_n \\ x_n & \dots & x_2 & x_1 \end{bmatrix} U_2 = \begin{bmatrix} 0 & y_n & \dots y_2 & \\ & 0 \ddots & \vdots & \\ & & \ddots & y_n \\ & & & 0 \end{bmatrix} \tag{3.2}$$

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{and} f = \begin{bmatrix} 0 \\ a_{n-1} - a_{-1} \\ \vdots a_2 - a_{-n+2} \\ a_1 - a_{-n+1} \end{bmatrix} \tag{3.3}$$

From the figure 1.1, we can see that in the case of the Toeplitz matrix, the first circulant component plays a very prominent role, i.e., the Toeplitz matrix can be approximated very easily with the first circulant component. This circulant component can be used to find the approximated inverse of the Toeplitz matrix,
This method has three stages in finding the solution of the Toeplitz linear solver.

- Find the first circulant component of the Toeplitz matrix (Can be done in $\mathcal{O}(n)$.
- Find the inverse of the circulant matrix by using the theorem 3.2 [3]($\mathcal{O}(n \log n)$.
- Multiply the inverted circulant matrix(which is again a circulant matrix) with the vector ($\mathcal{O}(n \log n)$.

Remark 3.2: In theorem 3.1, let the Toeplitz matrix $T = (a_{p-q})_{p,q=1}^{n}$ be a circulant Toeplitz matrix. That is to say, the elements of the matrix $T = (a_{p-q})_{p,q=1}^{n}$ satisfy $a_i = a_{i-n}$ for all $i = 1, \dots, n-1$. It is easy to see that $f = 0$. Thus, $x = T^{-1} f = 0$[**19**]. From theorem 3.1, we get

$$T^{-1} = \begin{bmatrix} y_1 & y_n & \dots & y_2 \\ y_2 & y_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & y_n \\ y_n & \dots & y_2 & y_1 \end{bmatrix} \tag{3.4}$$

To find y in remark 3.2 we need to solve the circulant system of equation $Cy = e_1$

Solving circulant linear system $Cy = e_1$: We have a Circulant linear system

$$\mathbf{C}\vec{y} = \vec{e1} \tag{3.5}$$

we need to covert this to

$$\widehat{\mathbf{C}}\widehat{\vec{y}} = \widehat{\vec{e1}} \tag{3.6}$$

From 3.5 we can write

$$\mathbf{F}\mathbf{C}\mathbf{F}^{-1}\mathbf{F}\vec{y} = \mathbf{F}\vec{e1} \tag{3.7}$$

where **F** is the DFT matrix of size n

$$\mathbf{F}_n = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & (w)^1 & (w)^2 & \dots & (w)^{(n-1)} \\ 1 & (w^2)^1 & (w^2)^2 & \dots & (w^2)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (w^{n-1})^1 & (w^{n-1})^2 & \dots & (w^{n-1})^{n-1} \end{bmatrix}$$

with $w = e^{\frac{\pi i}{n}}$ and $\mathbf{F}\mathbf{F}^{-1} = \mathbf{F}^{-1}\mathbf{F} = \mathbf{I}$

By comparing 3.6 and 3.7 we have

$$\widehat{\mathbf{C}} = \mathbf{F}\mathbf{C}\mathbf{F}^{-1} \tag{3.8}$$

$$\widehat{\vec{y}} = \mathbf{F}\vec{y} \tag{3.9}$$

and

$$\widehat{\vec{e1}} = \mathbf{F}\vec{e1} \tag{3.10}$$

In this 3.8 refers to Cauchy-like matrix of the Circulant matrix. 3.9 refers to the DFT of $\vec{y}$ and 3.10 results in $\vec{1}$. Any Circulant matrix $\mathbf{C}$ of the form

$$\mathbf{C} = \begin{bmatrix} a_0 & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_1 & a_0 & a_{n-1} & \dots & a_2 \\ a_2 & a_1 & a_0 & \dots & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_3 & \dots & a_0 \end{bmatrix}$$

can be written in the form

$$\mathbf{C} = \mathbf{F}^{-1} diag(\mathbf{F}\vec{a}_n)\mathbf{F} \tag{3.11}$$

Here, $\vec{a}_n = [a_0, a_1, \dots, a_{n-1}]$ is the vector representation of $\mathbf{C}$ Substituting eq 3.11 in eq 3.8 gives $\widehat{C} = diag(\mathbf{F}\vec{a}_n)$ Which results that the Cauchy-like matrix of a Circulant matrix is a diagonal matrix

Result 1: The Cauchy-like matrix of a Circulant matrix is a Diagonal matrix with elements Discrete Fourier transform of vector representation of Circulant matrix.

Solving Cauchy-like linear System

In order to solving eq 3.6 where $\mathbf{C}$ is a diagonal matrix with elements of vector $\mathbf{F}\vec{a}_n$ and from eq 3.10 resulting in unit vector. Hence, the solution of eq 3.6 results in a vector with elements which are inverse of elements of $\mathbf{F}\vec{a}_n$.

Result 2: The solution of the Cauchy-like system is the vector of elements inverse of elements of $\mathbf{F}\vec{a}_n$. From equation 3.9 $\vec{y} = \mathbf{F}^{-1}\widehat{\vec{y}}$ which is the first column of the inverse of Circulant matrix.

$$\mathbf{C}^{-1} = \begin{bmatrix} y_0 & y_{n-1} & y_{n-2} & \cdots & y_1 \\ y_1 & y_0 & y_{n-1} & \cdots & y_2 \\ y_2 & y_1 & y_0 & \cdots & y_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{n-1} & y_{n-2} & y_3 & \cdots & y_0 \end{bmatrix}$$

$\mathbf{C}^{-1}$ is the inverse of the first circulant component of the Toeplitz matrix $\mathbf{T}$ The time complexity of this algorithm is $O(nlog(n))$, where n is the size of the matrix.
However, this method is inefficient as the inverted circulant matrix cannot approximate the inverted Toeplitz matrix well. So, we cannot use this method to solve the Toeplitz linear solves. The errors in this method are unreliable, and the algorithm's time complexity is very low compared to the other methods. Many fast and super-fast algorithms are available solve the Toeplitz linear solvers. One of the approximated algorithms is the Sequential Semiseperable method

3.2 Sequential Semiseparable method:

Standard methods for solving systems of linear equations become computationally expensive for large Toeplitz matrices. The SSS method offers a faster alternative by exploiting the specific structure of these matrices.[**20**]
The SSS method leverages a two-step approach:

- **Transformation and Approximation:**
  - Displacement equation methods transform the Toeplitz matrix into a Cauchy-like matrix using techniques like Fast Fourier Transforms (FFTs).
  - This Cauchy-like matrix has a special property: its off-diagonal blocks have a low numerical rank.
  - Taking advantage of this low-rank property, the method approximates the Cauchy-like matrix with a more compact representation called a sequentially semi-separable (SSS) matrix.
- **Efficient Solution:**
  - The SSS representation allows solving the resulting Cauchy-like system effi-

ciently in linear time and with linear storage requirements. This is due to the precomputed compressions of the off-diagonal blocks (independent of the specific Toeplitz matrix entries) and the efficient structure of SSS matrices.

- **Benefits:**
  - The SSS method offers significant computational advantages compared to standard methods for solving large Toeplitz systems.
  - The precomputation step allows for efficient reuse when solving multiple Toeplitz systems with similar structures.
  - The use of FFTs and the low-rank property of the off-diagonal blocks contribute to the overall efficiency of the SSS method.
- **Stages of the Algorithm:**
  - *Precomputation:* Compress the off-diagonal blocks of the Cauchy-like matrices (independent of the Toeplitz matrix itself).
  - *Transformation:* Convert the Toeplitz matrix into a Cauchy-like matrix using FFTs (complexity: $\mathcal{O}(n \log n)$).
  - *SSS Construction and Solution:*
    - Construct a compact SSS representation from the precomputed compressions (complexity: $\mathcal{O}(p^2 n)$ where p is the SSS matrix rank).
    - Solve the resulting Cauchy-like system using the SSS representation (complexity: $\mathcal{O}(p^2 n)$).
  - *Solution Recovery:* Recover the solution of the original Toeplitz system (complexity: $\mathcal{O}(n \log n)$).

The SSS method offers a powerful tool for efficiently solving large Toeplitz systems arising in various scientific and engineering applications[**20**]

3.3 Analysis and Comparision:

3.3.1 Experiment:

A Toeplitz matrix is randomly generated from size 100 to 500 at step size 50. The Toeplitz matrix is approximated with the first circulant component, and an inverse is found for this approximated circulant component. The errors in the approximation of this inverse are plotted against the size and shown in figure 3.1. Similarly, for every matrix, a linear system is randomly generated, and the error in the solution using circulant decomposition is found relative to the exact solution, and the error is shown in figure 3.1

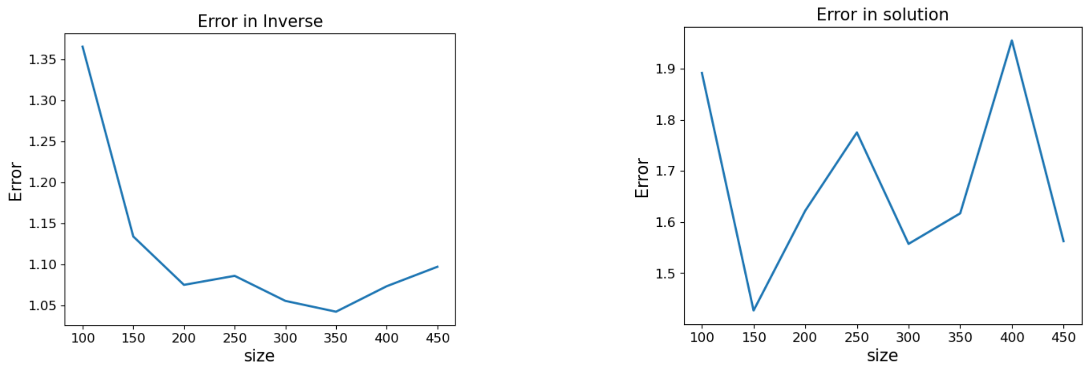


Figure 3.1: Left image shows the error in the inverse of an approximated matrix concerning the inverse of the original matrix. The right image shows the error in the solution of the linear system found using the inverse of circulant decomposition.

3.3.2 Conclusion:

- W.F.Trench algorithm can be used to find the exact inverse of a Toeplitz matrix to find the linear system's solution in $\mathcal{O}(n^2)$.
- Using the circulant decomposition method, the approximated inverse of a Toeplitz matrix can be found in $\mathcal{O}(n \log n)$, but the error due to approximation is not reliable, so this method cannot be used to find the solution of Toeplitz linear solvers.
- Many super-fast algorithms like sequential semi-separable methods are available to find the solution of Toeplitz linear solvers in $\mathcal{O}(n \log n)$.

Chapter 4

Conclusions

This work first applies the circulant decomposition method explained in [**2**]. This circulant decomposition can approximate a matrix as a sum of circulant matrices with fixed periodic relaxations, reducing the arithmetic complexity of various matrix operations. However, the error in any application depends on the specific application and the type of matrix used.

First, matrix multiplication is studied by approximating various matrices using the circulant decomposition followed by an iterative multiplication of these circulant matrices (Chapter 2). The method is then compared against two potential competitors: low-rank approximation and mean approximation. Later, the circulant decomposition method is used to find the solution for Toeplitz linear solvers and two algorithms better than circulant decomposition are also studied (Chapter 3).

These approximations and applications are studied for different types of matrices, including some randomly generated from distributions (mean zero and mean non-zero) (Appendix 2), some generated from experiments (synthetic DFT and dense DFT matrices), and some derived from practical applications (images and bus networks). The following conclusions are drawn from this work.

- **Circulant Decomposition:**
  - Approximating a matrix using the circulant decomposition method with a particular number of circulant components is attractive depending on the matrix type, i.e., the degree of dominance of a few circulant components representing the periodic nature of its entries. The weight distribution of the circulant components of matrices depends on the elements of the matrix(distribution

of the elements and/or periodicity of the elements). For a particular matrix, as the number of circulant components considered for the approximation increases, the error in the approximation decreases but at the cost of additional computing.

- **Multiplication**
  - The mean approximation and the low-rank approximation of matrix multiplication are applicable only in restricted cases, whereas the circulant decomposition is generally applicable to any two sufficiently large matrices.
  - One iteration of the algorithm 1 (matrix multiplication using circulant decomposition method) can be done in $\mathcal{O}(n^2 \log n^2)$, with some error in the multiplication. Still, this error can be reduced by recursively applying the algorithm to the residues.
  - For all matrix classes and their products studied (except the case of two mean-zero random matrices), the relative error in the multiplication reduces with the size of the matrices, for a given number of iterations.
- **Toeplitz Linear Solvers:**
  - Trench algorithm gives exact inverse in $\mathcal{O}(n^2)$.
  - Sequential Semi-seperable methods gives approximated solvers iteratively in $\mathcal{O}(n \log n)$, but its precision has to be studied further.
  - Circulant decomposition method is not suited for solving Toeplitz linear systems in generality, as the error in the approximation of the matrices can be unsatisfactory in many cases.

Chapter 5

Appendix-1

Discrete Fourier Transform:

An N-point Discrete Fourier Transform (DFT) is defined by the multiplication $X = Wx$, where x is the input vector, W is the $N \times N$ DFT matrix, and X is the DFT of x. The transformation matrix W can be described as $W = \left(\frac{w^{jk}}{\sqrt{N}}\right)_{j,k=0,1,\ldots,N-1}$, or more explicitly:

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & w^3 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & w^6 & \cdots & w^{2(N-1)} \\ 1 & w^3 & w^6 & w^9 & \cdots & w^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & w^{3(N-1)} & \cdots & w^{(N-1)(N-1)} \end{bmatrix}$$

where $w = e^{-2\pi i/N}$ is a primitive N-th root of unity, with i being the imaginary unit ($i^2 = -1$). Large exponents of w can be simplified using the identity $w^x = w^{x \bmod N}$. This matrix is a Vandermonde matrix for the roots of unity, adjusted by the normalization factor. The normalization factor $\frac{1}{\sqrt{N}}$ and the exponent sign in w are conventions that may vary in different contexts [**21**].

Transformation:

Direct Transformation:

Given a vector x of dimension n, where

$$x = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_{N-1} \end{bmatrix}^T,$$

the DFT of x, denoted as X, is computed as $X = Wx$. Here,

$$X = \begin{bmatrix} X_0 & X_1 & X_2 & \cdots & X_{N-1} \end{bmatrix}^T$$

represents the DFT of x. To find X, we multiply the input vector x by the DFT matrix W.

Inverse Transformation: To retrieve the original vector x from X, the inverse DFT is used, given by $x = W^{-1}X$, where W^{-1} is the inverse of the DFT matrix W. The inverse DFT matrix W^{-1} is the conjugate transpose of W, scaled by $\frac{1}{\sqrt{N}}$. Therefore,

$$W^{-1} = \frac{1}{\sqrt{N}} W^*,$$

where W^* denotes the conjugate transpose of W. By multiplying X by W^{-1}, we obtain the original vector x.

Time Complexity: Transformations using the matrix W typically require $\mathcal{O}(n^2)$ operations, where n is the size of the vector. However, by utilizing the Fast Fourier Transform (FFT), this time complexity can be reduced to $\mathcal{O}(n \log n)$.

Fast Fourier Transform (FFT): The Fast Fourier Transform (FFT) is an algorithm designed to compute the Discrete Fourier Transform (DFT) of a sequence, as well as its inverse (IDFT), in a more efficient manner. The FFT achieves this by factorizing the DFT matrix into a product of simpler matrices, significantly reducing the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, where n is the size of the vector. This difference in computational speed is substantial, particularly for large datasets where n can be in the thousands or millions. Additionally, many FFT algorithms are more accurate in the presence of round-off errors compared to directly evaluating the DFT. Various FFT algorithms exist, drawing from diverse mathematical theories, including complex-number arithmetic, group theory, and number theory [**8**]. There are several algorithms to compute the DFT of a vector with lower time complexity:

- The Cooley-Tukey FFT algorithm: A divide-and-conquer approach that recursively breaks down a DFT of any composite size $n = n_1 n_2$ into smaller DFTs of sizes n_1 and n_2, along with $\mathcal{O}(n)$ multiplications by complex roots of unity, traditionally called twiddle factors.
- Prime-Factor Algorithm: Applicable for $n = n_1 n_2$ with coprime n_1 and n_2, this

algorithm is similar to Cooley-Tukey but does not use twiddle factors.

- Rader-Brenner Algorithm: A Cooley-Tukey-like factorization that uses purely imaginary twiddle factors, which reduces the number of multiplications at the expense of increased additions and reduced stability.

These FFT algorithms provide the DFT of a vector of size n with a time complexity of $\mathcal{O}(n \log n)$, which is significantly less than the $\mathcal{O}(n^2)$ complexity of the straightforward DFT computation.

Circulant Matrix: In linear algebra, a circulant matrix is a square matrix in which all rows are composed of the same elements and each row is rotated one element to the right relative to the preceding row. It is a particular kind of Toeplitz Matrix.
Example: A circulant matrix C is a special form of Toeplitz matrix:

$$C = \begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & \ldots c_2 & \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & \cdots & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix} \tag{5.1}$$

In general, note that any circulant matrix C_n of size nxn has a vector representation:

$$\vec{c_n} = \begin{bmatrix} c_0 & c_1 & \ldots & c_{n-1} \end{bmatrix} \tag{5.2}$$

Concept of Diagonalizability: A matrix is said to be diagonalizable if it can be expressed as PDP^{-1}, where D is a diagonal matrix and P is an invertible matrix. In other words, diagonalizability allows us to simplify matrix operations by transforming the matrix into a diagonal form.

Introduction to Discrete Fourier Transform(DFT): The Discrete Fourier Transform(DFT) is a mathematical operation that transforms a sequence of complex numbers into another sequence of complex numbers. A unitary matrix, denoted as F, represents the DFT, which has special properties that make it useful for analyzing periodic signals and cyclic structures.

Theorem: *A circulant matrix C_n is diagonalizable by the DFT matrix. i.e, a circulant* *matrix C_n of size $n \times n$ can be written as: $C_n = (F_n)^{-1} diag(F_n \vec{c_n}) F_n$, $diag(F_n \vec{c})$ is the $n \times n$ diagonal matrix whose diagonal entries are the entries from dft of vector $\vec{c_n}$ where $\vec{c_n}$ is the vector representation of the circulant matrix C_n.*

For this theorem, we need to prove that the columns of the inverse DFT matrix are eigenvectors of any circulant matrix. The corresponding eigenvalues are the DFT values of the vector defining the circulant matrix. We can conclude that a circulant matrix C_n of size $n \times n$ can be written as:

$$C_n = (F_n)^{-1} diag(F_n \vec{c_n}) F_n \tag{5.3}$$

Multiplying a circulant matrix by vector Let $y = DFT(\vec{x}) = F_n \vec{x}$ denote the DFT of a vector $\vec{x}$ and led $\vec{x} = DFT^{-1}(y) = {F_n}^{-1} \vec{y}$ denote the inverse DFT. If C_n is circulant with vector representation $\vec{c_n}$, then multiplying it by a size-n vector $\vec{x}$ can be written as:

$$\begin{aligned}
C_n \vec{x} &= ((F_n)^{-1} diag(F_n \vec{c_n}) F_n) \vec{x} && (5.4)\\
&= (F_n)^{-1} (diag(F_n \vec{c_n}) (F_n \vec{x}) && (5.5)\\
&= DFT^{-1} (diag(DFT(\vec{c_n})) DFT(\vec{x}) && (5.6)\\
&= DFT^{-1} (diag(\vec{v}) \vec{y}) && (5.7)\\
&= DFT^{-1} (\vec{v} \odot \vec{y}) && (5.8)\\
&= DFT^{-1} (\vec{u}) && (5.9)
\end{aligned}$$

Time Complexity:

- $\vec{y} = DFT(\vec{x})$. Can be computed in $\mathcal{O}(n \log n)$.
- $\vec{v} = DFT(\vec{a}_n)$. Can be computed in $\mathcal{O}(n \log n)$.
- Hadamard product $\vec{u} = \vec{v} \cdot \vec{y}$.
- Inverse DFT on $\vec{u}$. Can be computed in $\mathcal{O}(n \log n)$.

Thus, we can compute $C_n \vec{x}$ in $\mathcal{O}(n \log n)$[**22**]. And any matrix multiplication with a circulant matrix $C_n A_n$ where C_n is a circulant matrix and A_n is any random matrix of size nn can be done in $\mathcal{O}(n^2 \log n)$

Chapter 6

Appendix-2

Randomly Generated Matrices(Mean=0 Two categories of $n \times n$ matrices were generated using a uniform distribution in numpy. One was periodic using a seed of $2n$ random entries, and the other was non-periodic matrices:

- Periodic Matrices
  - Toeplitz Matrix
  - Hankel Matrix
- Non-Periodic Matrices
  - General Matrix
  - Symmetric Matrix

Toeplitz Matrix: A Toeplitz matrix, also known as a diagonal-constant matrix, is characterized by having each descending diagonal from left to right being constant.
Example: Any $n \times n$ matrix A of the form

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & \ddots & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & \ddots & \ddots & a_1 & a_0 & a_{-1} \\ a_{(n-1)} & \cdots & \cdots & a_2 & a_1 & a_0 \end{bmatrix}$$

is a Toeplitz matrix. Denoting the element at i, j of A as $A_{i,j}$, we have
$A_{i,j} = A_{i+1,j+1} = a_{i-j}$. **Properties:**

- An $n \times n$ Toeplitz matrix may be defined as a matrix A where $A_{i,j} = c_{i-j}$, for constants $c_{1-n}, \ldots, c_{n-1}$. The set of $n \times n$ Toeplitz matrices forms a subspace of the vector space of $n \times n$ matrices (under matrix addition and scalar multiplication).
- Addition of two Toeplitz matrices can be performed in $\mathcal{O}(n)$ time (by sorting only on the value of each diagonal), while their multiplication can be done in $\mathcal{O}(n^2)$ time.
- Toeplitz matrices are closely connected with Fourier series, as the multiplication operator by a trigonometric polynomial, compressed to a finite-dimensional space, can be represented by such a matrix. Similarly, linear convolution can be represented as multiplication by a Toeplitz matrix.

Hankel Matrix: A Hankel matrix, also known as a catalecticant matrix, is a square matrix in which each ascending skew-diagonal from left to right is constant. A Hankel matrix A is any $n \times n$ matrix of the form

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_1 & a_2 & \cdots & \cdots & \vdots \\ a_2 & \cdots & \cdots & \cdots & a_{2n-4} \\ \vdots & \cdots & \cdots & a_{2n-4} & a_{2n-3} \\ a_{n-1} & \cdots & a_{2n-4} & a_{2n-3} & a_{2n-2} \end{bmatrix}$$

In terms of its components, if the i, j element of A is denoted as A_{ij}, and assuming $i \leq j$, then we have $A_{i,j} = A_{i+k,j-k}$ for all $k = 0, \ldots, j - i$.

Properties:

- Any Hankel matrix is symmetric.
- Let J_n be the $n \times n$ exchange matrix. If H is an $m \times n$ Hankel matrix, then $H = TJ_n$ where T is an $m \times n$ Toeplitz matrix.
  - If T is real symmetric, then $H = TJ_n$ will have the same eigenvalues as T up to sign.
- The Hilbert matrix is an example of a Hankel matrix.

Random Generated Matrix(Mean non-zero): Matrix generated from a Gaussian distribution was also studied for the application.

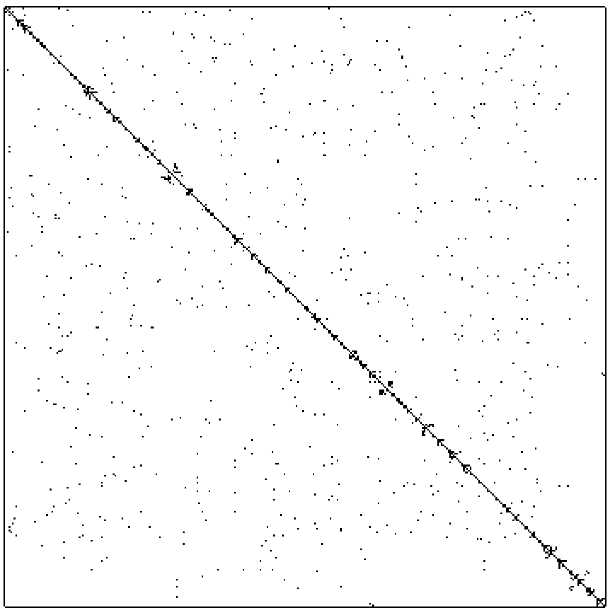


Figure 6.1: 494-Bus

Matrix Market

Introduction The Matrix Market is a comprehensive repository of matrices utilized for testing and benchmarking numerical algorithms across various domains within the computational sciences. Its purpose lies in providing researchers and practitioners with a standardized collection of matrices representing real-world problem instances, facilitating the evaluation and comparison of algorithms' performance and efficiency. Significantly, the Matrix Market houses diverse matrices, including sparse and dense structures, spanning disciplines such as linear algebra, optimization, graph theory, and finite element analysis. The Matrix Market is pivotal in advancing computational research, fostering collaboration, and driving innovation in algorithm development and optimization by offering a centralized platform for accessing and sharing benchmark datasets. We have selected the matrices bus-494 and bus-662 from the application area of power systems and networks.

Bus-494: It is a power system admittance matrix from set PSADMIT from the Harwell-Boeing Collection. The matrix size is 494 × 494, 1080 entries. The structure plot of the matrix is shown in figure 6.1. The matrix is real symmetric indefinite positive definite, with 1666 non-zero elements; 494 elements are on the diagonal, 586 above the diagonal, and 586 below the diagonal.

Bus-662: It is a power system admittance matrix from set PSADMIT and the

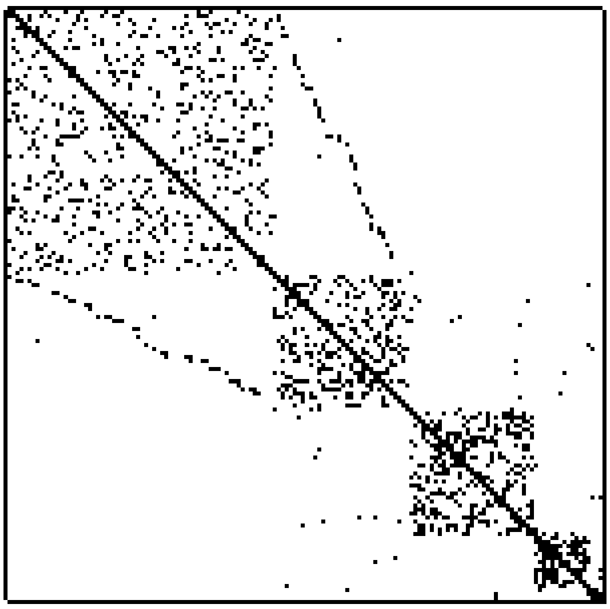


Figure 6.2: 662-Bus

Harwell-Boeing Collection. The matrix size is 662 × 662, 1568 entries. The structure plot of the matrix is shown in figure 6.2. The matrix is a real symmetric indefinite positive definite, and it has a total of 2474 non-zero elements, out of which are 662 on the diagonal, 906 above the diagonal, and 906 below the diagonal.

Images: Two images were selected from the Dog vs. Cat classification dataset available on Kaggle, a well-established resource in computer vision research. This dataset comprises numerous labeled images depicting household cats and dogs. The images chosen for analysis are depicted in Figure 6.5. The cat image has dimensions of (306, 512, 3), while the dog image measures (250, 400, 3). Both images were converted to grayscale and then resized to 2D arrays with dimensions of (250 and 250) to facilitate processing. This resizing ensures uniformity in image dimensions for algorithmic analysis.

DFT Matrices: A special structure of matrices of the form shown in the equation 6.1 is used to check the versatility of the algorithm developed. These matrices are symmetric structures and arise in the case of DFT and are synthetically generated using:

$$H_{ij}^{p} = H_{ji}^{p} = e^{-0.5\times\|i-j\|} \times sin(i+1) \tag{6.1}$$

A DFT matrix provided by the MATRIX lab which is generated from physical

Figure 6.3: Cat

Figure 6.4: Dog

Figure 6.5: Images taken from Dog-Vs-Cat classification dataset from kaggle

experiments was also studied.

Chapter 7

Appendix-3

Matrices discussed in Appendix 2. are generated in sizes ranging from 100 to 500, increasing by 50 each time. These matrices are analyzed to understand the approximation errors, considering the portion of weights used for approximations. Our experiments considered two scenarios: one where matrices are approximated using a $\log n$ number of circulant components and another where the matrices are approximated using only one.

The following figures show the error in multiplication for different categories of matrices for sizes ranging from 100 to 500. The algorithm 1 is executed $\log n$ times taking considering $\log n$ circulant components of each matrix in every iteration. 7.7 and 7.8 shows the error in squaring of DFT matrix and dense matrices. square blocks of size for different ranges are considered, and the errors for the corresponding size are plotted.

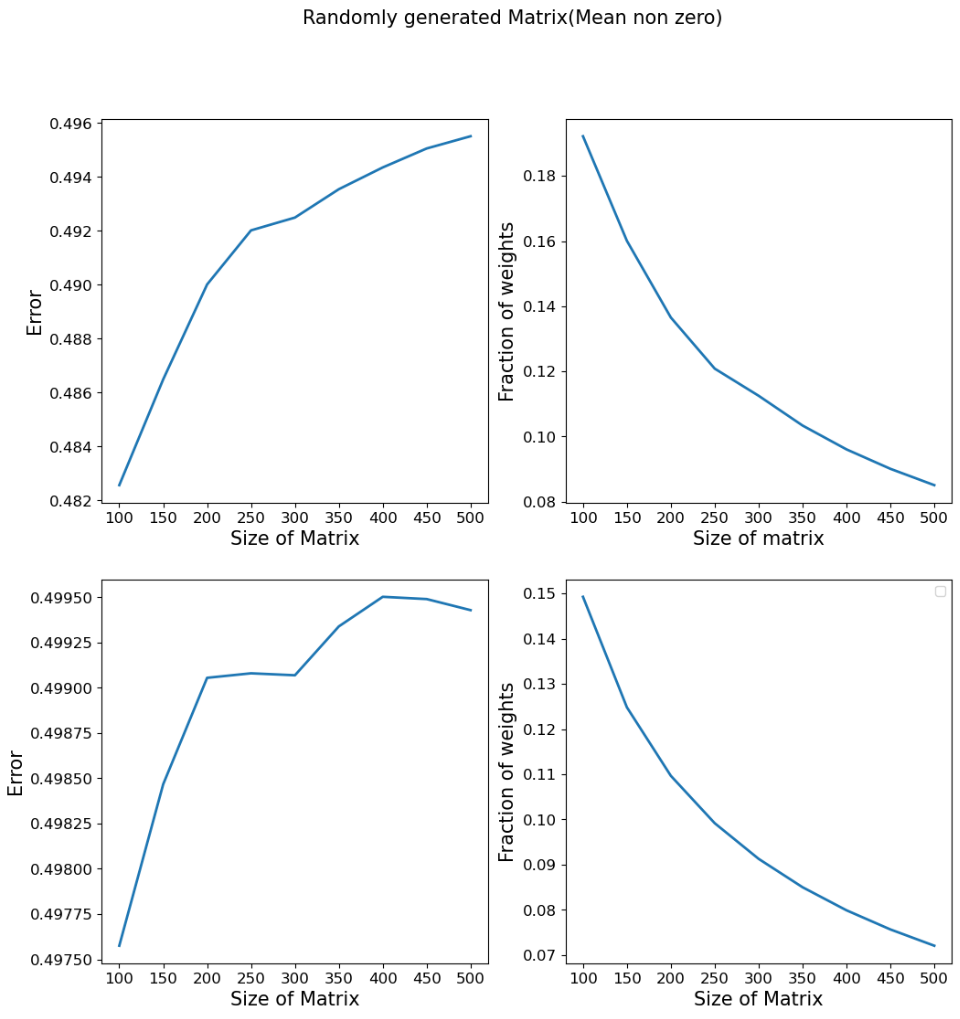


Figure 7.1: Mean of the error in matrix approximation for random general matrix generated from uniform distribution for 100 samples, the top two figures are by considering $\log n$ circulant components, and the bottom figures are by considering one circulant component.

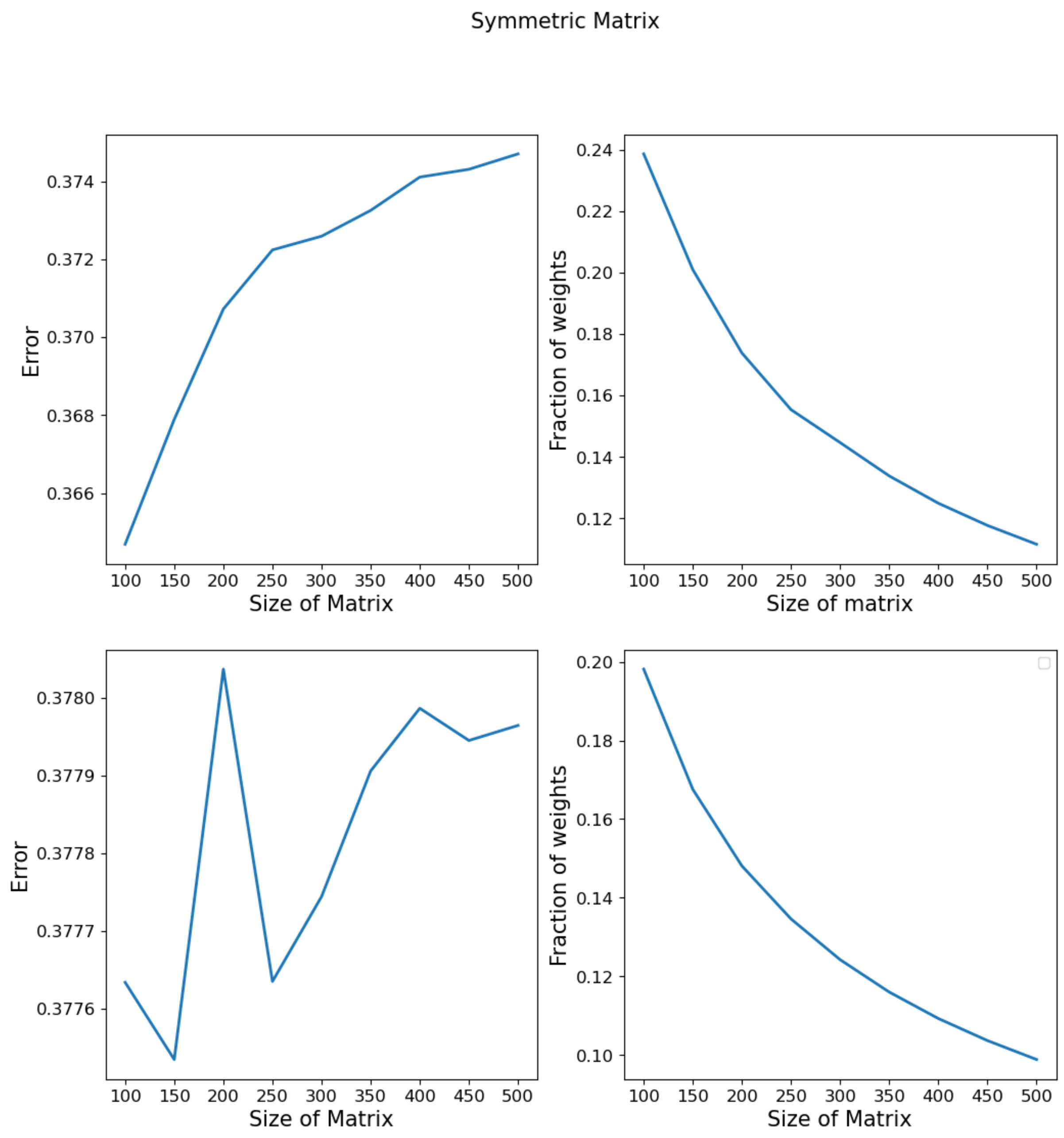


Figure 7.2: Mean of the error in matrix approximation for symmetric matrix for 100 samples, the top two figures are by considering $\log n$ circulant components, and the bottom figures are by considering one circulant component

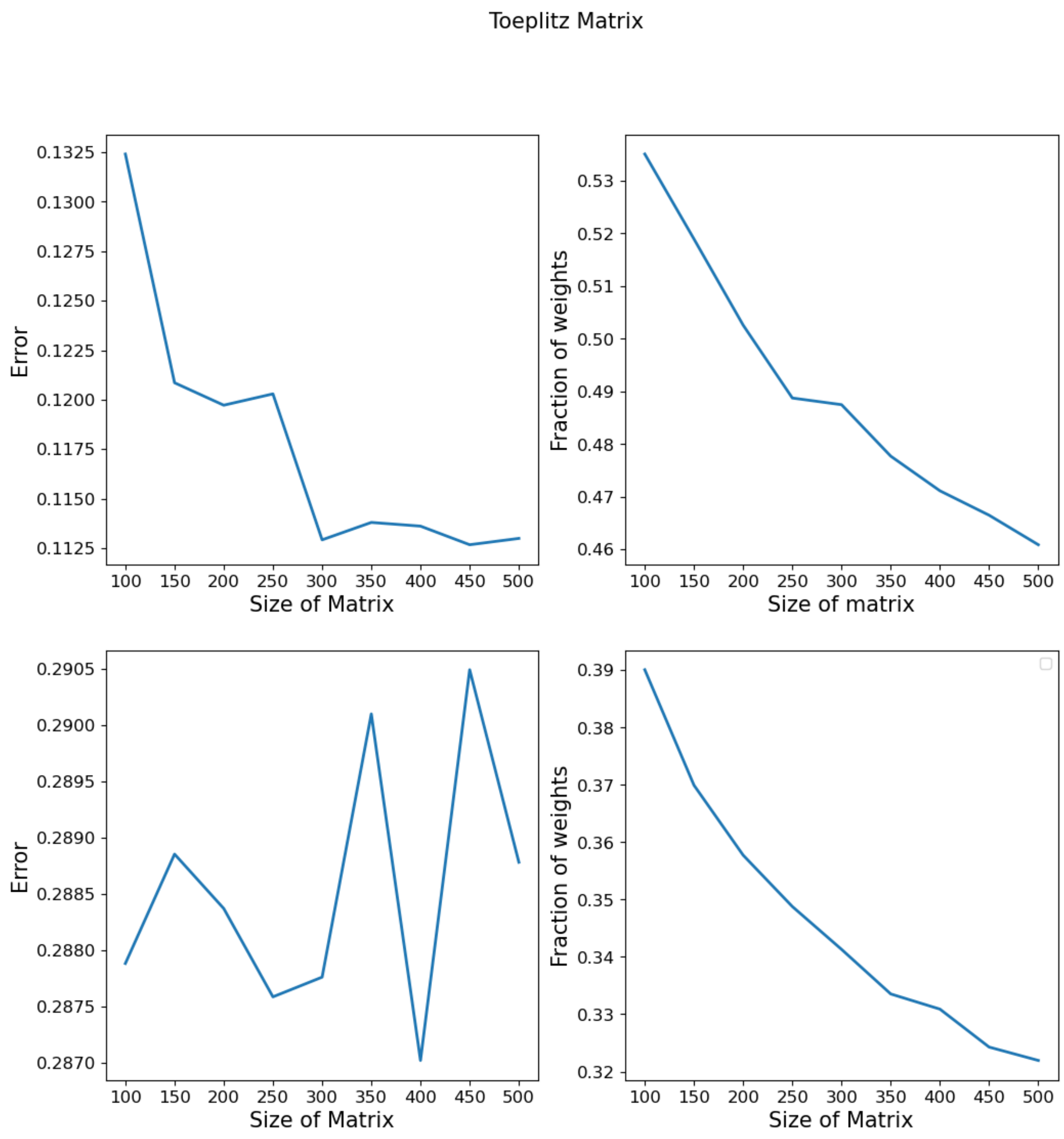


Figure 7.3: Mean of the error in matrix approximation for Toeplitz matrix for 100 samples, the top two figures are by considering $\log n$ circulant components, and the bottom figures are by considering one circulant component

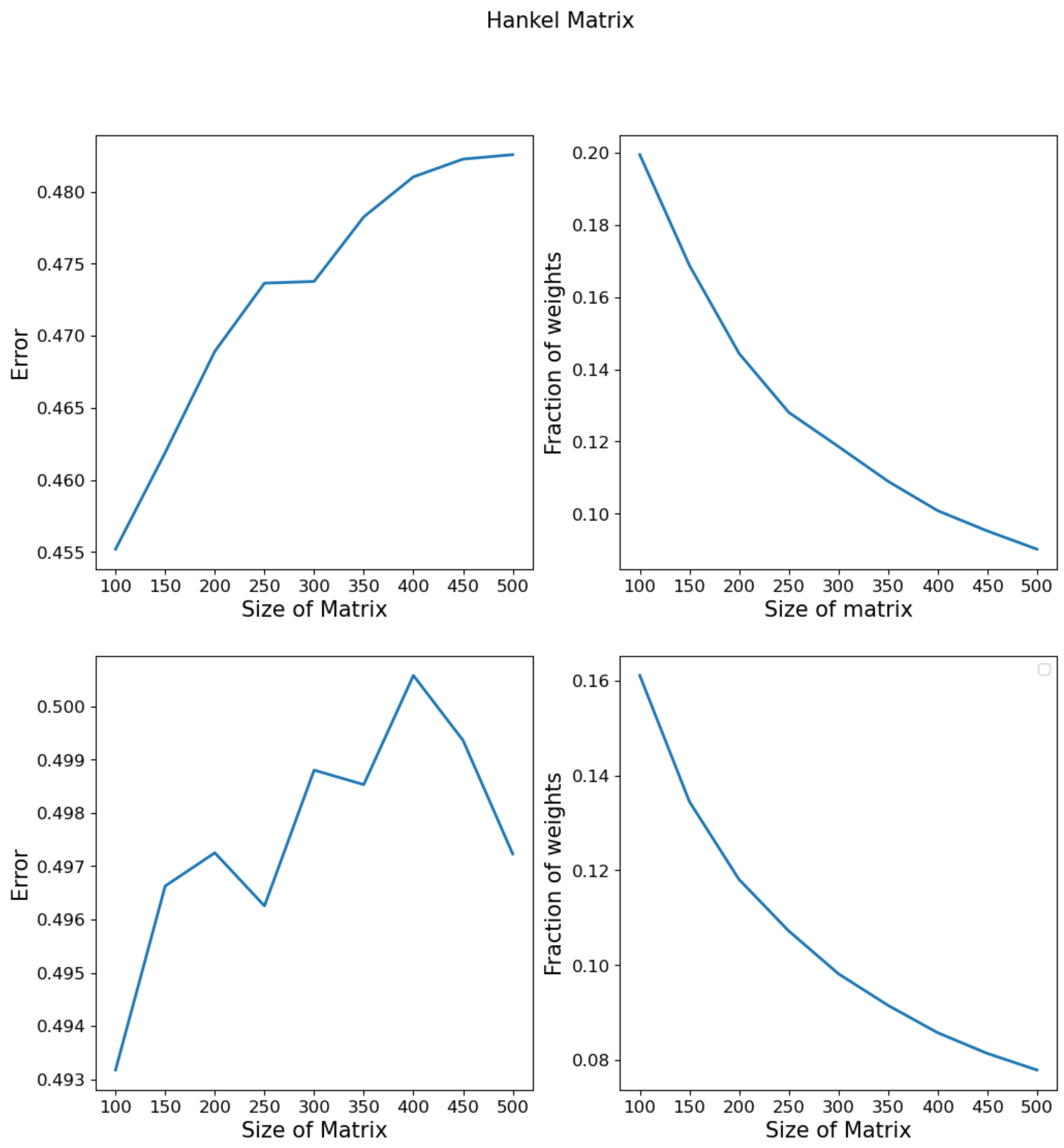


Figure 7.4: Mean of the error in matrix approximation for Hankel matrix for 100 samples, the top two figures are by considering $\log n$ circulant components, and the bottom figures are by considering one circulant component

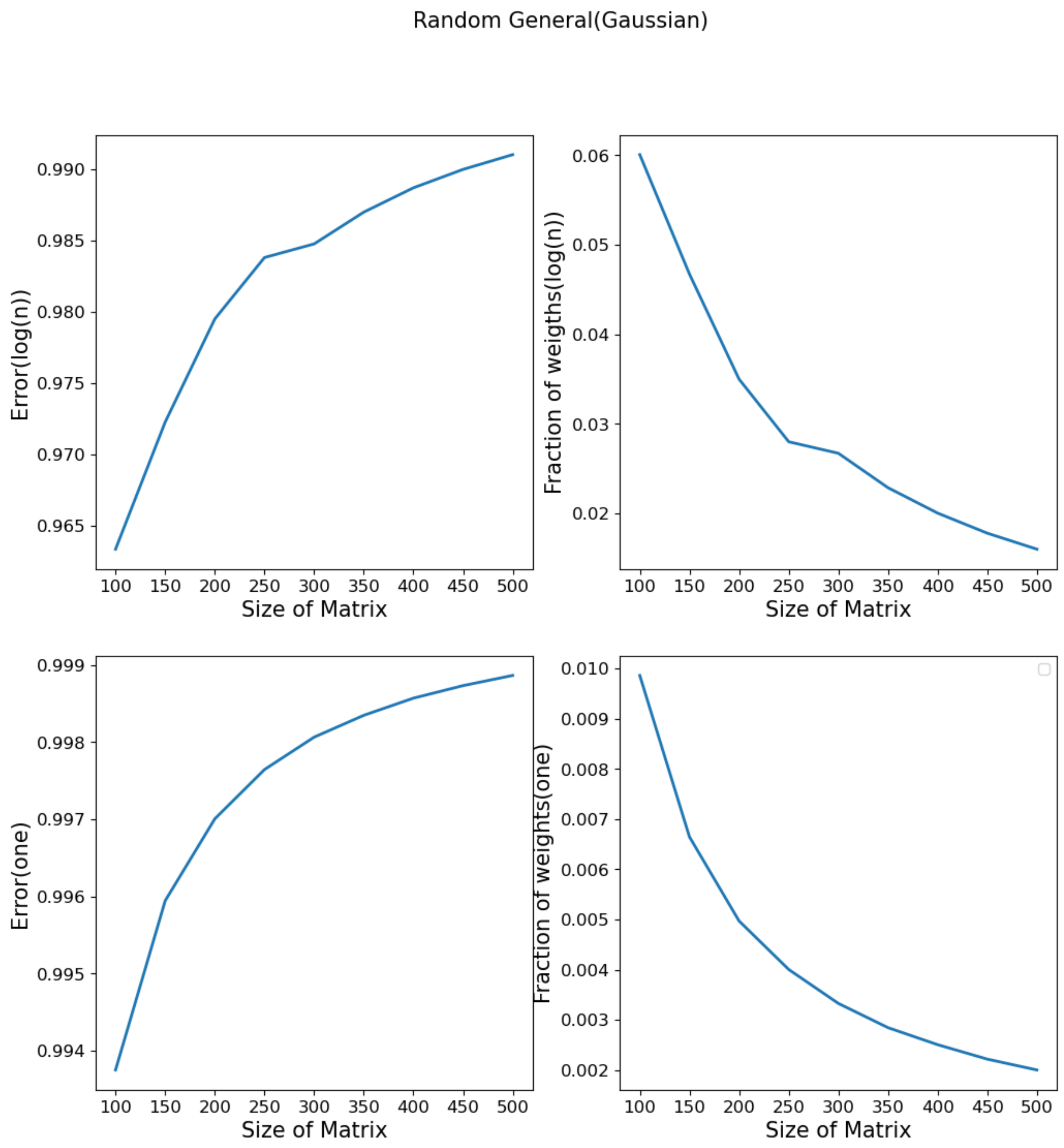


Figure 7.5: Mean of the error in matrix approximation for matrix generated from mean zero distribution for 100 samples, the top two figures are by considering $\log n$ circulant components, and the bottom figures are by considering one circulant component

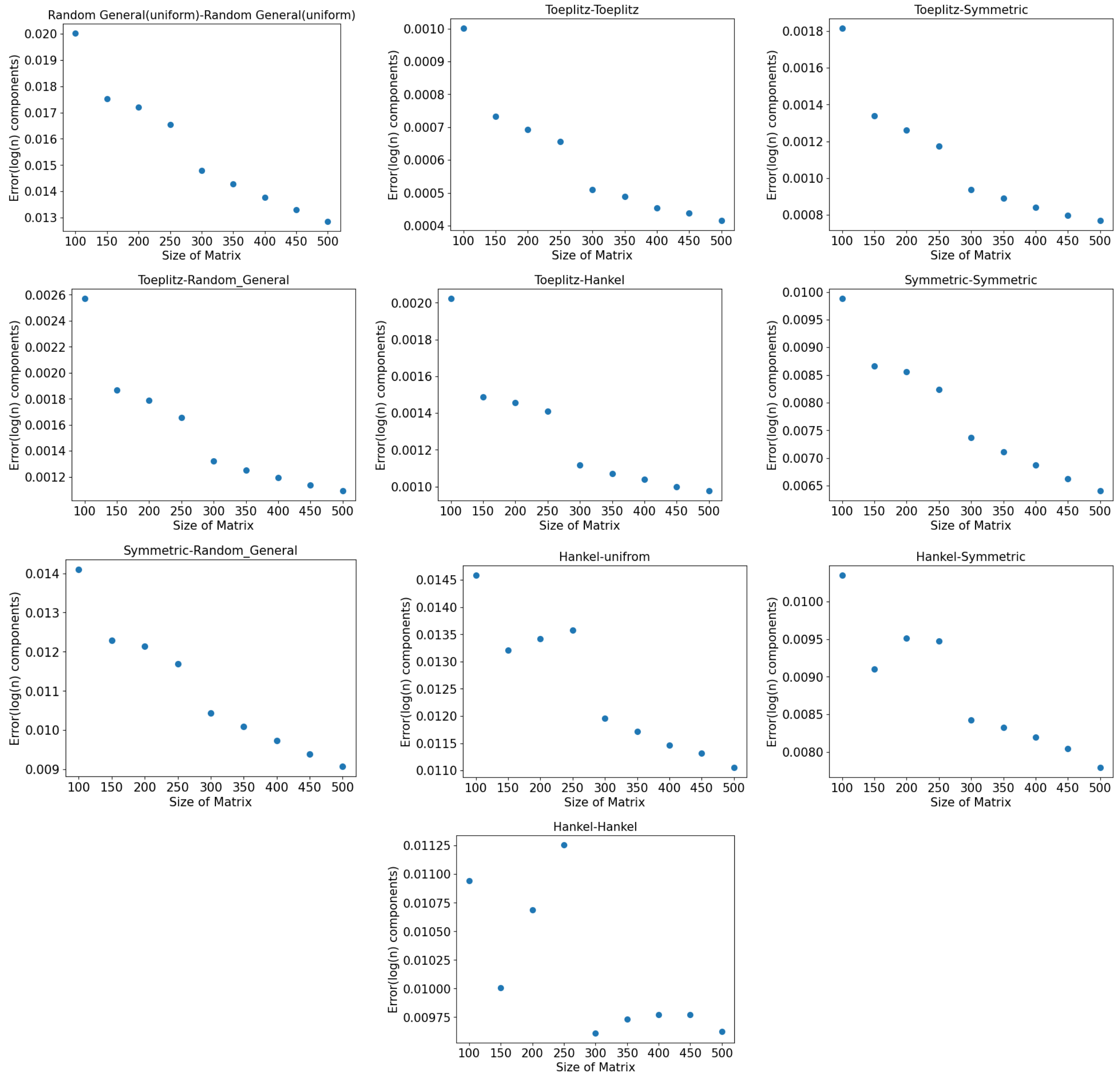


Figure 7.6: Mean of the errors in matrix multiplication for different combinations of randomly generated matrices for the first two categories explained in section 4. The matrix size varies from 100 to 500, with a step size of 50 running for 50 iterations for each size.

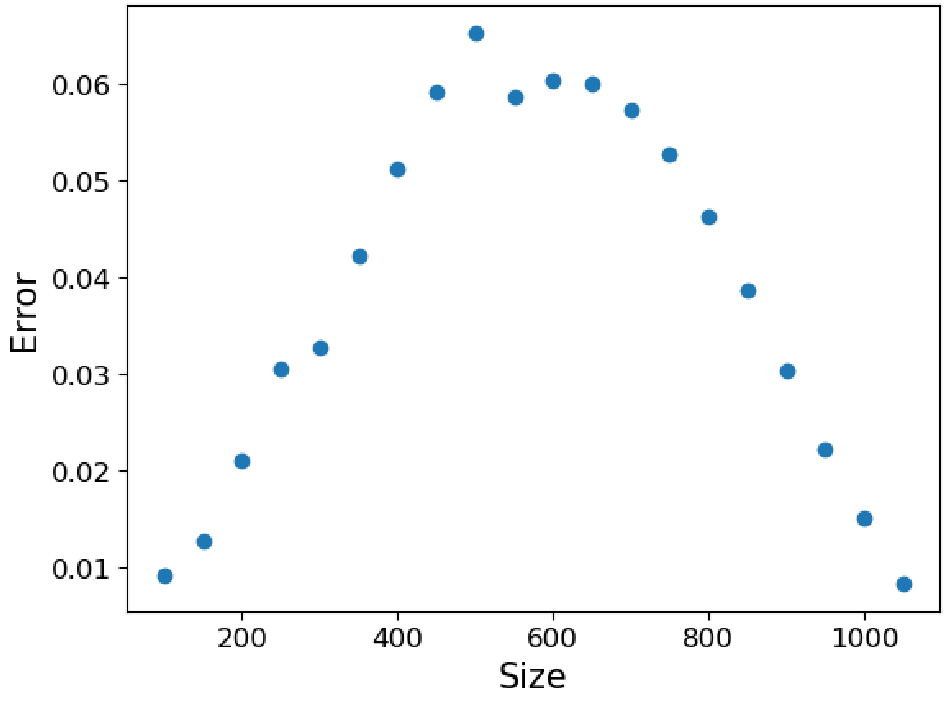


Figure 7.7: Error in multiplication of dft matrix with itself

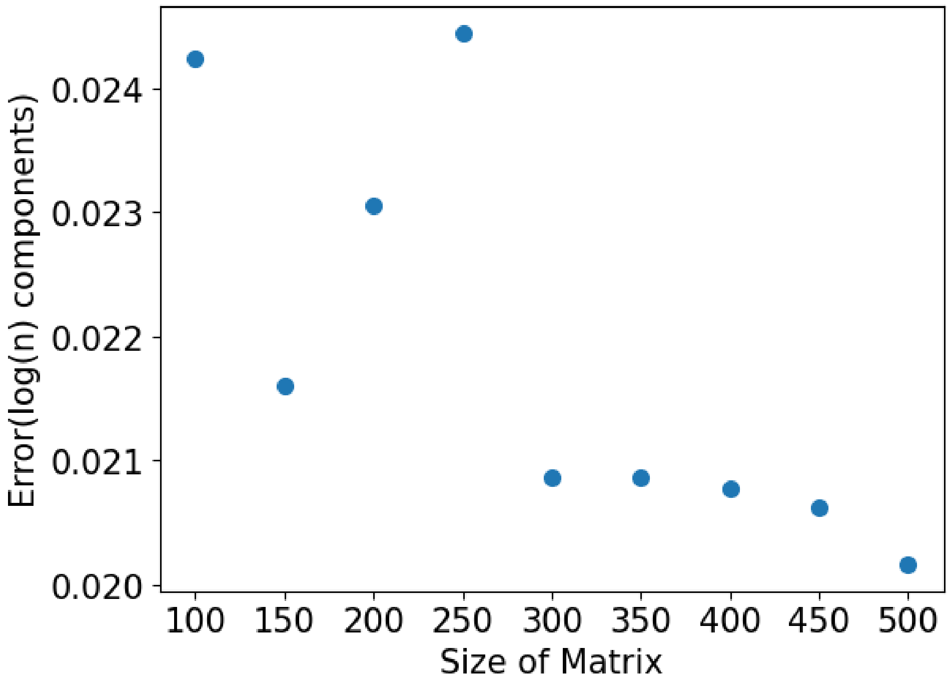


Figure 7.8: Error in the multiplication of Dense matrix with itself

Chapter 8

Reference

[**1**]Sivaram Ambikasaran and Eric Darve. *An $\mathcal{O}(n)$ fast direct solver for partial hierarchically semi-separable matrices.* Journal of Scientific Computing, 57(3):477–501, 2013

[**2**]*Circulant decomposition of a matrix and the eigenvalues of Toeplitz type matrices* by Hariprasad M. and Murugesan Venkatapathi

[**3**]Strassen, Volker (1969). *Gaussian Elimination is not Optimal.* Numer. Math. 13 (4): 354–356

[**4**]Virginia Vassilevska Williams. *Multiplying matrices faster than Coppersmith-Winograd.* In Proceedings of the 44th Symposium on Theory of Computing Conference, (STOC), pages 887–898, 2012.

[**5**] Ran Duan, Hongxun Wu, and Renfei Zhou. *Faster matrix multiplication via asymmetric hashing.* In Proceedings of the 64th IEEE Symposium on Foundations of Computer Science (FOCS), 2023

[**6**] François Le Gall. *Powers of tensors and fast matrix multiplication.* In Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC), pages 296–303, 2014

[**7**]Andris Ambainis, Yuval Filmus, and François Le Gall. *Fast matrix multiplication: limitations of the Coppersmith-Winograd method.* In Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC), pages 585–593, 2015.

[**8**]Josh Alman and Virginia Vassilevska Williams. *Further limitations of the known ap-*

proaches for matrix multiplication. In Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS), pages 25:1–25:15, 2018.

[**9**]Matthias Christandl, Péter Vrana, and Jeroen Zuiddam. *Barriers for fast matrix multiplication from irreversibility.* Theory Comput., 17:1–32, 2021.

[**10**]Josh Alman. *Limits on the universal method for matrix multiplication.* Theory Comput., 17:1–30, 2021.

[**11**]Josh Alman and Virginia Vassilevska Williams. *A refined laser method and faster matrix multiplication.* In Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 522–539, 2021.

[**12**] Kannan, Ravindran and Vempala, Santosh, ”Randomized algorithms in numerical linear algebra”, Acta Nuerica vol. 26, pp. 95-135,2017.

[**13**] A. W. Bojanczyk, R. P. Brent, F. R. de Hoog, and D. R. Sweet, *On the stability of the Bareiss and related Toeplitz factorization algorithms*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 40–57.

[**14**]G. Cybenko, *The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 303–319

[**15**]T. Kailath and A. H. Sayed, eds., Fast Reliable Algorithms for Matrices with Structure, SIAM, Philadelphia, 1999.

[**16**]G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices, in Linear Algebra for Signal Processing*, IMA Vol. Math. Appl. 69, Springer, New York, 1995, pp. 63–81.

[**17**]I. Gohberg, T. Kailath, and V. Olshevsky, *Fast Gaussian elimination with partial pivoting for matrices with displacement structure*, Math. Comp., 64 (1995), pp. 1557–1576.

[**18**] Zohar, Shalhav, ”Toeplitz Matrix Inversion: The Algorithm of W. F. Trench”, *Association for Computing Machinery, New York, NY, USA*, vol. 16, no. 4, 1969.

[**19**] X.-G. Lv and T.-Z Huang, ”A note on inversion of toeplitz matrices,” *Applied Mathematics Letters,* vol. 20, no. 12, pp. 1189-1193, 2007.

[**20**] Chandrasekaran, S. and Gu, M. and Sun, X. and Xia, J. and Zhu, J. ”A Superfast Algorithm for Toeplitz Systems of Linear Equations”, *SIAM Journal on Matrix Analysis and Applications,* vol. 29, no. 4, pp. 1247-1266, 2008.

[**21**]Strang, Gilbert (May–June 1994). ”Wavelets”. American Scientist. 82 (3): 250–255. JSTOR 29775194. ”This is the most important numerical algorithm of our lifetime...”

[**22**] https://alinush.github.io/2020/03/19/multiplying-a-vector-by-a-toeplitz-matrix.html