EXTREME EIGENVALUE CALCULATIONS THROUGH THE FEAST FRAMEWORK

An Honors Project Manuscript

Presented by

Ayodeji Marquis

Completion Date:

May 2018

Approved By:

Professor Eric Polizzi, ECE

Professor Neal Anderson, ECE

ABSTRACT

Title: **Extreme Eigenvalue Calculations Through the FEAST Framework**
Author: **Ayodeji Marquis**
Thesis/Project Type:**Independent Honors Project**
Approved By: **Professor Eric Polizzi, ECE**
Approved By: **Professor Neal Anderson, ECE**

The eigenvalue problem, $Ax = \lambda x$, is ubiquitous in Science and Engineering. Many numerical tools have been developed over the years to compute eigenvalues and eigenvectors. The FEAST eigenvalue solver belongs to a new generation of techniques that are ideally suited for large-scale parallel computing. FEAST requires the user to specify a search interval in the eigenvalue spectrum which may be difficult to know a priori for computing the extreme eigenvalues. This project proposes a tool for computing the search interval of M extreme lowest or largest eigenvalues. We employ procedures such as Gershgorin's circle theorem and FEAST stochastic techniques in our approach. The resulting search interval which is obtained at a relatively low cost can be used as an input for FEAST.

1 Introduction

When real-world problems are mapped into mathematical sets of linear systems, the size of the matrices obtained can be very large. We can have linear systems of dimension that are $10^6 \times 10^6$, if not more. With this in mind, imagine trying to find a common solution to a million or more linear equations! Solving this manually is nearly impossible because humans do not possess the computational speed and accuracy needed to efficiently calculate this; it would take us a lifetime to solve, and humans are known to be error-prone. Even modern computers will have a hard time computing these accurately in a reasonable amount of time using traditional brute force approaches to finding eigenvalues and eigenvectors.

This problem of solving a set of linear systems along a complex contour efficiently and quickly has baffled mathematicians, and engineers for decades. The solution to this problem has been highly sought after because the importance of a system that can quickly estimate eigenvalues and eigenvectors accurately cannot be overstated. Eigenvalues and eigenvectors are applicable to any field that requires some sort of computation. They are seen in Engineering, the Natural Sciences, Computer Science, and many others. They are popularly used in areas such as image processing, communications, statistics, numerical analysis, and mechanics. Eigenvalues enable us to understand linear transformations because they give us the factor by which compression occurs. Also, eigenvalues and eigenvectors equip us with the ability to store the important properties of a system in just one value and vector.

The FEAST algorithm, recently adopted into Intel's math library, fundamentally differs from previous approaches to solving the eigenvalue problem. Its most important feature is its scalability; it can process multiple levels of parallelism which makes it ideally suited for taking advantage of modern computing architecture that is comprised of thousands of computing nodes. The main computational aspect of the algorithm is solving a set of linear systems along a complex contour. Its computational robustness has been tested against other eigenvalue problem-solving software such as the ARPACK, and FEAST has been shown to outperform the ARPACK in both reliability and scalability[1][2][3].

FEAST requires the user to specify a search interval in the eigenvalue spectrum which may be difficult to know a priori for computing the extreme eigenvalues. This project aims to provide a tool to find the extreme lowest or largest eigenvalues that best describes the system, and meets the user's requirements without having them to specify a search interval. We are more concerned with extreme eigenvalues (i.e. minimum or maximum), because eigenvalue solvers are mostly used in legacy applications which require optimization, thus the need for extreme eigenvalues. This tool will provide an optimal input search for FEAST which can then be used to solve the eigenvalue problem.

Section 2, the review of literature, focuses on defining terminologies and looking into papers that would better our understanding of the Eigenvalue problem and the FEAST framework. Section 3, the methodology, discusses the approaches we use to find an efficient estimation of eigenvalue intervals and their corresponding eigenvalues for the standardized eigenvalue problem, $Ax = \lambda x$. Here we employ a combination Gershgorin's circle theorem and FEAST stochastic techniques to estimate our intervals. Section 4, the test sections, analyzes the results from our methodology by evaluating the efficiency and accuracy of our algorithm

2 Review of Literature

This section focuses on introducing the eigenvalue problem and further expanding our knowledge of it. Here we define the terminologies, know the importance of eigenvalues and look into traditional approaches of solving the eigenvalue problem.

Per a lecture note, "an eigenvalue is a scalar λ of the $n \times n$ matrix A, if there is a nontrivial solution x of $Ax = \lambda x$. Such that x is called the eigenvector of the corresponding to the to the eigenvalue λ.[4]" Note, an eigenvector cannot be zero, but an eigenvalue can be zero. What does this mean geometrically? The lecture note tells us that suppose "A is the standard matrix for a linear transformation $T : R^n \to R^n$. Then if $Ax = \lambda x$, it follows

that $T(x) = \lambda x$. This means that if x is an eigenvector of A, then the image of x under the transformation T is a scalar multiple of x – and the scalar involved is the corresponding eigenvalue λ. In other words, the image of x is parallel to x [4]".

The typical way of finding eigenvalues of a matrix A is to solve the characteristic polynomial $det(A - \lambda I) = 0$, where det means the determinant, and I is the identity matrix, i.e $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

$$\text{Given a resultant matrix } A - \lambda I = \begin{bmatrix} 2-\lambda & 5 & 8 \\ 0 & -\lambda & 8 \\ 0 & 0 & 4-\lambda \end{bmatrix},$$

the determinant of $A - \lambda I = (2 - \lambda)(-\lambda)(4 - \lambda)$. Setting the determinant equal to 0 and solving for λ, we get that $\lambda = 0, 2$, or 4 which are the three eigenvalues of A. Looking at the setup of the characteristic polynomial, one can easily see that this approach isn't efficient. The bigger the dimension of the matrix, the harder it is to solve the characteristic polynomial equation. Imagine trying to find all the eigenvalues of a matrix of dimension $10^6 \times 10^6$ using the approach above; this becomes nearly impossible for even a computer to solve in a reasonable amount of time, as it will require the computer to solve a polynomial equation of degree 10^6.

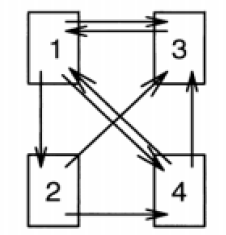
Why do we care so much about eigenvalue problems? Why have they been emphasized so much on? Well, most real-world problems can be mapped into series of mathematical equations which can then be mapped into sets of linear systems. The matrices obtained can be very difficult to understand and store in memory, but eigenvectors make understanding linear transformations easy. They are the directions along which a linear transformation acts simply by "stretching/compressing" and/or "flipping"; eigenvalues give you the factors by which this compression occurs [5]. Also, eigenvalues and eigenvectors allow one to capture some essential features of a system in just one value and one vector [5]. This is simpler than storing an $N \times N$ matrix. Due to this property, eigenvalues are very important in big

data analysis and statistics for dimensionality reduction. Also, eigenvalues are important in physics because we can easily characterize systems; their application is seen in quantum mechanics, Fourier analysis, and Markov chaining [5]. Most importantly, its application in Computer Science cannot be overemphasized. The most talked about beneficiary of the eigenvalue calculation is Google's PageRank Algorithm.

Since Google went online in the late 1990s, its success has been primarily based on the PageRank algorithm (aka the $25B Eigenvector Linear algebra). The PageRank algorithm numerically assesses the importance of each page on the web, allowing Google to rank the pages and thereby presenting users with the most relevant pages first [6]; this was what set Google apart from every other search engine back in the 1990s. The core idea behind this algorithm is assigning a score to each web page. The score of a web page is determined by the number of links to that web page from every other web page. The links to a given page are called the back-links for that page [6]. This sounds simple, right? But here is where Google gets clever, they make it such that a link to page k from an important page adds more weight to the score the of page k's more than a link from an unimportant page. They also ensure that a web page doesn't gain extra influence by linking to multiple pages. So, if page y contains n_y links, one which links to page z, then page z's score will be boosted by x_y/n_y, rather than by x_y [6]. Using this approach, each page gets a total of one vote, weighted by that web page's score, that's evenly divided up among all its outgoing links [6]. To summarize this for a web of n pages linking to page k, we get

$$\sum_{j=1}^{n} \frac{x_j}{n_j}[6] \tag{1}$$

Given an internet structure shown in figure 1a and using the equation above 1, for page 1, we have $X_1 = X_3/1 + X_4/2$, since pages 3 and 4 are backlinks for page 1, and page 3 contains only one link, while page 4 contains two links (splitting its vote in half) [6]. Similarly, $X_2 = X_1/3, X_3 = X_1/3 + X_2/2 + X_4/2$, and $X_4 = X_1/3 + X_2/2$. These linear



(a) An example of a web with only 4 pages. An arrow from page A to page B indicates a link from page A to page B

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}.$$

(b) Link matrix

Figure 1: After [6]

equations can be written $Ax = x$, where x is the transpose of $[X_1, X_2, X_3, X_4]$ and now we have a link matrix A, which transforms the web ranking problem into the standard problem of finding an eigenvector for a square matrix! Note the matrix is column stochastic; this means that all its entries are positive and the entries in each column sum to one. This kind of matrix leads to the idea of Markov chaining which is another topic of its own.

Having seen the importance of eigenvalues and eigenvectors, what methods are currently employed to solve such systems? The traditional algorithms include power Iteration, inverse iteration, the Jacobi eigenvalue algorithm, and the Krylov subspace techniques. Per Wolfram Alpha, "the Jacobi method is probably the simplest iterative method for solving eigenvalue problems. It is a method of solving matrix equation on a matrix that has no zeros along its main diagonal." This algorithm applies to real symmetric matrices, and it outputs every possible eigenvalue of a system. The process is iterated until it converges; it converges quadratically and its cost per step is $O(n^3)$. The simplicity of the Jacobi method is both a good and bad thing. Good in the sense that it's very easy to understand, and bad in the sense that it is not typically used in practice due to its speed limitation. Just like the Jacobi method, the Power method uses an iterative approach.

In this method, you pick a random nonzero vector(x_0) as an approximate for the eigenvector of the system, then you multiply the system 'A' with this vector to generate another approximate eigenvector x_1 [7]. You then continue to repeat these and the more you do it,

$$\begin{aligned}
\mathbf{x}_1 &= A\mathbf{x}_0 \\
\mathbf{x}_2 &= A\mathbf{x}_1 = A(A\mathbf{x}_0) = A^2\mathbf{x}_0 \\
\mathbf{x}_3 &= A\mathbf{x}_2 = A(A^2\mathbf{x}_0) = A^3\mathbf{x}_0 \\
&\vdots \\
\mathbf{x}_k &= A\mathbf{x}_{k-1} = A(A^{k-1}\mathbf{x}_0) = A^k\mathbf{x}_0.
\end{aligned}$$

Figure 2: Power Method (After [7])

the better the approximation of the dominant eigenvector of A (see figure 2). The power iteration method applies to general matrices, and it outputs the eigenpair with the largest value. This method converges linearly and the cost per step is $O(n^2)$, which is faster then the Jacobi method but still slow for very large matrices.

Now that we have seen how intuitive some of the old approaches were, we must also consider recent algorithms which have taken on the challenge to tackle both the speed limitation and efficiency of the eigenvalue problem. A new algorithm for finding the largest and smallest eigenvector was recently published by Yiguang and Zhisheng titled, "A concise Functional Neural Network (FNN) Computing the Largest (Smallest) eigenvalue and One Corresponding Eigenvector of a Real Symmetric matrix". In the paper [8], a previous FNN model was improved from

$$\frac{dx(t)}{dt} = Ax(t) - x^T(t)x(t)x(t) \tag{2}$$

to

$$\frac{dx(t)}{dt} = Ax(t) - sgn[x^T](t)x(t)x(t) \tag{3}$$

In equation 3, A represents the real symmetric matrix where $x(t)$ represents the state of the neurons. Since A is real symmetrical, all eigenvalues are denoted as $\lambda_1 > \lambda_2 > \ldots > \lambda_n$. Thus using the lemma in the article [8] we get the following results:

$$\epsilon = lim_{t\to\infty}x(t)$$

and

$$A\epsilon = \sum_{i=1}^{n} |\epsilon_i|\epsilon,$$

when ϵ is an eigenvector and the eigenvalue is λ is

$$\lambda = \sum_{i=1}^{n} |\epsilon_i|$$

Using the eigenvalue obtained, the analytic solution of FNN $x_i(t) = f(x_i(0), \lambda i, t)$ can then be solved. The detailed steps (algorithm) to compute the largest and smallest λ are concisely written in the article [8]. This method is adaptive to non-definite, positive definite and negative definite matrices [8].

Secondly, another modern algorithm has also been developed to compute extreme eigenvectors of a complex Hermitian matrix, that is, a symmetric matrix, $f(x) = 1/2trx^H Ax$, where tr(trace) is the sum of all elements in the main diagonal. A Hermitian matrix is a complex square matrix (A) that is equal to its conjugate transpose(A^H). This method differs from the first one in the type of matrix being solved, this solves complex Hermitian symmetric matrices while the other solve real symmetric matrices. Also, this algorithm is guaranteed to converge to a minimal eigenvector, given that the initial vector is not orthogonal to the Eigen space spanned by the minimal eigenvectors [9]. According to Manton, "this approach varies from the traditional ones, such as power and inverse iteration methods. Firstly, whereas traditional methods can fail to converge in a reasonable number of iterations if two or more eigenvalues are closely spaced, the proposed algorithm continues to converge rapidly in such situations. Secondly, unlike traditional methods which converge to the eigenvector associated with the eigenvalue having the smallest or the largest absolute value, the proposed algorithm converges to the eigenvector associated with the eigenvalue having the smallest or the largest value [9]." This new algorithm works by finding a descent, such that, given a point x, we compute a descent direction z and a step size γ such that $f(x + z) < f(x)$, where [9]. You want to start off by picking an x in the region, such that

$x^H x = 1$. Then you compute z, where $\lambda = x^H Ax$. If the $\sqrt{z^H z}$ is very small, then we are done. Otherwise, we compute the equations $\alpha = x^H A^2 x - \lambda^2$ and $\beta = x^H A^3 x - 3\alpha\lambda - \lambda^3$ to solve for the step size, which is the positive root of the equation $\alpha^2\gamma^2 + \beta\gamma - \alpha = 0$ [9]. Then we reset x to $x + \gamma z$, re-compute z and keep doing this until we get a z that's sufficiently small enough. Note, a small z produces the minimal λ that we are looking for.

This paper has shown how powerful tools have been developed to compute eigenvalues and eigenvectors from the Jacobi Method to the power iterations to the FNN, but these traditional numerical algorithms are extremely computationally challenging, slow, and linear scalability is not easily achieved. A fast and stable numerical algorithm for solving the symmetric eigenvalue problem named FEAST was published by Professor Eric Polizzi. The FEAST algorithm takes its inspiration from the contour integration and density – matrix representation in quantum mechanics [1].

The basic FEAST algorithm was curated for solving generalized eigenvalue problems of this form $Ax = \lambda Bx$, within a given interval $[\lambda min, \lambda max]$, where A is a real symmetric matrix or Hermitian matrix, and B is symmetric positive definite. The approach to deriving the FEAST algorithm is clever. The author [1] takes the Green's function and performs an exact mathematical factorization of its contour integration which represents the reduced density matrix ρ in quantum mechanics. Multiplying the ρ by a set random vectors(Y) of leads to a new set of M-independent vectors $Q_{NxM} = q_1, q_2, ..., q_m$ obtained by solving linear systems along the contour. Doing this gives us the following important equations mentioned

in the article [1]:

$$\rho = -\frac{1}{2\pi\iota}\int_C dZG(Z) = \sum_{m=1}^{M} X_m X_m^H \tag{4}$$

$$\begin{aligned} A_Q\Phi &= \epsilon B_Q\Phi, \text{ with} \\ A_Q = Q^T AQ \ and \ B_Q &= Q^T BQ, \\ \text{The Ritz values and vectors are then given by} & \\ \lambda_m &= \epsilon_m, m = 1, \ldots, M \end{aligned} \tag{5}$$

In practice, the contour integration region is usually replaced with a circular region. This simplifies the integral a bit because of the even symmetric property of a circle. Though this might be clever, but evaluating the integral is still computationally challenging because the size of the Green's function matrix could be very large. To overcome this challenge, Polizzi uses a clever numerical analysis approach that solves Q in seconds regardless of the size of $G(Z)$. He uses a N_e-point Gauss-Legendre quadrature on the positive half of the circle $C+$ with x_e the e_{th} Gauss node associated with the weight w_e, one obtains if A is Hermitian and Y, $Q \in \mathbb{C}^{N\times M}$ [1].

$$\begin{aligned} Q = -\sum_{e=1}^{N_e}\frac{1}{4}\omega_e r[exp(\iota\theta_e)G(Z_e) + exp(-\iota\theta_e)G^T(Z_e)]Y, \\ r = \frac{\lambda_{max} - \lambda_{min}}{2}, \theta_e = -(\pi/2)(x_e - 1) \end{aligned} \tag{6}$$

This method reduces the number of linear systems to be solved i.e a million sized linear system is reduced to like size eight. This way the value of Q is quickly obtained, the operator can proceed to find the value of A_q and B_q, and this results in getting the eigenvalue, λ_m and the corresponding eigenvector as seen in equation 5.

The numerical stability, robustness, and scalability of the FEAST algorithm were tested against that of the ARPACK. The ARPACK is a numerical software library that is also very popular for solving large scale eigenvalue problems[2]. The three simulations ran were as

follows: i) Simulation times and relative residual (%error) obtained by the solver ARPACK and FEAST on the Test Carbon nanotube system seeking M eigenpairs for different search intervals, ii) Variation of the relative error on the trace of the eigenvalues for different search intervals with the number of iterative refinement loops, and iii) Performance results obtained by FEAST seeking $M = 100$ eigenpairs for different values of Ne (number of linear systems to solve) [1]. In all three experiments, the FEAST significantly outperformed ARPACK in both runtime and percentage error. Not only does it outperform the ARPACK, it also exhibits important potentialities for parallelism at three different levels, per Polizzi, these are:

1. "Many search interval $[\lambda min, \Lambda max]$ can be run independently
2. Each linear system can be solved simultaneously (e.g., on each node of parallel architecture where the factorization of the linear system can be done only once for all the refinement loops
3. The linear system solver can be parallel [1]."

Due to its robustness, the FEAST algorithm has been adopted into Intel's Math Kernel library. As we have seen, it fundamentally differs from previous approaches to solving the eigenvalue problem. Most importantly, the algorithm is highly scalable; it can process multiple levels of parallelism which makes it ideally suited for taking advantage of modern computing architecture that is comprised of thousands of computing nodes [10]. This is very useful because most real-world problems can be mapped into series of mathematical linear equations, and the importance of having a framework that finds eigenvalues of large systems quickly like FEAST cannot be overstated.

As robust as the FEAST framework is, it requires the user to specify a search interval in the eigenvalue spectrum which may be difficult to know a priori for computing the extreme eigenvalues. Specifying an interval may seem trivial but surprisingly it's not. The biggest challenge for the user is trying to specify an interval that provides the number of eigenvalues

they want. If we were talking about simple systems, for example, matrices of size 5×5, then this would be an easy thing to do because you can tell the properties of the system, thus enabling you easily specify the interval. However FEAST was not made to solve very simple linear systems, it was made to find eigenvalues of very large systems (e.g $10^6 \times 10^6$ matrix). And specifying an interval for such a large system could be as difficult as solving for its eigenvalues by hand. Not to even mention the fact that linear systems and their eigenvalue problems can come in different forms (standardized, hermitian, non-symmetric, etc.). We have two main types of eigenvalue problems. There is the standardized eigenvalue problem and the generalized eigenvalue problem. The standard eigenvalue problem is of the form $Ax = \lambda x$ while the generalized eigenvalue problem is of the form $Ax = \lambda Bx$. In the generalized eigenvalue problem we focus on solving for eigenvalues (λ) and eigenvectors (x) that describe two systems A and B. When solving for generalized eigenvalue problems, most mathematicians and engineers try to find intuitive ways to convert the generalized eigenvalue problem to standardized eigenvalue problem such that the form $Ax = \lambda Bx$ becomes $(B^{-1}A)x = \lambda x$. Converting generalized eigenvalue problems to standardized eigenvalue problems enable us use the several approaches of solving eigenvalue problems like those mentioned in this document. Mathematically, this seems very easy and direct, however, as the size of matrices (A and B) increase, our computational and space complexity also increases. More storage is needed for both matrices, more processing power is needed because finding the inverse of B could be difficult, and most of the methods/algorithms being used to solve these systems are based on approximations, and in the case where we have to do with deal two matrices our approximations tend to be less precise. So solving for generalized eigenvalue problems tend to be more problematic than solving for standardized eigenvalue problems.

Polizzi and I have spent most the year looking at different techniques that can be used to implement a tool/feature that allows users to solve for extreme eigenvalues without having them specify a particular interval to the system. For the standardized eigenvalue problem, we have been successful in developing a tool to find the search interval of the M wanted

lowest or largest eigenvalues.

3 Explanation of Current Methodology and Goals

This project is so theoretical and analytical that we had to get acquainted with the FEAST framework first by exploring the routines for the five main family of eigenvalue problems which are real and symmetric, complex and hermitian, complex and symmetric, real and non-symmetric, and complex and general. We then studied the concepts, theorems and algorithms behind standardized and generalized eigenvalues problems. We also looked at how variant of this problem is being solved for different matrices such real, complex, symmetric, non- symmetric, hermitian and non-Hermitian matrices. The knowledge gained from our research was then used to optimize the FEAST algorithm by providing a tool that allow users to solve for extreme eigenvalues of the standard eigenvalue problem without having them specify a particular interval to the system.

The the rest of this section will be focused on explaining what we have done so far, and where we hope to get.

Have other people tried to address the specification of intervals for eigenvalue problems before? The answer is 'Yes, but indirectly'. They have addressed other variations of this problem. Gershgorin, a famous Mathematician, came up with a theorem that can be used to bound the spectrum of a square matrix. It's called the Gershgorin's circle theorem. This theorem is very straightforward in its explanation and proof, and it can be used to determine what range the eigenvalues would be in. The theorem says let A be a complex $n \times n$ matrix whose entries are denoted by a_{ij}, we define [11], [12]

$R_i(A) = \sum_{i \neq j} |a_{ij}|$ to be the absolute sum of all the non-diagonal element in the i_th row and

(7)

$$D_i(A) = z \in \mathbb{C} : |z - a_{ii}| \leq R_i(A) \tag{8}$$

Then every eigenvalue is contained inside the union of D_i. This theorem is very important because it serves as the basis for our current methodology being used to implement the feature.

Polizzi too worked on a variant of this problem. His work with Di Napoli in 2014 on an efficient estimation of eigenvalue counts in an interval is very vital. They estimated the number of eigenvalues located in a given interval of a large sparse hermitian matrix using stochastic approaches based on polynomial and rational approximation filtering[13]. Estimation of eigenvalue counts is important in certain applications and it is a prerequisite of eigensolvers based on a dichotomy[13]; it also forms the basis of our current methodology. The FEAST framework coupled along with Gershgorin's circle theorem and Polizzi's stochastic approach gives us a way of solving for a number of extreme eigenvalues without the user having to specify an interval.

Gershgorin's circle theorem which was explained in section 2 serves as the basis of our current methodology because it allows us to estimate the eigenvalue bound of a given symmetric matrix. When a user parses in a linear system (matrix) of a standardized eigenvalue problem into the FEAST framework, the first thing we do is get the eigenvalue bound using Gershgorin's circle theorem.

We have written an algorithm that implements Gershgorin's circle theorem into FEAST. For efficiency in memory and speed, we had to implement/consider the following:

1. We utilized what is called the compressed sparse row (CSR) format. This is useful because some matrices are really sparse (i.e has lot of zeroes) and the zero elements are redundant, so why waste memory storing this? CSR compresses the matrix while keeping the relevant information.

2. We also had to consider the type of matrices we are working with. In the case of a symmetric matrix A where $A = A^T$, we do not need to store or consider every single element in the matrix because knowing only the lower or upper half of the matrix is enough to recover the full matrix. Once you know one half of it, you know the other

half by simply taking the transpose of it, and this make Gershgorin applicable to both full and triangular matrices

Below is the code that performs Gershgorin's circle theorem (see figure 3 below). It was

```
subroutine dgershgorin1(UPLO,n,nnza,isa,jsa,dsa,dEmin,dEmax)
    !!! The array dsa is of length NNZ and holds all-
    !!! UPLO represents the shape of the matrix (whether full or triangular)
    !!! -the nonzero entries of the matrix(M) in left-to-right top-to-bottom ("row-major") order.
    !!! The array isa is of length n + 1. It's what enables us to iterate the CSR
    !!! jsa, contains the column index in M of each element of dsa and hence is of length NNZ as well.
    !!! dEmin and dEmax represent the variables that store the bound when gersh is complete

    implicit none
    character(len=1) :: UPLO
    integer :: n,nnza
    integer,dimension(*) :: isa,jsa
    double precision :: dEmin, dEmax
    double precision,dimension(*) :: dsa
    double precision, dimension(:), allocatable :: radius, center
    !!!
    integer :: i,index
    allocate(radius(n))
    allocate(center(n))

    radius = 0d0
    do i = 1, n
        do index = isa(i),isa(i+1)-1
            if (UPLO == 'F') then
              if (i/=jsa(index)) radius(i) = radius(i) + ABS(dsa(index))
              if (i == jsa(index)) center(i) = dsa(index)
            else
              if (i/=jsa(index)) then
                radius(i) = radius(i) + ABS(dsa(index))
                radius(jsa(index)) = radius(jsa(index)) + ABS(dsa(index))
              endif
              if (i == jsa(index)) center(i) = dsa(index)
            endif
        enddo
    enddo
    dEmin = huge(dEmin)
    dEmax = -dEmin
    do i = 1,n
        if ((center(i) - radius(i)) < dEmin) then
          dEmin = center(i) - radius(i)
        endif
        if ((center(i) + radius(i)) > dEmax) then
          dEmax = center(i) + radius(i)
        endif
    enddo
    print *,'Min/Max:    ',dEmin,dEmax
    deallocate(radius,center)
end subroutine dgershgorin1
```

Figure 3: Code that implements Gershgorin's circle theorem

tested using the following symmetric sparse matrix,

$$M = \begin{bmatrix} 10 & 0 & -5 & 0 & 0 \\ 0 & 30 & 0 & 10 & 0 \\ -5 & 0 & 40 & 0 & -10 \\ 0 & 10 & 0 & 50 & 0 \\ 0 & 0 & -10 & 0 & 60 \end{bmatrix}$$

which produced the following output in figure 4 that tells us the center and radius of the Gershgorin's disk corresponding to the i_{th} row of the Matrix. The Min/Max in figure 4 represent the eigenvalue bound, so we know our eigenvalues are within $5 \leq \lambda \leq 70$

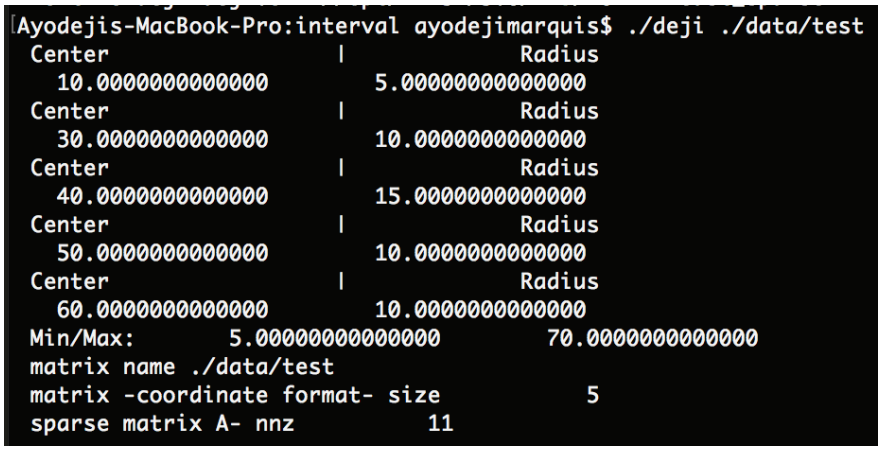


Figure 4: Gershgorin's output of Matrix M

From equation 7 and 8, we know that all the eigenvalues are contained inside the union of the disks, and this can be seen in the following plot done with Matlab (see below, figure 5).

Now that we can successfully estimate eigenvalue bounds, we then proceed to estimate the number (M) of eigenvalue values closest to our extreme bounds (either mimiumum of maximum) by utilizing Napoli and Polizzi's stochastic approach. If this number (M) is in a reasonable offset to the number of eigenvalues needed by the user (M_0), we then proceed solve for the eigenvalues using the FEAST solver, otherwise we keep adjusting our eigenvalue

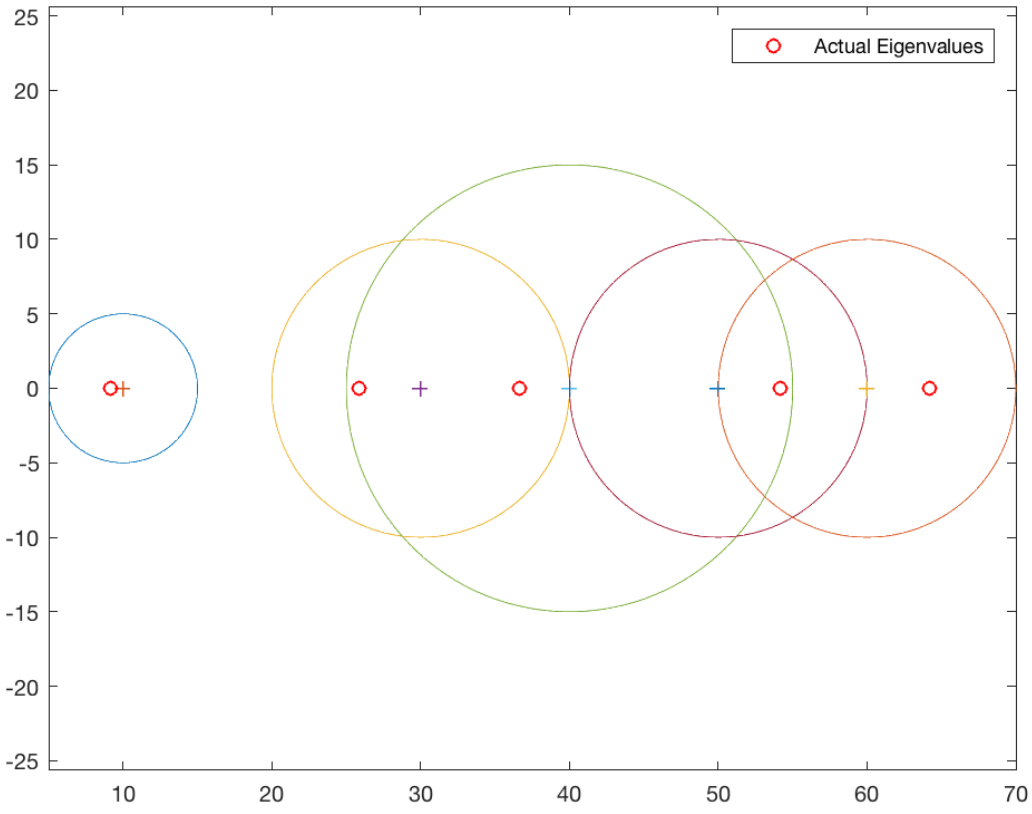


Figure 5: Gershgorin's discs of Matrix M

bound till we find an M that satisfies our condition; this is called dichotomy. Remember for FEAST to work properly we have to ensure our subspace M_0 is at least 2 times greater than than the number of eigenvalues we actually want, so $M_0 = 2M$. The pseudo-code below, 1, explains this process. For the FORTRAN code see figure **??**

The process was then tested using the Na5 matrix in FEAST. The Na5 matrix is symmetric matrix of size 5832×5832 with 155731 non-zero elements. The goal here was to solve for 25 ($M_0 = 2 * 25$) extreme eigenvalues to the standardized eigenvalue problem without the user specifying a search interval. See figure 6 for the output. Gershgorin gives us the Min/Max bound to be $[-10.086, 31.221]$, the dichotomy process only take 2 iterations to find a bound that satisfies the number of eigenvalues needed, and we observe that our eigenvalues are closer to the minimum bound -10.086. This method is very intuitive, and it works well

Algorithm 1 My approach to the interval problem

```
0: procedure MyProcedure
0:    Input matrix to FEAST
0:    FEAST calls gershgorin to get the eigenvalue bound [a,b]
0:    We initialize the number of eigenvalues (M) to be a huge number
0:    b_0 ← b
0:    a_0 ← a
0:    M_1 = M_0/2 ——// The number of eigenvalues the user really wants
0:    i ← 3
0:    if interested in min extreme λ then
0:       b ← (b_0 − a_0)/2^i + a_0
0:    if interested in max extreme  then
0:       a ← b_0 − (b_0 − a_0)/2^i
0:    loop:
0:    if  not M_0/2 + .05 ∗ M_0 < M < (M_0 ∗ 3/4) then
0:       i ← i + 1
0:       Δ ← (b_0 − a_0)/2^i
0:       if no eigenvalues found then
0:          reset i to 3
0:          if interested in min extreme λ then
0:             move our previous bounds slightly to the right
0:          if interested in max extreme  then
0:             move our previous bounds slightly to the left
0:       if  M > M_0 ∗ .75 then
0:          if interested in min extreme λ then
0:             b ← b − Δ ——// Decrease our max
0:          if interested in max extreme λ then
0:             a ← a + Δ
0:       if M_0/2 + .05 ∗ M_0 > M then
0:          if interested in min extremeλ then
0:             b ← b + Δ ——// increase our max by delta
0:          if if interested in max extreme λ then
0:             a ← a − Δ
0:       M ← stochastic() ——//we call stochastic, fpm14, to find a new estimate of M
0:    goto loop.
0:    FEAST solves for λs
```

for standard eigenvalue problems.

```
[^[[AAyodejis-MacBook-Pro:interval ayodejimarquis$ ./deji ../data/Na5
 Min/Max:      -10.0858386363924        31.2209791594811
 matrix name ../data/Na5
 matrix -coordinate format- size        5832
 sparse matrix A- nnz      155731

 Routine  dfeast_scsrev

*********************************************
********** FEAST- BEGIN *********************
*********************************************
Routine DFEAST_S{}{}
List of input parameters fpm(1:64)-- if different from default
   fpm(1)=1
   fpm(2)=5
Search interval [-9.017537408822562e-02; 4.352690452901943e-01]
Size system     5832
Size subspace   50
#Linear systems 5
-------------------------------------------------------------------------------------
#Loop | #Eig  |       Trace           |     Error-Trace       |     Max-Residual
-------------------------------------------------------------------------------------
0       25      5.480815113404439e+00   1.000000000000000e+00   1.622071351200287e-03
1       25      5.480812058040714e+00   7.019483141839841e-06   9.714775161952461e-07
2       25      5.480812058040160e+00   1.271248579379838e-12   5.712659804688475e-10
3       25      5.480812058040163e+00   6.121582244204679e-15   3.035427717838899e-13
==>FEAST has successfully converged (to desired tolerance)
*********************************************
********** FEAST- END************************
*********************************************

 Number of iterations          23
 FEAST OUTPUT INFO           0
 *************************************************
 ************** REPORT ***************************
 *************************************************
 Eigenvalues/Residuals
 inside interval
           1 -8.180518811284594E-002  3.035427717838899E-013
           2 -5.518014222472481E-002  5.138621341523038E-014
           3 -2.116800868554112E-002  4.434873532394522E-014
           4  4.121289015772360E-002  2.561604450567902E-014
           5  6.038410558363245E-002  2.669035302508639E-014
           6  8.825804236287421E-002  3.534786014041921E-014
           7  0.103448473876485       3.162861161294418E-014
           8  0.106022284441572       3.561993274572903E-014
           9  0.148596439686408       4.085006741401628E-014
          10  0.194012353299308       2.635088697477293E-014
          11  0.199435009778216       2.264020134920602E-014
          12  0.221298700102012       2.249219599671810E-014
          13  0.244631929573783       2.191389418390870E-014
          14  0.257679357303821       2.163040778941032E-014
          15  0.273749087220112       1.935638358662599E-014
          16  0.282732873782328       1.768946497076075E-014
          17  0.303148550153459       3.458168749981925E-014
          18  0.340871094771032       1.498863973567237E-014
          19  0.361310226499288       2.414540947820734E-014
          20  0.370182964879676       1.847187191478754E-014
          21  0.382186257687260       1.983082607586513E-014
          22  0.389754956936956       2.681119783718374E-014
          23  0.410713067278826       2.113852995388059E-014
          24  0.424667714932304       4.240836636263571E-014
          25  0.434669016756200       1.413216585514234E-013
```

Figure 6: Output of the dichotomy process

4 Test

Our gershgorin and dichotomy algorithms were tested on a numerous of PARSEC matrices and other matrices already present in FEAST. PARSEC matrices are being used in symmetric eigenvalue problems in density functional theory calculations. The matrices are real, symmetric, sparse, indefinite, with multiple and clustered eigenvalues—typical character of Hamiltonian matrices from the Kohn-Sham equations [14]. The PARSEC group serve as good test cases because they represent actual chemical compounds and these are being used in a lot of research entailing either chemistry or numerical algorithms. The table, 7, below describes the properties of these matrices. In FEAST we employ the following to solve for these matrices:

1. 5 contour points for Hermitian FEAST (half-contour).
2. 20 (default) number of FEAST refinement loops
3. We tell FEAST to allow estimation of eigenvalues inside a search interval. This enables us stochastically estimate the number of eigenvalues in a given spectrum when performing the dichotomy
4. A new parameter has been added to FEAST to specify whether we are interested in minimum eigenvalues or maximum eigenvalues

	---------- Matrix Properties ----------					Type of Eigenvalue Problem	
Matrix	Real	Complex	Symmetric	Hermitian	General	Standard	Generalized
PARSEC/Na5	x		x			x	
PARSEC/SiNa	x		x			x	
PARSEC/Si5H12	x		x			x	
cnt	x		x				x

Figure 7: Table describing the properties of the test matrices

The remainder of this section will be focused on evaluating the results of our methodology and determining how well these algorithms perform.

4.1 Testing with Na5

The Na5 matrix is a real symmetric matrix belonging to the standardized eigenvalue problem.It's of size 5832×5832 with 155731 non-zero elements. It's parsed in as a lower half triangular matrix into FEAST, and our goal is to find at least 50 extreme and at most 100 eigenvalues either near the minimum or the maximum. Since we are interested in 50λs, we set the the size of our subspace $M_0 = 2 * 50$.

The procedure for Gershgorin's circle theorem is then called which gives us an eigenvalue bound of [-10.086, 31.221]. The figure below, 8 is visual representation of the Gershgorin discs. The union of the discs contains every single eigenvalue, λ of the matrix Na5

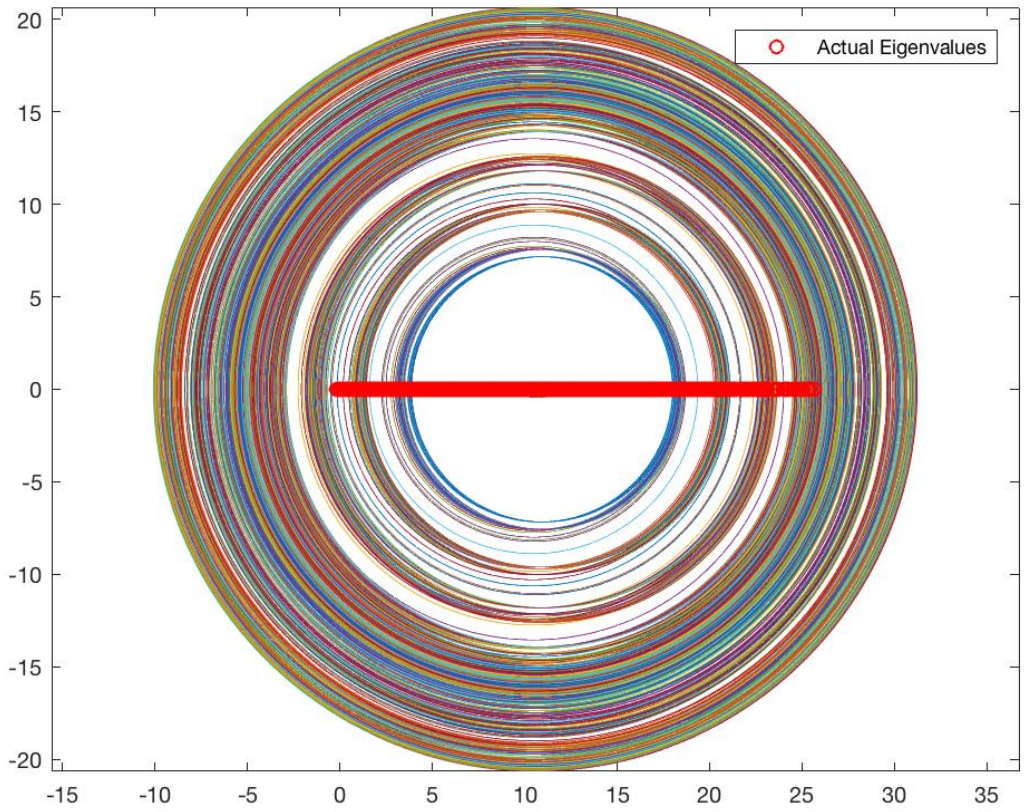


Figure 8: Gershgorin's Disk plot of PARSEC/Na5

We observe that the value of the X-axis at the leftmost span of the disk represents the

minimum eigenvalue bound and the value of the X-axis at the rightmost span of the disk represents the maximum eigenvalue bound. It is important to note that the union of the discs are concentric. This is because all diagonal elements of the matrix are almost identical, so the discs will be centered at almost the same point.

Now we have our bounds, we can then proceed to call the dichotomy process which was explained in pseudo-code **??**. This process continuously contrasts and compares the number of eigenvalues in different ranges till our desired result is obtained. See figures 13, 14, the figures are self explanatory.

```
Min/Max:      -10.0858386363924        31.2209791594811
matrix name ../data/Na5
matrix -coordinate format- size        5832
sparse matrix A- nnz      155731

Routine  dfeast_scsrev
min  -10.0858386363924      max  -4.92248641190819
Iteration count           1 Estimate           1
No eigenvalues found in this range. min = max, max = min + interval/8


min  -4.92248641190819      max -0.404553215484527
Iteration count           2 Estimate           1
No eigenvalues found in this range. min = max, max = min + interval/8


min -0.404553215484527      max   3.54863833138617
Iteration count           3 Estimate         468
Too many eigenvalues found. MAX is reduced by   2.58167611224209


min -0.404553215484527      max  0.966962219144084
Iteration count           4 Estimate          74
```

Figure 9: PARSEC/Na5: output of dichotomy process for extreme minimum values

```
Min/Max:      -10.0858386363924        31.2209791594811
matrix name ../data/Na5
matrix -coordinate format- size        5832
sparse matrix A- nnz      155731

Routine  dfeast_scsrev
min   26.0576269349969      max   31.2209791594811
Iteration count           1 Estimate           1
No eigenvalues found in this range. Max = min, min = max - interval/8


min   21.5396937385732      max   26.0576269349969
Iteration count           2 Estimate         129
Too many eigenvalues found. Min shrinks by   2.58167611224209


min   24.1213698508153      max   26.0576269349969
Iteration count           3 Estimate          24
 Few eigenvalues found. Min widens by   1.29083805612105


min   22.8305317946943      max   26.0576269349969
Iteration count           4 Estimate          68
```

Figure 10: PARSEC/Na5: output of dichotomy process for extreme maximum values

The goal was to find an interval that would give us 50 extreme eigenvalues either towards the minimum range or the maximum range. Our algorithm was successfully able to find a range with a stochastic estimate of 74 eigenvalues in the minimum case and an estimate of 68 eigenvalues in the maximum case. We then proceed to use this derived interval in the already existing FEAST solver to calculate our eigenvalues. The results of our eigenvalues can be seen in figure 11. The results displayed in this figure are those of our extreme minimum eigenvalues, and we observe the error trace and residuals are very minimal. Also the actual number (not stochastic estimates) of λs found is 69, which still meets our requirements of $50 \leq M \leq 100$. If interested in the result of the maximum extreme eigenvalues, please refer to figure 31 in the appendix.

```
==>FEAST has successfully converged (to desired tolerance)

 Eigenvalues/Residuals
 inside interval
           1 -0.163824726797767       7.238046224264992E-014
           2 -8.180518811284673E-002  6.704661469815915E-014
           3 -5.518014222472481E-002  8.565033250983422E-014
           4 -2.116800868554078E-002  1.722131177825223E-013
           5  4.121289015772357E-002  1.678616777695767E-013
           6  6.038410558363289E-002  6.638514424415505E-014
           7  8.825804236287420E-002  7.181797414324989E-014
           8  0.103448473876486       4.099323889848553E-014
           9  0.106022284441572       1.563859968417648E-013
          10  0.148596439686408       5.048819700067570E-014
          11  0.194012353299307       3.864287684541348E-014
          12  0.199435009778215       1.741646273957892E-013
          13  0.221298700102012       1.668484710907468E-013
          14  0.244631929573784       1.686209470037973E-013
          15  0.257679357303821       1.074600845816359E-013
          16  0.273749087220112       4.051299677001908E-014
          17  0.282732873782328       2.203352602654042E-014
          18  0.303148550153459       1.230718967458247E-013
          19  0.340871094771031       1.035162511798143E-013
          20  0.361310226499288       3.199678700927202E-014
          21  0.370182964879676       8.181683579554789E-014
          22  0.382186257687260       8.921905357942662E-014
          23  0.389754956936955       2.117988371905068E-014
          24  0.410713067278826       1.917905978223731E-014
          25  0.424667714932305       2.711675161613623E-014
          26  0.434669016756200       1.142783024761308E-014
          27  0.439125298381731       8.117493112309422E-014
          28  0.467684139719128       6.843584419388020E-014
          29  0.472954198191659       7.200779230172235E-014
          30  0.507310112435584       1.098161077712230E-013
          31  0.530220075937602       1.854012809973438E-014
          32  0.548819860276865       2.075761859507673E-014
          33  0.553184196141978       8.473051738443703E-014
          34  0.565648110791943       6.007512290509869E-014
          35  0.570230197838237       8.666602231020309E-014
          36  0.578835807667586       3.119916031869596E-014
          37  0.597940413887051       3.342949123274546E-014
          38  0.609124143735205       2.506554748000837E-014
          39  0.617866210234637       2.885382749544577E-014
          40  0.626024444292822       1.572767989451566E-013
          41  0.629457454756601       6.414668477228681E-014
          42  0.640785480837755       3.495063288489143E-014
          43  0.647087549532053       2.065413489504130E-014
          44  0.658689287931632       6.014079465593646E-014
          45  0.682432570492133       4.184899365047204E-014
          46  0.684889737190180       7.821247491568217E-014
          47  0.723120261138394       3.309161714127759E-014
          48  0.761676173975782       4.759428973195073E-014
          49  0.765299809584943       1.181155090254839E-014
          50  0.778832924784182       1.013552153912569E-014
          51  0.782587802724746       1.981015516517952E-014
          52  0.788900782075583       4.635167669746820E-014
          53  0.794636043886341       1.461119970745771E-014
          54  0.803422518343232       3.668071691932235E-014
          55  0.804853976042917       1.293706233474439E-014
          56  0.830269825361063       1.627254724700439E-014
          57  0.837906249328670       2.349760738138801E-014
          58  0.846179945358743       9.887123154889194E-015
          59  0.851055395399233       3.030333019749277E-014
          60  0.853906302793252       4.716136811824991E-014
          61  0.861195505372309       1.104207314060269E-014
          62  0.863061581693079       2.845246311642103E-014
          63  0.880427592630678       1.850778920769721E-014
          64  0.890028892014525       1.349040697706939E-014
          65  0.891590349333218       1.315620406643435E-014
          66  0.906023961943617       4.947270620638916E-014
          67  0.912596976246830       2.603636248255950E-014
          68  0.916379050369695       2.507784645524532E-014
          69  0.934041008929978       1.098934718407539E-013
|
 -----------------------------------
 SIMULATION TIME   42.0294000000000
 # Search interval [Emin,Emax]
 -0.404553215484527          0.966962219144084
 Subspace size M0             100
 # eigenvalue found M          69
 # FEAST iterations            5
 Relative error on the Trace    7.348195427832166E-015
 Maximum eigenvector residual   1.741646273957892E-013
```

Figure 11: Minimum extreme eigenvalues of Na5

4.2 Testing with cnt-A-mtx

The cnt_A matrix is a real symmetric matrix belonging to the standardized eigenvalue problem. It's a full matrix of size 12450×12450 with 86808 non-zero elements. Our goal is to find at least 50 and at most 100 extreme eigenvalues either near the minimum or the maximum. Since we are interested in 50λs, we set the the size of our subspace $M_0 = 2 * 50$.

The procedure for Gershgorin's circle theorem is then called which gives us an eigenvalue bound of [-0.1303, 2.0374]. The figure below, 12 is visual representation of the Gershgorin discs. The union of the discs contains every single eigenvalue, λ of the matrix cnt

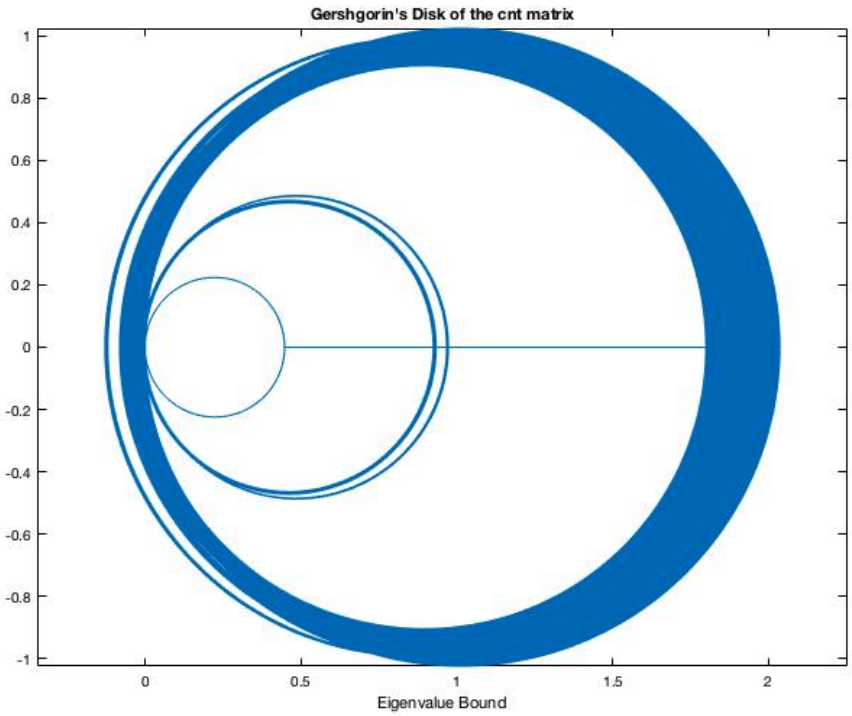


Figure 12: Gershgorin's Disk plot of cnt

We observe that the value of the X-axis at the leftmost span of the disk represents the minimum eigenvalue boundary and the value of the X-axis at the rightmost span of the disk represents the maximum eigenvalue boundary. For the most part, the union of the discs are concentric, but we observe that we have approximately three distinct center. We can conclude that the elements in the diagonals are split into three different sets, and the

elements of the same set tend to have identical values.

Now we have our bounds, we can then proceed to call the dichotomy process. This process continuously contrasts and compares the number of eigenvalues in different ranges till our desired result is obtained. See figures 13, 14, the figures are self explanatory.

```
Min/Max:     -0.130269024372307        2.03738796259222
matrix name ../data/cnt
matrix -coordinate format- size       12450
sparse matrix A- nnz       86808

Routine  dfeast_scsrev
min -0.130269024372307      max  0.140688098998259
Iteration count           1 Estimate         638
Too many eigenvalues found. MAX is reduced by  0.135478561685283


min -0.130269024372307      max  5.209537312975798E-003
Iteration count           2 Estimate          85
```

Figure 13: cnt: output of dichotomy process for extreme minimum values

```
Min/Max:     -0.130269024372307        2.03738796259222
matrix name ../data/cnt
matrix -coordinate format- size       12450
sparse matrix A- nnz       86808

Routine  dfeast_scsrev
min   1.76643083922165      max   2.03738796259222
Iteration count           1 Estimate           1
No eigenvalues found in this range. Max = min, min = max - interval/8


min   1.52934335627241      max   1.76643083922165
Iteration count           2 Estimate           2
 Few eigenvalues found. Min widens by  0.135478561685283


min   1.39386479458712      max   1.76643083922165
Iteration count           3 Estimate         510
Too many eigenvalues found. Min shrinks by  6.773928084264141E-002


min   1.46160407542977      max   1.76643083922165
Iteration count           4 Estimate          17
 Few eigenvalues found. Min widens by  3.386964042132071E-002


min   1.42773443500845      max   1.76643083922165
Iteration count           5 Estimate         153
Too many eigenvalues found. Min shrinks by  1.693482021066035E-002


min   1.44466925521911      max   1.76643083922165
Iteration count           6 Estimate          56
```

Figure 14: cnt: output of dichotomy process for extreme maximum values

The goal was to find an interval that would give us at least 50 and at most 100 extreme

eigenvalues either towards the minimum range or the maximum range. Our algorithm was successfully able to find a range with a stochastic estimate of 85 eigenvalues in the minimum case and an estimate of 56 eigenvalues in the maximum case. We then proceed to use this derived interval in the already existing FEAST solver to calculate our eigenvalues. The results of our eigenvalues can be seen in figure 15. The results displayed in this figure are those of our extreme minimum eigenvalues, and we observe the error trace and residuals are very minimal. Also the actual number (not stochastic estimates) of λs found is 86, which still meets our requirements of $50 \leq M \leq 100$. If interested in the maximum extreme eigenvalues, please refer to figure 33 in the appendix.

```
==>FEAST has successfully converged (to desired tolerance)
 Eigenvalues/Residuals
 inside interval
           1 -2.521085950322027E-002  1.521155772121843E-015
           2 -2.510433087137880E-002  1.957601033378535E-015
           3 -2.472754352732023E-002  2.030793364733282E-015
           4 -2.468077378229265E-002  2.244475843284878E-015
           5 -2.407067891478145E-002  2.021262305770293E-015
           6 -2.400472768871875E-002  1.962565681790223E-015
           7 -2.379644055111148E-002  2.112963587712986E-015
           8 -2.355051387896392E-002  2.014155580128623E-015
           9 -2.344104407434401E-002  2.142106385532545E-015
          10 -2.310557258101560E-002  1.938184840969329E-015
          11 -2.307988415910787E-002  2.065278789898603E-015
          12 -2.285844157179347E-002  2.034094926509314E-015
          13 -2.253743889579447E-002  1.976601743329955E-015
          14 -2.166635191654165E-002  1.970387734993501E-015
          15 -2.150769573882920E-002  2.214770384173688E-015
          16 -2.122728974774196E-002  1.981346610262589E-015
          17 -2.105713096933308E-002  1.979304735787559E-015
          18 -2.064873749431031E-002  1.958699968965993E-015
          19 -2.043567293808152E-002  2.010429267952642E-015
          20 -2.013925989650660E-002  1.960232481733986E-015
          21 -1.999418831105549E-002  2.025290656565478E-015
          22 -1.973977609792900E-002  2.065390748164189E-015
          23 -1.925254477106988E-002  2.052987876167378E-015
          24 -1.917153231296282E-002  1.779348612607204E-015
          25 -1.910543145593301E-002  1.865648440559786E-015
          26 -1.896553300813279E-002  1.817684918317888E-015
          27 -1.144694245451193E-002  1.845339783221964E-015
          28 -1.136232692077306E-002  1.830244367400129E-015
          29 -1.106902198754184E-002  1.836957001470177E-015
          30 -1.089725819962098E-002  1.871662100886324E-015
          31 -9.628946525968039E-003  1.912604591237089E-015
          32 -9.070843142842996E-003  1.902054224022854E-015
          33 -8.899532119330101E-003  1.887567818628122E-015
          34 -7.755222533485115E-003  1.964755441303532E-015
          35 -7.602341971529301E-003  1.968938491266853E-015
          36 -6.579310514479572E-003  1.885159276579951E-015
          37 -6.113498907501434E-003  1.980712511885041E-015
          38 -6.036595898000059E-003  1.812882735608421E-015
          39 -5.864105899815607E-003  1.807350948835312E-015
          40 -5.715904182669890E-003  1.723679422287559E-015
          41 -5.084173540451580E-003  1.780570774459968E-015
          42 -4.725373036495947E-003  1.947708168235919E-015
          43 -4.229131647894544E-003  1.820366019125131E-015
          44 -4.151423284444870E-003  1.938109195328612E-015
          45 -3.740525930278839E-003  1.783490863486341E-015
          46 -3.703210328375268E-003  1.772906903358372E-015
          47 -3.200503283441162E-003  1.772212362840090E-015
          48 -2.210332853339051E-003  1.837767798071347E-015
          49 -2.203914235126485E-003  1.820011289036242E-015
          50 -1.982010662336527E-003  1.805309351095018E-015
          51 -1.960214389403179E-003  1.817305336874571E-015
          52 -1.885054346126764E-003  1.778311893174618E-015
          53 -1.727826385945171E-003  1.766954227271427E-015
          54 -1.711553644625051E-003  1.725507260855370E-015
          55 -1.697391743702964E-003  1.651464602977314E-015
          56 -1.682925029310743E-003  1.721106145658496E-015
          57 -1.298505015448560E-003  1.719070279917559E-015
          58 -1.183411795274077E-003  1.622479319891224E-015
          59 -1.166852623487352E-003  1.677201055993432E-015
          60 -4.029796458842308E-004  1.627798600114956E-015
          61 -3.992249773733408E-004  1.730123774082912E-015
          62 -2.488705272446697E-004  1.641651158252249E-015
          63 -2.359686329867819E-004  1.669642766780521E-015
          64 -9.198854963661678E-005  1.693646642797239E-015
          65 -9.027395724578277E-005  1.591521887500262E-015
          66  3.801605921830975E-004  1.588554761426089E-015
          67  4.153150880014056E-004  1.761912949630215E-015
          68  4.746759427797469E-004  1.563899140354752E-015
          69  1.277088109824626E-003  1.723764385686012E-015
          70  1.529133730118874E-003  1.741219818177205E-015
          71  1.562376397554244E-003  1.562925396615360E-015
          72  1.613408489382139E-003  1.673252945872021E-015
          73  1.704164088028846E-003  1.553648269034157E-015
          74  2.086496945938690E-003  2.622964396239235E-015
          75  2.090550122419926E-003  2.536413691834055E-015
          76  2.468703669011605E-003  1.843389269324675E-015
          77  2.492572993106594E-003  2.426984197438740E-015
          78  2.651481641185094E-003  4.956182488300104E-015
          79  2.662104506528628E-003  1.036040179404116E-014
          80  3.789846021495842E-003  9.127939005873519E-014
          81  3.846671453195231E-003  4.132981885684994E-014
          82  3.859018329856451E-003  2.048427322773048E-013
          83  3.918621452642746E-003  6.408221990821533E-014
          84  4.776617342597260E-003  9.085065067964992E-013
          85  4.822123323369915E-003  7.702802742833724E-013
          86  4.835704932201517E-003  7.842202655881635E-013
```

Figure 15: Minimum extreme eigenvalues of cnt-A

4.3 Testing with SiNa

The SiNa matrix is a real symmetric matrix belonging to the standardized eigenvalue problem.It's of size 5743×5743 with 102265 non-zero elements. It's parsed in as a lower half triangular matrix into FEAST, and our goal is to find at least 50 and at most 100 extreme eigenvalues either near the minimum or the maximum.

The procedure for Gershgorin's circle theorem is then called which gives us an eigenvalue bound of [-11.076, 31.037]. The figure below, 16 is visual representation of the Gershgorin discs. The union of the discs contains every single eigenvalue, λ of the matrix Na5

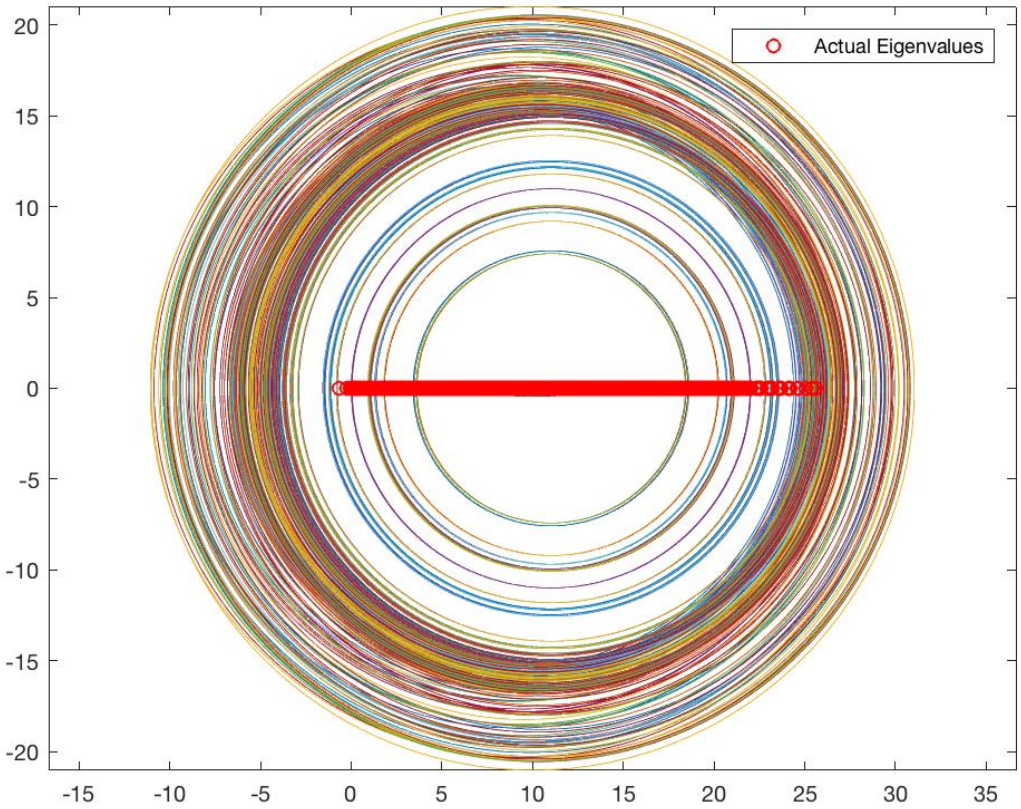


Figure 16: Gershgorin's Disk plot of PARSEC/SiNa

Now we have our bounds from Gershgorin, we can then proceed to call the dichotomy process. This process continuously contrasts and compares the number of eigenvalues in

different ranges till our desired result is obtained. See figrues 17, 18. The figures are self explanatory.

```
Min/Max:      -11.0763732978297        31.0368976972103
matrix name ../data/SiNa
matrix -coordinate format- size        5743
sparse matrix A- nnz      102265

Routine  dfeast_scsrev
min  -11.0763732978297      max  -5.81221442344974
Iteration count           1 Estimate           1
No eigenvalues found in this range. min = max, max = min + interval/8

min  -5.81221442344974      max  -1.20607540836724
Iteration count           2 Estimate           1
No eigenvalues found in this range. min = max, max = min + interval/8


min  -1.20607540836724      max   2.82429622982995
Iteration count           3 Estimate         321
Too many eigenvalues found. MAX is reduced by   2.63207943719000


min  -1.20607540836724      max  0.192216792639946
Iteration count           4 Estimate          13
Few eigenvalues found. MAX is increased by   1.31603971859500


min  -1.20607540836724      max   1.50825651123495
Iteration count           5 Estimate         128
Too many eigenvalues found. MAX is reduced by  0.658019859297500


min  -1.20607540836724      max  0.850236651937446
Iteration count           6 Estimate          57
```

Figure 17: SiNa: output of dichotomy process for extreme minimum values

```
Min/Max:      -11.0763732978297        31.0368976972103
matrix name ../data/SiNa
matrix -coordinate format- size        5743
sparse matrix A- nnz      102265

Routine  dfeast_scsrev
min   25.7727388228303      max   31.0368976972103
Iteration count           1 Estimate           1
No eigenvalues found in this range. Max = min, min = max - interval/8


min   21.1665998077478      max   25.7727388228303
Iteration count           2 Estimate         153
Too many eigenvalues found. Min shrinks by   2.63207943719000


min   23.7986792449378      max   25.7727388228303
Iteration count           3 Estimate          27
 Few eigenvalues found. Min widens by   1.31603971859500


min   22.4826395263428      max   25.7727388228303
Iteration count           4 Estimate          79
```

Figure 18: SiNa: output of dichotomy process for extreme maximum values

The goal was to find an interval that would give us 50 extreme eigenvalues either towards the minimum range or the maximum range. Our algorithm was successfully able to find a

range with a stochastic estimate of 57 eigenvalues in the minimum case and an estimate of 79 eigenvalues in the maximum case. We then proceed to use this derived interval in the already existing FEAST solver to calculate our eigenvalues. The results of our eigenvalues can be seen in figure 19. The results displayed in this figure are those of our extreme minimum eigenvalues, and the actual number (not stochastic estimates) of λs found is 56, which still meets our requirements of $50 \leq M \leq 100$. If interested in the maximum extreme eigenvalues, please refer to figure 32 in the appendix.

```
==>FEAST has successfully converged (to desired tolerance)

 Eigenvalues/Residuals
 inside interval
           1 -0.704184900484430       2.988758618846363E-014
           2 -0.263428711009114       1.466403318867289E-014
           3 -0.263070939003140       7.895621856833619E-015
           4 -0.225954645941191       8.173178655344279E-015
           5 -5.848623243714584E-002  6.285580538272144E-015
           6  5.051281591220180E-002  1.037613014687757E-014
           7  5.194751438500918E-002  1.000476826830591E-014
           8  8.593045566308793E-002  1.075386522902785E-014
           9  0.135590775474212       8.837313772573377E-015
          10  0.135790131392778       1.480439334500715E-014
          11  0.190231900952121       4.692479399460203E-015
          12  0.191082383259333       7.666387740877520E-015
          13  0.230880197411109       7.745949441084135E-015
          14  0.286085511613849       5.300985632507267E-015
          15  0.296801348483694       6.886918850037224E-015
          16  0.297630231916910       4.483502863330373E-015
          17  0.320869286442495       4.942139913048023E-015
          18  0.321228618629325       5.255107466301472E-015
          19  0.341626963364066       9.604832437808515E-015
          20  0.345843143851968       6.936651656485591E-015
          21  0.346274873538316       1.556204294804744E-014
          22  0.419435032882199       6.072599749278747E-015
          23  0.419846726389107       1.671619992515285E-014
          24  0.471633513795705       9.164583475514141E-015
          25  0.472271521427933       8.052193101618188E-015
          26  0.503185927639462       8.072914404751863E-015
          27  0.514683694661787       4.404390623214074E-015
          28  0.515791001007463       6.913401806390105E-015
          29  0.520619247338093       7.310897998287298E-015
          30  0.521556243809887       4.505391728683145E-015
          31  0.522205508614319       9.202001835441394E-015
          32  0.528309588764210       4.243659654747226E-015
          33  0.530060235279199       8.233411747192391E-015
          34  0.540122813696298       7.830827837269378E-015
          35  0.541663662565180       6.365844706074316E-015
          36  0.651468650270671       6.578431312215562E-015
          37  0.654525870354313       1.021335852591474E-014
          38  0.699394849029306       5.481586908060730E-015
          39  0.721885498143530       6.744875397423908E-015
          40  0.722708157941928       8.079045047651432E-015
          41  0.729593085900675       5.151525405655201E-015
          42  0.730133738018778       5.053143050694928E-015
          43  0.733117235787937       7.661111932612972E-015
          44  0.734494234383293       4.627655629783434E-015
          45  0.736825209482988       5.024611149855481E-015
          46  0.736883214681542       4.613586626370275E-015
          47  0.738286001402677       5.051273661010516E-015
          48  0.741496146795905       8.301465985404445E-015
          49  0.742797931942067       5.328106153080231E-015
          50  0.749912830420996       4.471508242928041E-015
          51  0.750869837701049       4.888926485729050E-015
          52  0.765406637694202       8.621897577150262E-015
          53  0.765937123535505       4.457556556649390E-015
          54  0.830720049621142       8.969057262081055E-014
          55  0.835036998636214       1.419334342368194E-013
          56  0.836605989983143       2.002451908847768E-013
```

Figure 19: Minimum extreme eigenvalues of SiNa

4.4 Testing with Si5H12

The Ga19As19H42 matrix is a real symmetric matrix belonging to the standardized eigenvalue problem.It's of size 19896×19896 with $738,598$ non-zero elements. It's parsed in as a lower half triangular matrix into FEAST, and our goal is to find at least 50 and at most

100 extreme eigenvalues either near the minimum or the maximum.

The procedure for Gershgorin's circle theorem is then called which gives us an eigenvalue bound of [-19.724, 64.276]. The figure below, 20 is visual representation of the Gershgorin discs. The union of the discs contains every single eigenvalue, λ of the matrix H2O

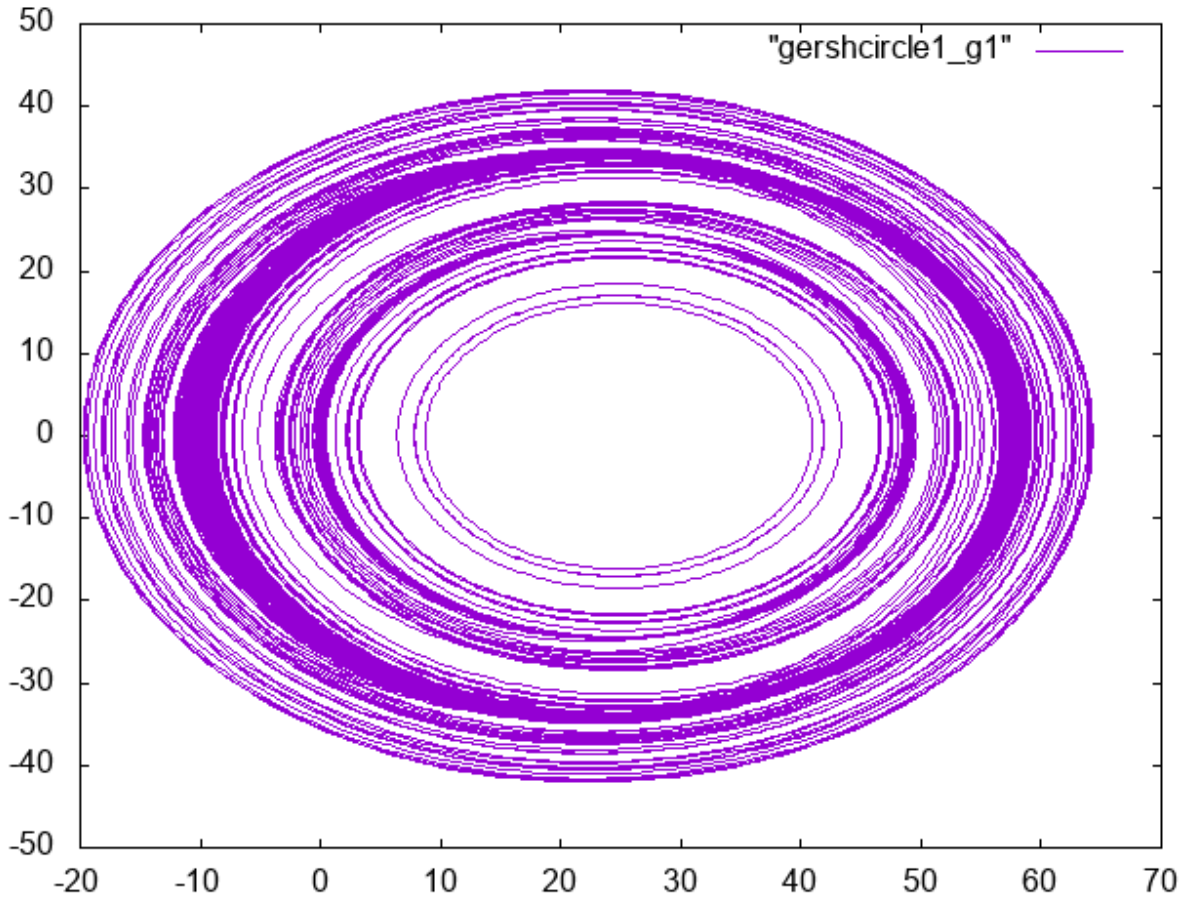


Figure 20: Gershgorin's Disk plot of PARSEC/Si5H12

Now we have our bounds from Gershgorin, we can then proceed to call the dichotomy process. This process continuously contrasts and compares the number of eigenvalues in different ranges till our desired result is obtained. See figures 21, 22. The figures are self explanatory.

```
min  -19.7240970585642      max  -9.22413402006686
Iteration count           1 Estimate           1
No eigenvalues found in this range. min = max, max = min + interval/8


min  -9.22413402006686      max -3.666636138164492E-002
Iteration count           2 Estimate           8
Few eigenvalues found. MAX is increased by   5.24998151924869


min  -9.22413402006686      max   5.21331515786705
Iteration count           3 Estimate         842
Too many eigenvalues found. MAX is reduced by   2.62499075962435


min  -9.22413402006686      max   2.58832439824270
Iteration count           4 Estimate         288
Too many eigenvalues found. MAX is reduced by   1.31249537981217


min  -9.22413402006686      max   1.27582901843053
Iteration count           5 Estimate         104
Too many eigenvalues found. MAX is reduced by  0.656247689906087


min  -9.22413402006686      max  0.619581328524442
Iteration count           6 Estimate          41
Few eigenvalues found. MAX is increased by  0.328123844953043


min  -9.22413402006686      max  0.947705173477485
Iteration count           7 Estimate          70
```

Figure 21: Si5H12: output of dichotomy process for extreme minimum values

```
min   53.7756442109175      max   64.2756072494149
Iteration count           1 Estimate         135
Too many eigenvalues found. Min shrinks by   5.24998151924869


min   59.0256257301662      max   64.2756072494149
Iteration count           2 Estimate           1
No eigenvalues found in this range. Max = min, min = max - interval/8


min   49.1819103815749      max   59.0256257301662
Iteration count           3 Estimate         435
Too many eigenvalues found. Min shrinks by   5.24998151924869


min   54.4318919008236      max   59.0256257301662
Iteration count           4 Estimate         107
Too many eigenvalues found. Min shrinks by   2.62499075962435


min   57.0568826604479      max   59.0256257301662
Iteration count           5 Estimate          22
 Few eigenvalues found. Min widens by   1.31249537981217


min   55.7443872806357      max   59.0256257301662
Iteration count           6 Estimate          56
```

Figure 22: Si5H12: output of dichotomy process for extreme maximum values

The goal was to find an interval that would give us 50 extreme eigenvalues either towards the minimum range or the maximum range. Our algorithm was successfully able to find a range with a stochastic estimate of 70 eigenvalues in the minimum case and an estimate of

56 eigenvalues in the maximum case. We then proceed to use this derived interval in the already existing FEAST solver to calculate our eigenvalues. For this specific matrix we set the number of refinement loops to 40, so as to let the sovler converge. The results of our eigenvalues can be seen in figure 23. The results displayed in this figure are those of our extreme minimum eigenvalues, and the actual number (not stochastic estimates) of λs found is 66, which still meets our requirements of $50 \leq M \leq 100$. If interested in the maximum extreme eigenvalues, please refer to figure 30 in the appendix.

```
==>FEAST has successfully converged (to desired tolerance)

 Eigenvalues/Residuals
 inside interval
           1 -0.996202324448980       1.336938049030517E-015
           2 -0.672220819590052       1.403366082109146E-015
           3 -0.628503768294800       1.470290474629105E-015
           4 -0.537692380039058       1.377645848270502E-015
           5 -0.104973266756380       1.372119870649435E-015
           6 -8.033737611634539E-003  1.295500677968698E-015
           7  1.568395231592195E-002  1.328358187311217E-015
           8  5.033833211756254E-002  1.317818971731918E-015
           9  8.364543395781575E-002  1.378416029683895E-015
          10  0.102114900263868       1.332572346933208E-015
          11  0.127640643714062       1.336648090782653E-015
          12  0.178435042495262       1.307623055930827E-015
          13  0.208566782499015       1.331368544161578E-015
          14  0.223789005619333       1.360591186620318E-015
          15  0.231687825294898       1.372786198722916E-015
          16  0.263930784329014       1.320030382617537E-015
          17  0.293323422020135       1.322291453750294E-015
          18  0.294863658503095       1.331814956277288E-015
          19  0.308812225936876       1.318531319652362E-015
          20  0.316164877145017       1.342764142633583E-015
          21  0.367050796947738       1.333701531837968E-015
          22  0.377782905247770       1.361913870469620E-015
          23  0.388963525419412       1.313032987700513E-015
          24  0.392593620501861       1.336136448402020E-015
          25  0.414381201398059       1.320401771175195E-015
          26  0.424220731931287       1.267636201117979E-015
          27  0.485172801419376       1.353092697232817E-015
          28  0.487953578513044       1.352196166454476E-015
          29  0.491241301321474       1.322539405924708E-015
          30  0.491855963657889       1.295539042777992E-015
          31  0.494401722127745       1.309800462793164E-015
          32  0.521242721194347       1.310793933445493E-015
          33  0.528339374670536       1.305100413948209E-015
          34  0.562419854943423       1.300378783648094E-015
          35  0.565896228988084       1.289515880524705E-015
          36  0.596148745781847       1.310262480428527E-015
          37  0.600361333114963       1.302696227431610E-015
          38  0.600602170645608       1.291875737917501E-015
          39  0.604049241229403       1.320813699790690E-015
          40  0.604156065565113       1.290383718540346E-015
          41  0.639216850365505       1.310711742552648E-015
          42  0.647897538333152       1.338902451463324E-015
          43  0.655616166428509       1.240506517347415E-015
          44  0.715628307991244       1.249206455784495E-015
          45  0.718867498514981       1.279122612112489E-015
          46  0.719237705170801       1.253725996854447E-015
          47  0.721632749055621       1.211337255344207E-015
          48  0.722051612887944       1.215792241049664E-015
          49  0.722229334379938       1.235347241384828E-015
          50  0.736666775416715       1.261971698193620E-015
          51  0.753953330144881       1.257764563409252E-015
          52  0.781176766732438       1.282181493204781E-015
          53  0.792789467160355       1.709554612515818E-015
          54  0.798349718106158       1.342427853981991E-015
          55  0.811494221505373       2.061865938188186E-015
          56  0.827201975775245       1.513940522057849E-015
          57  0.845993332349559       1.727556355093685E-015
          58  0.846696713853073       2.412095162787419E-015
          59  0.848329298355849       4.484339535321439E-015
          60  0.848768579253530       5.061715537178550E-015
          61  0.849346142596299       5.136693153823252E-015
          62  0.856261369992386       3.038555780449378E-015
          63  0.865831154153907       6.897831487160158E-015
          64  0.917776234632598       5.559490441464636E-014
          65  0.932192414433030       1.046469740085183E-013
          66  0.944117209728810       3.764940431272032E-013
```

Figure 23: Minimum extreme eigenvalues of Si5H12

The graphs displayed below represent the dichotomy process used to specify an interval for 50 extreme eigenvalues of the different matrices. The graphs on the left represent the

process for minimum extreme eigenvalues spectrum, while the ones on the right represent the maximum extreme eigenvalues spectrum. The blue line represents the maximum spectrum at each step while the red line represents the minimum spectrum at each step. The value in parentheses represent the number of stochastically estimated eigenvalues in that bound.

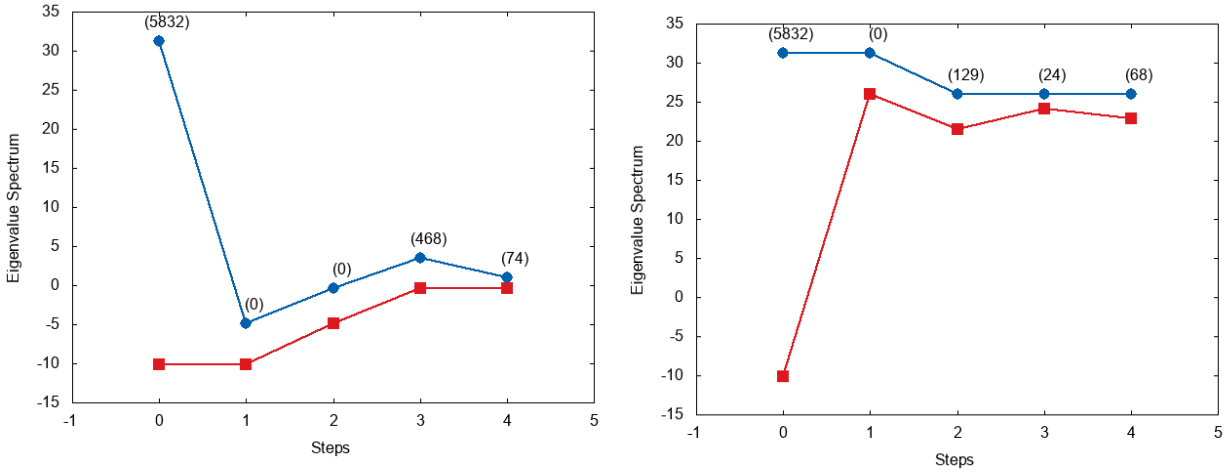


Figure 24: Na5

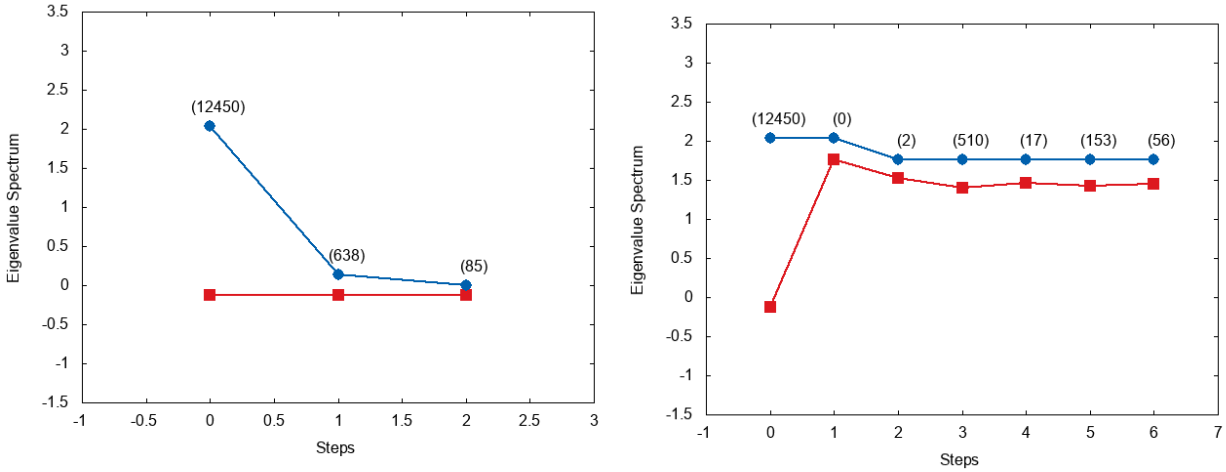
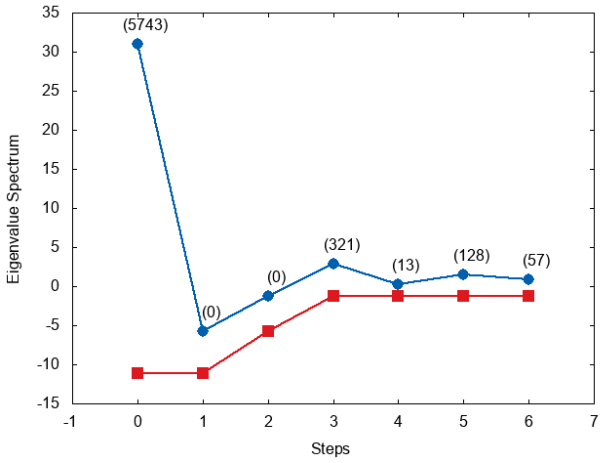


Figure 25: cnt



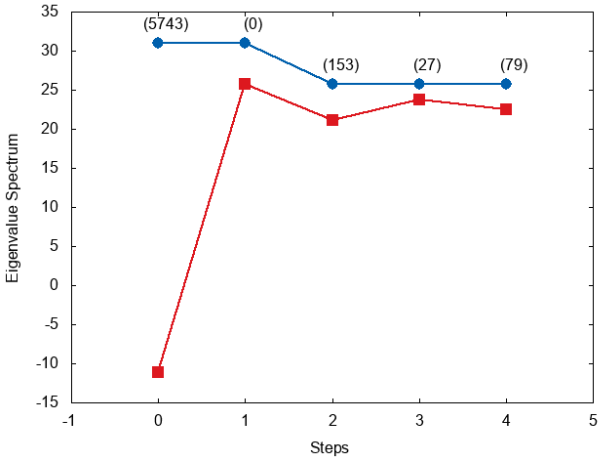
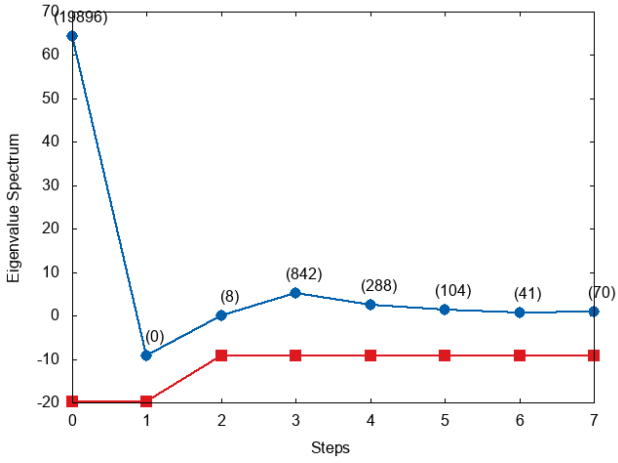


Figure 26: SiNa



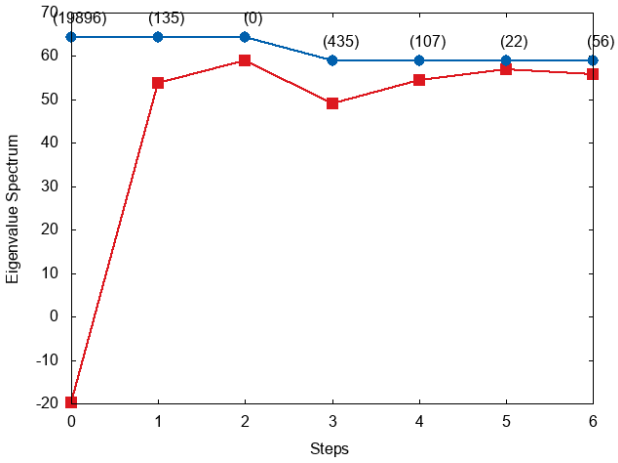


Figure 27: Si5H12

5 Summary

To summarize it all, section 2 focused on defining the terminologies and also looking into papers that helped us to better understand of the Eigenvalue problem and the FEAST framework. In section 3, we discussed the approaches used to find an efficient estimation of eigenvalue intervals and their corresponding eigenvalues for the standardized eigenvalue problem, $Ax = \lambda x$. We employed a combination Gershgorin's circle theorem and FEAST stochastic techniques to estimate our intervals. And section 4 analyzed the results from our

methodology by evaluating the efficiency and accuracy of our algorithm.

In conclusion, we successfully developed a tool to find the search interval of the M wanted lowest or largest eigenvalues. Gershgorin's circle theorem and the dichotomy process can now be used in the FEAST framework.We can now provide an optimal input search interval for FEAST, which can then be used to solve standard eigenvalue problems in the framework. Lastly, this tool can further be extended to work with the generalized eigenvalue problem.

6 Appendix

6.1 Artifact - Code

Figure 28 and 29 represent the FOTRAN-90 program written to automate Gersgorin's circle theorem and the dichotomy process. Gershgorin's theorem is used to bound the spectrum of a square matrix. With the boundary obtained from Gershgorin, the dichotomy process is used to continuously compare and contrast the number of eigenvalues in these different sub-ranges of the boundary until we obtain the desired number of λs needed. Refer back to Section 3 for a more detailed explanation.

```
subroutine dgershgorin1(UPLO,n,nnza,isa,jsa,dsa,dEmin,dEmax)
    !!! The array dsa is of length NNZ and holds all-
    !!! UPLO represents the shape of the matrix (whether full or triangular)
    !!! -the nonzero entries of the matrix(M) in left-to-right top-to-bottom ("row-major") order.
    !!! The array isa is of length n + 1. It's what enables us to iterate the CSR
    !!! jsa, contains the column index in M of each element of dsa and hence is of length NNZ as well.
    !!! dEmin and dEmax represent the variables that store the bound when gersh is complete

    implicit none
    character(len=1) :: UPLO
    integer :: n,nnza
    integer,dimension(*) :: isa,jsa
    double precision :: dEmin, dEmax
    double precision,dimension(*) :: dsa
    double precision, dimension(:), allocatable :: radius, center
    !!!
    integer :: i,index
    allocate(radius(n))
    allocate(center(n))

    radius = 0d0
    do i = 1, n
        do index = isa(i),isa(i+1)-1
            if (UPLO == 'F') then
              if (i/=jsa(index)) radius(i) = radius(i) + ABS(dsa(index))
              if (i == jsa(index)) center(i) = dsa(index)
            else
              if (i/=jsa(index)) then
                radius(i) = radius(i) + ABS(dsa(index))
                radius(jsa(index)) = radius(jsa(index)) + ABS(dsa(index))
              endif
              if (i == jsa(index)) center(i) = dsa(index)
            endif
        enddo
    enddo
    dEmin = huge(dEmin)
    dEmax = -dEmin
    do i = 1,n
        if ((center(i) - radius(i)) < dEmin) then
          dEmin = center(i) - radius(i)
        endif
        if ((center(i) + radius(i)) > dEmax) then
          dEmax = center(i) + radius(i)
        endif
    enddo
    print *,'Min/Max:    ',dEmin,dEmax
    deallocate(radius,center)
end subroutine dgershgorin1
```

Figure 28: FOTRAN code that implements Gershgorin's circle theorem

```
if  ((PRE=='d').and.(EG=='e')) then
   print *,'Routine  ','dfeast_scsrev'
   allocate(dE(1:M0))      ! Eigenvalue
   allocate(dres(1:M0))    ! Residual
   allocate(dX(1:n,1:M0))
   allocate(track(1:3))
   dEmax0 = dEmax
   dEmin0 = dEmin
   M1 = M0/2 ! number of eigenvalues
   i = 3
   ms = 10
   if (fpm(56) == 0) then
     dEmax = (dEmax - dEmin)/(2d0**i) + dEmin
   else
     dEmin = dEmax - (dEmax - dEmin)/(2d0**i)
   endif
   call dfeast_scsrev(UPLO,N,dsa,isa,jsa,fpm,depsout,loop,dEmin,dEmax,ms,dE,dX,M,dres,info)
   iterations = 1
   track(1) = M
   print *, 'min', dEmin, 'max', dEmax
   print *, 'Iteration count', iterations, 'Estimate', M

   do while ((M < (M0/2 + 0.05*M0)).or.(M > (M0 * 3/4)))
     i = i+1
     ddiff = (dEmax0-dEmin0)/(2d0**i)
     if (M == 1) then
       i =3
       if (fpm(56) == 0) then
         dEmin = dEmax
          dEmax = (dEmax0 - dEmin)/(2d0**i) + dEmin
         print *, '---min = max'
       else
         dEmax = dEmin
         dEmin = dEmax - (dEmax - dEmin0)/(2d0**i)
         print *, '---min = max'
      endif
     else if (M > M0 * 3/4) then
       if(fpm(56) == 0) then
         dEmax = dEmax - ddiff
         print *,'---MAX is reduced by', ddiff
       else
         dEmin = dEmin + ddiff
         print *,'---Min shirnks by' ,ddiff
       endif
     else if (M < (M0/2 + 0.05*M0)) then
       if (M < 0.2*M0/2) then
       endif
       if (fpm(56) == 0) then
         dEmax = dEmax + ddiff
         print *, '---MAX is increased by', ddiff
       else
         dEmin = dEmin - ddiff
         print *,'---Min widens by', ddiff
       endif
     endif
     call dfeast_scsrev(UPLO,N,dsa,isa,jsa,fpm,depsout,loop,dEmin,dEmax,ms,dE,dX,M,dres,info)
     iterations = iterations + 1
     print *, 'min', dEmin, 'max', dEmax
     print *, 'Iteration count', iterations, 'Estimate', M
   enddo

   fpm(1)=1
   fpm(4)=20
   fpm(14)=0
   call dfeast_scsrev(UPLO,N,dsa,isa,jsa,fpm,depsout,loop,dEmin,dEmax,M0,dE,dX,M,dres,info)
   print *, 'Number of iterations' , iterations
```

Figure 29: FORTRAN code that implements the Dichotomy process

6.2 More results from Testing

This subsection has the results of the maximum extreme eigenvalues obtained from our test matrices in section 5

```
==>FEAST has successfully converged (to desired tolerance)

 Eigenvalues/Residuals
 inside interval
           1   55.7558215367701       9.692445669512748E-014
           2   55.7642326728044       7.678222638600424E-014
           3   55.7999673167034       2.804368500401478E-014
           4   55.8029561983058       2.171116440715020E-014
           5   55.8633400296487       1.721265073617175E-014
           6   55.9374859120795       7.501059814215153E-015
           7   55.9439973054795       1.187143594665083E-014
           8   55.9649706597652       9.869562111904075E-015
           9   55.9949625919111       5.624126500464532E-015
          10   55.9998313460894       4.959272868483522E-015
          11   56.0804000431033       9.027442235893894E-015
          12   56.1361615487303       7.153180911350564E-015
          13   56.1415520040589       7.780457903598719E-015
          14   56.1580458643386       7.668496028155610E-015
          15   56.1612038994729       9.223839916917795E-015
          16   56.1870940297552       1.025521340400362E-014
          17   56.2875885637390       1.034186682375452E-014
          18   56.3183998720350       1.024536673343759E-014
          19   56.3553420811112       8.127417694472286E-015
          20   56.3553557817466       7.184408784162405E-015
          21   56.3921619236754       1.123388425550582E-014
          22   56.5184729517354       1.204375792839577E-014
          23   56.5216047939666       1.069607577503505E-014
          24   56.5347440978626       8.432799938413289E-015
          25   56.5553805989270       1.380513961310104E-014
          26   56.5575933344592       9.749091260928826E-015
          27   56.6408943719624       1.310106796419324E-014
          28   56.7226959206690       1.039928843242594E-014
          29   56.7474282074482       1.107860287317588E-014
          30   56.7697264658029       1.125233212776423E-014
          31   56.8724334586320       1.664444664297239E-014
          32   56.8785840642665       1.003047212106153E-014
          33   56.8869686439105       1.493526286742599E-014
          34   56.9019519960441       1.149289025999820E-014
          35   57.0310975424887       1.148334519293657E-014
          36   57.0904378168547       7.854942263129619E-015
          37   57.1025104943915       1.294278900424071E-014
          38   57.2054805127747       1.378237461451294E-014
          39   57.2113949076646       1.634043312521236E-014
          40   57.2266028981339       1.073516192812221E-014
          41   57.2291754279321       1.153917851478629E-014
          42   57.3720115298508       1.447573829166976E-014
          43   57.4046304313718       1.653034788054057E-014
          44   57.5118177496685       1.500445160172526E-014
          45   57.5147792852398       1.712485081290605E-014
          46   57.5275319248999       1.482312488504654E-014
          47   57.6254684310398       1.915922699306026E-014
          48   57.7881513231582       1.821470079146201E-014
          49   57.7972919399719       1.782376968122100E-014
          50   57.8049264298960       1.374011145040103E-014
          51   57.8868111675242       1.144871540888115E-014
          52   58.0362744268783       1.605849459856741E-014
          53   58.0463083877430       2.216245455705993E-014
          54   58.2479968855932       2.324306111281647E-014
          55   58.2623444774359       1.608664769751519E-014
          56   58.4205537520298       1.342761748857964E-014
          57   58.5609420018651       1.812622260707028E-014
```

Figure 30: Maximum extreme eigenvalues of Si5H12

```
==>FEAST has successfully converged (to desired tolerance)

 Eigenvalues/Residuals
 inside interval
           1   22.8909269301840       1.543037077259070E-013
           2   22.9150018695189       4.215294917058801E-014
           3   22.9173575471062       4.533155948407508E-014
           4   22.9355237278415       1.984028892437206E-014
           5   22.9537331861223       3.396720472918484E-014
           6   22.9636607342990       1.410669972616218E-014
           7   22.9772674309914       2.029543184917802E-014
           8   22.9931540819788       1.095311395058170E-014
           9   23.0031784173295       1.098354535942945E-014
          10   23.0118215429772       1.148034155084849E-014
          11   23.0292160960403       9.530153503535874E-015
          12   23.0474423115437       1.261463848017133E-014
          13   23.0638675809722       3.097440961871352E-014
          14   23.0726249807013       8.722355550210310E-015
          15   23.0797203226832       8.597760047334987E-015
          16   23.0923885066817       1.047399497211082E-014
          17   23.0930252738206       7.782703155899840E-015
          18   23.1081784216742       1.076463706939384E-014
          19   23.1333620829671       1.467722989816556E-014
          20   23.1354078308828       7.645582164653692E-015
          21   23.4530597724387       2.236442145454486E-014
          22   23.4720191854323       1.930304559834326E-014
          23   23.4920174086555       9.301846274173578E-015
          24   23.5027190013966       2.605383404321692E-014
          25   23.5273135536373       1.307726104895658E-014
          26   23.5375695791101       1.167698984855554E-014
          27   23.5692697990255       2.455343687512928E-014
          28   23.5815913637518       1.427789546012000E-014
          29   23.5878908587268       1.392521181812401E-014
          30   23.6046735379557       2.210289009154013E-014
          31   23.6115673499592       2.532042854270239E-014
          32   23.6218192770720       1.244419033494539E-014
          33   23.6290589924983       1.032275020645965E-014
          34   23.6395447261039       2.306471899400597E-014
          35   23.6438227517674       2.100162946882916E-014
          36   23.6835336152871       7.690718169752883E-015
          37   23.7364730012792       9.802274869812495E-015
          38   23.9961877116744       2.381972146017538E-014
          39   24.0043465309026       3.475340363781365E-014
          40   24.0360913168477       1.649093813794522E-014
          41   24.0729502773118       2.650845727485503E-014
          42   24.0819101036544       2.335169987193351E-014
          43   24.1091271444467       2.632520894636257E-014
          44   24.1146373089887       5.007519387111418E-014
          45   24.1346660088138       2.812744363425571E-014
          46   24.1469249763585       2.892159304551747E-014
          47   24.2034094756825       2.314060005336341E-014
          48   24.2365974236412       4.492785529033829E-014
          49   24.2871937852009       2.219498862980043E-014
          50   24.4847306768003       2.251453961075864E-014
          51   24.5040366735282       5.697674820304240E-014
          52   24.5314433498810       3.674226304138295E-014
          53   24.5637684956328       4.875392509334710E-014
          54   24.5832889174507       1.814372237972889E-014
          55   24.6019512299973       2.077295549910946E-014
          56   24.6113766785276       2.236332771728699E-014
          57   24.7937336500557       4.563552414483061E-014
          58   24.9217570979499       4.001257463942602E-014
          59   24.9653839005360       6.977088376070414E-014
          60   24.9804754978643       3.220188701609983E-014
          61   25.0084654838420       4.494135800299326E-014
          62   25.0413059210886       1.143665274178254E-014
          63   25.3064019648368       6.117384699725976E-014
          64   25.3487899346925       2.367164263892067E-014
          65   25.4027912173089       1.285687693229977E-013
          66   25.6604021643452       8.347581534843290E-014
 -------------------------------------
 SIMULATION TIME   35.5947000000000
 # Search interval [Emin,Emax]
   22.8305317946943        26.0576269349969
 Subspace size M0             100
 # eigenvalue found M          66
 # FEAST iterations            4
 Relative error on the Trace    8.725801317611774E-015
 Maximum eigenvector residual   1.543037077259070E-013
```

Figure 31: Maximum extreme eigenvalues of Na5

```
==>FEAST has successfully converged (to desired tolerance)

 Eigenvalues/Residuals
 inside interval
           1   22.4928606071593       7.033305764610305E-013
           2   22.4932345379568       4.273072985951200E-013
           3   22.5310235536158       1.586461676719354E-013
           4   22.5626644022628       4.071477528850318E-014
           5   22.5645481818245       1.510863442520712E-013
           6   22.5689685836064       6.447763351624758E-014
           7   22.5717795287930       1.191941959038270E-013
           8   22.5731297241973       6.072620237234014E-014
           9   22.5983659237471       2.100283149766658E-014
          10   22.6043270802450       2.455582982967743E-014
          11   22.6072049435915       8.573166038245522E-015
          12   22.9114702547908       2.560911912120583E-014
          13   22.9841276407417       1.108955224872632E-014
          14   23.0088624621637       3.086385578300241E-014
          15   23.0265101548146       4.874916287902735E-014
          16   23.0399699540593       2.772481031750234E-014
          17   23.0421020668533       1.177458883907865E-014
          18   23.0422653238441       2.493166040926699E-014
          19   23.0459227850640       4.861004901026596E-014
          20   23.0463845953019       1.135713349534263E-014
          21   23.0516784499617       4.236834466249742E-014
          22   23.0532190376137       2.441242541147432E-014
          23   23.0671767690029       1.920811155104512E-014
          24   23.0810747314356       1.993107741610433E-014
          25   23.0891623431158       1.846072947788551E-014
          26   23.1142431176860       7.379937412642825E-014
          27   23.1303341992019       1.738058493960387E-014
          28   23.1341244376544       1.762425207888464E-014
          29   23.1400587024145       6.840922224540137E-014
          30   23.1470975620742       8.884835703429863E-015
          31   23.1529287030266       5.394534996399077E-014
          32   23.2042607947960       3.691972258408555E-014
          33   23.5375995101908       4.224749202214392E-014
          34   23.5558094178406       4.051218711645990E-014
          35   23.5992901089292       2.625897605750518E-014
          36   23.6004417212954       4.182463155174516E-014
          37   23.6011184834245       4.386436558051644E-014
          38   23.6021298273983       2.411017064314051E-014
          39   23.6095459959762       5.463283310277944E-014
          40   23.6104965881579       2.386779846249369E-014
          41   23.6317122572429       1.456057172019602E-014
          42   23.6545671326071       3.690995035152015E-014
          43   23.6558771374416       1.944277276415070E-014
          44   23.6583147911659       2.391982586926154E-014
          45   23.6741882761929       2.866185961121218E-014
          46   23.6763086991774       2.434007742420410E-014
          47   23.6808968959085       5.011856454296128E-014
          48   23.6828495640961       2.978826289402349E-014
          49   24.0665046542166       1.633233239334731E-014
          50   24.1226185360930       3.706437656086068E-014
          51   24.1291706230795       7.316352869721090E-014
          52   24.1303590963426       2.710991117893895E-014
          53   24.1503008310090       5.639174733416510E-014
          54   24.1523759261145       5.468754868605256E-014
          55   24.1610723069454       6.241678466133880E-014
          56   24.1660077041693       6.031172381882546E-014
          57   24.1679035247447       5.140651127922229E-014
          58   24.1709311231031       8.394288331913258E-014
          59   24.1754407676346       9.277548683500427E-014
          60   24.1926147966530       5.198639937465488E-014
          61   24.5604699444104       1.028509502262480E-013
          62   24.6015377504349       2.895698017940783E-014
          63   24.6202621422763       6.577047609397453E-014
          64   24.6276997551049       4.363132302262485E-014
          65   24.6324781003702       2.264166261526348E-014
          66   24.6407539147439       3.787444374105594E-014
          67   24.6493384598045       7.211023586993803E-014
          68   24.6520155248034       1.937879677132462E-014
          69   25.0300444744151       8.364011630420913E-014
          70   25.0353401102795       1.299531772904279E-013
          71   25.0503480244986       9.687923461587529E-014
          72   25.0585870287607       7.354993524000125E-014
          73   25.0723872670250       1.059819334298939E-013
          74   25.3618136741706       3.849435729340638E-014
          75   25.4036738120072       6.849804456978279E-014
          76   25.4042548920292       5.866047963087053E-014
          77   25.6158970643388       1.099591837357857E-013
```

Figure 32: Maximum extreme eigenvalues of SiNa

```
==>FEAST has successfully converged (to desired tolerance)

 Eigenvalues/Residuals
 inside interval
           1   1.44477792694849       5.663548535137325E-013
           2   1.44519692219645       4.754279543362265E-013
           3   1.44543331853536       3.184127088677789E-013
           4   1.44603941648245       3.041318355318949E-013
           5   1.44606194728945       3.635585435183280E-013
           6   1.44614837643549       9.098557590720591E-014
           7   1.44644469700648       4.778001132540904E-014
           8   1.44711591123342       6.125984249454085E-014
           9   1.44757785498424       1.620015569924142E-014
          10   1.44758869466176       2.580205724732772E-014
          11   1.44829642314667       8.704934064972564E-015
          12   1.44891577522194       1.259763640842065E-014
          13   1.44894824177651       1.400419980030161E-014
          14   1.44896427006087       1.322711592629826E-014
          15   1.44958784298739       8.599044904342955E-015
          16   1.44986113606725       7.174505662483007E-015
          17   1.45076468008881       8.451057325838996E-015
          18   1.45117323684439       7.759413082013943E-015
          19   1.45136167313701       7.707186983307302E-015
          20   1.45181868503455       5.458055357458313E-015
          21   1.45191089537480       5.285091449694620E-015
          22   1.45234891945469       4.787552429971632E-015
          23   1.45292215673493       6.626763470249798E-015
          24   1.45293868536399       5.021441168309142E-015
          25   1.45343064633640       5.002749886460571E-015
          26   1.45446966037580       4.385375474815364E-015
          27   1.45460782872456       4.619062334079112E-015
          28   1.45730511597249       4.761317672006315E-015
          29   1.45746290063035       5.717631907206682E-015
          30   1.45815567526571       4.898720925036327E-015
          31   1.45834704045913       2.716226068681063E-015
          32   1.45850773804757       3.830902405094591E-015
          33   1.45994481356690       3.358838963234151E-015
          34   1.46043472880728       4.853918405828667E-015
          35   1.46087150666151       6.218714341676726E-015
          36   1.46220428002512       4.464418204954026E-015
          37   1.46338060038774       4.041794067701852E-015
          38   1.46356579035940       3.910647795337413E-015
          39   1.46424446988092       6.726664397476691E-015
          40   1.46499372264112       5.611045502953078E-015
          41   1.46635444258063       4.935399788683366E-015
          42   1.46647447381448       4.701130322254640E-015
          43   1.46951532070274       5.824788912440024E-015
          44   1.47042546984662       5.249010439637113E-015
          45   1.47136294936063       5.778415703224987E-015
          46   1.47236174881261       6.553547176751897E-015
          47   1.47307136035528       4.434472624447688E-015
          48   1.47722154144494       6.047602012046792E-015
          49   1.47796998609753       5.217344660387804E-015
          50   1.47842950436113       8.104134006267435E-015
          51   1.48116041709345       4.941142263491964E-015
          52   1.48555370295671       6.610430704400326E-015
          53   1.48856380287286       5.574942694528916E-015
          54   1.55154854669761       5.778716154050998E-015
```

Figure 33: Maximum extreme eigenvalues of cnt-A

References

[1] E. Polizzi, "Density-matrix-based algorithm for solving eigenvalue problems," *Physical Review B Condensed matter and materials physics*, vol. 79, no. 11, 2009. DOI: 10.1103/PhysRevB.79.115112.

[2] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users' guide*. Philadelphia: SIAM, 1998, ISBN: 9780898714074.

[3] E. Polizzi. (). Feast eigenvalue solver, [Online]. Available: http://www.feast-solver.org/.

[4] Harvard, *Harvard math lecture note on eigenvlaues*.

[5] *Quora*. [Online]. Available: https://www.quora.com/Why-do-eigenvalues-matter-What-are-their-real-world-applications.

[6] K. Bryan and T. Leise, "The $25,000,000,000 eigenvector : The linear algebra behind google$," *SIAM Review*, vol. 48, no. 3, pp. 569–581, 2006. [Online]. Available: http://www.jstor.org/stable/20453840.

[7] R. Larson, *Elementary linear algebra.* English. 2017, pp. 586–595, ID: 957642721, ISBN: 9781305658004 1305658000 9781305953208 1305953207.

[8] Y. Liu, Z. You, and L. Cao, "A concise functional neural network computing the largest (smallest) eigenvalue and one corresponding eigenvector of a real symmetric matrix," in *2005 International Conference on Neural Networks and Brain*, ID: 7, vol. 3, 2005, pp. 1334–1339. DOI: 10.1109/ICNNB.2005.1614878.

[9] J. H. Manton, "A new algorithm for computing the extreme eigenvectors of a complex hermitian matrix," in *Proceedings of the 11th IEEE Signal Processing Workshop on Statistical Signal Processing (Cat. No.01TH8563)*, ID: 5, 2001, pp. 225–228. DOI: 10.1109/SSP.2001.955263.

[10] A. Marquis, *Ece 499y semester plan for fall 2017*.

[11] *Gershgorin circle theoremnbsp;* [Online]. Available: https://en.wikipedia.org/wiki/Gershgorin_circle_theorem.

[12] L. DeVille, "Optimizing gershgorin for symmetric matrices," 2016, ID: 1605.07239; Accession Number: 1605.07239; DocumentType: working paper; Archive Set: Mathematics; Last Revision Date: 20160523. [Online]. Available: http://silk.library.umass.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=1605.07239&site=eds-live&scope=site%20http://arxiv.org/abs/1605.07239.

[13] E. D. Napoli, E. Polizzi, and Y. Saad, "Efficient estimation of eigenvalue counts in an interval," 2013, ID: 1308.4275; Accession Number: 1308.4275; DocumentType: working paper; Archive Set: Computer Science; Last Revision Date: 20130820. [Online]. Available: http://silk.library.umass.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=1308.4275&site=eds-live&scope=site%20http://arxiv.org/abs/1308.4275.

[14] (). Parsec matrices, [Online]. Available: https://www.cise.ufl.edu/research/sparse/matrices/PARSEC/index.html.