Exploration of FEAST Algorithm

Stephanie Kajpust, Chathuri Samarasinghe, Nathasha Weerasinghe

April 27, 2012

Abstract

[1] In 2009 a numerical algorithm for solving the symmetric eigenvalue problem, named FEAST, was presented by Eric Polizzi. Using this algorithm we wrote a code in Mathematica in order to find a given number of eigenvalues in a specified interval. The accuracy of the algorithm was tested for finding single, repeated, clustered, and small eigenvalues, and the results are presented in this report.

[1] NW

Contents

1 Introduction

[2] In this paper we explore the FEAST algorithm written by Eric Polizzi at the University of Massachusetts [1]. FEAST enables you to find the eigenvalues in a given range.

The goal of this project was to find out just how well Polizzi's algorithm works. He has written a computer package [2] to implement his algorithm, and it is available for distribution to anyone. However, we chose to follow Polizzi's pseudocode and write the algorithm in Mathematica. Polizzi ran his code on a 12450×12450 matrix and looked for various numbers of eigenvalues. He used a relative residual

$$\max_i \frac{\|Ax_i - \lambda_i B x_i\|_1}{\|Ax_i\|_1} \tag{1}$$

and his residuals are all 10^{-10} or better [1]. Using our Mathematica code, we tested to see if ours held up to Polizzi's claims, using the same residual and using a different relative residual.

$$\frac{\|\lambda_{actual} - \lambda_{calculated}\|_2}{\|\lambda_{actual}\|_2} \tag{2}$$

2 Algorithm

[3] Figure 1 is the pseudocode that Polizzi gives in [2]. The variables are as follows:

- N is the matrix size
- $[\lambda_{\min}, \lambda_{\max}]$ is the interval in which you want to find the eigenvalues
- N_e is the number of points for Gauss-Legendre quadrature. Polizzi says he gets consistent results with $N_e = 8$ [2], so that is what we used.
- M is the number of eigenvalues in the search interval.
- M_0 is the size of the working subspace, which Polizzi says works well if $M_0 \geq 1.5M$ [2]

[2] SK
[3] SK

For this verson of FEAST, the matrix A is real symmetric and B is symmetric positive definite.

1- Select $\mathrm{M}_0 > \mathrm{M}$ random vectors $\mathbf{Y}_{\mathrm{N}\times\mathrm{M}_0} \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}_0}$

2- Set $\mathbf{Q} = 0$ with $\mathbf{Q} \in \mathbb{R}^{\mathrm{N}\times\mathrm{M}_0}$; $r = (\lambda_{\max} - \lambda_{\min})/2$;

For $e = 1, \ldots \mathrm{N}_e$

compute $\theta_e = -(\pi/2)(x_e - 1)$,

compute $\mathrm{Z}_e = (\lambda_{\max} + \lambda_{\min})/2 + r\exp(\imath\theta_e)$,

solve $(\mathrm{Z}_e\mathbf{B} - \mathbf{A})\mathbf{Q_e} = \mathbf{Y}$ to obtain $\mathbf{Q_e} \in \mathbb{C}^{\mathrm{N}\times\mathrm{M}_0}$

compute $\mathbf{Q} = \mathbf{Q} - (\omega_e/2)\Re\{r\exp(\imath\theta_e)\ \mathbf{Q_e}\}$

End

3- Form $\mathbf{A_Q}_{\mathrm{M}_0\times\mathrm{M}_0} = \mathbf{Q^T A Q}$ and $\mathbf{B_Q}_{\mathrm{M}_0\times\mathrm{M}_0} = \mathbf{Q^T B Q}$

rescale value of M_0 if $\mathbf{B_Q}$ is not spd.

4- Solve $\mathbf{A_Q \Phi} = \epsilon\mathbf{B_Q \Phi}$ to obtain the M_0 eigenvalue $\epsilon_\mathbf{m}$,

and eigenvectors $\mathbf{\Phi}_{\mathrm{M}_0\times\mathrm{M}_0} \in \mathbb{R}^{\mathrm{M}_0\times\mathrm{M}_0}$

5- Set $\lambda_m = \epsilon_m$ and compute $\mathbf{X}_{\mathrm{N}\times\mathrm{M}_0} = \mathbf{Q}_{\mathrm{N}\times\mathrm{M}_0}\mathbf{\Phi}_{\mathrm{M}_0\times\mathrm{M}_0}$

If $\lambda_\mathrm{m} \in [\lambda_{\min}, \lambda_{\max}]$, λ_m is an eigenvalue solution

and its eigenvector is $\mathbf{X_m}$ (the m^{th} column of $\mathbf{X}$).

6- Check convergence for the trace of the eigenvalues λ_m

If iterative refinement is needed, compute $\mathbf{Y} = \mathbf{BX}$

and go back to step 2

Figure 1: FEAST pseudocode for real symmetric matrices [2]

3 Code

[4] We wrote several Mathematica modules to implement FEAST.

Our first module (see figure 2) created the Q matrix. Our code uses the special case where $B = Id$ to solve $Ax = \lambda Bx$.

The second module was the main part of the FEAST code itself. In the pseudocode seen in figure 1, the Q portion is run several times, along with steps 3-5. The goal is to run until we get accurate values. Figure 3 is our Mathematica code for the rest of the algorithm.

[4] SK

```
Clear[FeastQ]
FeastQ[{A_, Y_}, {Lmin_, Lmax_}]:=Module[
{Ne = 8, r, c, B, n = Length[A], XW, Q, θe, Ze, Qe},
{r, c} = 0.5{Lmax − Lmin, Lmax + Lmin};
B = SparseArray[Band[{1, 1}] → 1.0, {n, n}];
(* XW are the Gaussian integration data from Polizzi *)
XW = {{0.183434642495649, 0.362683783378361},
{−0.183434642495649, 0.362683783378361},
{0.525532409916328, 0.313706645877887},
{−0.525532409916328, 0.313706645877887},
{0.796666477413626, 0.222381034453374},
{−0.796666477413626, 0.222381034453374},
{0.960289856497536, 0.101228536290376},
{−0.960289856497536, 0.101228536290376}};
(* Initializing then forming Q*)
Q = 0 * Y;
For[e = 1, e ≤ Ne, e++,
θe = −(π/2)(XW[[e, 1]] − 1);
Ze = c + r * E^(I * θe);
Qe = LinearSolve[Ze * B − A, Y];
Q = Q − (XW[[e, 2]]/2) * Re[r * E^(I * θe) * Qe];
];
Q
]
```

Figure 2: Module to create the Q matrix

A is an $n \times n$ matrix that is real and symmetric. $Lmin$ and $Lmax$ are the values for the search interval, and M is the number of eigenvalues in that interval.

The algorithm uses Q to create smaller matrices, Aq and Bq. These matrices are small enough that we can use the built-in Mathematica command Eigensystem to find the eigenvectors and eigenvalues of the smaller matrix.

Q is always of a bigger dimension than the number of eigenvalues in the interval. This means the code always outputs more eigenvalues than we need. In fact, it outputs 1.5 times more than we need, rounded up. The assumption would be that we would use the first M eigenvalues output, but the built-in eigensystem solver automatically sorts the values from largest to smallest, so

```
FEAST[A_, {Lmin_, Lmax_}, M_]:=Module[
{n = Length[A], M0 = Ceiling[1.5 * M], Y, evals, evecs},
(* Initial Random Directions *)
Y = RandomReal[{-1, 1}, {n, M0}];
(* Initializing then forming Q*)
Q = FeastQ[{A, Y}, {Lmin, Lmax}];
(* forming Aq and Bq *)
(*Note doing special case with B = Id*)
Aq = Transpose[Q].A.Q;
Bq = Transpose[Q].Q;
{evals, evecs} = Eigensystem[{Aq, Bq}];
{evals, Q.Transpose[evecs]}
]
```

Figure 3: Module that does the main FEAST code

there is a question of which ones are the ones we are supposed to look at. The solution would be to run the algorithm multiple times. The value that shows up the most would be the correct eigenvalue.

To make testing easier, we also have a module (seen in figure 4) that creates symmetric matrices when given specified eigenvalues. That way we could control the testing process.

```
Clear[SymMatWithEvals]
SymMatWithEvals[evals_List]:=Module[
{n = Length[evals], Q, A},
(* Form a random rotation *)
Q = QRDecomposition[RandomVariate[NormalDistribution[0, 1], {n, n}]][[1]];
(* Similarity transforms the input eigenvalues to be unrecognizable *)
A = Q.DiagonalMatrix[evals].Transpose[Q];
(* A is already symmetric in exact arithmetic *)
(* but for some purposes I need to fix any floating point asymmetry *)
0.5(A + Transpose[A])
]
```

Figure 4: Module that creates symmetric matrices given eigenvalues

4 Results

4.1 General testing of algorithm

[5] The majority of our work was done on the real version of Polizzi's algorithm. The initial goal was to see if the algorithm actually worked.

4.1.1 Find 1 eigenvalue

The first step of testing the FEAST algorithm was to see if it could accurately find one specific eigenvalue. Table 1 shows the result of testing a 5×5 real symmetric matrix. Each run found the correct eigenvalue and placed it first on the list. FEAST output 2 eigenvalues becaue it created a 2×2 matrix to find the eigenvalues. The second eigevalue can be ignored as it is not actually an eigenvalue of the original matrix.

Eigenvalues	λ_{min}	λ_{max}	M	Output
1, 25, 50, 400, 1000	0	5	1	{1., 0.}
1, 25, 50, 400, 1000	20	30	1	{25., 0.}
1, 25, 50, 400, 1000	45	55	1	{50., Indeterminate}
1, 25, 50, 400, 1000	350	500	1	{400., 123.586}
1, 25, 50, 400, 1000	900	1200	1	{1000., 365.714}

Table 1: Testing the ability to find one specific eigenvalue

4.1.2 Find 2 eigenvalues

The next test was to see how well the algorithm performed finding 2 eigenvalues. The results can be seen in table 2.

Polizzi specifically says in his algorithm (see figure 1) that only the values within the search interval are eigenvalues. The algorithm performs as expected. The value of 50.0013 in the first row of the table is not considered an eigenvalue by Polizzi since it is not in the range. However, we know that 50 is actually an eigenvalue. It was just not one we were looking for.

Row 3 shows that the output isn't always exact. As seen in table 3, our relative residual is 10^{-6} which is less than the residual that Polizzi said he

[5] SK

Eigenvalues	λ_{min}	λ_{max}	M	Output
1, 25, 50, 400, 1000	-2	30	2	{50.0013, 25., 1.}
1, 25, 50, 400, 1000	20	75	2	{50., 25., 1.}
1, 25, 50, 400, 1000	40	500	2	{399.94894163701554, 49.99974501865308, 19.06911267847751}
1, 25, 50, 400, 1000	350	1200	2	{1000., 400., 1.81701}

Table 2: Testing the ability to find two specific eigenvalues

was getting, and much smaller than the residual we found using Polizzi's residual in equation 1.

Correct Eigenvalue	Algorithm Eigenvalue	Equation 1	Equation 2
400	399.94894163701554	0.011914984061113687	0.00012764590746115802
50	49.99974501865308	0.002148514867156731	$5.099626938402935 \times 10^{-6}$

Table 3: Comparing the Accuracy of Eigenvalues

4.1.3 Finding 3-5 eigenvalues

In table 4 you can see that sometimes there is an eigenvalue found that is nearly the same as the one we were looking for. In the case of looking for all 5, you can see that 50 is found twice, though the second one is very close. 25 is found twice in row 1 of the same table.

Eigenvalues	λ_{min}	λ_{max}	M	Output
1, 25, 50, 400, 1000	0	60	3	{50., 25., 24.6902, 1., 0.}
1, 25, 50, 400, 1000	20	450	3	{1006.24, 400., 50., 25., 1.}
1, 25, 50, 400, 1000	40	1500	3	{1000., 400., 50., 25., 1.}
1, 25, 50, 400, 1000	0	500	4	{997.753, 400., 50., 25., 1.00061, 1.}
1, 25, 50, 400, 1000	20	1200	4	{1000., 400., 50., 40.0445, 25., 1.}
1, 25, 50, 400, 1000	0	1200	5	{1000., 400., 50., 49.9727, 25., 22.3177, 1., Indeterminate}

Table 4: Looking for 3-5 eigenvalues

Table 5 shows the relative residuals for the results from row 1 of table 4. In this case the residuals are all better than Polizzi's numbers for our relative residual. Polizzi's residuals are respectable, but the difference implies that perhaps the eigenvectors are not as accurate as they should be.

Correct Eigenvalue	Algorithm Eigenvalue	Equation 1	Equation 2
50	50.	4.04765×10^{-10}	7.10543×10^{-16}
25	25.	4.13696×10^{-8}	8.5123×10^{-14}
1	1.	1.68769×10^{-7}	1.9762×10^{-14}

Table 5: Comparing the Accuracy of Eigenvalues found in row 1 of table 4

4.1.4 Testing accuracy of finding 1 eigenvalue multiple times

The next test was to see how accurately it finds one eigenvalue when done repeatedly. We looked for the eigenvalue 1 given a matrix with eigenvalues of {1, 25, 50, 400, 1000}. The search interval was [0,2]. The FEAST algorithm was run 200 times with the same input. Figure 5 shows the result. The consistency is very good.

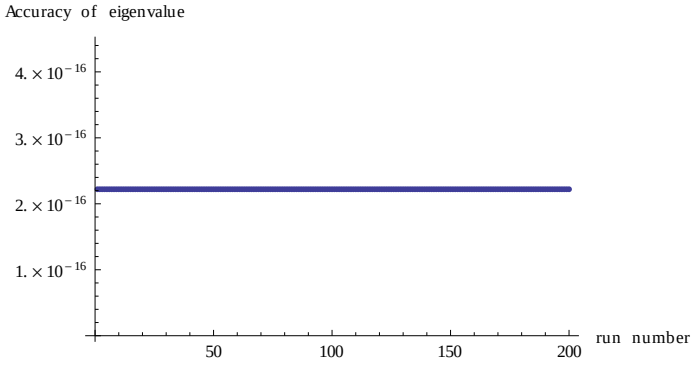

Figure 5: Relative Residual (eqn 2) of looking for 1 eigenvalue over 200 runs

Figure 6 shows another run using the same information as in the previous test. This time the algorithm was run 400 times. Red dots show the first eigenvalue, and blue dots show the second eigenvalue. Clearly 1 is found as it should be. However, there are times when either the first value is incorrect or the second one is incorrect.

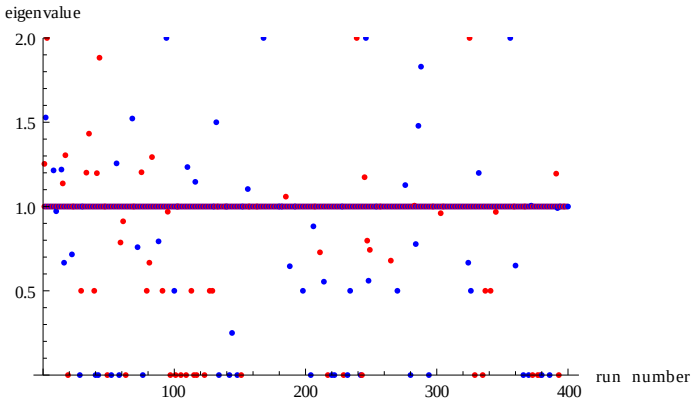

Figure 6: Graph of resulting eigenvalues when looking for an eigenvalue of 1 on range [0,2]

The interpretation of figure 6 is supported by figure 7, which is a table of the results of 25 runs looking for an eigenvalue of 1.

2.323	1.
1.	0.
1.	Indeterminate
1.	Indeterminate
1.	Indeterminate
1.	Indeterminate
1.	0.
1.	Indeterminate
1.	1.
1.	0.993538
1.	Indeterminate
∞	1.
1.25424	1.
$-2.$	1.
1.	Indeterminate
∞	1.
1.	Indeterminate
1.	Indeterminate
1.	1.
1.	Indeterminate
1.	1.
1.	0.
1.	0.962441
1.	1.
1.	Indeterminate

Figure 7: Eigenvalues for 25 runs looking for $\lambda = 1$ in interval [0,2]

4.2 Testing M

[6] Once we established that the algorithm worked, we were interested in how picking M effects the results.

4.2.1 Underestimating the number of eigenvalues

The first test was to see what happens if you underestimate the number of eigevalues by 1. Table 6 shows that the accuracy is not affected in any way. 1, 200.003, 200.004, 200.005, 200.006, 200.007, 400

Eigenvalues	λ_{min}	λ_{max}	M	Output
{1, 25, 50, 400, 1000}	0	30	1	{25., 1.}
{1, 25, 50, 400, 1000}	20	55	1	{50., 25.}
{1, 25, 50, 400, 1000}	40	500	1	{399.276, 49.9987}
{1, 25, 50, 400, 1000}	350	1200	1	{1000., 400.}
{1, 25, 50, 400, 1000}	0	55	2	{50., 25., 1.}
{1, 25, 50, 400, 1000}	0	450	3	{829.44, 400., 50., 25., 1.}
{1, 25, 50, 400, 1000}	0	1200	4	{1000., 400., 50., 49.043, 25., 1.}

Table 6: Eigenvalues found when underestimating M by 1

Underestimating the number of eigenvalues in a range by 2 means that the FEAST output is extremely inaccurate. Figure 8 shows the results of this. We were looking for 1 eigenvalue in the range [0,55], which actually contains 3 eigenvalues (original eigenvalues were {1, 25, 50, 400, 1000}). The resulting plot (FEAST run 25 times) shows that the values are all over the place.

[6] SK

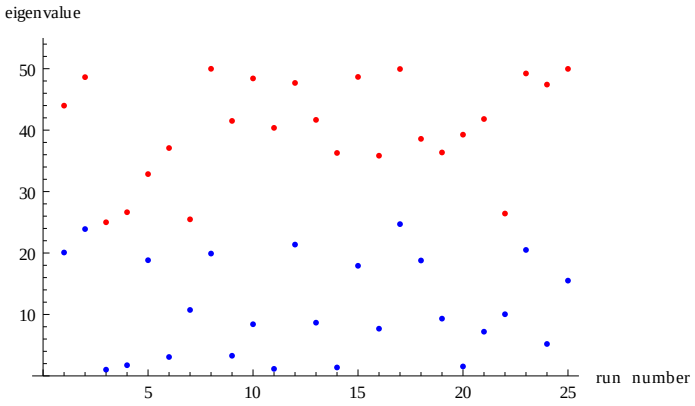

Figure 8: Plot of eigenvalues found when underestimating M by 2

4.2.2 Overestimating the number of eigenvalues

Based on how the algorithm works, overestimating the number of eigenvalues will not affect the output. Overestimating M just means we use a larger matrix size to find eigenvalues from. We already overestimated the number of eigenvalues by using the ceiling of $1.5M$.

4.3 Testing search interval

[7] Surprisingly, the search interval is not so important when looking for the specific eigenvalues. The algorithm will find eigenvalues that are near the search interval, though technically they are not considered eigenvalues because they are not in the range. So we find none in the range, but we do find the nearby ones. This can be seen in table 7.

Eigenvalues	λ_{min}	λ_{max}	M	Desired λ	Output
{1, 25, 50, 400, 1000, 20, 45, 500}	0	1	1	1	{1.,0.5}
{1, 25, 50, 400, 1000, 20, 45, 500}	0	.9	1	1	{1.,0.}
{1, 25, 50, 400, 1000, 20, 45, 500}	-1	0	1	1	{23.6169,1.}
{1, 25, 50, 400, 1000, 20, 45, 500}	55	60	1	50	{50.,44.9802}

Table 7: Testing to see if an eigenvalue near the range can be found

[7] SK

4.4 Testing clustered eigenvalues

[8] We tested how accurate the algorithm is when there are a bunch of clustered eigenvalues.

4.4.1 Eigenvalues {1, 298, 299, 300, 301, 302, 600}

We created a matrix with eigenvalues {1, 298, 299,300, 301, 302,600}. With search range [290,305] and $M = 5$ we ran the algorithm 25 times and plotted the results, which can be seen in figure 9.

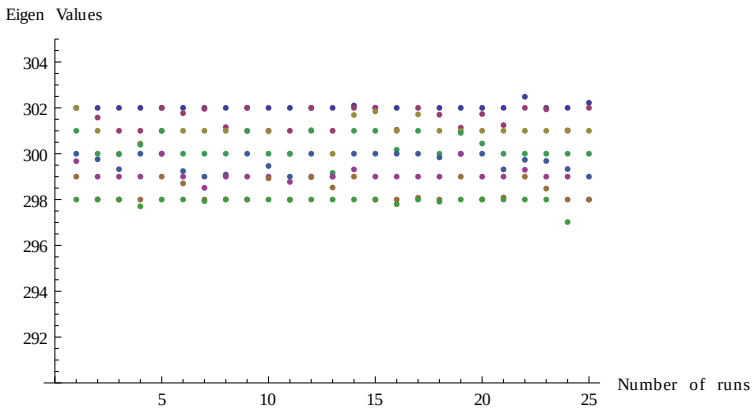

Figure 9: Plot of eigenvalues in a cluster, run 25 times with search interval [290,305]

We asked for the 5 eigenvalues clustered together but the algorithm gives us 8 eigenvalues, all inside the range we need and close to the exact eigen values. The first run had an output of {353.488, 302.001, 302.,301., 300., 299.674, 299., 298.}. All the needed eigenvalues are there, along with others that are extremely close. However, the difference between them are relatively large. For example, the residual between 299 and 299.674 is 0.00225475 (using 2), which shows that they are different enough.

Using the same initial eigenvalues, we then tested the results of 25 runs. This time we had a range of [300,305] so we were looking for the eigenvalues 300, 301, and 302. Then we plotted the resulting output on a range from [295,305] where all the clustered eigenvalues were. The idea was to see the eigenvalues near the range turned up. The results are in figure 10.

[8] CS

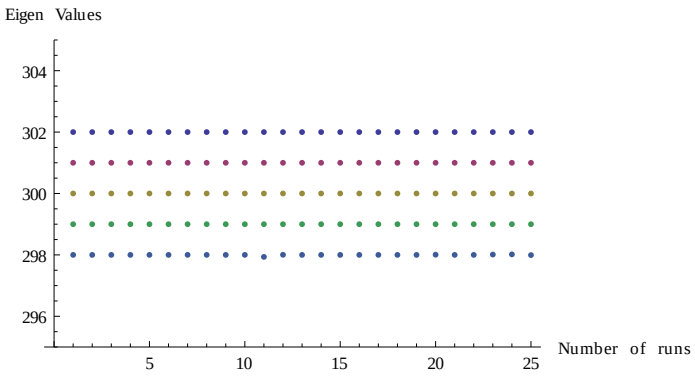

Figure 10: Plot of eigenvalues in a cluster when run 25 times. Search interval was [300,305]

We asked for 3 eigenvalues in the interval [300, 305], but we got all the correct clustered eigenvalues instead. This time there were no really close values to the ones we were looking for, as seen in the following sample output: {302., 301., 300., 299., 297.993}.

4.4.2 Eigenvalues {1, 4998, 4999, 5000, 5001, 5002, 10000}

We looked at clustered eigenvalues again, with a matrix that had the eigenvalues {1, 4998, 4999, 5000, 5001, 5002, 10000}. We did 25 runs looking for the eigenvalues in the range [4900,5050]. This time some of the results gave us complex eigenvalues, which should not occur since in the algorithm we are taking the real part. One run gave us eigenvalues of $\{5002. + 0.i, 5001.01 + 0.i, 5000.74 + 0.i, 5000. + 0.i, 4999. + 0.i, 4998. + 0.i, 4872.16 - 279.761i, 4872.16 + 279.761i\}$. Clearly there are some with quite large imaginary parts, and they are complex conjugates.

Figure 11 shows the graph of the 25 runs where complex eigenvalues showed up. We got 8 eigenvalues in the interval we wanted. The results changed as we got some complex eigenvalues for some runs.

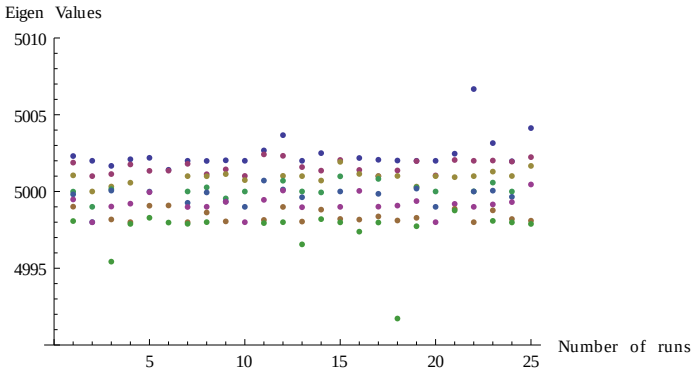

Figure 11: 25 runs looking for eigenvalues in the range [4900,5050]

4.4.3 Eigenvalues {1, 200.003, 200.004, 200.005, 200.006, 200.007, 400}

Next we tested clustered eigenvalues that were very close together. We created a matrix with the eigenvalues {1, 200.003, 200.004, 200.005, 200.006, 200.007, 400}. A single run looking for the 5 eigenvalues near 200 (range [200,201] resulted in {203.178, 200.021, 200.008, 200.007, 200.006, 200.005, 200.004, 200.003}. When the clustered eigenvalues were very close to each other we got the 5 correct eigenvalues and some extras.

Figure 12 shows the graph of the 30 runs to find the very closely clustered eigenvalues. We wanted 5 eigenvalues but received all 8 of the clustered ones.

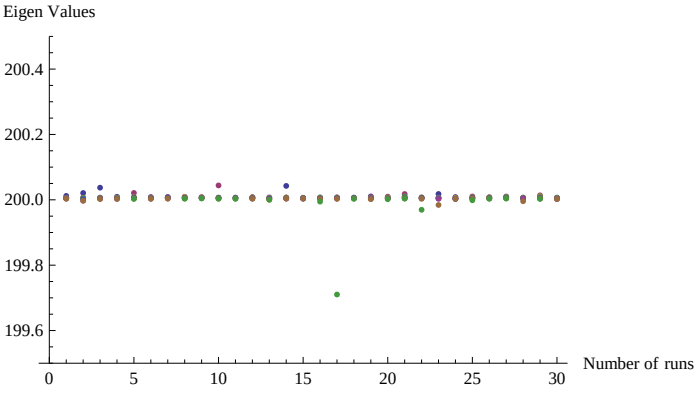

Figure 12: 30 runs looking for eigenvalues in the range [200,201]

4.4.4 Matrix with 30 very close eigenvalues and 2 eigenvalues far away

To test larger matrices, we created a matrix with 30 eigenvalues between [290,310] and eigenvalues of 1 and 600. Then we compared the output to the correct values. In this case we underestimated by 10 eigenvalues, setting $M = 20$.

The 32 eigenvalues that were correct were {1, 290.034, 290.227, 290.658, 291.621, 293.748, 294.924, 298.976, 299.017, 299.294, 299.449, 299.581, 301.276, 302.009, 302.254, 303.125, 303.321, 303.913, 304.455, 305.458, 305.798, 306.337, 306.445, 306.642, 306.655, 306.729, 307.044, 307.221, 308.003, 309.644, 309.848, 600}.

The output of FEAST was {290.028, 290.227, 290.658, 291.667, 293.748, 294.924, 298.976, 299.017, 299.294, 299.449, 299.581, 301.276, 302.009, 302.254, 303.125, 303.321, 303.913, 304.455, 305.458, 305.798, 306.337, 306.445, 306.642, 306.655, 306.729, 307.044, 307.221, 308.003, 309.644, 309.846}.

We get all 30 clustered eigenvalues even though we only ask for 20 but other than 2 eigenvalues (291.621 and 309.848) the other eigenvalues are a perfect match.

Figure 13 shows the graph of the 30 runs to find all the close eigenvalues. We get all 30 eigenvalues in the interval we want and they are almost always equal to the original eigenvalues.

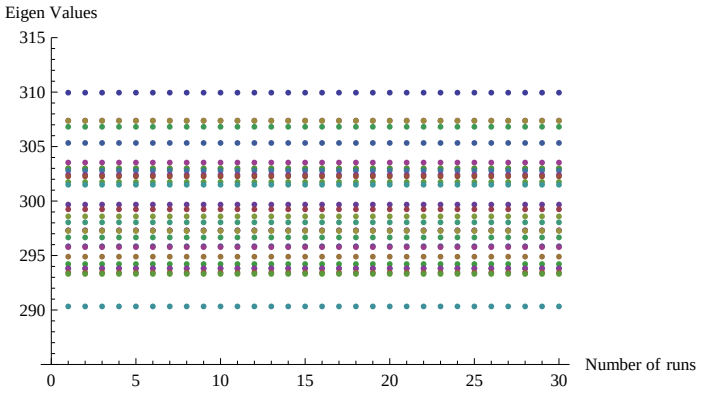

Figure 13: 30 runs looking for eigenvalues in the range [290,310]

We then did the same thing, this time asking for all the eigenvalues in the range ($M = 30$). Everything else was the same.

The 32 correct eigenvalues: {1, 290.193, 290.24, 291.473, 292.34, 292.821, 292.947, 293.636, 295.315, 295.914, 296.55, 298.423, 300.702, 301.269, 301.434, 301.533, 302.801, 305.344, 305.456, 305.504, 305.531, 306.007, 306.338, 306.562, 306.597, 306.973, 307.108, 307.362, 308.361, 308.443, 309.09, 600}.

The output of FEAST: $\{278.742 + 7.92246i, 278.742 - 7.92246i, 290.242 + 0.i, 290.431 + 0.i, 291.634 + 0.i, 292.715 + 0.i, 292.958 + 0.i, 293.682 + 0.i, 295.079 + 0.i, 295.392 + 0.i, 295.605 + 15.9858i, 295.605 - 15.9858i, 295.949 + 0.i, 298.438 + 0.i, 299.581 + 0.i, 300.224 - 4.36456i, 300.224 + 4.36456i, 301.392 + 0.0646473i, 301.392 - 0.0646473i, 301.795 + 0.i, 302.557 + 0.i, 304.112 - 2.24116i, 304.112 + 2.24116i, 305.358 + 0.i, 305.486 + 0.i, 305.52 + 0.i, 306.177 + 0.i, 306.354 + 0.i, 306.596 + 0.i, 306.793 + 0.i, 306.967 + 0.i, 307.383 + 0.i, 308.563 + 0.i, 308.713 - 10.8968i, 308.713 + 10.8968i, 309.685 + 0.419621i, 309.685 - 0.419621i, 311.316 - 25.7048i, 311.316 + 25.7048i, 313.172 + 0.i, 342.176 + 0.i, 392.884 + 0.i, 404.573 + 0.i, 462.882 + 0.i, 689.925 + 0.i\}$.

When we ask for all 30 of the eigenvalues inside the interval we do not get an accurate solution. We end up with more complex results, and while some of them have a complex part of essentially zero, not all of them do.

$$291.621 \rightarrow 291.667$$
$$309.848 \rightarrow 309.846$$

The last thing we did was to graph the results when there are 30 close eigenvalues. Figure 14 shows that the results are consistent.

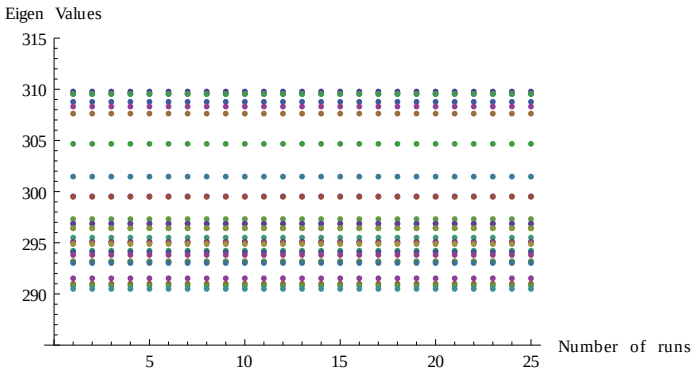

Figure 14: Multiple runs looking for 30 close eigenvalues

Then we wanted to see what would happen if we only asked for 19 eigenvalues in the interval. We get 29 eigenvalues and they were not as accurate as our earlier attempts.

The eigenvalues we get are: 309.905, 309.882, 308.201, 308.116, 308.08, 307.572, 307.559, 307.473, 305.6, 304.957, 304.665, 303.918, 303.888, 303.793, 302.924, 301.92, 301.369, 300.722, 299.96, 297.833, 297.399, 297.204, 295.279, 294.833, 294.294, 292.784, 292.491, 291.328, 290.526

$$290.034 \rightarrow 290.525$$
$$294.924 \rightarrow 294.833$$

As Figure 15 shows the results are not consistent.

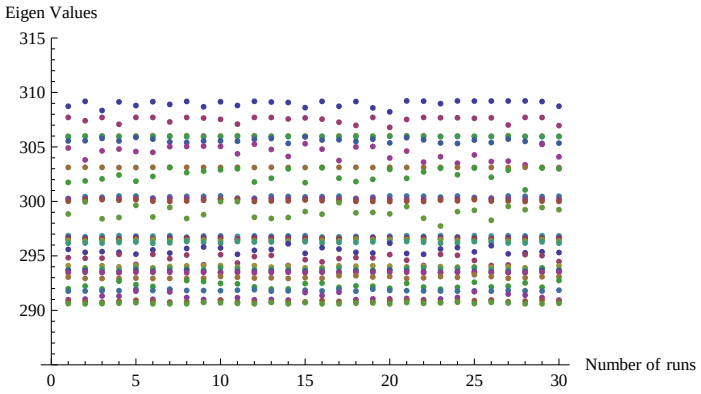

Figure 15: Multiple runs looking for 30 close eigenvalues for M = 19

4.5 Testing small eigenvalues

[9] We tested to see how accurately FEAST performs with small eigenvalues.

4.5.1 Looking for one specific eigenvalue

Before extensive testing of small eigenvalues, we wanted to make sure that we could get a good result looking for 1 of them. With a matrix that had eigenvalues of {0.0001,0.0025,0.005,0.04,0.1}, we ran FEAST on an interval of [0,0.0002] and $M = 1$. {0.0001,0.0000496878} was the resulting output. It correctly found the eigenvalue of 0.0001.

[9] NW

The next step was to find the accuracy of finding 0.0001 over 300 runs. Using the same interval and M, we plotted the relative residual using equation 2. Figure 16 shows that the relative residual is quite small, even smaller than for large eigenvalues.

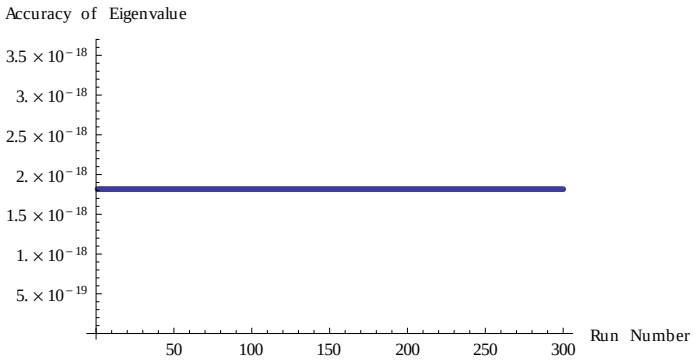

Figure 16: Plot of the accuracy of finding a small eigenvalue of 0.0001 over 300 runs

We decided to run this algorithm multiple times and compare plots of the eigenvalues found. Figure 17 shows the plot of eigenvalues over 200, 400, and 800 runs. Comparing the 3 plots you can see that a "ghost" eigenvalue shows up around 0.00012. The more runs, the more times it shows up. Our theory is that because it is such a small range, the likelihood of the same number showing up is higher than if the range is larger.

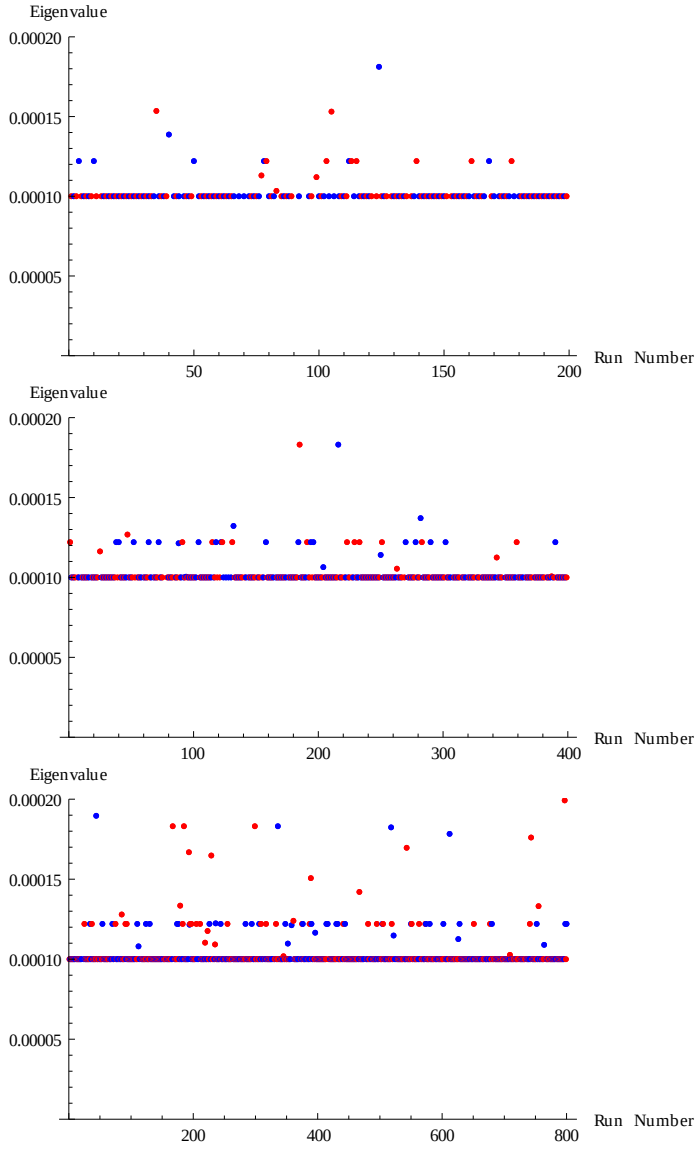

Figure 17: Multiple runs finding 0.0001 results in a "ghost" eigenvalue showing up

4.5.2 Looking for multiple eigenvalues

Using a matrix with the same eigenvalues as before ($\{0.0001, 0.0025, 0.005, 0.04, 0.1\}$) we tested finding 3 eigenvalues in a range. Figure 18 shows FEAST run 300 times to find the eigenvalues between 0 and 0.0051. There were 3 eigenvalues in this range. Because of how the algorithm works, FEAST gives $\lceil 1.5 \times 3 \rceil = 5$ possible eigenvalues. In this plot each eigenvalue was given a different color in the following order: Red, Yellow, Blue, Magenta, Green. As can be seen in figure 18, the first eigenvalue output from the algorithm was always an eigenvalue because the red dots appear only in front of the eigenvalue 0.005. This was the largest eigenvalue in the range since the output is always sorted. Therefore we can assume that for small eigenvalues, the largest would always be an actual eigenvalue.

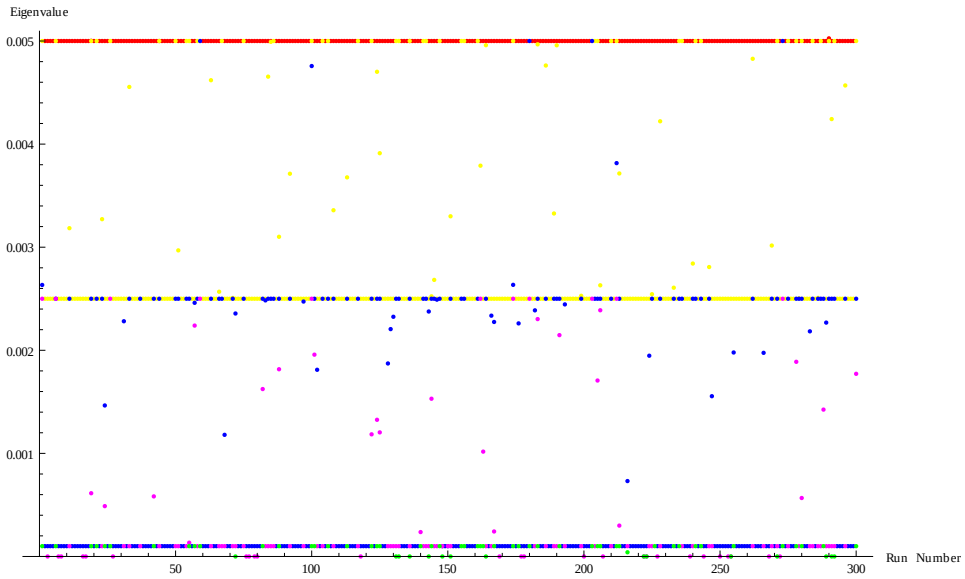

Figure 18: Plot of FEAST output looking for 3 small eigenvalues, run 300 times

Note that even though this algorithm outputs values (which are also in the range of consideration) that are not eigenvalues, we can always run the algorithm several times and figure out what the real eigenvalues are. As seen in figure 18, the actual eigenvalues appear frequently when the algorithm is run multiple times. The algorithm does not take much time to run, so this a feasible way to find which of the resulting numbers are eigenvales.

One sample from the run of 300 gave an output of {0.005, 0.00301705, -0.00252876, 0.0025, 0.0001}. We computed the relative residual of the correct eigenvalues using our norm and Polizzi's norm. Table 8 shows these residuals. In general they are more accurate by a power of ten than the residuals obtained in table 5.

Correct Eigenvalue	Algorithm Eigenvalue	Equation 1	Equation 2
0.0050	0.005	1.97687×10^{-11}	1.73472×10^{-16}
0.0025	0.0025	1.87104×10^{-9}	1.21431×10^{-15}
0.0001	0.0001	7.91298×10^{-9}	1.07065×10^{-14}

Table 8: Comparing the accuracy of small eigenvalues

4.6 Testing repeated eigenvalues

[10] We tested what happened when there were multiple eigenvalues in an interval.

4.6.1 Eigenvalues {1, 5, 10, 10, 10, 15, 20, 25}

We did 25 runs of FEAST looking for 4 eigenvalues in the interval [4,12] from a matrix with eigenvalues of {1, 5, 10, 10, 10, 15, 20, 25}. In that interval 10 appears 3 times.

Output for one run was {15.0003, 10., 10., 10., 5., 1.00222}. It has the correct eigenvalues of 10 in there 3 times, and also has the eigenvalue of 5. In fact, when we plot the eigenvalues in the interval for the 25 runs, we find that it is extremely consistent in finding the correct eigenvalues. Figure 19 portrays this.

Table 9 shows the residual between the calculated eigenvalues and the correct eigenvalues for the run previously mentioned.

[10] CS

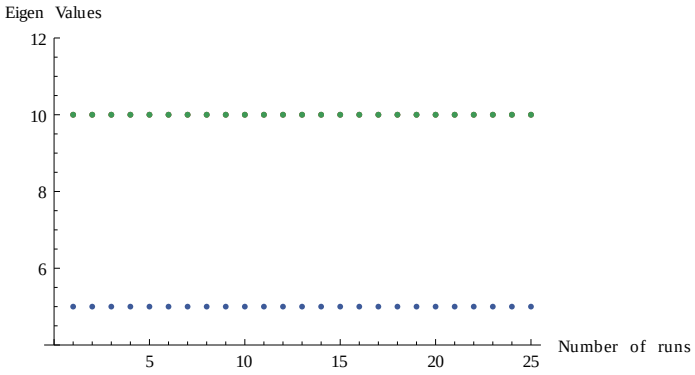

Figure 19: Looking for eigenvalues of 10,10,10 and 5 over 25 runs

Correct λ	Calculated λ	Difference
5	5.000000000000304	$3.0375701953744283 \times 10^{-13}$
10	10.000000000000004	$3.552713678800501 \times 10^{-15}$
10	10.000000000000012	$1.2434497875801753 \times 10^{-14}$
10	10.000000000000357	$3.5704772471945034 \times 10^{-13}$

Table 9: Residual for repeated eigenvalues

5 Complex Version

[11] We used Polizzi's pseudocode [2] for complex eigenvalues to write the algorithm in Mathematica. Figure 20 shows Polizzi's pseudocode.

The code we use is almost the same as earlier with the main difference being that we do not ignore the complex parts on the way to compute Q. The modified Q code is in figure 21.

Our general code for running the complex version of FEAST can be seen in figure 22.

Testing of the complex version was not complete, as there seemed to be a lot more issues in getting it started.

[11] CS

1- Select $M_0 > M$ random vectors $\mathbf{Y}_{N\times M_0} \in \mathbb{C}^{N\times M_0}$

2- Set $\mathbf{Q} = 0$ with $\mathbf{Q} \in \mathbb{C}^{N\times M_0}$; $r = (\lambda_{max} - \lambda_{min})/2$;

For $e = 1, \ldots N_e$

compute $\theta_e = -(\pi/2)(x_e - 1)$,

compute $Z_e = (\lambda_{max} + \lambda_{min})/2 + r\exp(\imath\theta_e)$,

solve $(Z_e\mathbf{B} - \mathbf{A})\mathbf{Q_e} = \mathbf{Y}$ to obtain $\mathbf{Q_e} \in \mathbb{C}^{N\times M_0}$

solve $(Z_e\mathbf{B} - \mathbf{A})^{\dagger}\hat{\mathbf{Q}}_\mathbf{e} = \mathbf{Y}$ to obtain $\hat{\mathbf{Q}}_\mathbf{e} \in \mathbb{C}^{N\times M_0}$

$\mathbf{Q} = \mathbf{Q} - (\omega_e/4)r\left(\exp(\imath\theta_e)\ \mathbf{Q_e} + \exp(-\imath\theta_e)\ \hat{\mathbf{Q}}_\mathbf{e}\right)$

End

3- Form $\mathbf{A_{Q_{M_0\times M_0}}} = \mathbf{Q^T A Q}$ and $\mathbf{B_{Q_{M_0\times M_0}}} = \mathbf{Q^T B Q}$

rescale value of M_0 if $\mathbf{B_Q}$ is not hpd.

4- Solve $\mathbf{A_Q\Phi} = \epsilon\mathbf{B_Q\Phi}$ to obtain the M_0 eigenvalue $\epsilon_\mathbf{m}$, and eigenvectors $\mathbf{\Phi}_{M_0\times M_0} \in \mathbb{C}^{M_0\times M_0}$

5- Set $\lambda_m = \epsilon_m$ and compute $\mathbf{X}_{N\times M_0} = \mathbf{Q}_{N\times M_0}\mathbf{\Phi}_{M_0\times M_0}$

If $\lambda_\mathbf{m} \in [\lambda_{min}, \lambda_{max}]$, $\lambda_\mathbf{m}$ is an eigenvalue solution and its eigenvector is $\mathbf{X_m}$ (the $\mathbf{m}^{th}$ column of $\mathbf{X}$).

6- Check convergence for the trace of the eigenvalues λ_m

If iterative refinement is needed, compute $\mathbf{Y} = \mathbf{BX}$ and go back to step 2

Figure 20: FEAST pseudocode for the complex version

```
Clear[FeastQComplex]
FeastQComplex[{A_, Y_}, {Lmin_, Lmax_}]:=Module[
{Ne = 8, r, c, B, n = Length[A], XW, Q, θe, Ze, Qe, Qhe},
{r, c} = 0.5{Lmax - Lmin, Lmax + Lmin};
B = SparseArray[Band[{1, 1}] → 1.0, {n, n}];
XW = {{0.183434642495649, 0.362683783378361},
{-0.183434642495649, 0.362683783378361},
{0.525532409916328, 0.313706645877887},
{-0.525532409916328, 0.313706645877887},
{0.796666477413626, 0.222381034453374},
{-0.796666477413626, 0.222381034453374},
{0.960289856497536, 0.101228536290376},
{-0.960289856497536, 0.101228536290376}};
Q = 0 * Y;
For[e = 1, e ≤ Ne, e++,
θe = -(π/2)(XW[[e, 1]] - 1);
Ze = c + r * E^(I * θe);
Qe = LinearSolve[Ze * B - A, Y];
Qhe = LinearSolve[Transpose[Ze * B - A], Y];
Q = Q - (XW[[e, 2]]/4) * r * ((e^(i*θe) * Qe) - (e^(-i*θe) * Qhe));
];
Q
]
```

Figure 21: Complex version of code to create the Q matrix

```
FEASTComplex[A_, {Lmin_, Lmax_}, M_]:=Module[
{n = Length[A], M0 = Ceiling[1.5 * M], Y, evals, evecs},
Y = RandomReal[{-1, 1}, {n, M0}];
Q = FeastQComplex[{A, Y}, {Lmin, Lmax}];
Aq = Transpose[Q].A.Q;
Bq = Transpose[Q].Q;
{evals, evecs} = Eigensystem[{Aq, Bq}];
{evals, Q.Transpose[evecs]}
]
```

Figure 22: Complex version of FEAST

5.1 Testing for Eigenvalues $\{1+i, 2+2i, 3-5i\}$

We tested the code on a complex diagonal matrix with eigenvalues $1+i, 2+2i, 3-5i$ and we got the result $\{3.-5.i, 2.+2.i, 1.+1.i\}$.

It is to be noted that no matter the interval for $M=2$ we always get all 3 correct eigenvalues. The error was found to be: $\{-2.6645352591003757 \times 10^{-15} - 8.881784197001252 \times 10^{-16}, 3.9968028886505635 \times 10^{-15} - 1.3322676295501878 \times 10^{-15} i, -1.3322676295501878 \times 10^{-15} - 8.881784197001252 \times 10^{-16} i\}$.

Then we tried to just find one eigenvalue $(3-5i)$ in the interval and we got the following results which is not accurate at all: $\{3.546 - 4.64787i, 1.83322 + 2.24231i\}$.

When we tried $M=3$ we got the correct eigenvalues: $\{3.-5.i, 2.+2.i, 1.+1.i, \text{Indeterminate}, \text{Indeterminate}\}$.

5.2 Testing for Eigenvalues $\{3-5i, 2-3i, 2+2i, 2-i, 1-i\}$

We tried to find 2 eigenvalues out of the 5 but we get 3 eigenvalues and the error is very large as can be seen from the results we obtained: $\{2.02814 - 1.9919i, 1.81547 + 1.86346i, 1.03128 - 1.13449i\}$.

Next we tried for $M=3$. We get all 5 eigenvalues and the error is negligible: $\{3.-5.i, 2.-3.i, 2.+2.i, 2.-1.i, 1.-1.i\}$.

6 Future Work

[12] The Complex FEAST algorithm looses its accuracy for larger complex matrices. We would like to find out why this happens. Also we would like to find a better way to give the search interval for the complex FEAST algorithm rather than giving λ_{min}, λ_{max}. Thirdly, it would be nice to test the complex version with full matrices.

Multiple questions popped up regaring the input range. Is there is a minimum range for λ_{min} and λ_{max}. Is there a maximum range? What happens as we decrease the range? Do we get more "ghost" eigenvalues? Does it fail? Does it get even more accurate?

Additional hopes are to:

1. Get the algorithm working for non-symmetric matrices.

[12] CS, SK, NW

2. Find out why it works better on small eigenvalues.

3. Find out the threshold for underestimating M.

4. Find out why sometimes we get complex results even though we take the real part.

References

[1] Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79:115112–115117, 2009.

[2] Eric Polizzi. *A High-Performance Numerical Library for Solving Eigenvalue Problems: FEAST Solver v2.0 User's Guide*, March 2012. http://arxiv.org/abs/1203.4031.