



Michigan Technological University
Create the Future Digital Commons @ Michigan Tech

Dissertations, Master's Theses and Master's
Reports - Open

Dissertations, Master's Theses and Master's
Reports

2014

Variations of the FEAST Eigenvalue Algorithm

Stephanie Kajpust
Michigan Technological University

Follow this and additional works at: <https://digitalcommons.mtu.edu/etds>



Part of the [Mathematics Commons](#)

Copyright 2014 Stephanie Kajpust

Recommended Citation

Kajpust, Stephanie, "Variations of the FEAST Eigenvalue Algorithm", Master's Thesis, Michigan Technological University, 2014.
<https://doi.org/10.37099/mtu.dc.etds/737>

Follow this and additional works at: <https://digitalcommons.mtu.edu/etds>



Part of the [Mathematics Commons](#)

VARIATIONS OF THE FEAST EIGENVALUE ALGORITHM

By

Stephanie Kajpust

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Mathematical Sciences

MICHIGAN TECHNOLOGICAL UNIVERSITY

2014

© 2014 Stephanie Kajpust

This thesis has been approved in partial fulfillment of the requirements for the Degree of
MASTER OF SCIENCE in Mathematical Sciences.

Department of Mathematical Sciences

Thesis Advisor: *Dr. Allan Struthers*

Committee Member: *Dr. Rebecca Schmitz*

Committee Member: *Dr. Jiguang Sun*

Committee Member: *Dr. Linda Ott*

Department Chair: *Dr. Mark Gockenbach*

Contents

List of Algorithms	ix
List of Figures	x
List of Tables	xvii
Acknowledgments	xviii
Preface	xix
Abstract	xx
1 Introduction	1
2 FEAST	3
2.1 Introduction to the Standard FEAST Algorithm	3
2.2 Performance of the Standard Feast Algorithm	10
2.2.1 Real Symmetric Matrix Creation	11
2.2.2 Testing Accuracy of Algorithm	12
2.2.3 Testing Effect of Eigenvalue Distribution	14

2.2.4	Orthonormal Modification	16
2.3	Hermitian Matrix Modification	23
2.3.1	Hermitian Matrix Creation	24
2.3.2	Testing Accuracy of Hermitian Version	24
2.4	Conclusions	25
3	Recursive FEAST Variation	27
3.1	Why a Recursive Algorithm?	27
3.2	Modifications of the Standard FEAST Algorithm	28
3.3	Code Details	29
3.3.1	Search Region and Contours	30
3.3.1.1	Formation of Rectangle Contour	30
3.3.1.2	Formation of Diamond Contour	31
3.3.2	Recursion Level	33
3.3.3	Test Matrix Creation	36
3.4	Testing The Algorithm with Rectangular Contours	36
3.4.1	Testing Real Eigenvalues	37
3.4.2	Testing Complex Eigenvalues	41
3.4.2.1	Random Eigenvalues	41
3.4.2.2	Clustered Eigenvalues	44
3.4.2.3	Complex-Conjugate Eigenvalues	44
3.5	Testing the Algorithm with Diamond Contours	49

3.5.1	Testing Real Eigenvalues	49
3.5.1.1	Random Eigenvalues	49
3.5.1.2	Clustered Eigenvalues	50
3.5.2	Testing Complex Eigenvalues	52
3.5.2.1	Random Eigenvalues	52
3.5.2.2	Clustered Eigenvalues	54
3.5.2.3	Complex-Conjugate Eigenvalues	56
3.6	Conclusions	60
4	Investigation of Quadrature Schemes	61
4.1	Mathematica Integration Schemes	61
4.2	Introduction to Quadrature Tests	64
4.2.1	Ellipse Parameterization	67
4.2.2	Rectangle Parameterization	67
4.2.3	Diamond Parameterization	69
4.3	Testing Quadrature on Circular Contours	69
4.3.1	Real Poles	70
4.3.2	Complex-Conjugate Poles	76
4.3.3	Complex Poles	83
4.4	Testing Quadrature on Rectangular Contours	90
4.4.1	Real Poles	91
4.4.2	Complex-Conjugate Poles	92

4.4.3	Complex Poles	94
4.5	Testing Quadrature on Diamond Contours	97
4.5.1	Real Poles	97
4.5.2	Complex-Conjugate Poles	101
4.5.3	Complex Poles	102
4.6	Conclusions	104
5	Conclusions	107
	References	109

List of Algorithms

1	Standard FEAST Algorithm for Real Symmetric Matrices	11
2	Random Real Symmetric Matrix Generator	11
3	FEAST Algorithm for Hermitian Matrices	23
4	Random Hermitian Matrix Generator	24
5	FEAST Recursive Algorithm	29
6	Rectangle Creation	30
7	Diamond Creation	32
8	Random Complex Matrix Generator	36
9	Random Real Matrix Generator	36
10	Complex-Conjugate Eigenvalue Generator	46

List of Figures

2.1	Projection schematic for Polizzi's FEAST	9
2.2	Projection schematic for FEAST with orthogonalization by SVD (U) or using the pseudoinverse (\widehat{PY}^\dagger)	10
2.3	Random matrix eigenvalue distribution: specific eigenvalues $\{1, 25, 50\}$, $n = 1000$	13
2.4	Matrix C eigenvalue distribution: 19 eigenvalues in $[1, 2]$, $n = 500$	15
2.5	Distribution of number of output eigenvalues from standard FEAST: random real symmetric matrix, $n = 500$, $m = 18$, $s = 27$	17
2.6	Gap in singular values of \hat{P} : random real symmetric matrix, $n = 500$, $m = 8$, $s = 12$	18
2.7	Distribution of found eigenvalues for standard FEAST, orthogonalized FEAST, and orthogonalized knowing m FEAST: $n = 500$, $m = 19$, $s = 29$.	19
2.8	Distribution of found eigenvalues for standard FEAST, orthogonalized FEAST, and orthogonalized knowing m FEAST: $n = 500$, $m = 18$, $s = 27$.	20
2.9	Distribution of found eigenvalues of matrix C for orthogonalized FEAST, and orthogonalized with a tolerances $10^{-1}, 10^{-2}$: $n = 500$, $m = 18$, $s = 27$.	20

2.10	Distribution of found eigenvalues for orthogonalized FEAST with various tolerances: $n = 500, m = 100, s = 150$	21
2.11	List plot of singular values for randomly generated 500×500 matrix with 100 eigenvalues in the range $[-101, 101]$: $m = 100, s = 150$	22
3.1	Rectangle recursive subdivision contour, $\lambda_1 = -3 - i, \lambda_2 = 1 + 2i$	31
3.2	Diamond recursive subdivision contour, $\lambda_1 = -3 - i, \lambda_2 = 1 + 2i$	32
3.3	All singular values of \hat{P} : random complex matrix, $n = 20, m = 1, s = 16$. .	34
3.4	All singular values of \hat{P} : random complex matrix, $n = 500, m = 1, s = 8$. .	35
3.5	All singular values of \hat{P} : random complex matrix, $n = 500, m = 1, s = 16$.	35
3.6	Recursive rectangular FEAST subdivision process: random real symmetric matrix, $n = 1000, m = 16, s = 16$, red dots indicate eigenvalues	38
3.7	Recursive rectangular FEAST subdivision process, zoomed in: random real symmetric matrix, $n = 1000, m = 16, s = 16$, red dots indicate eigenvalues .	39
3.8	Recursive rectangular FEAST subdivision process: random Hermitian matrix, $n = 1000, m = 29, s = 16$	40
3.9	Recursive rectangular FEAST subdivision process: random complex matrix, $n = 500, m = 25, s = 16$	42
3.10	Recursive rectangular FEAST subdivision process zoomed in: random complex matrix, $n = 500, m = 25, s = 16$	43
3.11	Recursive rectangular FEAST subdivision process: random complex matrix, clustered eigenvalues, $n = 500, m = 281, s = 16$	45

3.12 Recursive rectangular FEAST subdivision process: random complex matrix, clustered eigenvalues, maximum recursion depth 10, $n = 500$, $m = 281$, $s = 16$	46
3.13 Recursive rectangular FEAST subdivision process: random complex matrix, clustered eigenvalues, maximum recursion depth 15, $n = 500$, $m = 281$, $s = 16$	47
3.14 Recursive rectangular FEAST subdivision process: random complex matrix, complex conjugate eigenvalues, $n = 500$, $m = 18$, $s = 16$	48
3.15 Recursive diamond FEAST subdivision process: random real symmetric matrix, $n = 500$, $m = 20$, $s = 16$	50
3.16 Recursive diamond FEAST subdivision process: random real symmetric matrix zoomed in, $n = 500$, $m = 20$, $s = 16$	51
3.17 Recursive diamond FEAST subdivision process: random Hermitian matrix, $n = 500$, $m = 20$, $s = 16$	52
3.18 Recursive diamond FEAST subdivision process: real symmetric matrix, clustered eigenvalues, $n = 200$, $m = 19$, $s = 32$	53
3.19 Recursive diamond FEAST subdivision process: Hermitian matrix, clustered eigenvalues $n = 200$, $m = 19$, $s = 32$	54
3.20 Recursive diamond FEAST subdivision process: random complex matrix, $n = 500$, $m = 8$, $s = 16$	55

3.21	Recursive diamond FEAST subdivision process: random complex matrix, clustered eigenvalues, $n = 500, m = 241, s = 16$	56
3.22	Recursive diamond FEAST subdivision process zoomed in: random complex matrix, clustered eigenvalues, $n = 500, m = 241, s = 16$	57
3.23	Recursive diamond FEAST subdivision process: random complex matrix, complex conjugate eigenvalues, $n = 500, m = 19, s = 16$	58
3.24	Recursive diamond FEAST subdivision process: random complex matrix, complex conjugate eigenvalues, $n = 500, m = 19, s = 16$	59
4.1	Unit Square Contour	68
4.2	Unit Diamond Contour	70
4.3	Quadrature comparison: Aspect Ratio = 1, $q = 7$, real poles, circle contour .	71
4.4	Comparison of quadrature error: Aspect Ratio = 1, $q = 7$, real poles, circle contour	72
4.5	Comparisons of quadrature for different aspect ratios: $q = 7$, pole at $x_0 = 0$, circle contour	73
4.6	Quadrature comparison: Aspect Ratio = 0.5, $q = 7$, real poles, circle contour	74
4.7	Quadrature comparison: Aspect Ratio = 0.25, $q = 7$, real poles, circle contour	75
4.8	Quadrature comparison: Aspect Ratio = 0.1, $q = 7$, real poles, circle contour	75
4.9	Comparison of quadrature error: Aspect Ratio = 1, $q = 7$, complex-conjugate poles, circle contour	77

4.10 Dependence on aspect ratio: Clenshaw-Curtis Quadrature, $q = 7$, complex-conjugate poles, circle contour	79
4.11 Dependence on aspect ratio: Gauss-Legendre Quadrature, $q = 7$, complex-conjugate poles, circle contour	80
4.12 Dependence on aspect ratio: Trapezoid Quadrature, $q = 7$, complex-conjugate poles, circle contour	81
4.13 Dependence on aspect ratio: Lobatto-Kronrod Quadrature, $q = 7$, complex-conjugate poles, circle contour	82
4.14 Comparison of quadrature error: Aspect Ratio = 1, $q = 7$, complex poles, circle contour	84
4.15 Dependence on aspect ratio: Clenshaw-Curtis Quadrature, $q = 7$, complex poles, circle contour	86
4.16 Dependence on aspect ratio: Gauss-Legendre Quadrature, $q = 7$, complex poles, circle contour	87
4.17 Dependence on aspect ratio: Trapezoid Quadrature, $q = 7$, complex poles, circle contour	88
4.18 Dependence on aspect ratio: Lobatto-Kronrod Quadrature, $q = 7$, complex poles, circle contour	89
4.19 Quadrature comparison: Aspect Ratio = 1, real eigenvalues, rectangle contour	91

4.20 Comparisons of quadrature for different aspect ratios: real eigenvalues, rectangle contour, $x_0 = 0$	92
4.21 Comparison of quadrature error: Aspect Ratio = 1, complex-conjugate eigenvalues, rectangle contour	93
4.22 Comparison of quadrature error: Aspect Ratio = 1, complex eigenvalues, rectangle contour	95
4.23 Comparison of quadrature error: Aspect Ratio = 0.5, complex eigenvalues, rectangle contour	96
4.24 Quadrature comparison: Aspect Ratio = 1, real eigenvalues, diamond contour	97
4.25 Comparisons of quadrature for different aspect ratios: real eigenvalues, diamond contour, $x_0 = 0$	98
4.26 Quadrature comparison: Aspect Ratio = 0.6, real eigenvalues, diamond contour	99
4.27 Quadrature comparison: Aspect Ratio = 0.45, real eigenvalues, diamond contour	100
4.28 Quadrature comparison: Aspect Ratio = 0.15, real eigenvalues, diamond contour	100
4.29 Comparison of quadrature error: Aspect Ratio = 1, complex-conjugate eigenvalues, diamond contour	102
4.30 Comparison of quadrature error: Aspect Ratio = 1, complex eigenvalues, diamond contour	103

4.31 Comparison of quadrature error: Aspect Ratio = 0.5, complex eigenvalues,
diamond contour 104

List of Tables

2.1	Eigenvalue accuracy of standard FEAST: real symmetric matrix, $n = 1000$, $s = 2, m = 1$, 100 runs.	13
2.2	Eigenvalue accuracy: real symmetric matrices, 100 runs.	14
2.3	Standard FEAST: Number of eigenvalues returned, real symmetric matrix, $n = 500$, 100 runs.	16
2.4	Eigenvalue accuracy for orthogonalized FEAST: real symmetric matrices, 100 runs on different matrices	22
2.5	Eigenvalue accuracy: Hermitian matrices, 100 runs	24
4.1	Dependence of quadrature node count on Mathematica argument n	63
4.2	Quadrature scheme type	64
4.3	Quadrature scheme accuracy as a function of Mathematica argument	64

Acknowledgments

I'd like to thank Allan for meeting with me every day to help me get my thesis become a professional and complete document. Thank you for keeping me organized and on track when I got lost among all the details.

Thank you Rebecca, for helping me out with the application to Finlandia. Without the stellar editing job, I wouldn't have gotten the job and had the motivation to finish my thesis in time.

A huge thanks goes out to Steve for supporting me through this whole process. You always believed in me (even when you were being a pain in the behind) and I'm very grateful.

Mom and Dad: thanks for supporting me when I decided to completely change my life and get a degree in teaching math. Without your support I wouldn't have come on the path I'm on now, and found out that teaching college is my dream job.

A shout-out goes to CJ and Nathasha, who helped me on the project that became my thesis.

Preface

In MA5627 Numerical Linear Algebra, I worked on a project which implemented an eigenvalue-finding algorithm named FEAST [7]. The goal of the project was to see how well this algorithm worked in practice. In implementing the algorithm, many questions arose which became the basis and direction of this research. At the end I ended up in a very different place than I expected when I began in Spring 2012.

This thesis is in three parts: exploration of the standard FEAST algorithm, recursive extensions of FEAST, and examination of various quadrature schemes. I have organized this in essentially chronological order, so you can follow along in the journey I went on during my research.

Abstract

FEAST is a recently developed eigenvalue algorithm which computes selected interior eigenvalues of real symmetric matrices. It uses contour integral resolvent based projections. A weakness is that the existing algorithm relies on accurate reasoned estimates of the number of eigenvalues within the contour. Examining the singular values of the projections on moderately-sized, randomly-generated test problems motivates orthogonalization-based improvements to the algorithm. The singular value distributions provide experimentally robust estimates of the number of eigenvalues within the contour. The algorithm is modified to handle both Hermitian and general complex matrices. The original algorithm (based on circular contours and Gauss-Legendre quadrature) is extended to contours and quadrature schemes that are recursively subdividable. A general complex recursive algorithm is implemented on rectangular and diamond contours. The accuracy of different quadrature schemes for various contours is investigated.

Chapter 1

Introduction

The FEAST eigenvalue algorithm as described by Eric Polizzi [7] computes eigenvalues of real symmetric matrices in a specified interval of the real axis. It creates small matrices containing the eigenvalues in the interval using contour integrals in the complex plane. Polizzi [7] uses circular contours centered on the real axis, and exploits symmetry to reduce the contour to a semi-circle in the upper half plane. He approximates the contour integrals using Gauss-Legendre quadrature. A weakness of the existing algorithm is that it requires an accurate estimate of the number of targeted eigenvalues. In contrast to most traditional algorithms, the FEAST eigenvalue algorithm exposes hierarchical parallelism: multiple parallel linear solves, each with multiple right hand sides.

The following questions arose:

- Can the FEAST algorithm be used on non-symmetric matrices?
- What if instead of circles for contours, the algorithm used rectangles or diamonds?
- What if the algorithm did not use “unit” shapes, but rather variations of the shapes?
- Is using Gauss-Legendre quadrature really the best thing to do on circles?
- Is Simpson’s Rule the best quadrature to use on rectangles and diamonds?
- Can the weakness in the algorithm (foreseeing how many eigenvalues are in the interval) be removed by making the algorithm recursive?
- Can the algorithm work in parallel?

Chapter 2

FEAST

2.1 Introduction to the Standard FEAST Algorithm

The original FEAST algorithm [7] computes generalized eigenvalues and eigenvectors [12] for square $n \times n$ matrices A and B , where A is a Hermitian matrix and B is symmetric positive definite. In other words it computes eigenvalues λ and eigenvectors \vec{v} satisfying

$$A\vec{v} = \lambda B\vec{v} \tag{2.1}$$

In all computations we will use $B = I$ (which recovers the standard eigenvalue problem) but several of the arguments are more general and so B is included. When various hypotheses are necessary they are noted in footnotes.

Generalized eigenvalues and eigenvectors satisfy

$$A\vec{v} - \lambda B\vec{v} = \vec{0} \implies (A - \lambda B)\vec{v} = \vec{0} \quad (2.2)$$

which motivates the definition of the resolvent [5]

$$(A - zI)^{-1} \quad (2.3)$$

which for the generalized eigenvalue problem is

$$(A - zB)^{-1} \quad (2.4)$$

To avoid negative signs Polizzi [7] uses the variation

$$R(z) = (zB - A)^{-1} \quad (2.5)$$

Integrating the resolvent $R(z)$ around a simple closed contour γ in the complex plane [6]

and dividing by $2\pi i$ defines the $n \times n$ matrix

$$P = \frac{1}{2\pi i} \oint_{\gamma} (zB - A)^{-1} dz \quad (2.6)$$

which is essentially a projection and contains complete spectral information about the

eigenvalues within γ .

We will show why this is the case.

Pre-multiplying (2.1) by B^{-1} gives a standard eigenvalue problem for $B^{-1}A$

$$B^{-1}A\vec{v} = \lambda\vec{v} \quad (2.7)$$

Since $B^{-1}A$ is Hermitian the standard eigenvalue decomposition [12] is

$$B^{-1}A = Q\Lambda Q^T \implies A = BQ\Lambda Q^T \quad (2.8)$$

where Λ is the diagonal matrix of eigenvalues $\lambda_1, \dots, \lambda_n$ and Q is the orthogonal eigenvector¹.

Substituting (2.8) into (2.6) gives

$$\begin{aligned} P &= \frac{1}{2\pi i} \oint_{\gamma} (zB - BQ\Lambda Q^T)^{-1} dz = \frac{1}{2\pi i} \oint_{\gamma} (zI - Q\Lambda Q^T)^{-1} B^{-1} dz \\ &= \frac{1}{2\pi i} \oint_{\gamma} (Q[zI - \Lambda]Q^T)^{-1} B^{-1} dz = \frac{1}{2\pi i} \oint_{\gamma} Q[zI - \Lambda]^{-1} Q^T B^{-1} dz \\ &= QZ_{\gamma}Q^T B^{-1} \end{aligned} \quad (2.9)$$

¹If $B^{-1}A$ exists and is diagonalizable $A = BX\Lambda X^{-1}$ where Λ is as before and X is the matrix of non-orthogonal eigenvectors.

where

$$Z_\gamma = \begin{bmatrix} \frac{1}{2\pi i} \oint_\gamma \frac{1}{z - \lambda_1} dz & 0 & \dots & 0 \\ 0 & \frac{1}{2\pi i} \oint_\gamma \frac{1}{z - \lambda_2} dz & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{2\pi i} \oint_\gamma \frac{1}{z - \lambda_n} dz \end{bmatrix} \quad (2.10)$$

This argument reproduces in concise mathematical language the quantum mechanical arguments given by Polizzi in [7]. Note in [10] Polizzi modifies his construction to

$$\frac{1}{2\pi i} \oint_\gamma (R(z))^{-1} B dz \quad (2.11)$$

to remove the B^{-1} , where $R(z)$ is as given in (2.5).

By the Cauchy Integral Theorem [6] each integral in (2.10) evaluates to either 0 or 1: 1 if λ_j ($j = 1, \dots, n$) is inside γ and 0 otherwise.

A projector [12] E satisfies $E^2 = E$. Since $(Z_\gamma)^2 = Z_\gamma$, then Z_γ is a projector in the coordinate system defined by the orthonormal eigenvectors. Clearly $PB = QZ_\gamma Q^T$ (from

(2.9)) is a projection in the original coordinate system ² since

$$(QZ_\gamma Q^T)^2 = QZ_\gamma Q^T QZ_\gamma Q^T = Q(Z_\gamma)^2 Q^T = QZ_\gamma Q^T \quad (2.12)$$

P is not analytically or numerically computable for large matrices A and B . However, the approximate action of P on a subspace is computable using linear solves and numerical quadrature. Given a sample of s n -vectors ($s \ll n$) we assemble the $n \times s$ sample matrix Y . The action of P on Y is

$$PY = \frac{1}{2\pi i} \oint_{\gamma} (zB - A)^{-1} Y dz \quad (2.13)$$

For the standard parameterization of a circle $z = e^{i\theta}$ this gives us

$$PY = \frac{1}{2\pi i} \int_0^{2\pi} (e^{i\theta} B - A)^{-1} Y (ie^{i\theta}) d\theta \quad (2.14)$$

Approximating the integral numerically with a q -point quadrature scheme gives

$$\widehat{PY} = \frac{1}{2\pi i} \sum_{i=1}^q \omega_i (z_i B - A)^{-1} Y \quad (2.15)$$

where $\vec{\omega} = (\omega_1, \dots, \omega_q)$ are the quadrature weights and $\vec{z} = (z_1, \dots, z_q)$ combines the

²For diagonalizable $B^{-1}A$ the result is the same since $XZ_\gamma X^{-1}XZ_\gamma X^{-1} = XZ_\gamma X^{-1}$.

quadrature nodes and the contour parameterization.

Polizzi [7, 10] uses a symmetrized 8-point Gaussian quadrature scheme on a circular contour. Other quadrature schemes and contours are possible. Implementing (2.15) in code requires q linear solves of distinct $n \times n$ systems each with s right-hand sides.

Let m be the number of eigenvalues within γ . If $s > m$ the s vectors in \widehat{PY} span the m -dimensional eigenspace with high probability. Polizzi [7] iterates his projection j times until he obtains a matrix $P_{pol} = P^j Y$ with approximately orthogonal columns. Polizzi uses the orthogonal projection onto the space this defines to compute the eigenvalues and eigenvectors of $A_{pol} = (P_{pol})^T A P_{pol}$ and $B_{pol} = (P_{pol})^T B P_{pol}$.

Polizzi and Tang [10] state,

Intuitively, these m eigenvalues will be among the p eigenvalues of the reduced problem $A_Q z = \lambda z$, $A_Q = Q^H A Q$ or that of the [generalized Hermitian eigenvalue problem] $A_Y w = \lambda B_Y w$, $A_Y = Y^H A Y$ and $B_Y = Y^H B Y$. The corresponding eigenvectors will be among the p vectors Qz or Yw , respectively.³

Figure 2.1 shows how the projection works for Polizzi [7].

³ A_Q is our A_{pol} , Q is our P_{pol} , Q^H indicates the complex-conjugate transpose, and p is used to denote the sample size

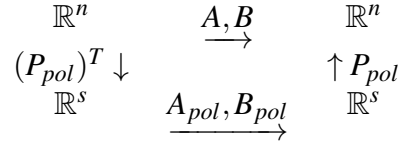


Figure 2.1: Projection schematic for Polizzi’s FEAST

For nonsingular matrices, equivalent pencils have the same eigenvalues [3]. Since P_{pol} is singular, A_{pol} and B_{pol} are not considered equivalent. However, since A_{pol} and B_{pol} are built with a projection that spans the eigenspace, their Ritz values are good approximations to the eigenvalues of A and B [12].

One of our discoveries is that aggressive orthogonalization can approximate the number of eigenvalues m and the eigenvalues and eigenvectors with one iteration. We construct orthogonal eigenvector candidates using the Singular Value Decomposition (SVD) as suggested in a nonlinear eigenvalue setting by Beyn [2]. We select candidate eigen-directions that are significantly represented in the SVD of \widehat{PY} . In Mathematica this is sometimes implemented using `SingularValueDecomposition` [4] calls and sometimes implemented as `PseudoInverse` [4] calls. Note both calls involve a tolerance (Mathematica’s default is 10^{-14} times the largest singular value) below which singular values are neglected. We examined the singular value distributions to identify appropriate relative tolerances (typically much larger than 10^{-14}) for different matrix types, matrix dimensions, and sample sizes. We also experimented with selecting known numbers of vectors.

The projection scheme used in code is demonstrated as:

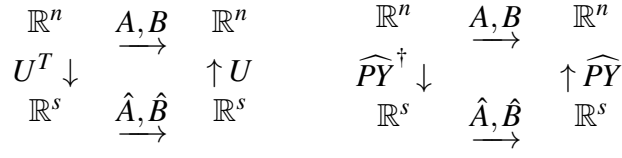


Figure 2.2: Projection schematic for FEAST with orthogonalization by SVD (U) or using the pseudoinverse (\widehat{PY}^\dagger)

2.2 Performance of the Standard Feast Algorithm⁴

The standard FEAST algorithm is shown in Algorithm 1. Polizzi [7] gives the following recommendations for input variables to the algorithm:

- Knowing m eigenvalues lie in interval $[\lambda_{min}, \lambda_{max}]$, choose $s \geq 1.5m$.
- Using Gauss-Legendre quadrature points, choose $q = 8$.

The following tests are run with $B = I$ in (2.1) and no refinement done. The algorithm was coded in Mathematica [4], where the built-in LinearSolve, Eigensystem, Transpose, and SingularValueDecomposition commands could be utilized.

⁴Contributions to research by Chathuri Samarasinghe and Nathasha Weerasinghe.

Algorithm 1 Standard FEAST Algorithm for Real Symmetric Matrices

Given $A^{n \times n}, B^{n \times n}, \lambda_{\min}, \lambda_{\max} \in \mathbb{R}$, and $m \in \mathbb{R}$
 $s \leftarrow 1.5m$
 $Y^{n \times s} \in \mathbb{R} \leftarrow$ random vector
 $\hat{P} \in \mathbb{R}^{n \times s} \leftarrow 0$
 $r = 0.5(\lambda_{\min} + \lambda_{\max})$
for $j = 1, q$ **do**
 $\theta_j \leftarrow -\frac{\pi}{2}(\alpha_j - 1)$ where α_j is the node for Gauss quadrature
 $z_j \leftarrow 0.5(\lambda_{\min} + \lambda_{\max}) + re^{i\theta_j}$
 Solve $(z_j B - A)p_j = Y$
 $\hat{P} \leftarrow \hat{P} + 0.5\omega_j \Re(re^{i\theta_j} p_j)$ where ω_j is the weight for Gauss quadrature
end for
 $\hat{A} \leftarrow \hat{P}^T A \hat{P}$
 $\hat{B} \leftarrow \hat{P}^T B \hat{P}$
Find eigenvalues and eigenvectors of $\hat{A}\vec{v} = \lambda \hat{B}\vec{v}$

2.2.1 Real Symmetric Matrix Creation

Real symmetric test matrices with specific eigenvalues were randomly generated using the eigenvalue decomposition [12]; pseudocode is in Algorithm 2.

Algorithm 2 Random Real Symmetric Matrix Generator

Given list of eigenvalues L
 $n \leftarrow \text{Length}(L)$
 $A \leftarrow \text{RandomReal}[\{-1, 1\}, \{n, n\}]$
 $Q \leftarrow$ from QR decomposition of A
 $\tilde{A} \leftarrow Q \cdot \text{Diag}(L) \cdot Q^T$
 $A \leftarrow \frac{1}{2}(\tilde{A} + \tilde{A}^T)$ ensures the matrix is symmetric in floating-point

2.2.2 Testing Accuracy of Algorithm

Eigenvalue accuracy is measured using the relative residual

$$\frac{\|\vec{\lambda}_{actual} - \vec{\lambda}_{calculated}\|_2}{\|\vec{\lambda}_{actual}\|_2} \quad (2.16)$$

where $\vec{\lambda}_{actual}$ is a vector containing the actual eigenvalues of the matrix, and $\vec{\lambda}_{calculated}$ is a vector containing the eigenvalues found with FEAST. Both vectors are sorted so $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_s|$.

The initial testing was done on a test matrix for size $n = 1000$. The eigenvalues were distributed uniformly in the interval $[-1000, 100] \cup [100, 1000]$. Three of the eigenvalues were specifically set at $\{1, 25, 50\}$. Tests were run to find the three specific eigenvalues. By having the uniformly distributed eigenvalues far away from the eigenvalues of interest, we ensured they would not affect the results of the calculations. The additional advantage of the uniform distribution is that there is no clusters of eigenvalues, which generally pose problems in eigenvalue algorithms. Figure 2.3 shows the eigenvalue distribution of this matrix, with the eigenvalues sorted so $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$. There is a clear gap between the eigenvalues of interest and the remaining eigenvalues.

For this matrix, one eigenvalue was sought. With $m = 1$, sample size $s = 2$, and 100 runs of FEAST on the same matrix, Table 2.1 shows accuracy near machine precision (10^{-16} in

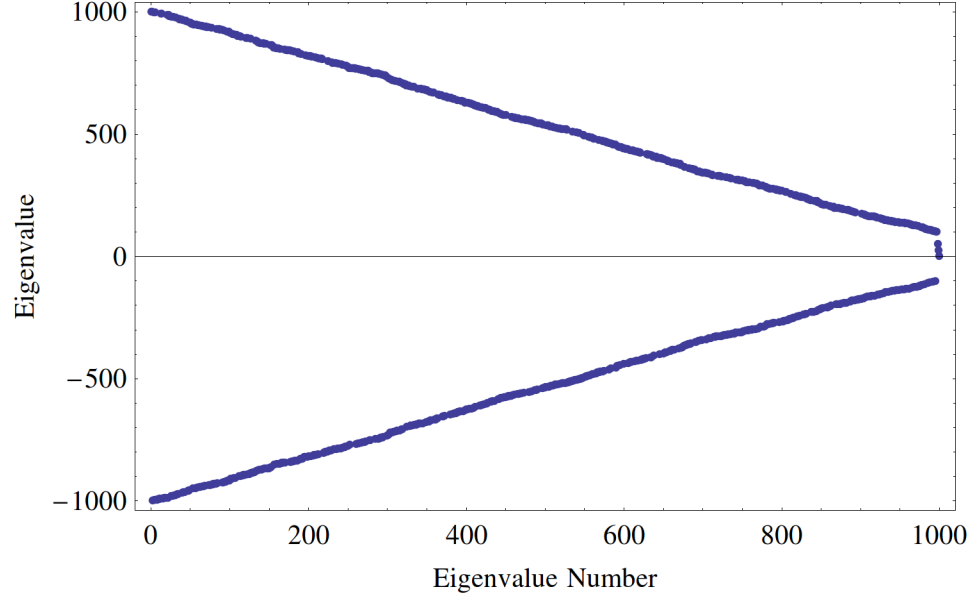


Figure 2.3: Random matrix eigenvalue distribution: specific eigenvalues $\{1, 25, 50\}$, $n = 1000$

Mathematica [4]).

Table 2.1

Eigenvalue accuracy of standard FEAST: real symmetric matrix, $n = 1000$, $s = 2$, $m = 1$, 100 runs.

λ_{min}	λ_{max}	m	Mean Residual	Max Residual
-1	5	1	1.86939×10^{-12}	2.88147×10^{-12}
20	30	1	6.2684×10^{-15}	1.93268×10^{-14}
45	55	1	2.5929×10^{-14}	3.33955×10^{-14}

This test was repeated on larger matrices generated by Algorithm 2. The goal of this test was to investigate the accuracy of FEAST for different matrix sizes, where the eigenvalues may not necessarily be far apart from each other. The search interval for each run was $[10, 50]$ was used each time ($\lambda_{min} = 10$ and $\lambda_{max} = 50$). Eigenvalues for each matrix were uniformly distributed in $[-1000, 1000]$. Each run used a different randomly-generated matrix. Results are in Table 2.2. As the matrix size gets larger, the gaps between the

eigenvalues decrease, due to how we set up the matrices. The table shows that the accuracy decreases as n increases, which is most likely due to the eigenvalues being closer together. It is known that the gaps between eigenvalues can affect the accuracy of eigenvalue calculations.

Table 2.2
Eigenvalue accuracy: real symmetric matrices, 100 runs.

n	Mean Residual	Max Residual
100	3.68326×10^{-13}	2.1502×10^{-11}
250	1.16823×10^{-8}	3.59027×10^{-7}
500	1.23188×10^{-7}	5.38811×10^{-7}
1000	1.12556×10^{-8}	2.04451×10^{-7}
2000	1.12556×10^{-8}	2.04451×10^{-7}

2.2.3 Testing Effect of Eigenvalue Distribution

Call matrix C a random real symmetric $n = 500$ matrix with 19 eigenvalues clustered in the range $[1,2]$ generated by

$$1 + \frac{1}{2^i} \quad i = 0, \dots, 18 \tag{2.17}$$

and the rest of the eigenvalues uniformly distributed in $[-1000, 200] \cup [200, 1000]$. Figure 2.4 shows a large gap between the clustered eigenvalues and the others. Finding all 19 clustered eigenvalues should be easy. However, finding 18 of them within an interval might be hard.

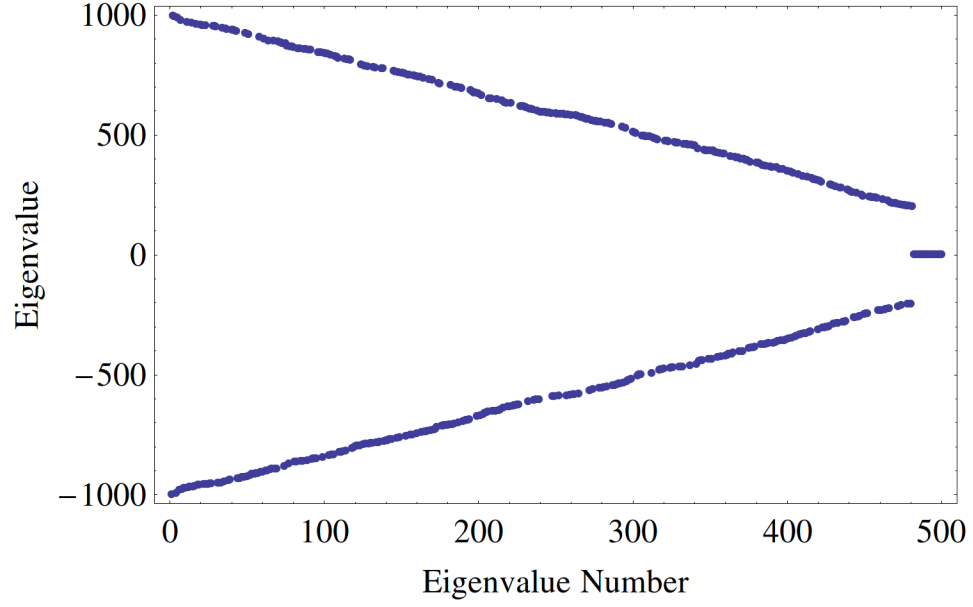


Figure 2.4: Matrix C eigenvalue distribution: 19 eigenvalues in $[1,2]$, $n = 500$

Table 2.3 shows the mean and mode number of eigenvalues found for matrix C using the standard FEAST algorithm. Polizzi [7] uses a strict cut-off so any eigenvalues on the edge of the interval may be missed if they are off by a few decimal places. The same goes for eigenvalues just inside the interval: they may be off enough to fall inside the cut-off, resulting in too many eigenvalues inside the interval. This is an issue when one does not know how many eigenvalues to expect.

For 1000 runs on C , Figure 2.5 shows that for $m = 18$ and $s = 27$, 25 eigenvalues are the most frequent output from FEAST. Since the purpose of FEAST is to find eigenvalues, there is no way of knowing which of the extra values returned are not actually eigenvalues of the original matrix.

Table 2.3

Standard FEAST: Number of eigenvalues returned, real symmetric matrix,
 $n = 500$, 100 runs.

λ_{\min}	λ_{\max}	m	s	Mean	Mode
1.75	3	1	2	1.	1
1.375	3	2	3	2.	2
1.1875	3	3	5	3.	3
1.09375	3	4	6	3.99	4
1.04688	3	5	8	5.	5
1.02344	3	6	9	5.48	5
1.01172	3	7	11	6.24	6
1.00586	3	8	12	7.08	7
1.00293	3	9	14	8.18	8
1.00146	3	10	15	9.17	9
1.00073	3	11	17	10.66	11
1.00037	3	12	18	11.84	12
1.00018	3	13	20	13.88	14
1.00009	3	14	21	15.75	16
1.00005	3	15	23	18.51	19
1.00002	3	16	24	20.49	21
1.00001	3	17	26	23.28	23
1.000006	3	18	27	25.03	25

The obvious problem with the algorithm is the need to input m . Running with $s < m$ does not catch all the eigenvalues, and running with $s \geq m$ does not guarantee the correct number of values returned.

2.2.4 Orthonormal Modification

A solution to the extra eigenvalue problem is to compute the SVD of \hat{P}

$$\hat{P} = U\Sigma V^T \tag{2.18}$$

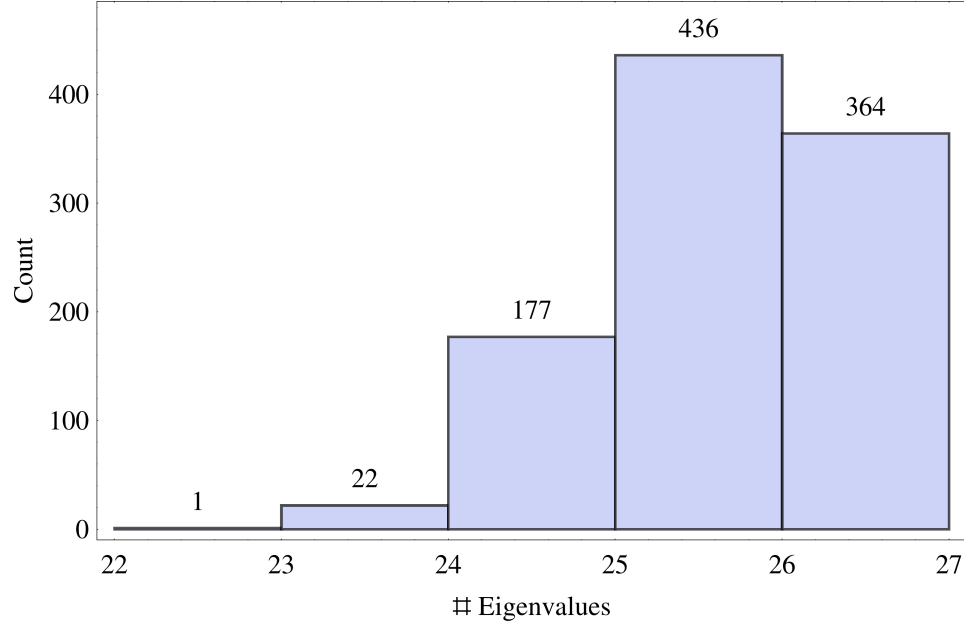


Figure 2.5: Distribution of number of output eigenvalues from standard FEAST: random real symmetric matrix, $n = 500$, $m = 18$, $s = 27$

where $U \in \mathbb{R}^{s \times s}$ is orthogonal, $\Sigma \in \mathbb{R}^{s \times s}$ is a diagonal matrix of eigenvalues, and $V \in \mathbb{R}^{s \times s}$ is also orthogonal.

Take an $n = 500$ randomly generated real symmetric matrix with $m = 8$ eigenvalues in the interval $[10, 50]$. Looking at a log-plot of the singular values of \hat{P} (in Figure 2.6) there is a large gap in magnitude after 8 values, corresponding with m .

U from the SVD is formed of unit vectors in the direction of \hat{P} [11]. These vectors span the output space, which should be the eigenspace. V spans the input space which has the potential to be affected badly by the random vector Y used in the standard FEAST algorithm. The advantage to using the SVD is that Mathematica's [4] `SingularValueDecomposition` command can be given a tolerance with which to determine

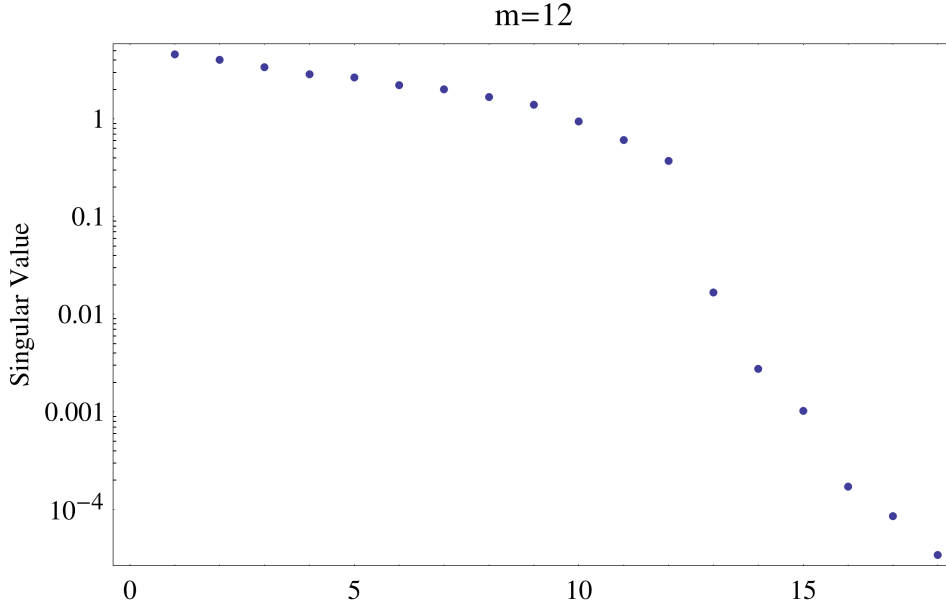


Figure 2.6: Gap in singular values of \hat{P} : random real symmetric matrix, $n = 500, m = 8, s = 12$

when a singular value is considered zero. The default is of magnitude 10^{-14} . Setting the tolerance to 0 results in U being fully orthogonal based on all the singular values of \hat{P} . A hard tolerance can be set by telling the `SingularValueDecomposition` to only use m number of singular values and to set everything else to zero. A soft tolerance can be set by choosing a magnitude based on the singular values.

Using matrix C from earlier, Figure 2.7 shows the distribution of the number of eigenvalues found for standard FEAST, the modification using U with a hard cut-off, and the modification using U with a tolerance of zero. The interval was $[-1, 3]$ which includes all 19 clustered eigenvalues. The standard FEAST algorithm finds extra eigenvalues, while orthogonalizing finds the exact number in the interval.

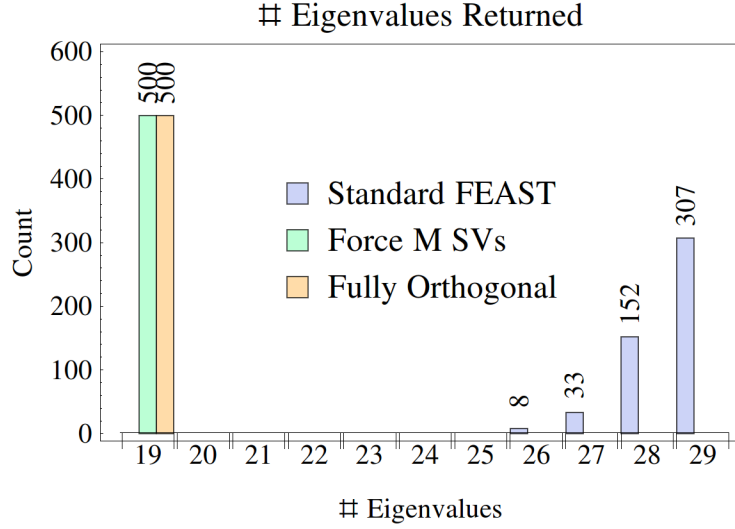


Figure 2.7: Distribution of found eigenvalues for standard FEAST, orthogonalized FEAST, and orthogonalized knowing m FEAST: $n = 500$, $m = 19$, $s = 29$

Figure 2.8 shows that for the same matrix and interval $[1.000005, 3]$ which has $m = 18$ eigenvalues inside (one eigenvalue is just outside the interval), using the hard cut-off actually misses an eigenvalue most of the time, due to the one just outside the interval influencing the computations. Fully orthogonalizing gives the exact number once again, showing it is not affected by clustering. Clearly orthogonalizing \hat{P} improves the algorithm and eliminates the need to accurately know m .

A soft tolerance could be more beneficial than fully orthogonalizing, as it means orthogonalizing fewer columns of \hat{P} . Figure 2.6 shows 10^{-1} could be a reasonable setting for a tolerance, as the number of eigenvalues in the interval m is equal to the number of singular values that are greater than 10^{-1} . To test this, the orthogonalized FEAST was run on matrix C with 500 randomly generated matrices (with the same eigenvalues). The goal was 18 eigenvalues as before, and figure 2.9 shows that a tolerance of 10^{-1} is fine for

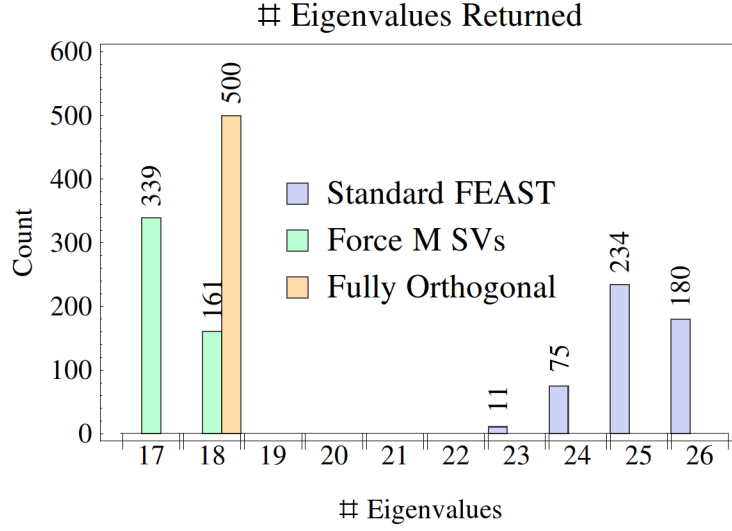


Figure 2.8: Distribution of found eigenvalues for standard FEAST, orthogonalized FEAST, and orthogonalized knowing m FEAST: $n = 500$, $m = 18$, $s = 27$

catching all 18. This tolerance shows that any singular value under 0.1 can be considered zero, which is quite generous.

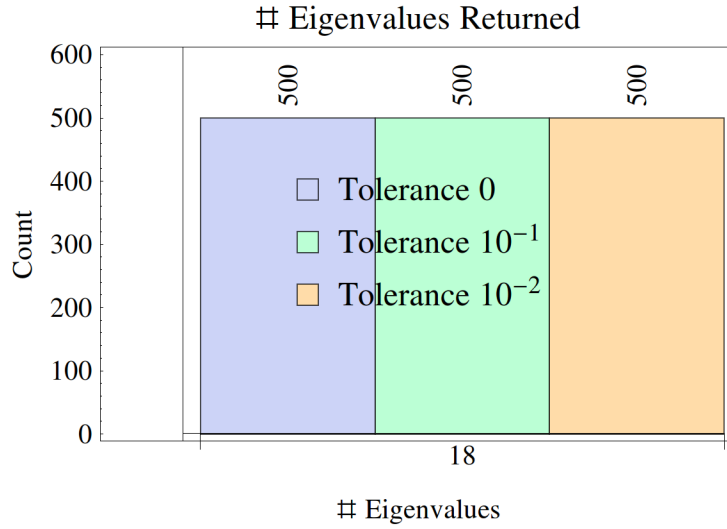


Figure 2.9: Distribution of found eigenvalues of matrix C for orthogonalized FEAST, and orthogonalized with a tolerances 10^{-1} , 10^{-2} : $n = 500$, $m = 18$, $s = 27$

To verify this tolerance works with different eigenvalue distributions, standard FEAST plus

orthogonalized FEAST with tolerances of 10^{-1} and 10^{-2} were run on randomly generated matrices with 100 eigenvalues in the search interval of $[-101, 101]$. For each of 500 runs, a different matrix was used with different eigenvalues. The only thing constant with the size of the matrices, the number of eigenvalues in the interval, and the sample size ($s = 150$). Figure 2.10 shows that a tolerance of 10^{-1} is again sufficient to get the correct number of eigenvalues returned. This is confirmed with a plot of the singular values for one of the matrices generated in this test. Figure 2.11 shows that the gap between the singular values is quite obvious at the 100th singular value.

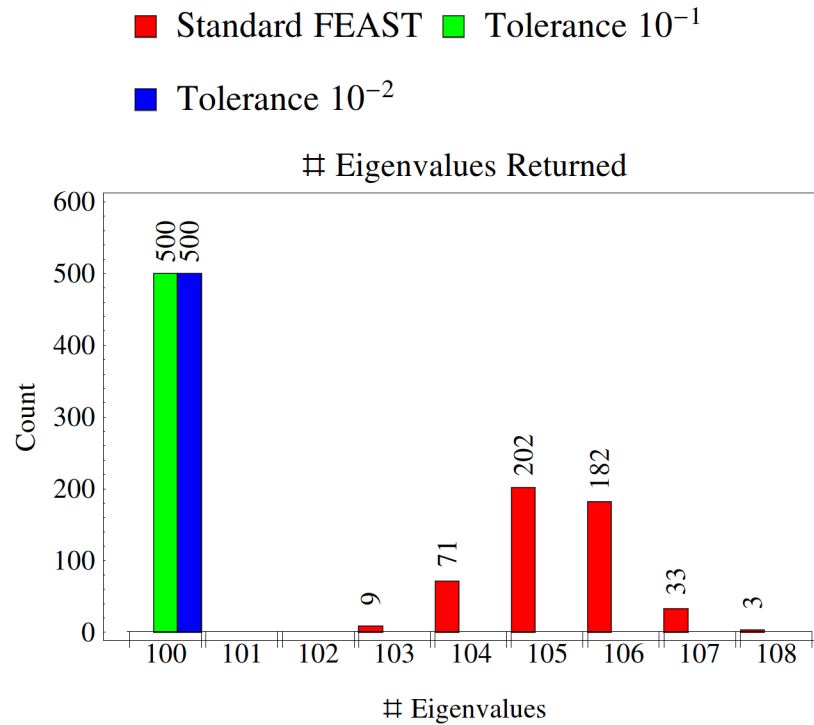


Figure 2.10: Distribution of found eigenvalues for orthogonalized FEAST with various tolerances: $n = 500$, $m = 100$, $s = 150$

Table 2.4 compares the maximum residual for standard FEAST with the orthogonalized version using a tolerance of 10^{-2} . For each matrix size, the maximum residual was

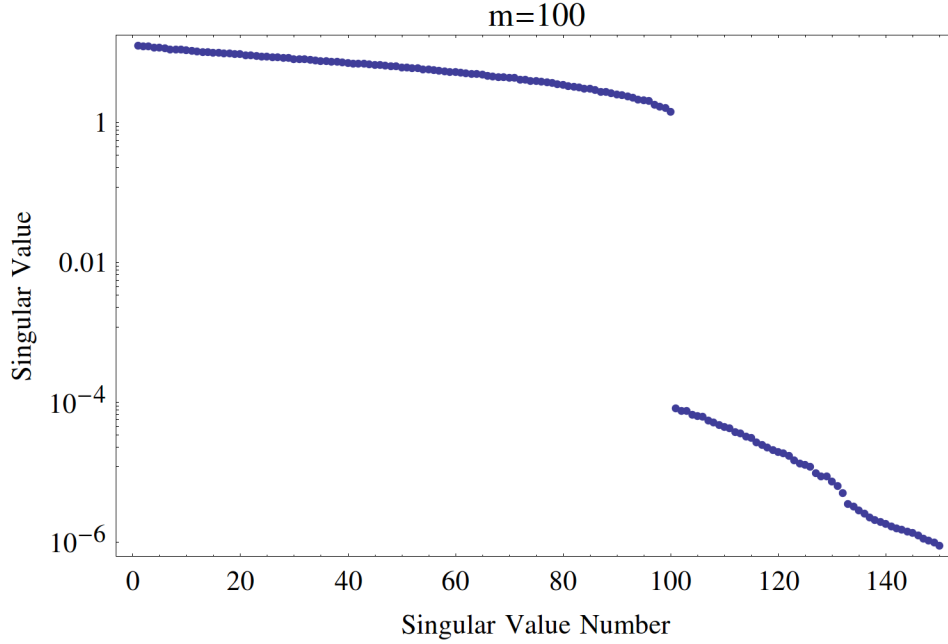


Figure 2.11: List plot of singular values for randomly generated 500×500 matrix with 100 eigenvalues in the range $[-101, 101]$: $m = 100$, $s = 150$

found out of 100 real symmetric matrices with eigenvalues uniformly distributed between $[-1000, 1000]$.

Table 2.4

Eigenvalue accuracy for orthogonalized FEAST: real symmetric matrices, 100 runs on different matrices

n	Max Residual Without Orthogonalization	Max Residual With Orthogonalization
100	9.8753×10^{-10}	7.22141×10^{-14}
250	505733×10^{-3}	5.16829×10^{-14}
500	2.03409×10^{-6}	4.52094×10^{-14}

2.3 Hermitian Matrix Modification

Polizzi [7] describes how to convert the standard FEAST algorithm so that it handles Hermitian matrices. As in the standard FEAST algorithm, no orthogonalizing is done. Our modification shown in algorithm 3 gives the pseudocode, with orthogonalization included since it was shown in previous sections that it eliminates extra eigenvalues being returned. The important different between the real version and the hermitian version is that in the hermitian version the whole contour is integrated, as it is not necessarily symmetric. The integration points used are complex-conjugates of each other.

Algorithm 3 FEAST Algorithm for Hermitian Matrices

```

Given  $A^{n \times n}, B^{n \times n}, \lambda_{\min}, \lambda_{\max} \in \mathbb{R}$ , and  $m \in \mathbb{R}$ 
 $s \leftarrow 1.5m$ 
 $Y^{n \times s} \in \mathbb{C} \leftarrow$  random vector
 $\hat{P} \in \mathbb{C}^{n \times s} \leftarrow 0$ 
 $r = 0.5(\lambda_{\min} + \lambda_{\max})$ 
for  $j = 1, q$  do
     $\theta_j \leftarrow -\frac{\pi}{2}(\alpha_j - 1)$  where  $\alpha_j$  is the node for Gauss quadrature
     $z_j \leftarrow 0.5(\lambda_{\min} + \lambda_{\max}) + re^{i\theta_j}$ 
    Solve  $(z_j B - A)p_j = Y$ 
    Solve  $(\bar{z}_j B - A)\tilde{p}_j = Y$  where  $\bar{z}_j$  is the conjugate
     $\hat{P} \leftarrow \hat{P} - 0.25\omega_j(re^{i\theta_j}p_j + re^{-i\theta_j}\tilde{p}_j)$  where  $\omega_j$  is the weight for Gauss quadrature
end for
 $\{U, \Sigma, V\} \leftarrow \text{SingularValueDecomposition}[\hat{P}]$ 
 $\hat{A} \leftarrow U^T A U$ 
 $\hat{B} \leftarrow U^T B U$ 
Find eigenvalues and eigenvectors of  $\hat{A}\vec{v} = \lambda\hat{B}\vec{v}$ 

```

2.3.1 Hermitian Matrix Creation

Hermitian matrices are randomly created with code from Algorithm 4.

Algorithm 4 Random Hermitian Matrix Generator

Given list of eigenvalues L
 $n \leftarrow \text{Length}(L)$
 $A \leftarrow \text{RandomComplex}[\{-1 - i, 1 + i\}, \{n, n\}]$
 $Q \leftarrow$ from QR decomposition of A
 $\tilde{A} \leftarrow Q.\text{Diag}(L).Q^*$ where $*$ indicates the conjugate transpose
 $A \leftarrow \frac{1}{2}(\tilde{A} + \tilde{A}^T)$ ensures the matrix is symmetric in floating-point

2.3.2 Testing Accuracy of Hermitian Version

The Hermitian version of FEAST utilizing full orthogonalization is very accurate at finding eigenvalues. Table 2.5 shows the results for 100 runs at each n . Each of these runs used a different randomly-generated matrix.

Table 2.5
Eigenvalue accuracy: Hermitian matrices, 100 runs

n	Mean Residual	Max Residual
100	9.99574×10^{-15}	4.03225×10^{-14}
250	1.15543×10^{-14}	4.11833×10^{-14}
500	1.20306×10^{-14}	2.47436×10^{-14}
1000	1.64923×10^{-14}	4.1934×10^{-14}

2.4 Conclusions

This investigation shows that provided one knows the number of eigenvalues in an interval, the single pass standard FEAST algorithm is accurate and consistent. The algorithm struggles in separating eigenvalues just inside and just outside the interval. Modifying the algorithm to orthogonalize the quadrature matrix \hat{P} results in identifying the correct number of eigenvalues (providing one uses sample sizes significantly larger than m) even when there are eigenvalues just inside or outside the interval. This can be done without additional iteration.

The Hermitian version of FEAST with the orthogonalization works similarly.

Further testing is needed to investigate the effects of quadrature schemes and contours. This is addressed in Chapter 4. The remaining issue is estimating the number of eigenvalues without wasting computational resources. The proposed solution is recursion.

Chapter 3

Recursive FEAST Variation

3.1 Why a Recursive Algorithm?

The previous chapter showed that the standard FEAST algorithm works well when the number of eigenvalues in a region are known, and there are no clusters near the edge. Most of the work for the algorithm is in the linear solves. If the sample size is not large enough to get all eigenvalues inside the interval, that work is wasted. Recursion solves the problem of needing a good estimate of the number of eigenvalues in the interval.

3.2 Modifications of the Standard FEAST Algorithm

To generalize the standard FEAST algorithm to handle complex eigenvalues and non-symmetric matrices, the entire contour is integrated as it is no longer guaranteed symmetric. The implication is that the search interval is now a search region. This search region can be anywhere in the complex plane.

Two changes were made to the standard FEAST algorithm to implement recursion:

- Rectangular and diamond contours replaced circles
- Simpson's Rule replaced Gauss-Legendre quadrature

A recursive FEAST algorithm reuses code and linear solves, so the contour needs to be a shape that can be split evenly. Circles cannot be split easily, but rectangles and diamonds can. For example, a rectangle can be split into four parts, and each of those parts can have the FEAST algorithm operate on them.

Calculations can be reused because the four sub-shapes share sides. Simpson's Rule can reuse the linear solve data at each quadrature point, and is exact for polynomials of degree less than four [1]. It makes sense to use Simpson's Rule since each side of the shape is a line.

B was kept as the identity matrix because the code to implement the recursive version was modified from the version tested in Section 2.2.

3.3 Code Details

Algorithm 5 shows the basic algorithm for recursive FEAST.

Algorithm 5 FEAST Recursive Algorithm

Given $A^{n \times n}$, $\lambda_{min}, \lambda_{max} \in \mathbb{C}$, and $s \in \mathbb{R}$
 $Y^{n \times s} \in \mathbb{C} \leftarrow$ random vector
 Create contour using λ_1 and λ_2
 \triangleright Find the function value at each point (4 corners and 4 midpoints)
for $i = 1, 8$ **do**
 $B \leftarrow Id$
 Solve $(z_i B - A)g_i = Y$.
end for
 \triangleright Find the result of the quadrature on the 4 sides
for $j = 1, 4$ **do**
 $\hat{P}_+ \leftarrow \frac{width}{6}(g_{2j-1} + 4g_{2j} + g_{(2j+1)})$
end for
 $\hat{P} \leftarrow \hat{P} \div (2\pi i)$
if $||\hat{P}|| < \text{tol}$ **then**
 $\hat{A} \leftarrow \hat{P}^\dagger . A . \hat{P}$
 $\hat{B} \leftarrow \hat{P}^\dagger . \hat{P}$
 Find eigenvalues and eigenvectors of $\hat{A}\vec{v} = \lambda \hat{B}\vec{v}$
else
 Split contour into 4 parts and repeat
end if

3.3.1 Search Region and Contours

For standard FEAST $\lambda_1, \lambda_2 \in \mathbb{R}$ form a search interval. Recursive FEAST is given $\lambda_1, \lambda_2 \in \mathbb{C}$. It is assumed that there are distinct real and imaginary parts, \Re and \Im respectively. The formation of the region is given by the shape of the contours.

3.3.1.1 Formation of Rectangle Contour

Rectangular contours are derived by opposite corners. Algorithm 6 shows this process, with \Re representing the real part of λ and \Im representing the imaginary part of λ . Figure 3.1 shows how this works for the initial rectangular contour.

Algorithm 6 Rectangle Creation

Given λ_1, λ_2
 $c_1 \leftarrow (\min(\Re(\lambda_1), \Re(\lambda_2)), \min(\Im(\lambda_1), \Im(\lambda_2)))$
 $c_2 \leftarrow (\max(\Re(\lambda_1), \Re(\lambda_2)), \min(\Im(\lambda_1), \Im(\lambda_2)))$
 $c_3 \leftarrow (\max(\Re(\lambda_1), \Re(\lambda_2)), \max(\Im(\lambda_1), \Im(\lambda_2)))$
 $c_4 \leftarrow (\min(\Re(\lambda_1), \Re(\lambda_2)), \max(\Im(\lambda_1), \Im(\lambda_2)))$

Contour integration proceeds counter-clockwise, starting at c_1 with Simpson's Rule as the quadrature scheme on each side.

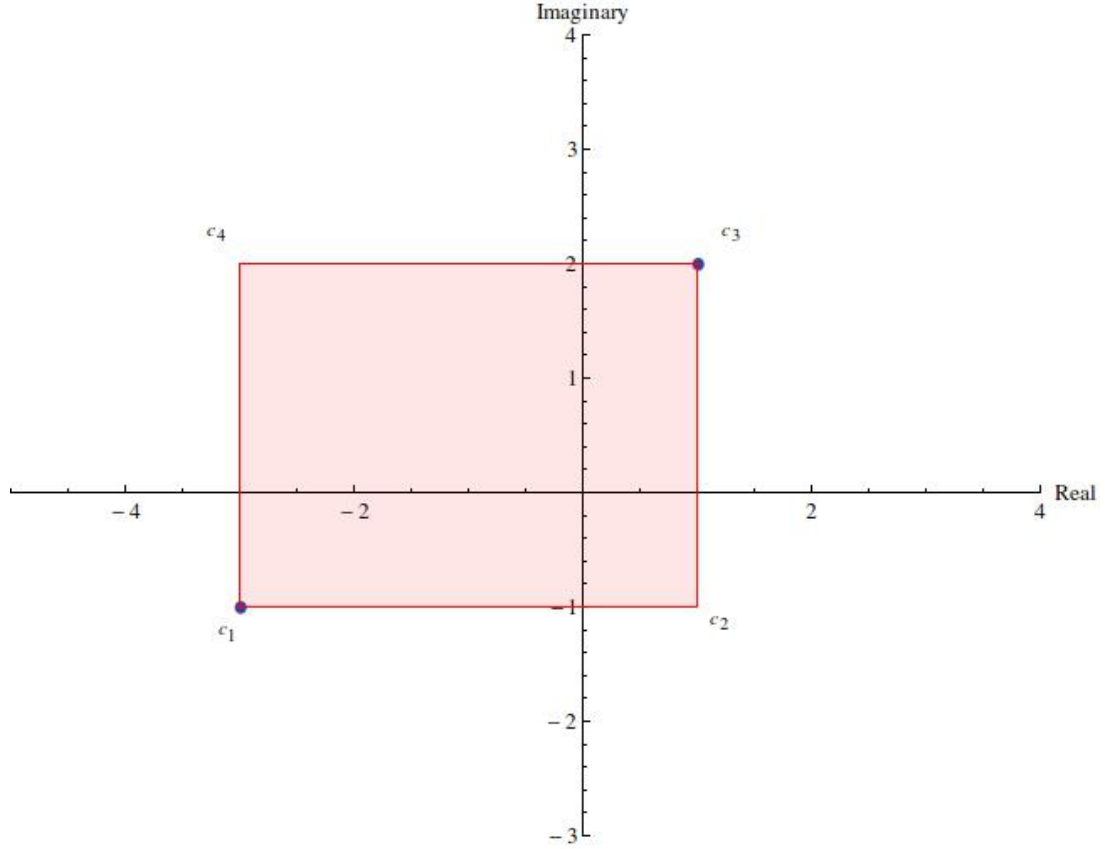


Figure 3.1: Rectangle recursive subdivision contour, $\lambda_1 = -3 - i$, $\lambda_2 = 1 + 2i$

3.3.1.2 Formation of Diamond Contour

To create a diamond given $\lambda_1, \lambda_2 \in \mathbb{C}$ find the midpoint between them. The four corners are created based on the distances the given values are from the middle. The two values are never corners but are always on the contour. Figure 3.2 demonstrates this, and the Mathematica code is given in Algorithm 7.

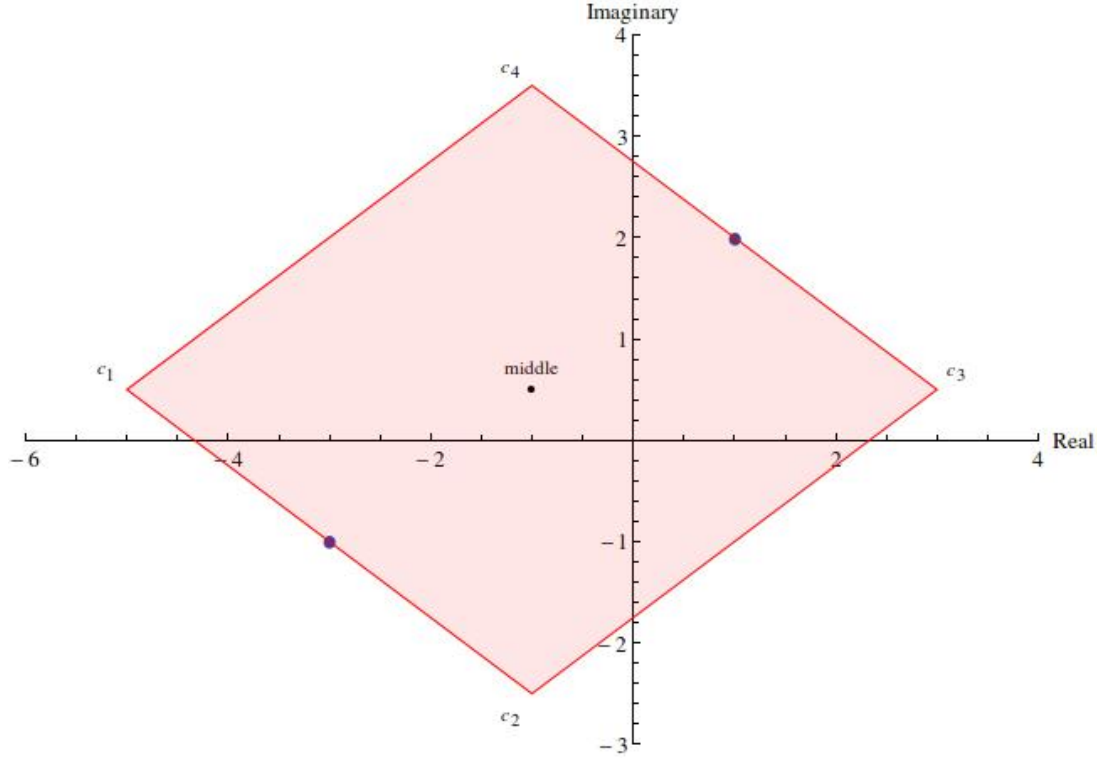


Figure 3.2: Diamond recursive subdivision contour, $\lambda_1 = -3 - i$, $\lambda_2 = 1 + 2i$

Algorithm 7 Diamond Creation

Given λ_1, λ_2
 $middle \leftarrow \frac{1}{2}(\lambda_1 + \lambda_2)$
 $w \leftarrow \max(\Re(\lambda_1), \Re(\lambda_2)) - \min(\Re(\lambda_1), \Re(\lambda_2))$
 $h \leftarrow \max(\Im(\lambda_1), \Im(\lambda_2)) - \min(\Im(\lambda_1), \Im(\lambda_2))$
 $c_1 \leftarrow middle - \{w, 0\}$
 $c_2 \leftarrow middle - \{0, h\}$
 $c_3 \leftarrow middle + \{w, 0\}$
 $c_4 \leftarrow middle + \{0, h\}$

3.3.2 Recursion Level

As seen previously, the singular values of the quadrature matrix can give information about the number of eigenvalues in the contour. The Frobenius norm [11]

$$\|A\|_F = \left(\sum_{i=1}^n \sum_{j=1}^m a_{ij}^2 \right)^{\frac{1}{2}} \quad (3.1)$$

contains information about the singular values. Theorem 5.3 [11] states

$$\|A\|_F = \left(\sum_{i=1}^n \sigma_i^2 \right)^{\frac{1}{2}} \quad (3.2)$$

By construction \hat{P} approximates P which has singular values 1 and 0, thus

$$\|\hat{P}\|_F \approx \sqrt{m} \quad (3.3)$$

The algorithm recursively subdivides each region until $\|\hat{P}\| \leq 0.1$ or until the maximum recursion depth is reached (currently 10). Experimental results showed that eigenvalues were located after 4 or 5 recursive steps, so 10 seemed like a good maximum. The value of 0.1 was chosen by running several matrices on intervals containing one eigenvalue with different s and n and identifying where there is a clear gap on a plot of the singular values

of \hat{P} . Figure 3.3 shows the singular values on a complex matrix with $n = 20$, $m = 1$, $s = 16$.

There is a clear gap between the non-zero and the zero singular values.

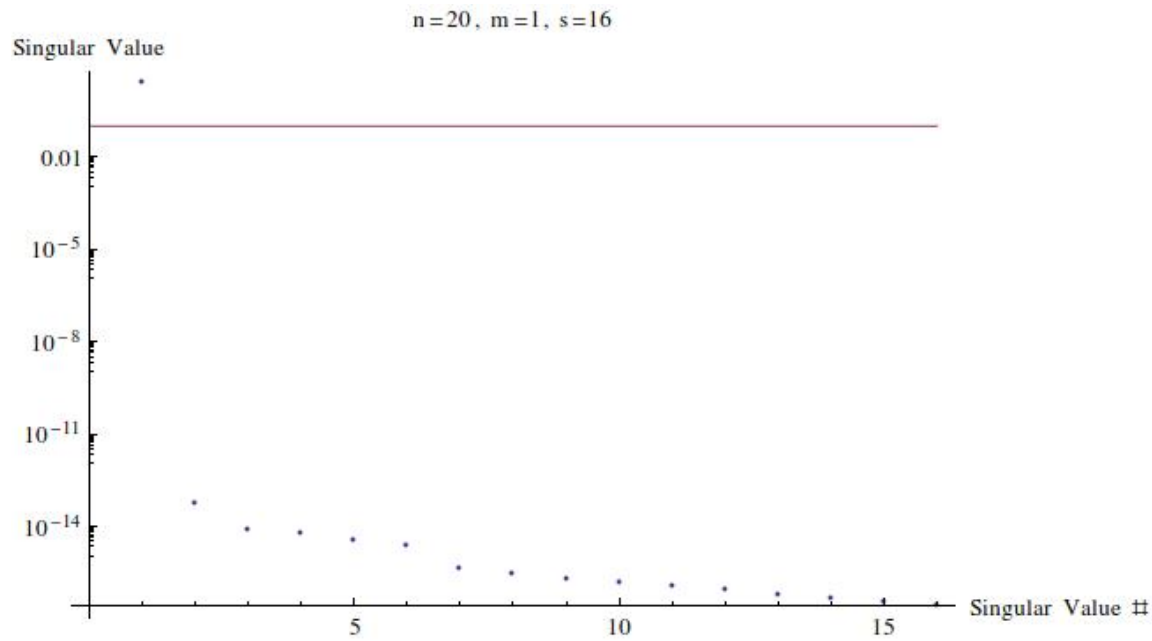


Figure 3.3: All singular values of \hat{P} : random complex matrix, $n = 20$, $m = 1$, $s = 16$

This tolerance holds for larger matrices and different sample sizes. Figure 3.4 shows that a tolerance of 0.1 works on an $n = 200$ random complex matrix with $m = 1$ and $s = 8$.

Figure 3.5 uses the same matrix but with $s = 16$. The results are consistent.

In the pictures that follow, regions with no eigenvalues rarely go more than 2 recursion levels deep, so a maximum recursion of 10 is enough to decide there are no eigenvalues inside.

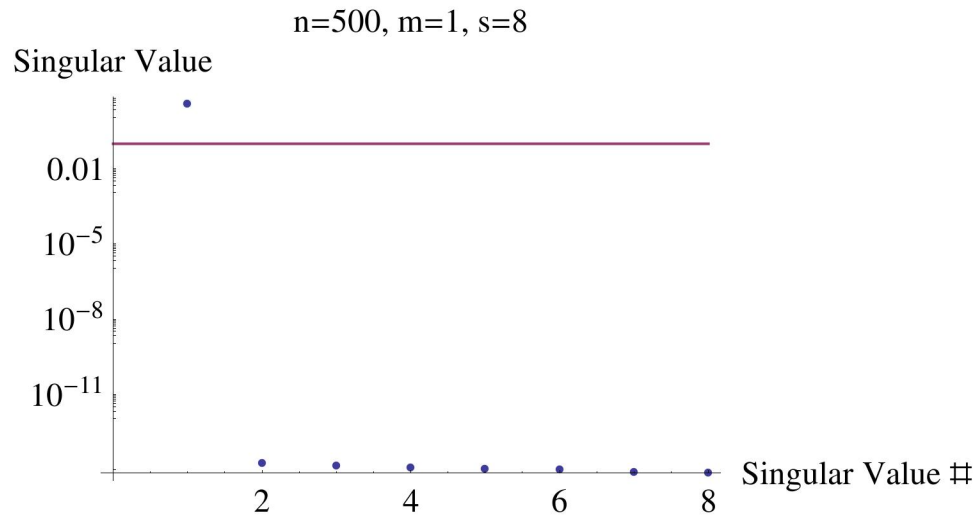


Figure 3.4: All singular values of \hat{P} : random complex matrix, $n = 500$, $m = 1$, $s = 8$

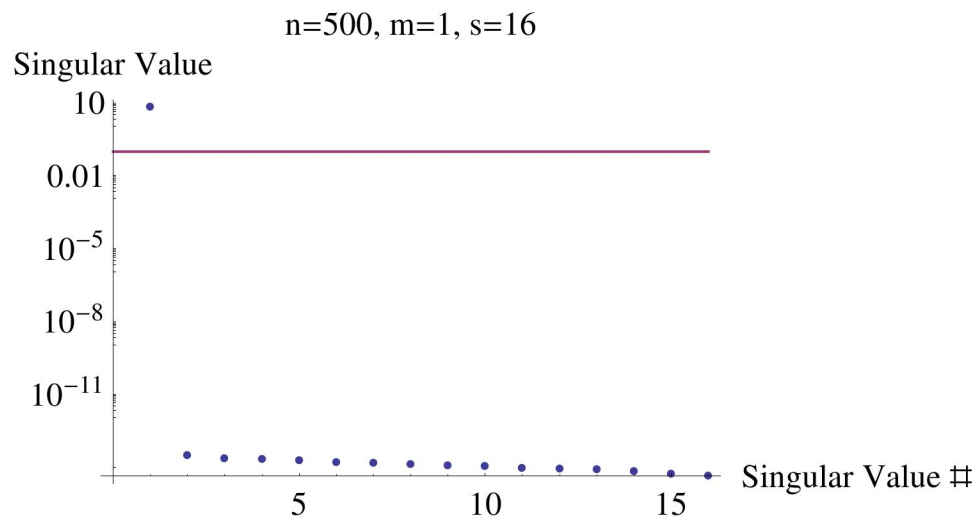


Figure 3.5: All singular values of \hat{P} : random complex matrix, $n = 500$, $m = 1$, $s = 16$

3.3.3 Test Matrix Creation

The algorithms for real-symmetric matrices and Hermitian matrices are the same as from Chapter 2.

Non-symmetric complex matrices are created with Algorithm 8.

Algorithm 8 Random Complex Matrix Generator

Given list of eigenvalues L
 $n \leftarrow \text{Length}(L)$
 $V \leftarrow \text{RandomComplex}[\{-1 - i, 1 + i\}, \{n, n\}]$
Normalize V
 $D \leftarrow \text{Diag}(L)$
 $A \leftarrow V.D.V^{-1}$

Non-symmetric real matrices are created with Algorithm 9.

Algorithm 9 Random Real Matrix Generator

Given list of eigenvalues L
 $n \leftarrow \text{Length}(L)$
 $V \leftarrow \text{RandomReal}[\{-1, 1\}, \{n, n\}]$
Normalize V
 $D \leftarrow \text{Diag}(L)$
 $A \leftarrow V.D.V^{-1}$

3.4 Testing The Algorithm with Rectangular Contours

The code works on non-symmetric matrices, which Polizzi [7] does not consider. The code at its current inception can be used to pinpoint where eigenvalues exist, though it has not

been written to output these eigenvalues and eigenvectors.

3.4.1 Testing Real Eigenvalues

To be comparable to the standard FEAST algorithm, the recursive version must be able to handle real eigenvalues.

A real symmetric matrix with random eigenvalues of size $n = 100$ was run on FEAST with $s = 16$ and interval $\{\lambda_1, \lambda_2\} = \{-5 - 4i, 5 + 5i\}$. Figure 3.6 shows that the recursive FEAST algorithm finds the eigenvalues. Notice that there are multiple eigenvalues inside what appear to be the smallest of the boxes. Figure 3.7 shows the same plot but zoomed in from $\{\lambda_1, \lambda_2\} = \{1.1 - .1i, 1.5 + .1i\}$ to see what happens with the eigenvalues near each other. It's clear that the eigenvalues are isolated from each other before the maximum recursion depth of 10 is reached. The eigenvalue on the left could have had 2 fewer divisions made.

To verify the behavior, the recursive FEAST was run on a Hermitian matrix of size $n = 1000$. For $s = 16$ and interval $\{\lambda_1, \lambda_2\} = \{-5 - 3.5i, 5 + 5i\}$ the same behavior is seen (see Figure 3.8).

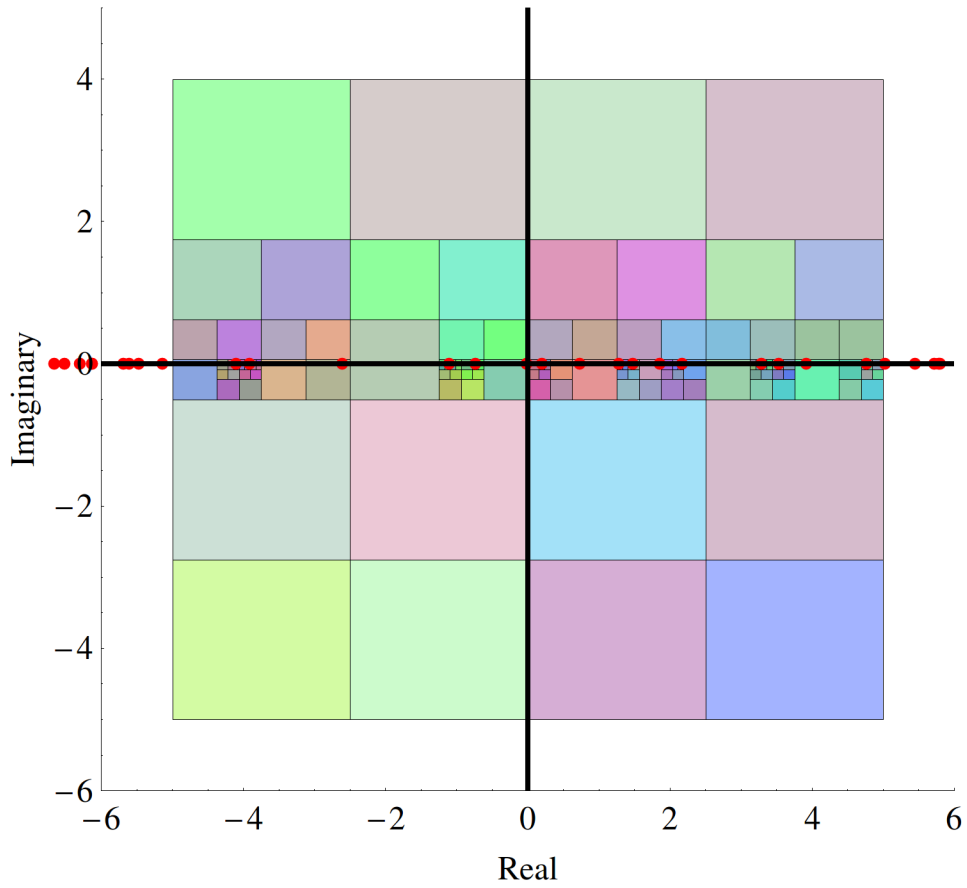


Figure 3.6: Recursive rectangular FEAST subdivision process: random real symmetric matrix, $n = 1000$, $m = 16$, $s = 16$, red dots indicate eigenvalues

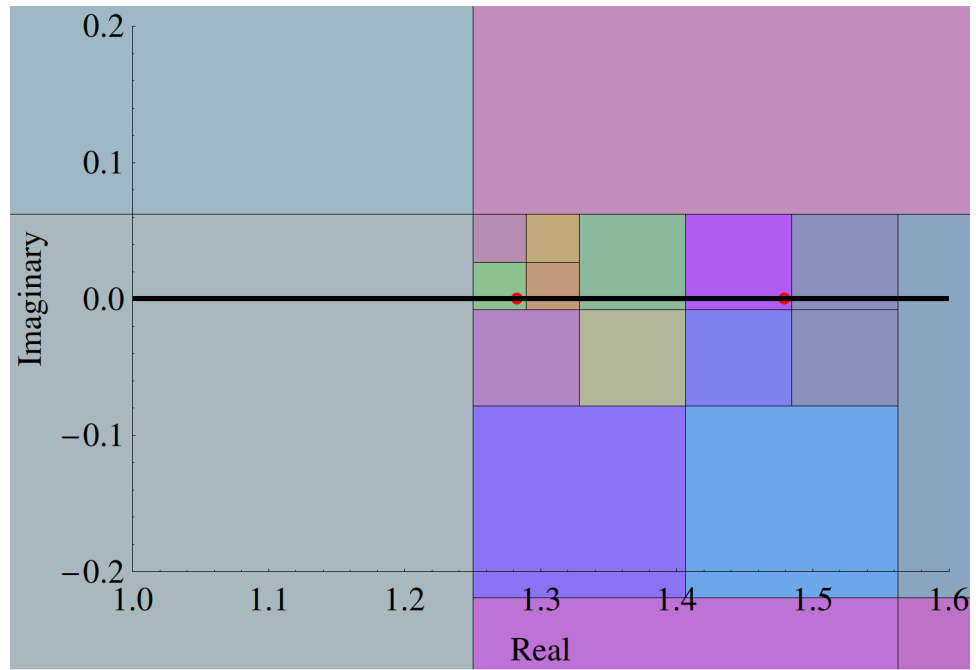


Figure 3.7: Recursive rectangular FEAST subdivision process, zoomed in: random real symmetric matrix, $n = 1000$, $m = 16$, $s = 16$, red dots indicate eigenvalues

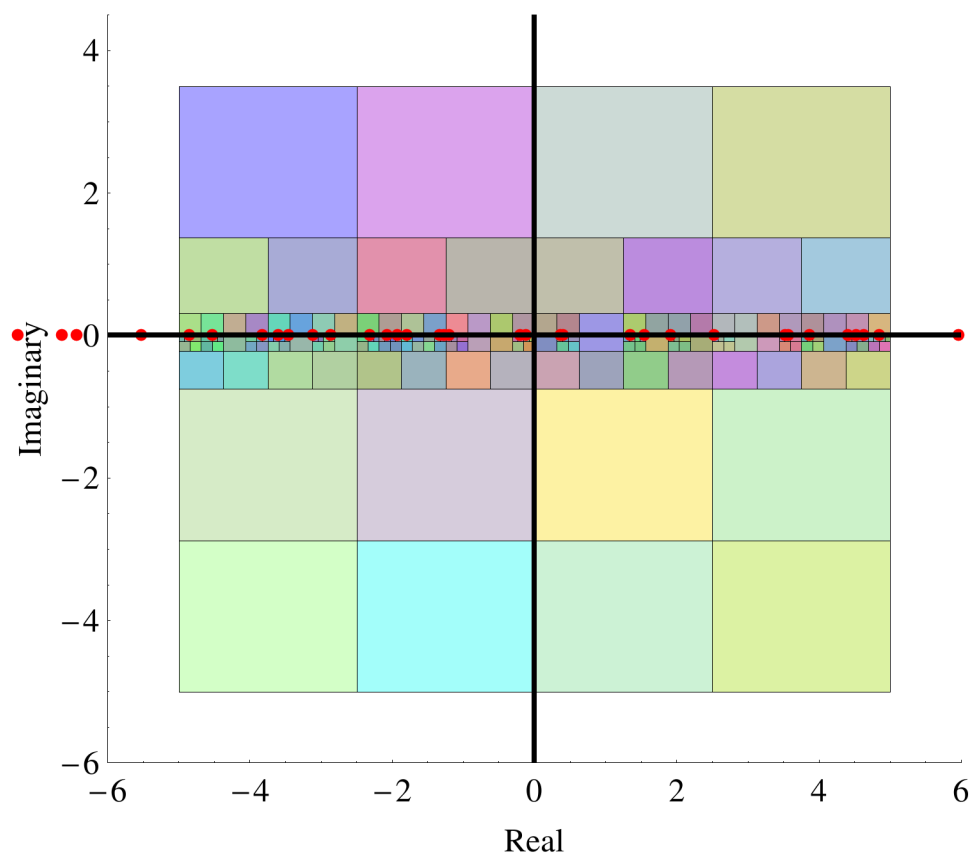


Figure 3.8: Recursive rectangular FEAST subdivision process: random Hermitian matrix, $n = 1000$, $m = 29$, $s = 16$

3.4.2 Testing Complex Eigenvalues

3.4.2.1 Random Eigenvalues

The recursive algorithm with rectangular contours successfully finds complex eigenvalues, which extends the standard FEAST algorithm. For a random complex matrix generated from Algorithm 8 of size $n = 500$, $\{\lambda_1, \lambda_2\} = \{-5 + 5i, 5 - 5i\}$, $s = 16$, Figure 3.9 shows how the algorithm splits into successively smaller rectangles in an effort to narrow down where the eigenvalues are. Eigenvalues near the edge do not cause extra eigenvalues to be found, unlike the standard FEAST code.

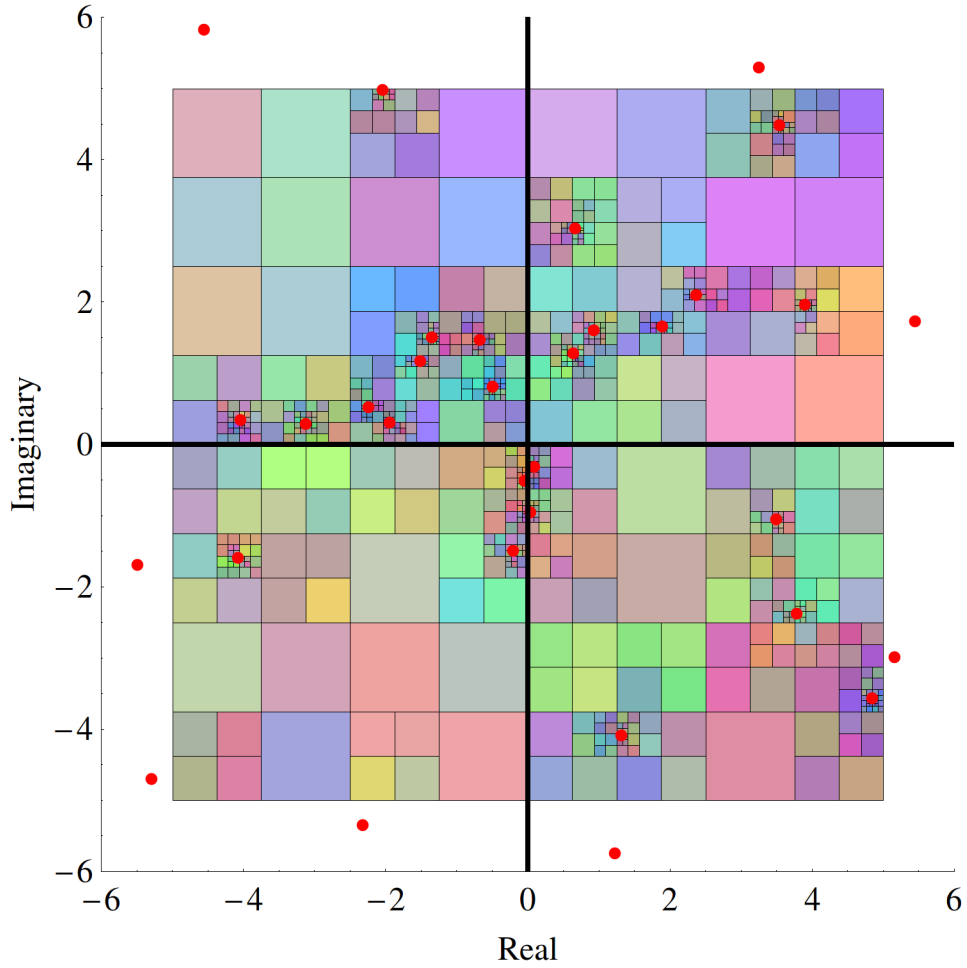


Figure 3.9: Recursive rectangular FEAST subdivision process: random complex matrix, $n = 500$, $m = 25$, $s = 16$

Note the two eigenvalues near $0 - 0.5i$. Since the two points are very close to each other, there potentially may not be enough iterations for them to be in separate rectangles. However, Figure 3.10 shows that in fact the two eigenvalues were isolate well before the maximum recursion depth was reached. From counting the divisions, it appears that 5 recursive steps would have been enough to isolate those two eigenvalues in separate rectangles.

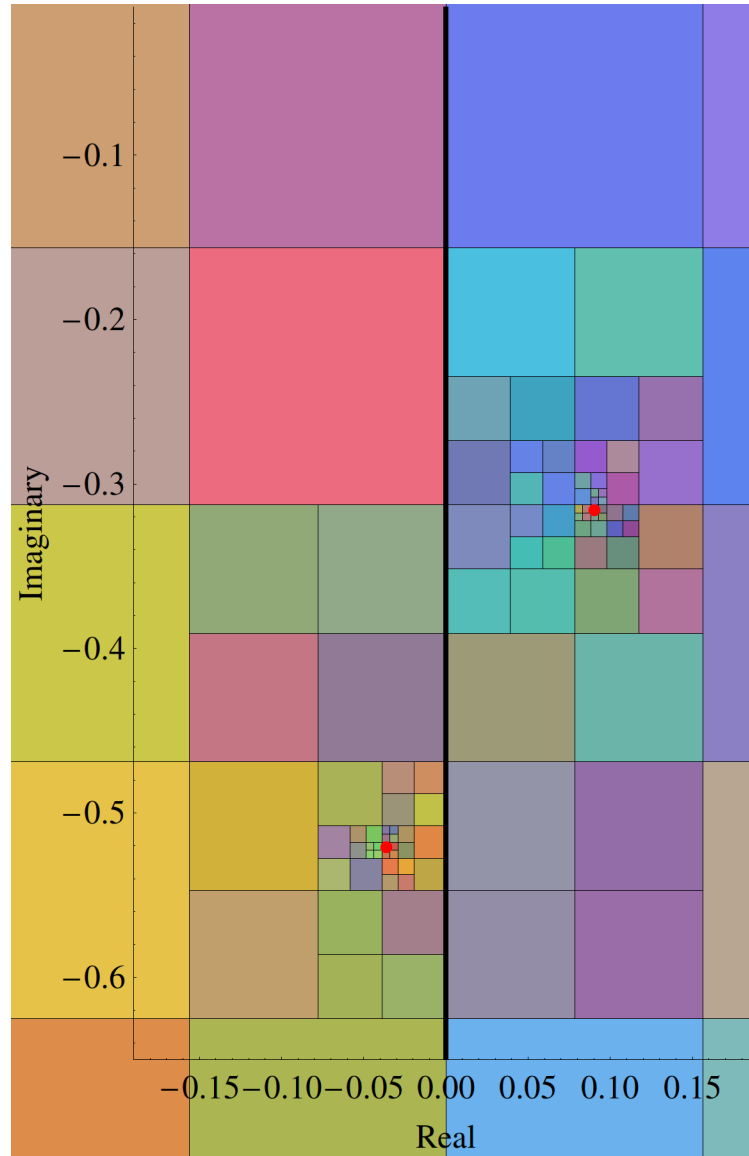


Figure 3.10: Recursive rectangular FEAST subdivision process zoomed in:
random complex matrix, $n = 500$, $m = 25$, $s = 16$

3.4.2.2 Clustered Eigenvalues

A 500×500 random complex matrix with 281 clustered complex eigenvalues was run with $s = 16$ and $\{\lambda_1, \lambda_2\} = \{20 + 32i, -7 - 2i\}$. Figure 3.11 shows the recursive version does not recurse where there are no eigenvalues. This is an improvement from the standard FEAST algorithm which does not handle clusters well.

Zooming in to the most clustered area however finds that a maximum recursion depth of 10 is not enough to get each eigenvalue inside a rectangle by itself. Figure 3.12 shows that some of the rectangles have more than one eigenvalue inside. However, note that there are no areas where there are smaller rectangles created with no eigenvalues nearby.

The maximum recursion depth was changed to 15 and then the same matrix was rerun. Figure 3.13 shows that increasing the maximum recursion means that some of the eigenvalues that were not isolated in figure 3.12 are now isolated.

3.4.2.3 Complex-Conjugate Eigenvalues

Complex-conjugate eigenvalues are generated via Algorithm 10.

For a randomly generated $n = 500$ matrix with complex-conjugate eigenvalues, recursive FEAST, with $s = 16$ and $\{\lambda_1, \lambda_2\} = \{-3 - 6i, 5 + 6i\}$, finds all 18 of the target eigenvalues

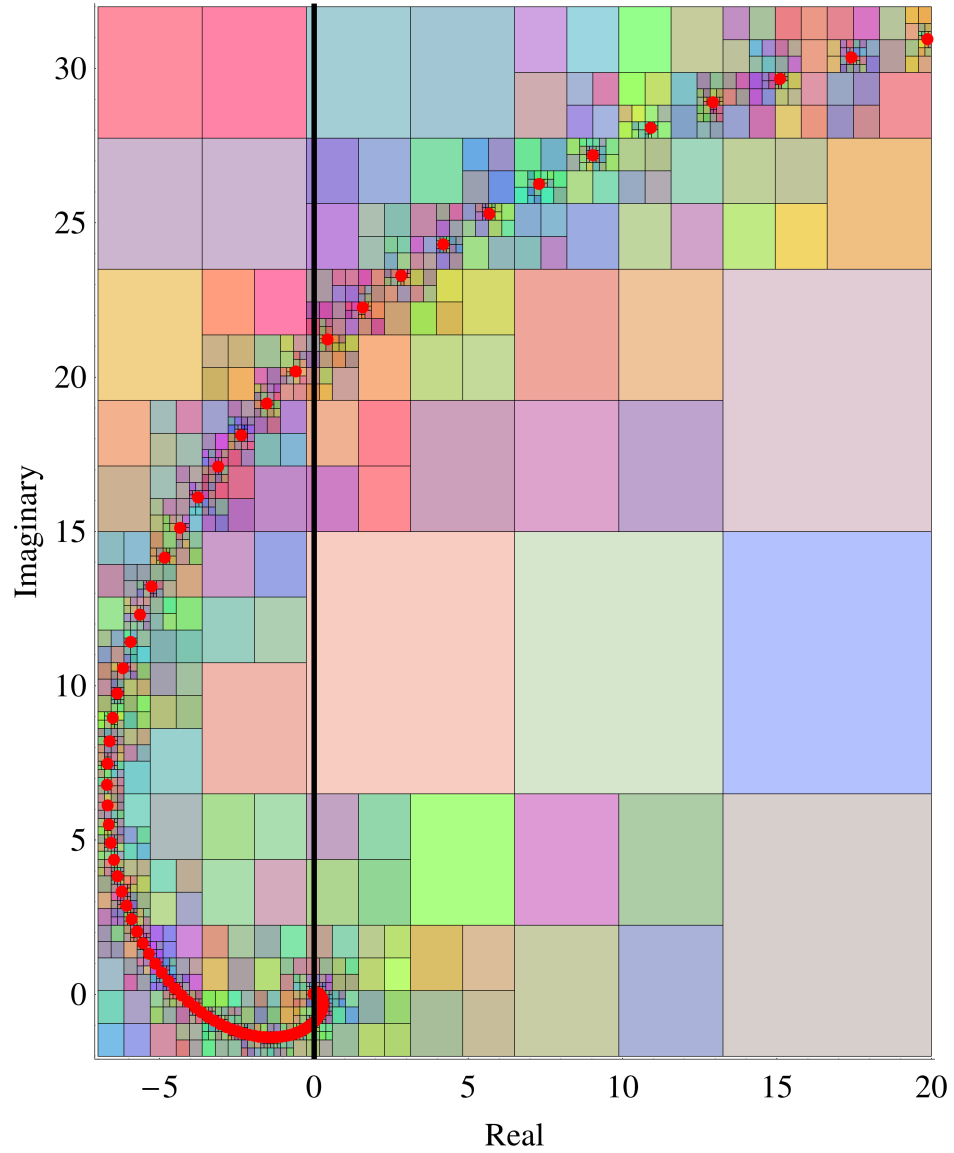


Figure 3.11: Recursive rectangular FEAST subdivision process: random complex matrix, clustered eigenvalues, $n = 500$, $m = 281$, $s = 16$

(see Figure 3.14).

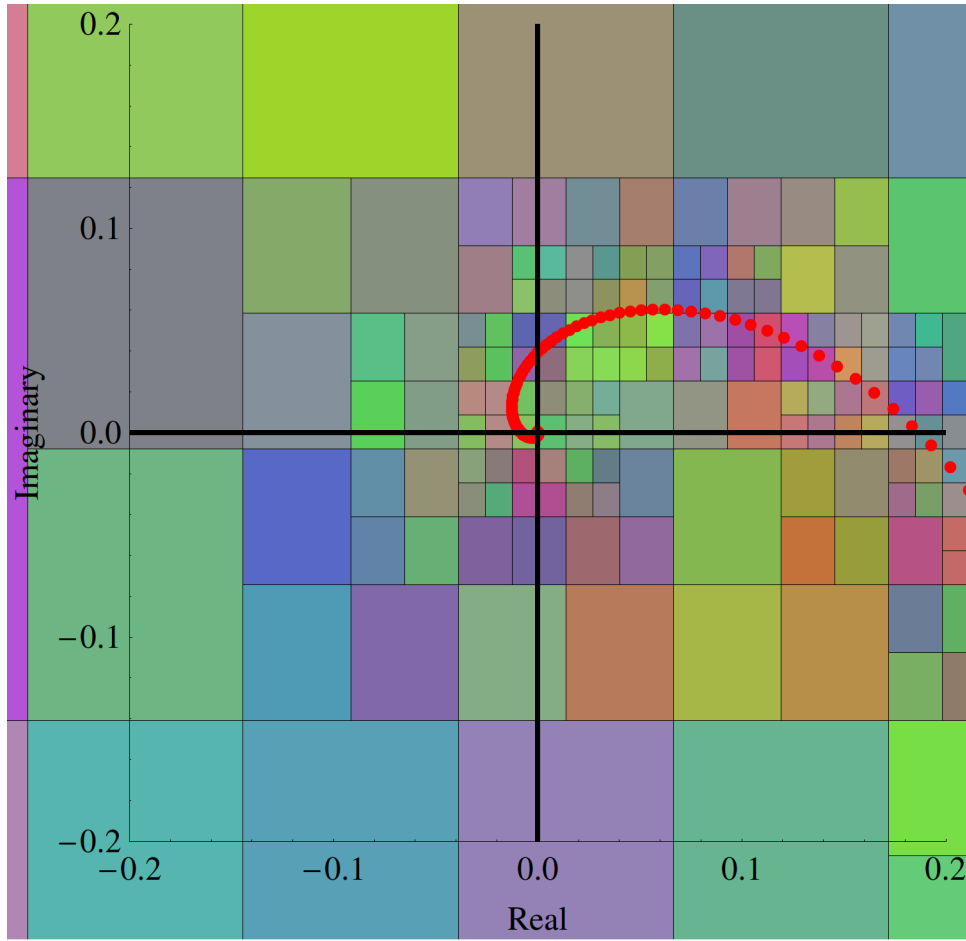


Figure 3.12: Recursive rectangular FEAST subdivision process: random complex matrix, clustered eigenvalues, maximum recursion depth 10, $n = 500$, $m = 281$, $s = 16$

Algorithm 10 Complex-Conjugate Eigenvalue Generator

```

for  $j \leftarrow 0, \dots, n \text{ by } 2$  do
   $a \leftarrow \text{RandomReal}[\{-10, 10\}]$ 
   $b \leftarrow \text{RandomReal}[\{-10, 10\}]$ 
   $\text{evalList}[j] \leftarrow a + bi$ 
   $\text{evalList}[j + 1] \leftarrow a - bi$ 
end for

```

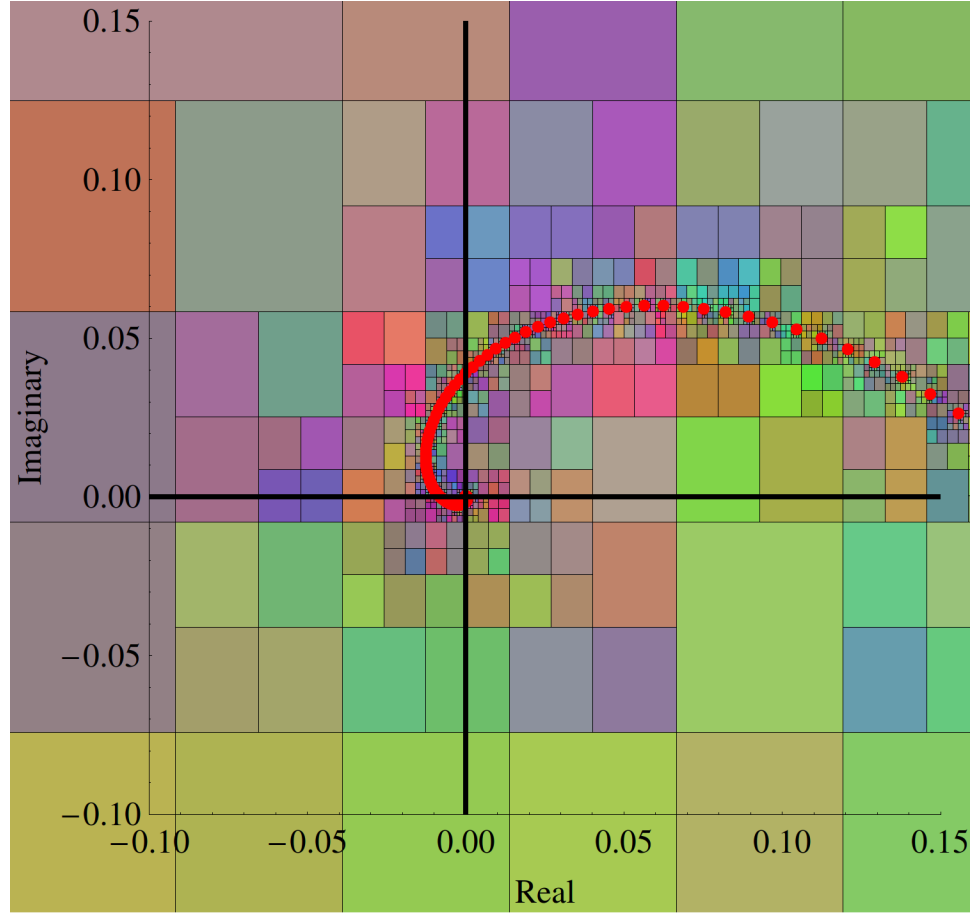


Figure 3.13: Recursive rectangular FEAST subdivision process: random complex matrix, clustered eigenvalues, maximum recursion depth 15, $n = 500$, $m = 281$, $s = 16$

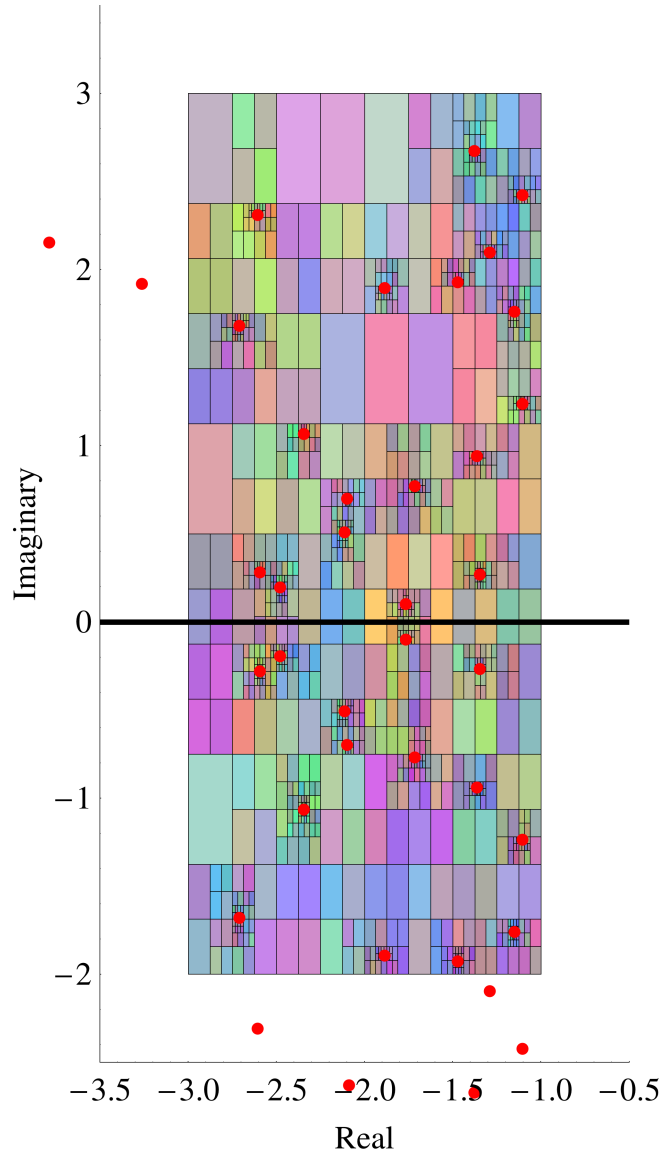


Figure 3.14: Recursive rectangular FEAST subdivision process: random complex matrix, complex conjugate eigenvalues, $n = 500$, $m = 18$, $s = 16$

3.5 Testing the Algorithm with Diamond Contours

Since clustered eigenvalues on the rectangles required at least 15 recursive steps, the maximum recursion depth was changed to 15 for all tests of the diamonds.

3.5.1 Testing Real Eigenvalues

3.5.1.1 Random Eigenvalues

On a real symmetric 500×500 matrix with $s = 15$ and $\{\lambda_1, \lambda_2\} = \{-2 - 2i, 2 + 2i\}$, the diamond contour gets all the eigenvalues. This is shown in Figure 3.15. More divisions are necessary when the eigenvalues are closer together.

Figure 3.16 zooms in on the real axis near -1.5. This close-up picture shows that the two eigenvalues near each other are resolved well before the maximum recursion is hit.

Figure 3.17 demonstrates that on a random Hermitian matrix with $n = 500$, $s = 16$, and $\{\lambda_1, \lambda_2\} = \{-1 - 1i, 1 + 1i\}$, no eigenvalues are missed inside the contour, and the algorithm subdivides appropriately. Notice how the top and bottom diamonds are not divided more than twice, as it is clear there are no eigenvalues there. When there are more eigenvalues clustered as is on negative real axis, more divisions are necessary to isolate the

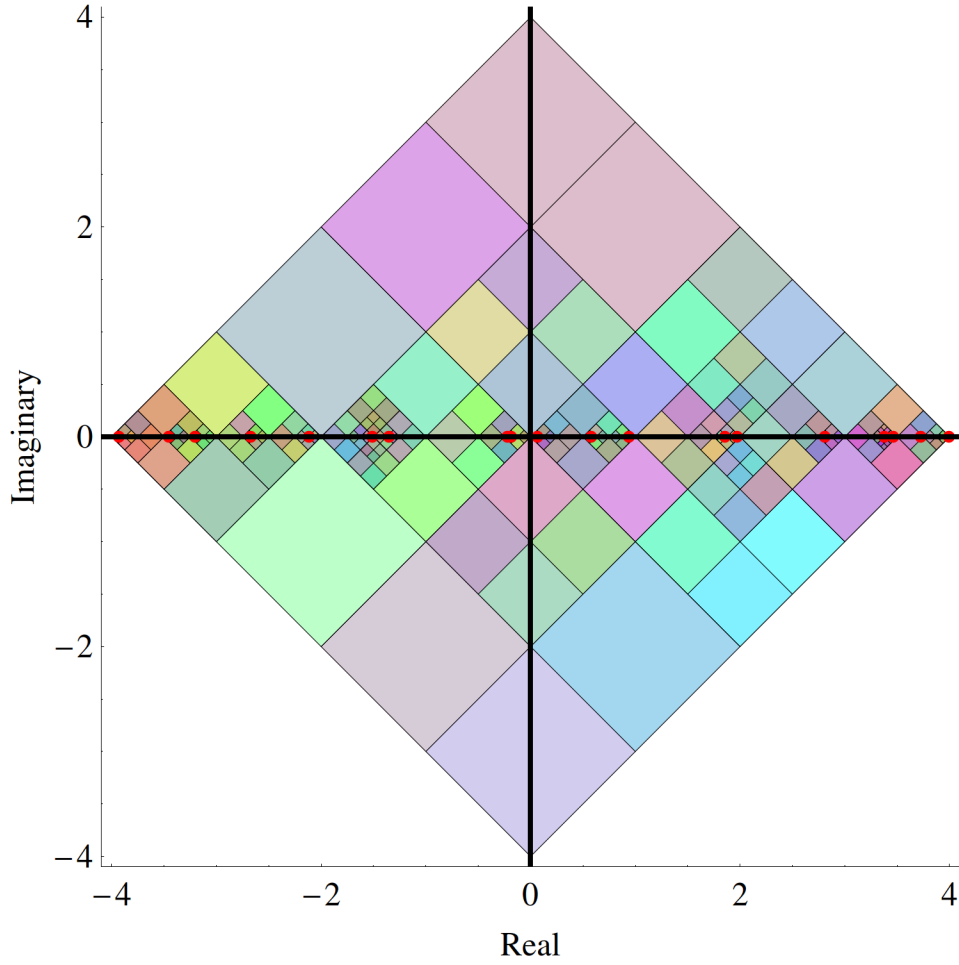


Figure 3.15: Recursive diamond FEAST subdivision process: random real symmetric matrix, $n = 500$, $m = 20$, $s = 16$

eigenvalues.

3.5.1.2 Clustered Eigenvalues

Clustered eigenvalues were created of the form

$$-1 + \frac{2}{i}, \quad i = 1, \dots, 20 \quad (3.4)$$

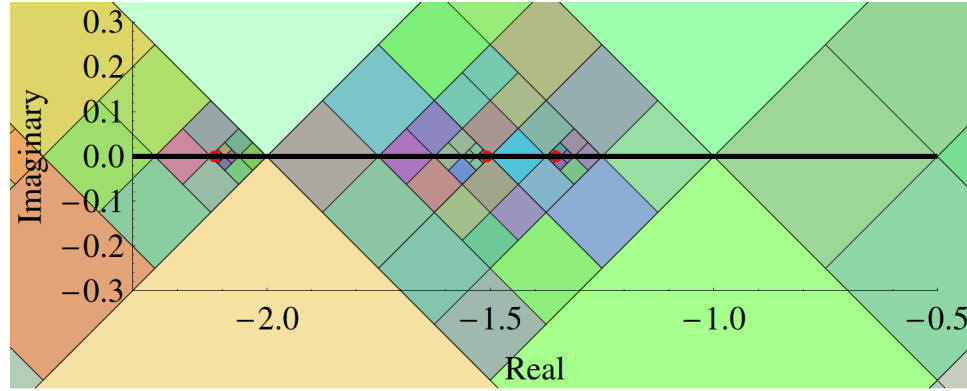


Figure 3.16: Recursive diamond FEAST subdivision process: random real symmetric matrix zoomed in, $n = 500$, $m = 20$, $s = 16$

Diamond contours successfully find clustered eigenvalues on the real axis. This was tested on a 200×200 real symmetric matrix with clustered eigenvalues generated from (3.4) and the remaining eigenvalues uniformly distributed between $[-1000, 0200]$ and $[200, 1000]$. Figure 3.18 shows run with $s = 32$ and interval $\{\lambda_1, \lambda_2\} = \{-1 - i, 1 + i\}$ finds all the eigenvalues in the interval.

The same clustered eigenvalues were run in Figure 3.19 with a Hermitian matrix of size $n = 200$, $s = 32$, and $\{\lambda_1, \lambda_2\} = \{-1 - i, 1 + i\}$. Again, all the eigenvalues were found.

The diamond is an improvement over the rectangles for real eigenvalues. Does it work on complex eigenvalues?

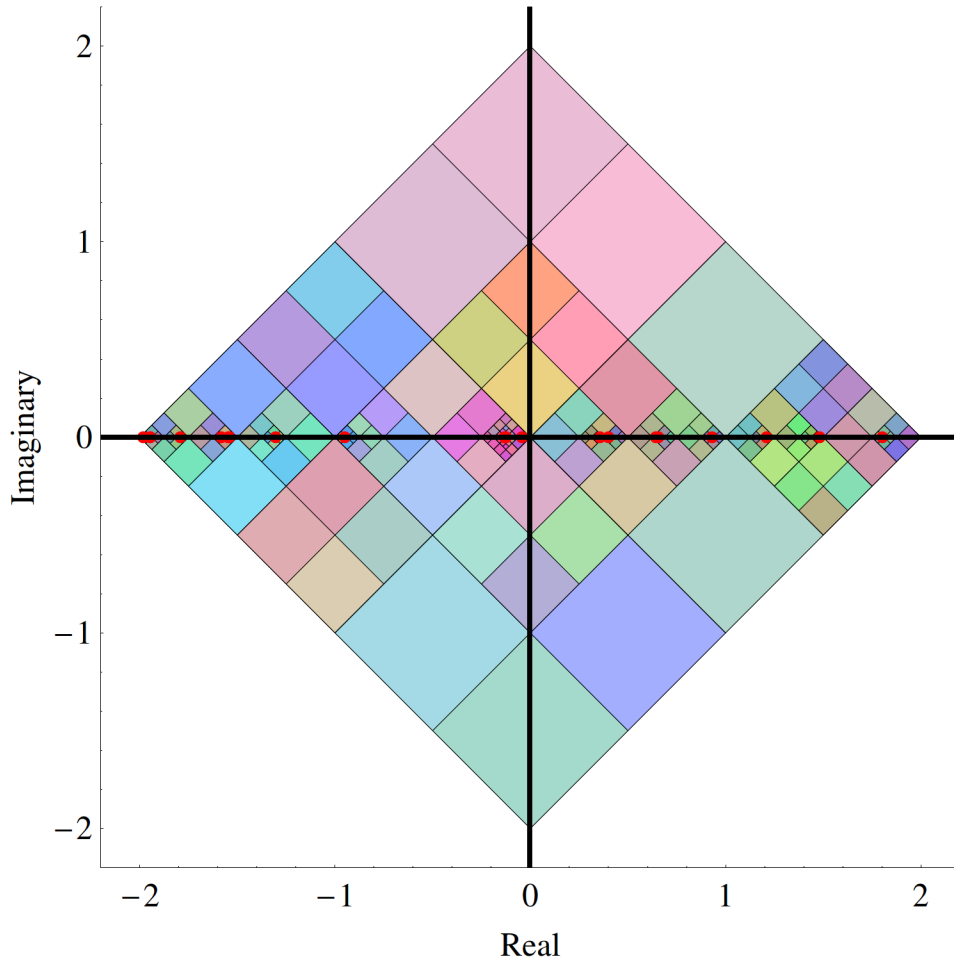


Figure 3.17: Recursive diamond FEAST subdivision process: random Hermitian matrix, $n = 500$, $m = 20$, $s = 16$

3.5.2 Testing Complex Eigenvalues

3.5.2.1 Random Eigenvalues

Figure 3.20 shows the division process works exactly the same as for the rectangles, with the same behavior near edges of the contour. Eigenvalues near the edges of the contour

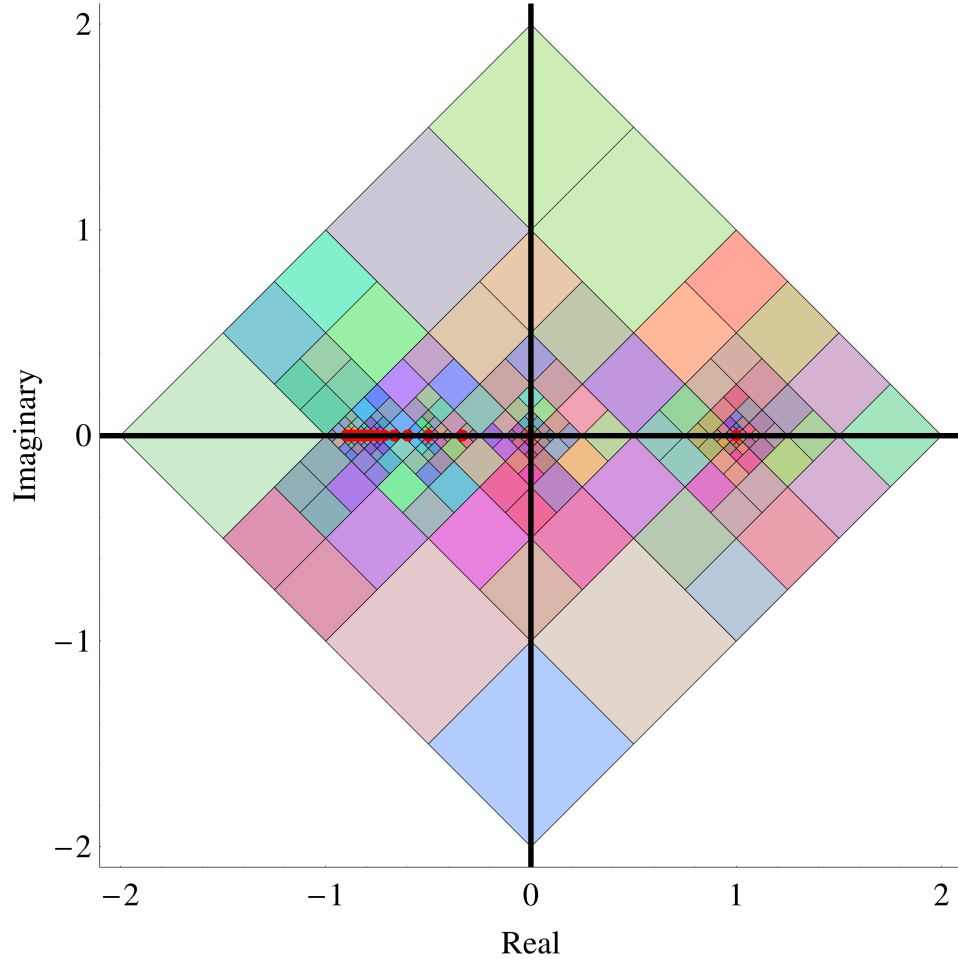


Figure 3.18: Recursive diamond FEAST subdivision process: real symmetric matrix, clustered eigenvalues, $n = 200$, $m = 19$, $s = 32$

only cause divisions up to 3 or 4 levels before the algorithm determines no eigenvalue is inside. For actual eigenvalues the algorithm will divide until the maximum recursion depth is reached. For this run, a random complex non-symmetric matrix of size $n = 500$ was used with $s = 16$ and $\{\lambda_1, \lambda_2\} = \{-2 - 2i, 2 + 2i\}$.

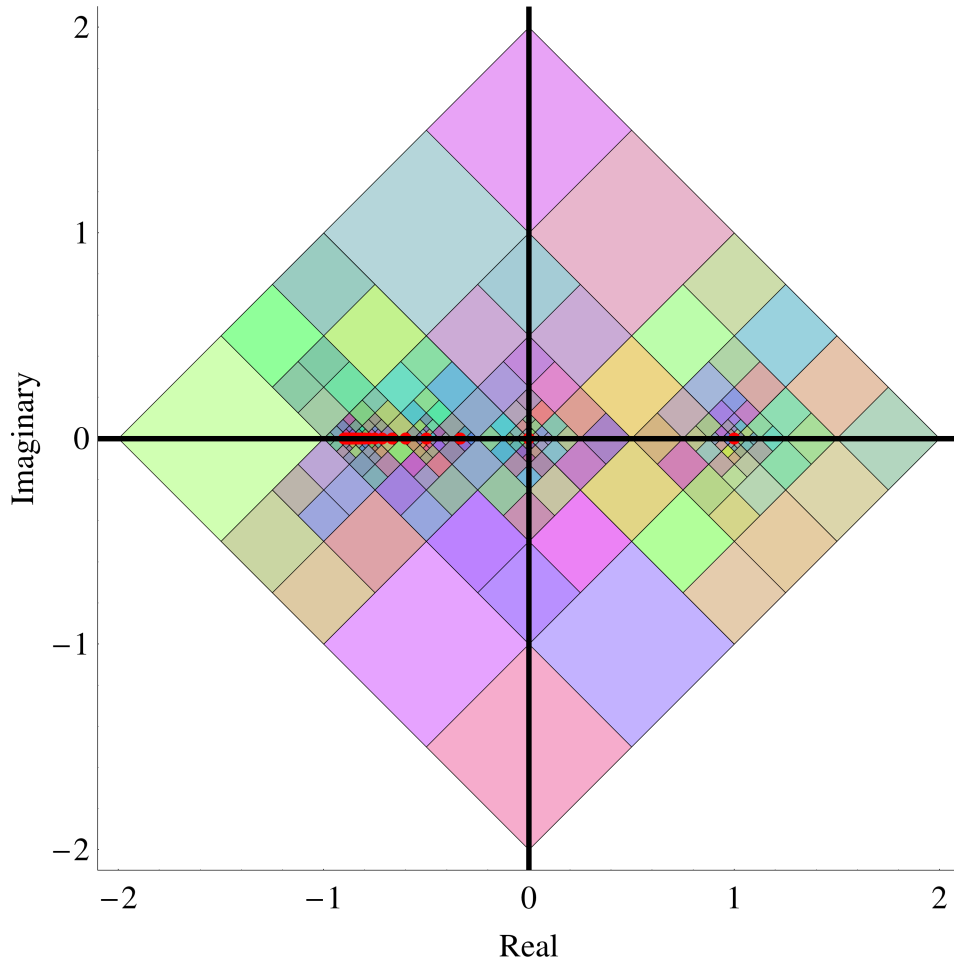


Figure 3.19: Recursive diamond FEAST subdivision process: Hermitian matrix, clustered eigenvalues $n = 200$, $m = 19$, $s = 32$

3.5.2.2 Clustered Eigenvalues

Figure 3.21 shows that diamond contours handle clustered complex eigenvalues just fine. The sections of the complex plane with no eigenvalues don't divide more than four times. Those that do contain eigenvalues divide until the maximum recursion level is hit. Areas near eigenvalues divide more, but not to the maximum depth. There are no areas where the algorithm recurses more than is necessary, showing that it does not look for eigenvalues in

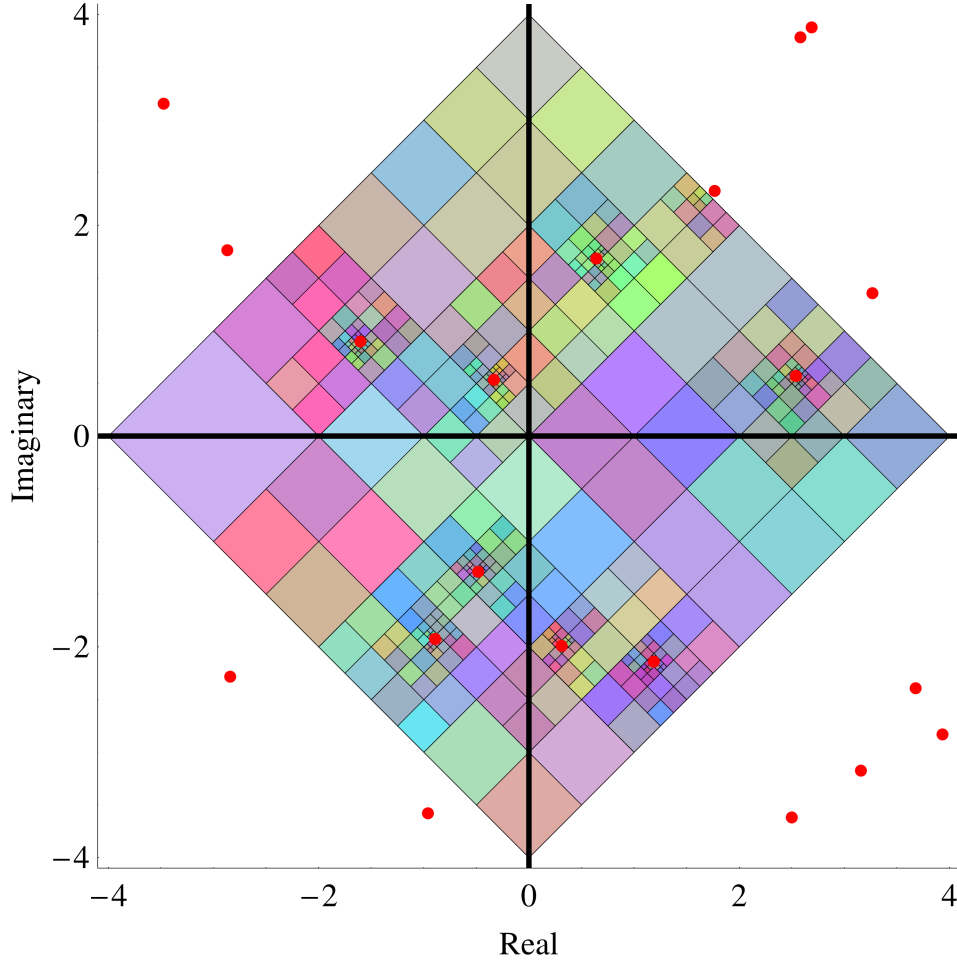


Figure 3.20: Recursive diamond FEAST subdivision process: random complex matrix, $n = 500$, $m = 8$, $s = 16$

areas where there are none. This run was on a $n = 500$ complex matrix, with $s = 16$ and $\{\lambda_1, \lambda_2\} = \{-5 + 12i, 5 - 2i\}$.

Figure 3.22 is a close-up of the recursion process near the left edge. The region is formed from the points $\{\lambda_1, \lambda_2\} = \{-6 + 8i, -4 + 12i\}$. The figure shows that the areas with the smallest boxes are where the eigenvalues are. When the eigenvalue is near the edge the boxes do not recurse as far as they do when there is an actual eigenvalue.

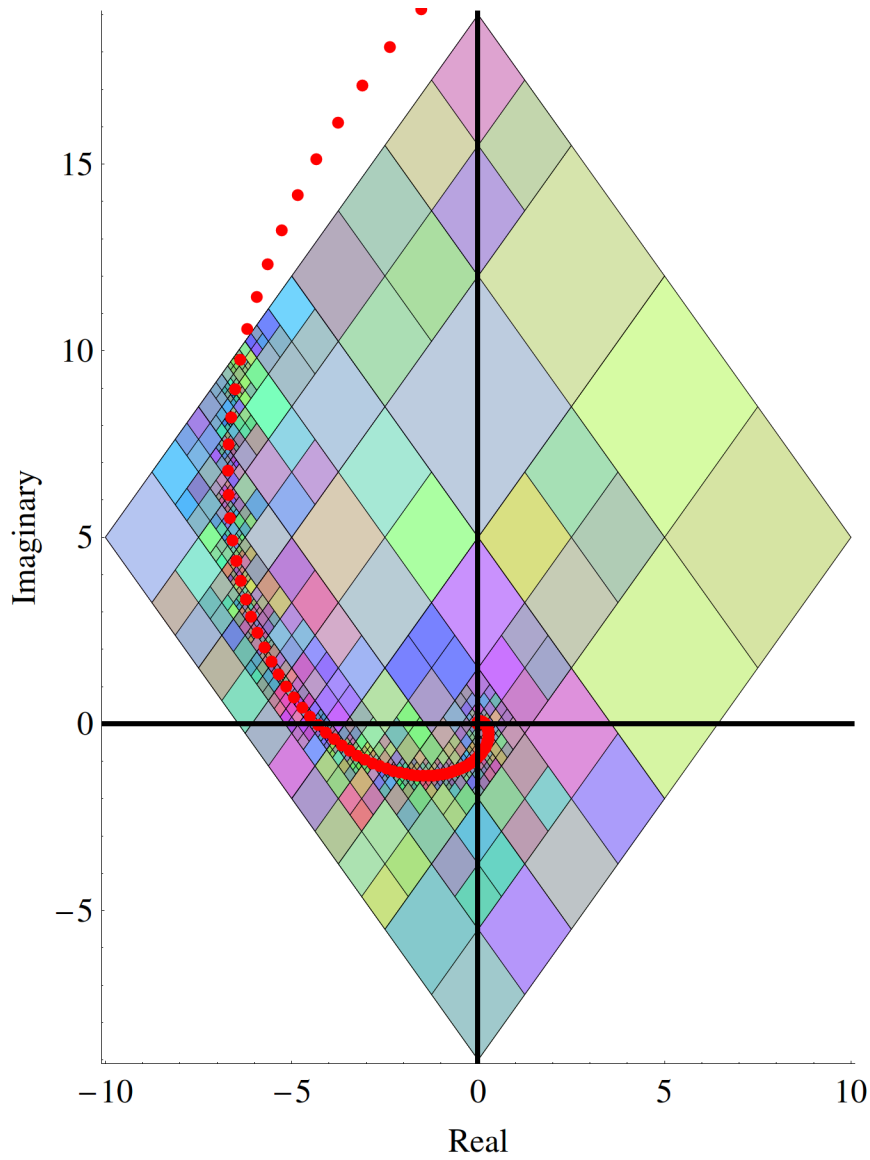


Figure 3.21: Recursive diamond FEAST subdivision process: random complex matrix, clustered eigenvalues, $n = 500$, $m = 241$, $s = 16$

3.5.2.3 Complex-Conjugate Eigenvalues

Figure 3.23 shows that diamond contours handle complex-conjugate eigenvalues. This run was on a $n = 500$ complex matrix, with $s = 16$ and $\{\lambda_1, \lambda_2\} = \{-3 - 2i, -1 + 3i\}$. A

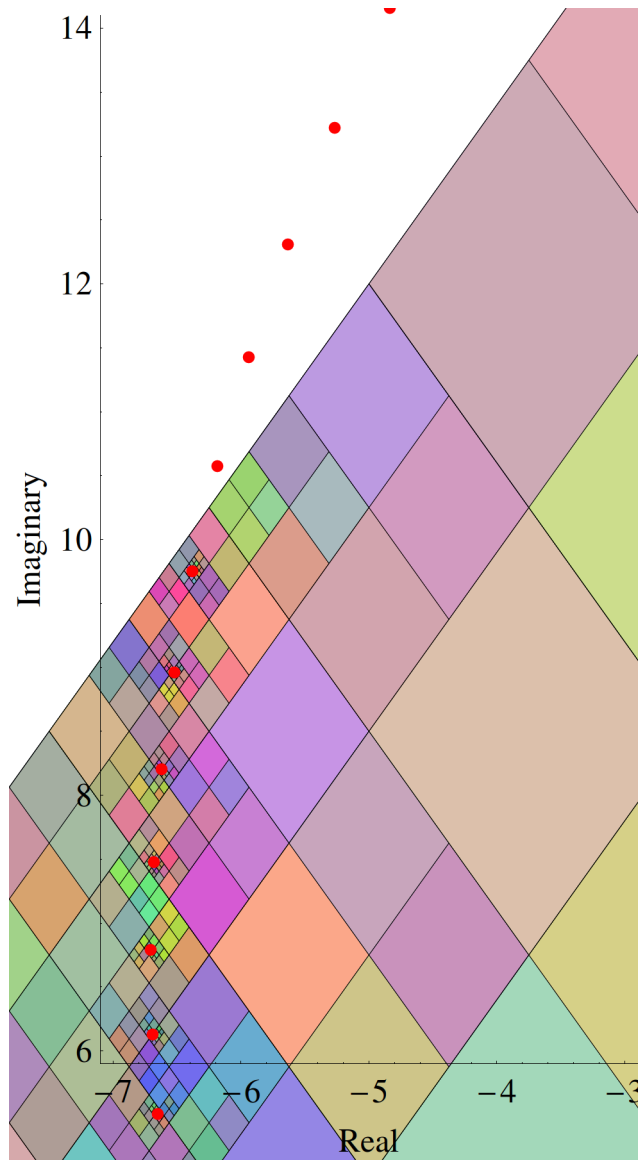


Figure 3.22: Recursive diamond FEAST subdivision process zoomed in: random complex matrix, clustered eigenvalues, $n = 500$, $m = 241$, $s = 16$

zoomed-in version is shown in figure 3.24. This version is focused around the real axis in one of the areas with some clustered eigenvalues. The figure shows that only the areas of the red dots (the eigenvalue locations) have the smallest diamonds, which means those are the areas that hit the maximum recursion depth. This should happen only where there are eigenvalues, and this is the case in this test.

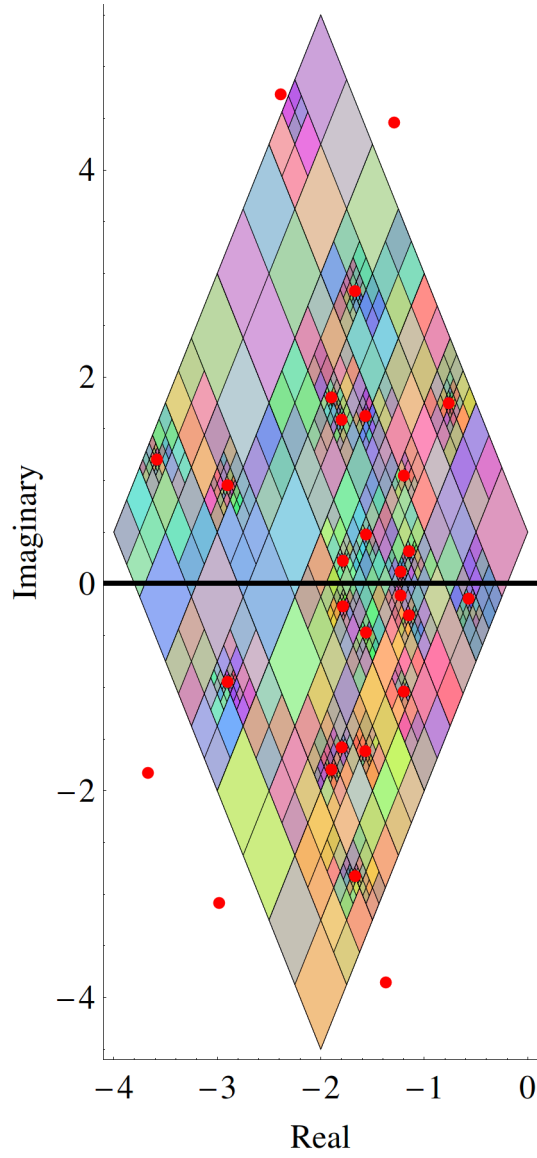


Figure 3.23: Recursive diamond FEAST subdivision process: random complex matrix, complex conjugate eigenvalues, $n = 500$, $m = 19$, $s = 16$

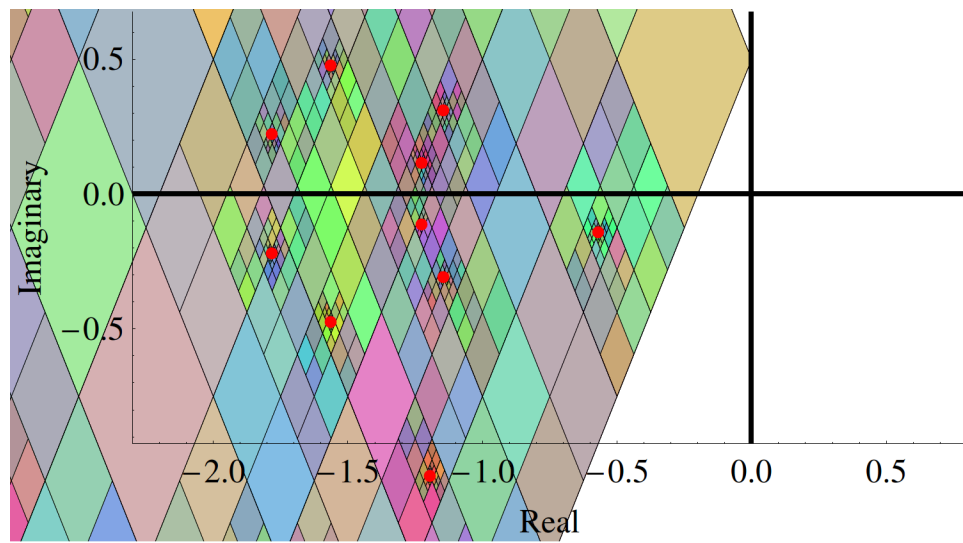


Figure 3.24: Recursive diamond FEAST subdivision process: random complex matrix, complex conjugate eigenvalues, $n = 500$, $m = 19$, $s = 16$

3.6 Conclusions

Based on the tests done on both the rectangles and diamonds, both shapes work equally well in all situations: real vs. complex eigenvalues, clustered vs. complex-conjugate vs. randomized eigenvalues. Regardless of contour shape or location or type of eigenvalues, the algorithm effectively pinpoints the eigenvalues inside the contour.

The next part investigates the contour shapes full, in addition to various numerical integration methods.

Chapter 4

Investigation of Quadrature Schemes

The contour integral around a closed curve can be approximated by various numerical integration methods. The goal was to find out which numerical integration technique gives the most accurate result on various shapes. This accuracy was measured by comparing the computed integration around a single pole to the known value.

4.1 Mathematica Integration Schemes

Mathematica [4] has the following integration schemes built in:

- Trapezoid
- Newton-Cotes

- Gauss-Berntsen-Espelid
- Cartesian
- Gauss-Kronrod
- Multidimensional
- Lobatto-Kronrod
- Levin
- Clenshaw-Curtis
- Monte-Carlo
- Multipanel

Not all of them make sense to use, such as Cartesian or Monte-Carlo. The following integration schemes make sense to use on the circle: Clenshaw-Curtis, Gauss-Berntsen-Espelid, Trapezoid, Gauss-Legendre, Gauss-Kronrod, and Lobatto-Kronrod. Mathematica has code that will give the node locations, weights, and errors for a specified number of integration points. For example, if the values of Gauss-Legendre are desired for q points, call:

NIntegrate ‘GaussRuleData[q, **MachinePrecision**]

In order to get error information, Mathematica compares the scheme using the given number n points with a scheme using the number of points determined by Table 4.1. The value for the higher number of points is returned. In the case of the Trapezoid Rule, we get $2n - 1$ points returned. The code was modified to account for this: if q quadrature points are desired, then $n = \frac{q+1}{2}$ for the input into Clenshaw-Curtis, Lobatto-Kronrod, and Trapezoid. The input for Gauss-Bernsten-Espelid and Gauss-Kronrod were modified

similarly.

Table 4.1
Dependence of quadrature node count on Mathematica argument n

Scheme	Number of nodes returned
Clenshaw-Curtis, Lobatto-Kronrod, Trapezoid	$2n - 1$
Gauss-Bernsten-Espelid, Gauss-Kronrod	$2n + 1$
Gauss-Legendre	n

When the number of points returned is odd, Gauss-Legendre and Gauss-Kronrod have identical node locations and weights; since we only use odd number of nodes we will drop Gauss-Kronrod.

Mathematica outputs the nodes on $[0, 1]$. Since our circles are described in radians the nodes and weights need to be scaled and shifted by:

$$\begin{aligned} \text{nodes} &= \theta_{min} + \text{nodes}(\theta_{max} - \theta_{min}); \\ \{\text{weights}, \text{errweights}\} &= (\theta_{max} - \theta_{min})\{\text{weights}, \text{errweights}\}; \end{aligned} \tag{4.1}$$

Some of the methods are closed (they include the endpoints) and some are open (they do not include endpoints). Table 4.2 lists which scheme is open or closed. Closed schemes with nodes on the real axis affect quadrature accuracy for real poles.

Each quadrature scheme has a type of problem that it works well on. For example, Trapezoid Rule is exact for sine and cosine functions (as exact one can get in floating-point arithmetic). Since circles can be parameterized using sines and cosines, the Trapezoid

Table 4.2
Quadrature scheme type

Method	Type	End Points
Trapezoid	Closed	included
Gauss-Legendre	Open	not included
Lobatto-Kronrod	Closed	included
Clenshaw-Curtis	Closed	included
Gauss-Berntsen-Espelid	Open	not included

Rule should be exact on these shapes. The other schemes are all designed to be exact on polynomials. Table 4.3 shows the maximum degree of a polynomial at which each scheme is exact.

Table 4.3
Quadrature scheme accuracy as a function of Mathematica argument

Scheme	Degree
Trapezoid	$n - 1$
Gauss-Legendre	$2n - 1$
Lobatto-Kronrod	$\begin{cases} 3n - 3 & n \text{ even} \\ 3n - 2 & n \text{ odd} \end{cases}$
Clenshaw-Curtis	$2n - 1$
Gauss-Berntsen-Espelid	$2n + 1$

4.2 Introduction to Quadrature Tests

To test the accuracy of various quadrature schemes, we focus on the approximation of the contour integral from Chapter 2. Equation (2.9) involves the inverse of $zB - A$. Theoretically the inverse of a matrix E can be computed (Theorem 13 from [9]) by

$$E^{-1} = \frac{1}{\text{Det}(E)} \text{Adj}(E) \quad (4.2)$$

where $Det(E)$ is the determinate of E and $Adj(E)$ is the adjugate of E .

For $(zB - A)^{-1}$, the determinate is of degree n and each entry of the adjugate (also called the classical adjoint) is of degree $n - 1$, since it is the transpose of the cofactor matrix (a cofactor matrix has a row/column removed). This gives a matrix of rational functions with simple poles. This was also seen in equation (2.10).

We consider three different categories of matrices: Hermitian, general real, and general complex. Hermitian matrices have only simple real poles, real matrices have simple real poles and complex-conjugate poles, and general complex matrices have simple complex poles.

For real poles the accuracy is determined by the accuracy of the quadrature scheme on simple real poles where x_0 is the pole location:

$$f(z) = \frac{1}{z - x_0} \quad (4.3)$$

For complex-conjugate poles the accuracy is determined by the accuracy of the quadrature on pairs of complex-conjugate poles z_0, \bar{z}_0 :

$$f(z) = \frac{1}{2} \left(\frac{1}{z - z_0} + \frac{1}{z - \bar{z}_0} \right) \quad (4.4)$$

The $\frac{1}{2}$ is included to make the result comparable to a real pole.

For general complex poles the accuracy is determined by the accuracy of the quadrature scheme on simple complex poles for complex z_0 (replace x_0 in (4.3) with z_0).

The contour integral

$$G = \frac{1}{2\pi i} \oint_{\gamma} f(z) dz \quad (4.5)$$

is approximated with q quadrature points, contour parameterization $\gamma(t)$, $\vec{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_q)$ weights, and $\vec{\beta} = (\beta_1, \beta_2, \dots, \beta_q)$ nodes as

$$\hat{G} = \frac{1}{2\pi i} \sum_{j=1}^q \alpha_j f(\gamma(\beta_j)) \gamma'(\beta_j) \quad (4.6)$$

An optimal quadrature scheme would minimize the error inside the contour, as well as the error outside the contour.

The goal of these tests was twofold: do the quadrature schemes get the expected values, and is there a large enough difference in the quadrature results inside and outside the contour that it can be used as an indicator of whether an eigenvalue has been found?

4.2.1 Ellipse Parameterization

Ellipses can be parameterized by

$$z = r_x \cos(\theta) + r_y i \sin(\theta) \quad 0 \leq \theta \leq 2\pi \quad (4.7)$$

where r_x is the horizontal distance from the center, and r_y is the vertical distance from the center. The unit circle is $r_y = r_x = 1$.

4.2.2 Rectangle Parameterization

Define the “unit” square as follows: center at $(0, 0)$ on the complex plane, with sides of length 2, and corners at $(-1, \pm 1i)$ and $(1, \pm 1i)$. See Figure 4.1.

The parameterized equations for a rectangle with constant width of 2 and variable vertical distance r_y are

$$\begin{aligned} \gamma_1(t) &= (-1 + r_y i) - (2r_y i)t & 0 \leq t \leq 1 \\ \gamma_2(t) &= (-1 - r_y i) + 2t & 0 \leq t \leq 1 \\ \gamma_3(t) &= (1 - r_y i) + (2r_y i)t & 0 \leq t \leq 1 \\ \gamma_4(t) &= (1 + r_y i) - 2t & 0 \leq t \leq 1 \end{aligned} \quad (4.8)$$

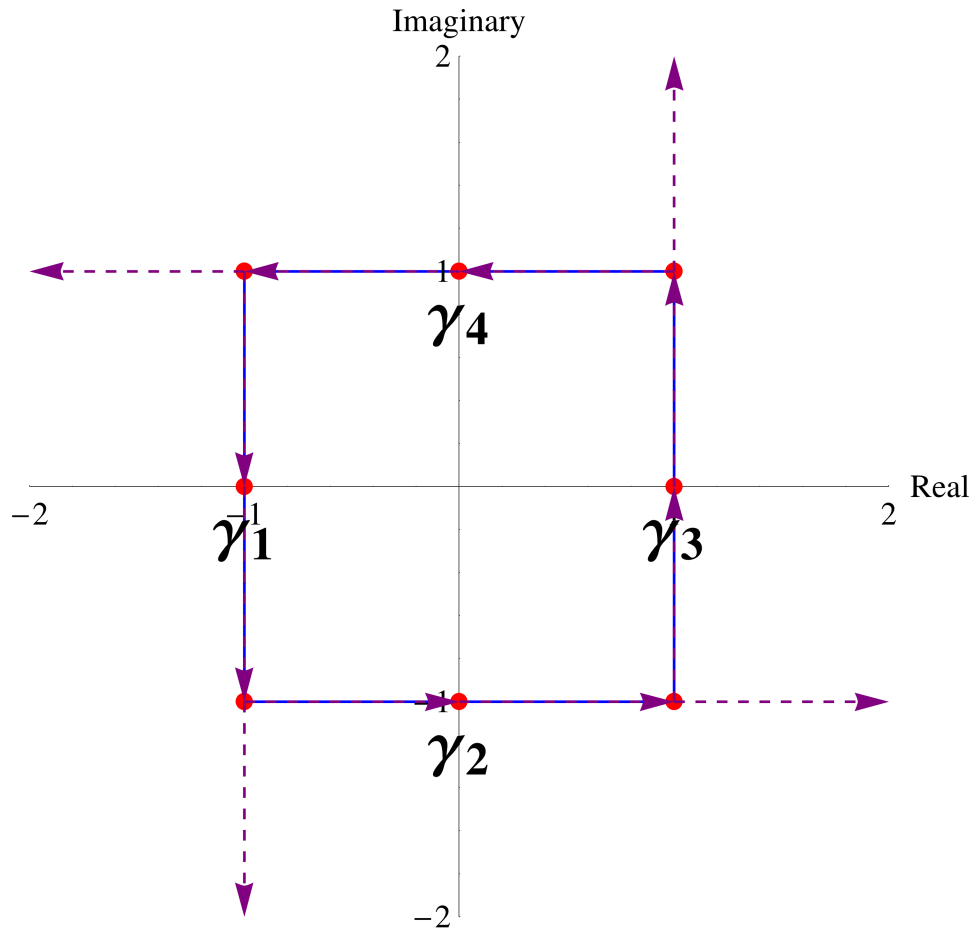


Figure 4.1: Unit Square Contour

The integration of the rectangle is denoted by

$$\hat{G}_{rectangle} = \sum_{i=1}^4 \left[\int_0^1 f(\gamma_i(t)) \gamma_i'(t) dt \right] \quad (4.9)$$

4.2.3 Diamond Parameterization

Define the “unit” diamond as having the corners at $(0, \pm i)$ and $(\pm 1, 0)$, as in Figure 4.2.

We can parameterize the equations for a diamond with constant width 2 as

$$\begin{aligned}
 \gamma_1(t) &= -1 + (1 - r_y i)t & 0 \leq t \leq 1 \\
 \gamma_2(t) &= -r_y i + (1 + r_y i)t & 0 \leq t \leq 1 \\
 \gamma_3(t) &= 1 - (1 - r_y i)t & 0 \leq t \leq 1 \\
 \gamma_4(t) &= r_y i - (1 + r_y i)t & 0 \leq t \leq 1
 \end{aligned} \tag{4.10}$$

with r_y representing the distance the vertical corners are from the $(0, 0)$.

4.3 Testing Quadrature on Circular Contours

The indicator function χ is 1 inside the circle and 0 outside and is used for error checking purposes.

$$\chi(a + bi) = \begin{cases} 1, & (\frac{a}{r_x})^2 + (\frac{b}{r_y})^2 \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{4.11}$$

The quadrature schemes tested were Clenshaw-Curtis (CC), Gauss-Legendre (G),

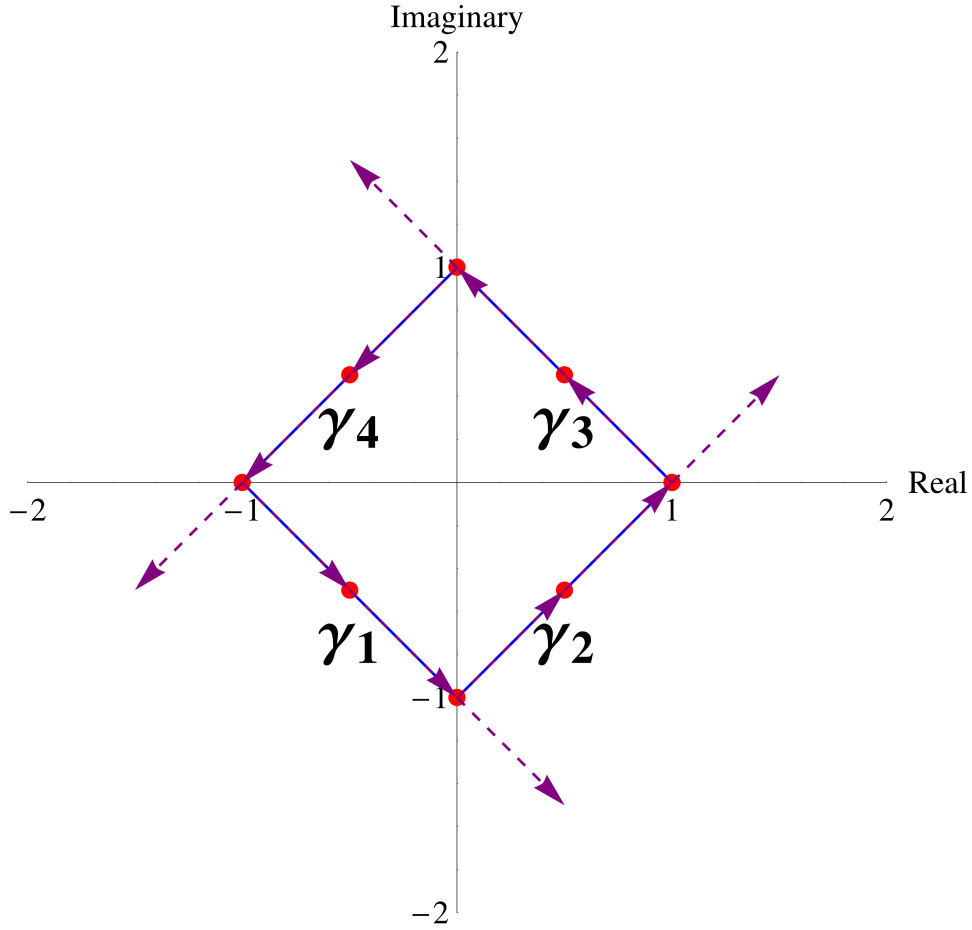


Figure 4.2: Unit Diamond Contour

Lobatto-Kronrod (LK), and Trapezoid (T).

4.3.1 Real Poles

To compare quadrature schemes with Polizzi's [7], use $0 \leq \theta_j \leq \pi$ and multiply the result of the quadrature by 2 due to the symmetry. Since he uses $q = 8$ and Mathematica only outputs an odd number of quadrature points, we decided to make all comparisons with

$q = 7$. Note that increasing the number of quadrature points should decrease error. Since one of goals is to have a difference in output for inside poles versus outside poles, the hope is that this can be achieved with fewer quadrature points.

The expected value of \hat{G} for a pole x_0 inside the contour is 1, while 0 for a pole outside. Figure 4.3 shows that we get a very clear 1 inside the unit circle, and a very clear 0 outside the unit circle for all tested schemes. Clenshaw-Curtis, Lobatto-Kronrod, and Trapezoid all have asymptotes near the contour edge. This greatly affects the accuracy, especially for Trapezoid where there is a steep change in value around ± 0.9 . This implies that for values near the contour edge, Trapezoid is less accurate than other schemes. Gauss has no asymptotes, so would be the most accurate on the edges.

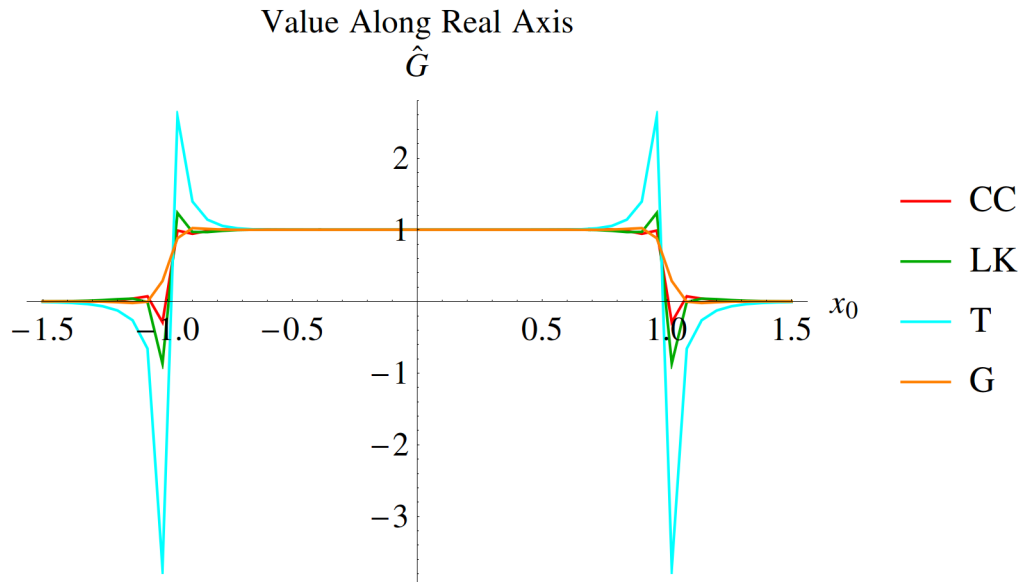


Figure 4.3: Quadrature comparison: Aspect Ratio = 1, $q = 7$, real poles, circle contour

The second question of interest is if there is a large difference in \hat{G} inside and outside the

contour. Figure 4.3 shows that there is a large difference most of the time, showing that the algorithm can work as an indicator. The problem is again near the contour edge, where Gauss gradually drops off. The gradual change in value means that near the contour it may be hard to tell if the result indicate an eigenvalue inside or outside the contour. If one is looking for a contrast, then perhaps Trapezoid would be best because the asymptotes on the edges give a clear picture whether the eigenvalue is inside or outside.

It is worth taking a look at the quadrature error by comparing it to the indicator function (4.11). The results are in Figure 4.4. All schemes struggle on the endpoints of the contour. The Trapezoid rule has the smallest error in the center of the circle.

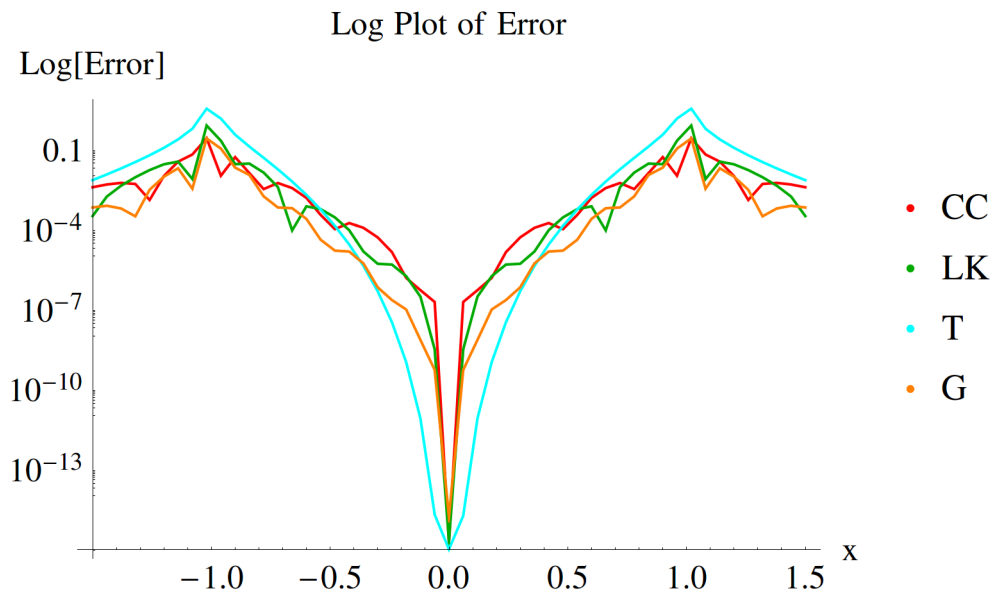


Figure 4.4: Comparison of quadrature error: Aspect Ratio = 1, $q = 7$, real poles, circle contour

What happens at different aspect ratios (vertical distance from the center)? Does the quadrature still approximate 1? As a specific pole to test in equation (4.3) set $x_0 = 0$.

Figure 4.5 shows that the accuracy of the integration depends on the aspect ratio of the circle. As expected, the Trapezoid Rule gets values closest to 1 at all radii, though for an aspect ratio < 0.5 all schemes start to vary from the expected value.

Figure 4.6 shows that for aspect ratio = 0.5 the integration does get the proper values for most of the inside of the contour. The results are comparable to those of the unit circle in Figure 4.3. Notice that Gauss has a smaller section near the contour edge (± 1) that has a gradual drop-off, which means that as an indicator the smaller aspect ratio is better when using that particular quadrature scheme.

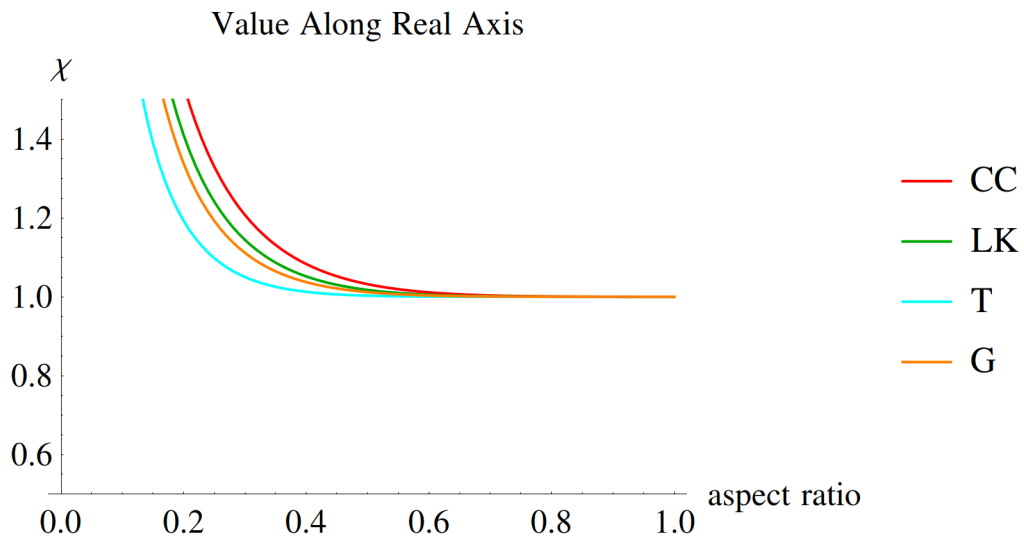


Figure 4.5: Comparisons of quadrature for different aspect ratios: $q = 7$, pole at $x_0 = 0$, circle contour

Things get interesting as the aspect ratio decreases. Figure 4.7 shows that the output of the integration for values inside the circle vary but are near 1, while the result for values outside the contour stay near 0. These plots are where the Trapezoid rule shines, as the difference from 1 inside the circle is much smaller than for the other methods. When considering the

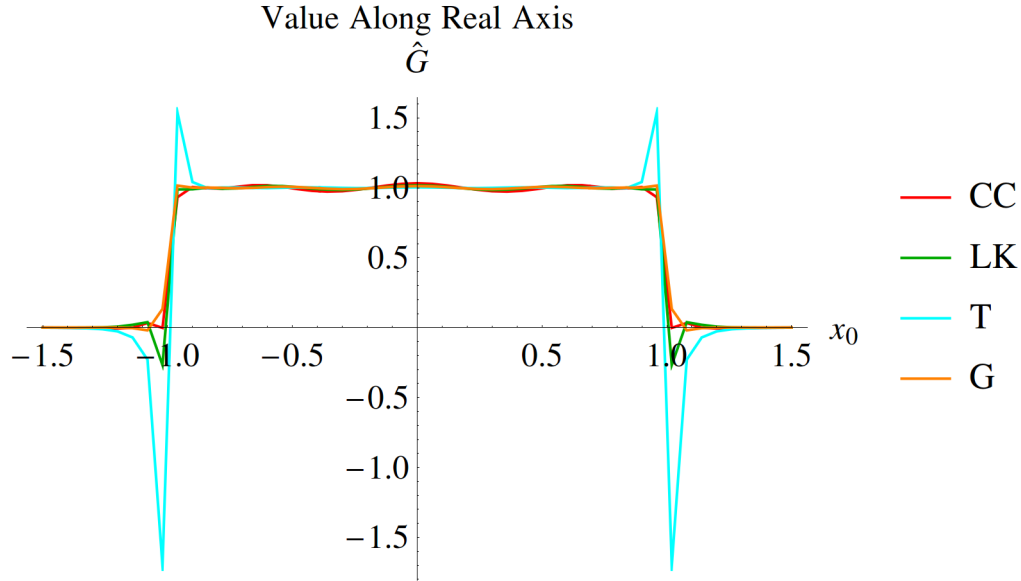


Figure 4.6: Quadrature comparison: Aspect Ratio = 0.5, $q = 7$, real poles, circle contour

alternate question of acting as an indicator, any of the schemes would work because in that case the actual value of \hat{G} doesn't matter so much as the difference between the inside and outside. There is quite a large difference for all schemes tested.

Figure 4.8 shows that the value of the contour integral cannot be trusted at all for poles inside an circle of aspect ratio 0.1. However, there is a very clear zero for poles outside the contour, so if one wanted to use the integration result as an indicator of whether a pole is inside or outside, any value of integration above zero would indicate a pole inside the contour.

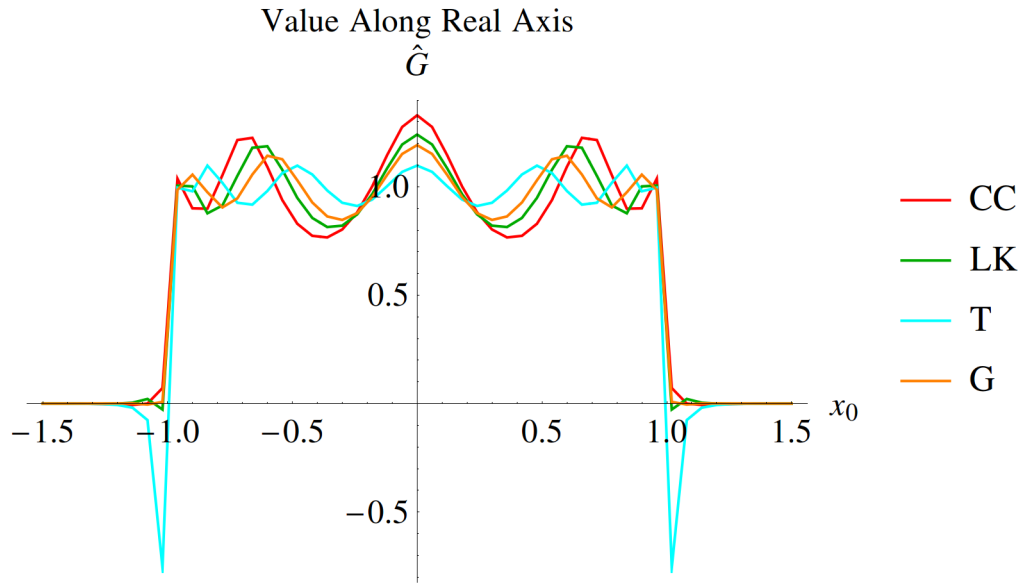


Figure 4.7: Quadrature comparison: Aspect Ratio = 0.25, $q = 7$, real poles, circle contour

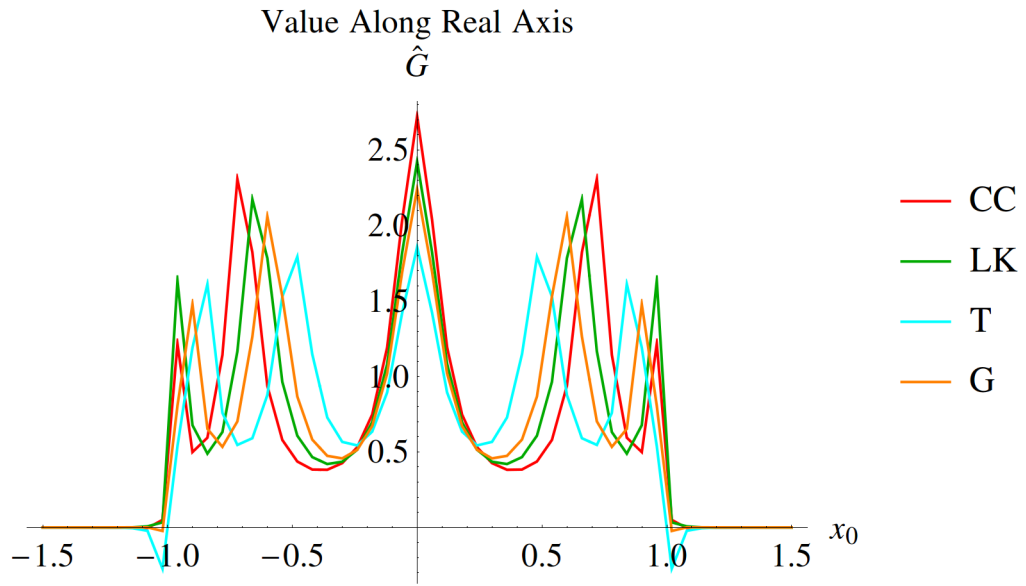


Figure 4.8: Quadrature comparison: Aspect Ratio = 0.1, $q = 7$, real poles, circle contour

4.3.2 Complex-Conjugate Poles

The contour integral on an circle with complex-conjugate poles is similar to that of real poles. There is symmetry across the real-axis. All nodes are between 0 and π and the result is multiplied by 2, as with real poles.

Equation (4.4) was used as the function we integrated over. Figure 4.9 shows the error of the integration of the circle when there are 2 poles that are complex-conjugates. The error is very consistent no matter what quadrature scheme is used, nor where the complex-conjugate poles are located. The only locations that have high error are near the quadrature points. Clenshaw-Curtis has the highest error at above 0.4, as seen from the legend in figure 4.9. Trapezoid rule appears to have the smallest band of error above zero, which is expected based on the nature of the shape (See the discussion in Section 4.1).

Since all four figures have the bulk of the area in dark purple representing an error below 0.1, the schemes are quite accurate. Additionally, aside from near the contour edge, this means that there is a large gap between the inside and outside integration values (inside is 1 ± 0.1 and outside is 0 ± 0.1). It would be problematic if the error was close to 0.5 on both sides, as it means that \hat{Q} could be the same for the poles inside the contour as the for poles outside.

What happens to the accuracy as the aspect ratio of the circle decreases? Figure 4.10 shows

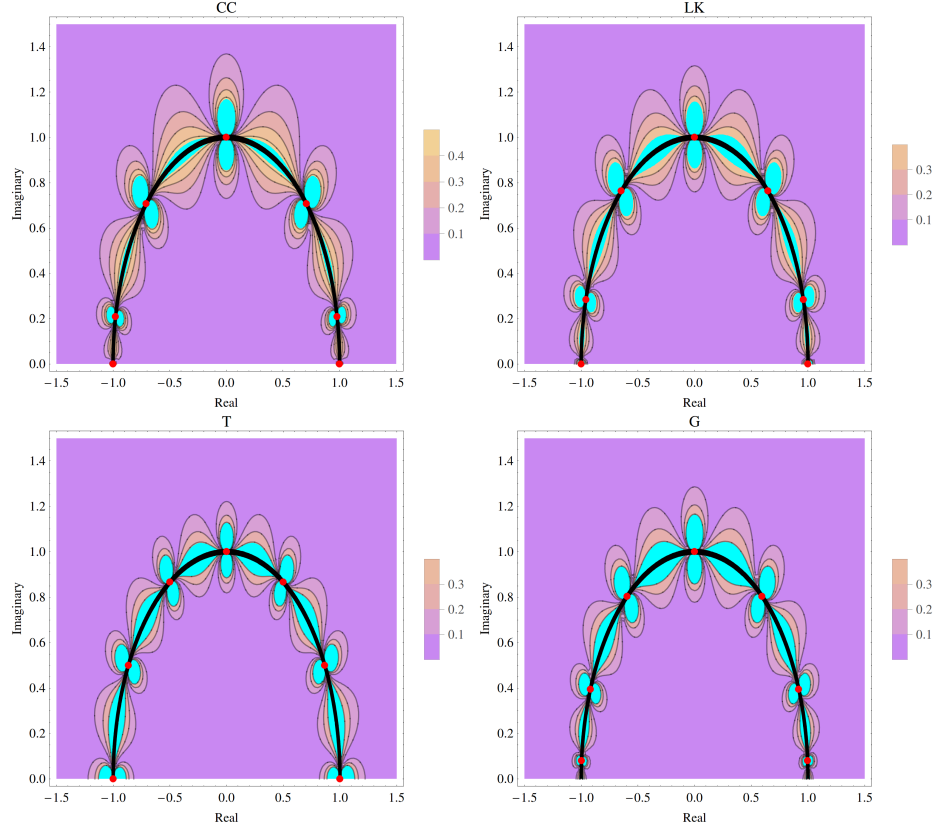


Figure 4.9: Comparison of quadrature error: Aspect Ratio = 1, $q = 7$, complex-conjugate poles, circle contour

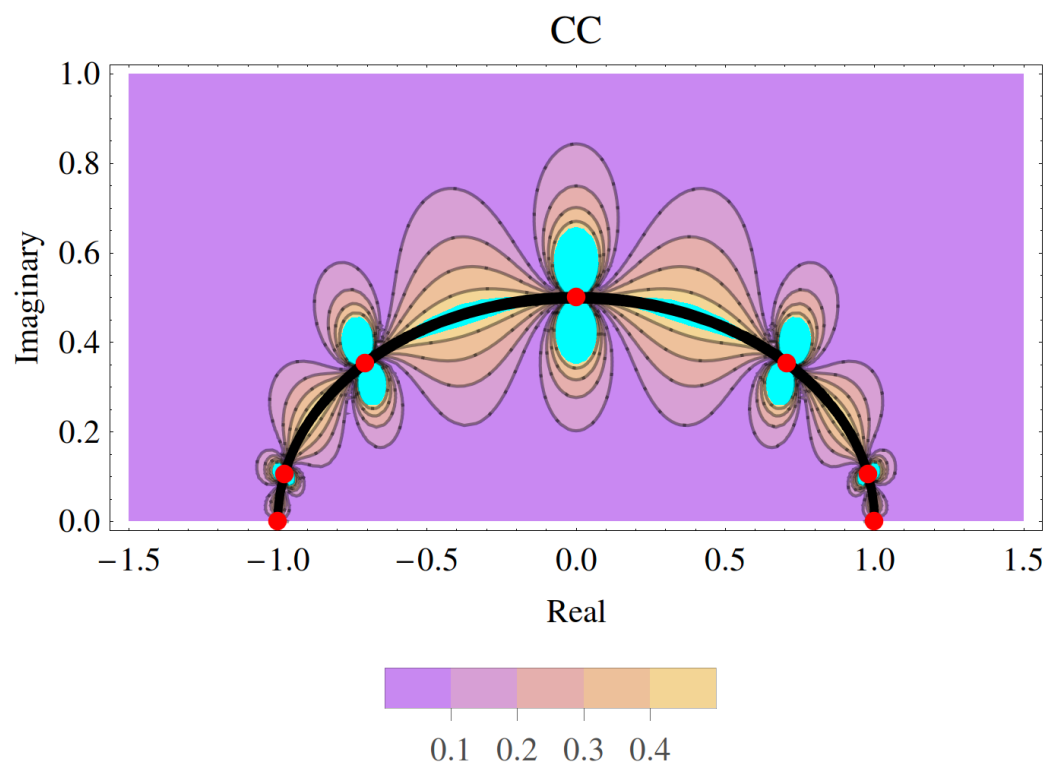
the error for Clenshaw-Curtis as the aspect ratio decreases. At an aspect ratio of 0.5 the real-axis has no error, but as the aspect ratio decreases the error on the real-axis increases. This can be seen in Figure 4.10b, where the real axis (bottom of the plot) no longer is dark purple but rather has some contours of 0.2 and 0.3.

What about Gauss-Legendre? Figure 4.11 shows what happens as the aspect ratio decreases. The same phenomenon that occurred for Clenshaw-Curtis occurs for Gauss-Legendre, though there is a larger area with no error: For Clenshaw-Curtis the contour of 0.1 is hit at about $0 + 0.2i$ in Figure 4.10a, while for Gauss-Legendre it appears to

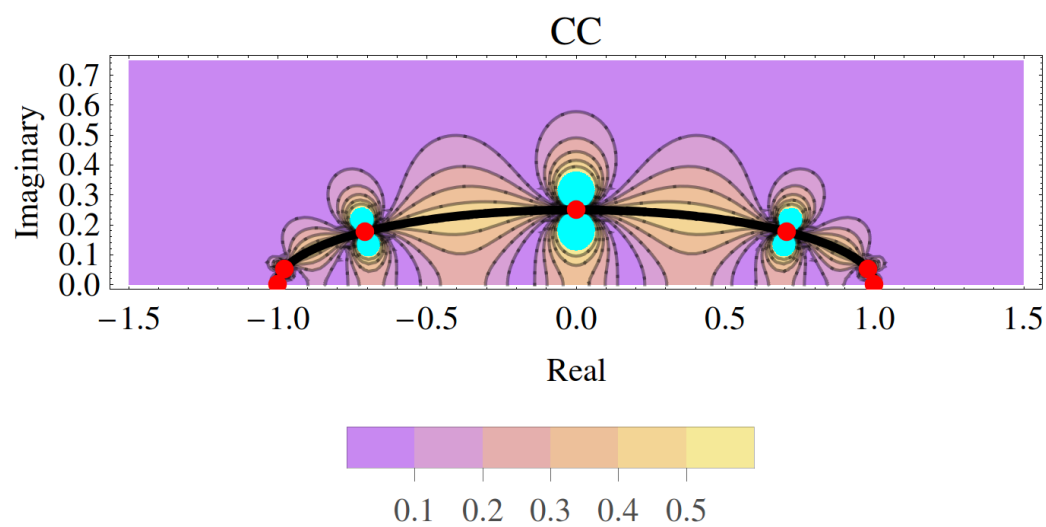
be hit at $0 + 0.25i$ in Figure 4.11a. Also note the difference in the legends: Clenshaw-Curtis in Figure 4.10a has a contour of 0.5 while Gauss-Legendre in Figure 4.11a has its highest contour at 0.4.

Trapezoid Rule is even better: Figure 4.12 shows an even larger area of zero error. Figure 4.12a shows that the next contour is hit at $0 + 0.3i$ and the maximum contour on the legend is 0.3 as opposed to 0.4 and 0.5 for Gauss-Legendre and Clenshaw-Curtis respectively.

For Lobatto-Kronrod the data is similar to that of Gauss-Legendre: a contour in Figure 4.13a is hit around $0 + .25i$, and the highest contour on that figure is 0.4.

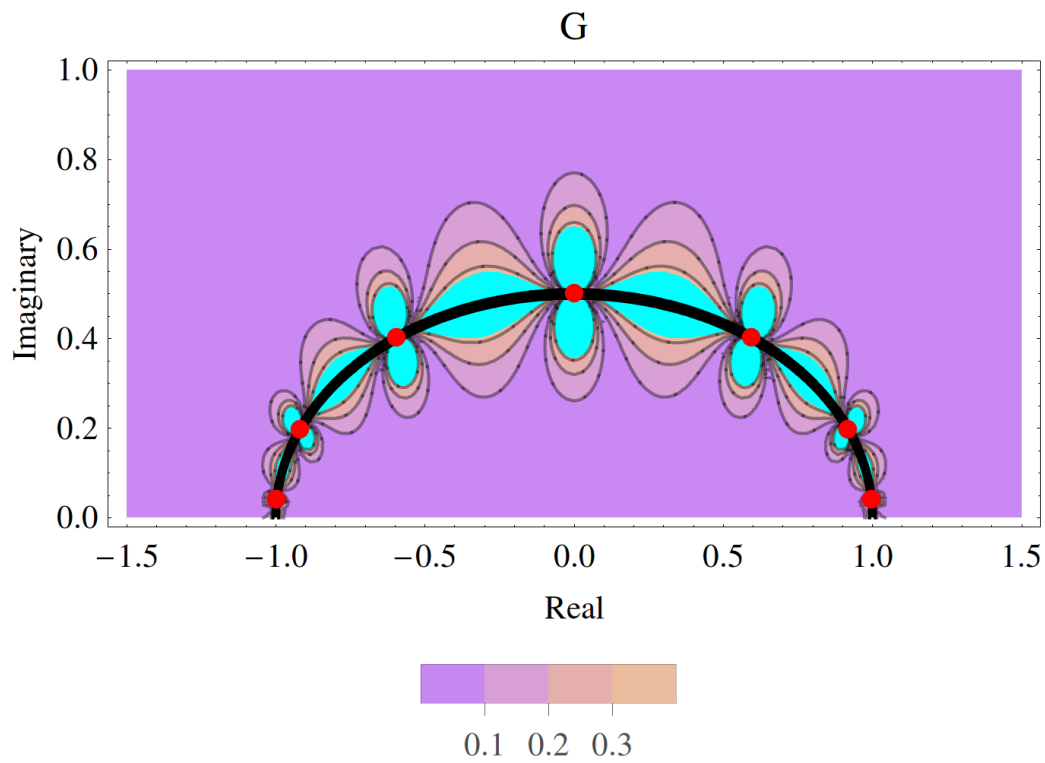


(a) Aspect Ratio 0.5

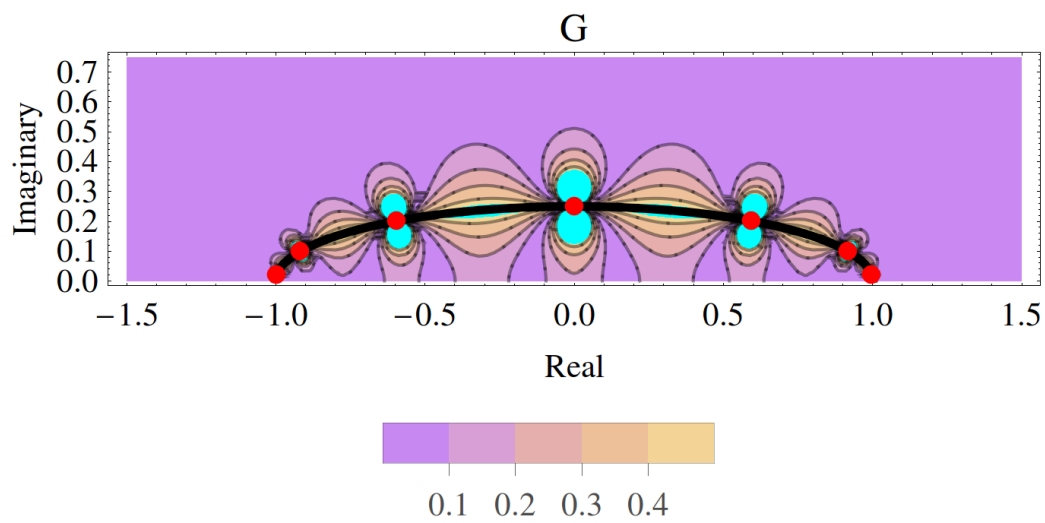


(b) Aspect Ratio 0.25

Figure 4.10: Dependence on aspect ratio: Clenshaw-Curtis Quadrature, $q = 7$, complex-conjugate poles, circle contour



(a) Aspect Ratio 0.5



(b) Aspect Ratio 0.25

Figure 4.11: Dependence on aspect ratio: Gauss-Legendre Quadrature, $q = 7$, complex-conjugate poles, circle contour

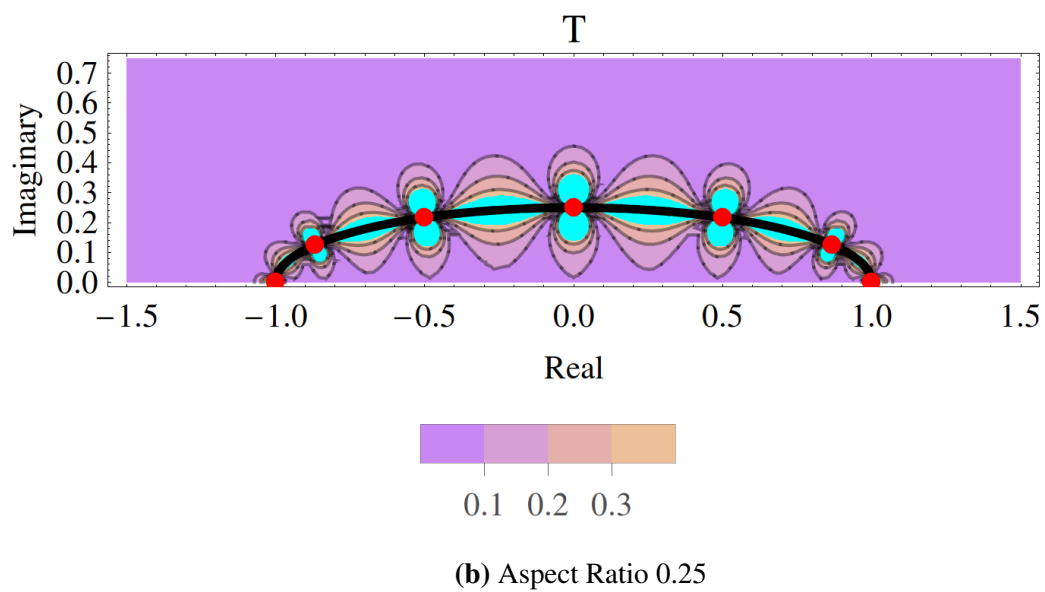
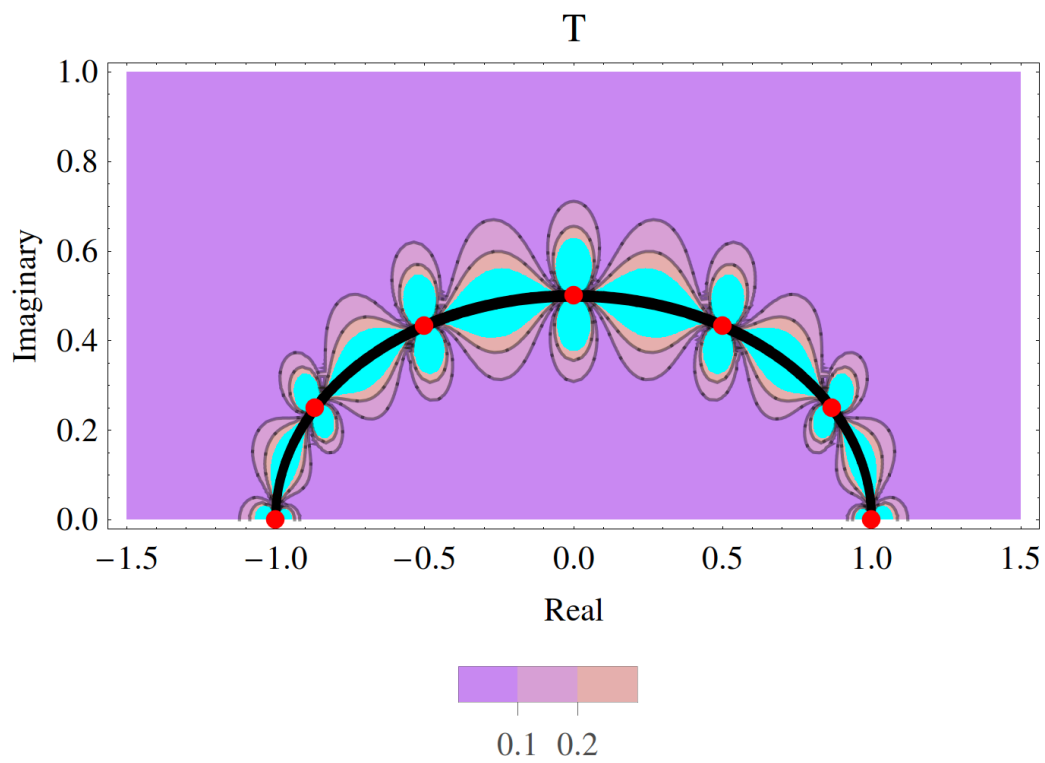
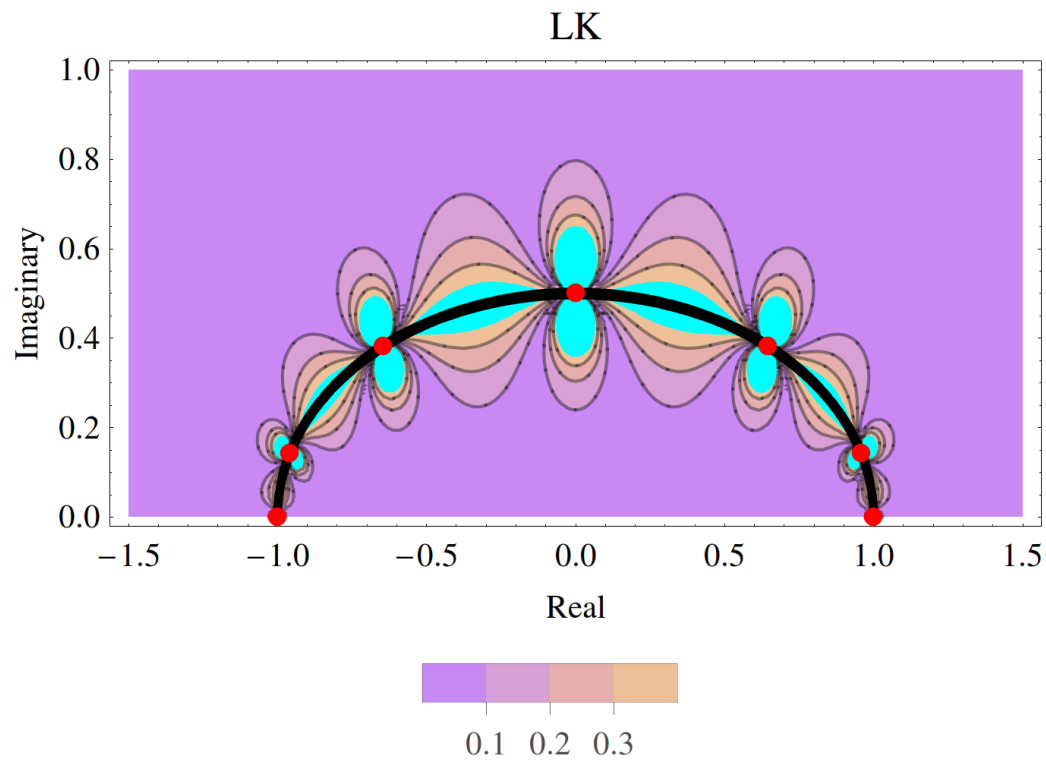
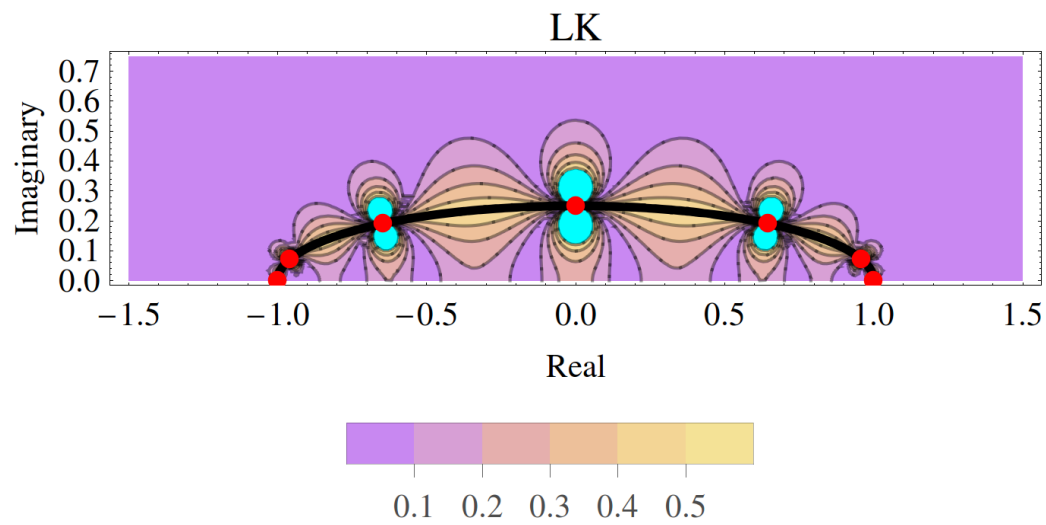


Figure 4.12: Dependence on aspect ratio: Trapezoid Quadrature, $q = 7$, complex-conjugate poles, circle contour



(a) Aspect Ratio 0.5



(b) Aspect Ratio 0.25

Figure 4.13: Dependence on aspect ratio: Lobatto-Kronrod Quadrature, $q = 7$, complex-conjugate poles, circle contour

4.3.3 Complex Poles

The third case of the contour integral involves a complex pole. Since there is no guaranteed symmetry within the shape, we integrate over the whole circle. Figure 4.14 shows the error for all 4 methods. In these plots it is easy to see how the methods differ in regards to their node locations. Note that Clenshaw-Curtis, Gauss-Legendre, and Trapezoid all show six node locations because the seventh is overlapped on the first one. In these plots notice how when the nodes are close to each other, the bound of the error on the edge of the contour is much smaller than those nodes that are further away. This implies that increasing the number of nodes should decrease the error, as the nodes will be closer to each other.

Figure 4.15 shows how the aspect ratio affects the error when using the Clenshaw-Curtis quadrature points. The plot of the complete circle gives a very different story than that of the semi-circle. There is symmetry across the real-axis due to the quadrature evaluation points being symmetric over the real-axis. The error near the quadrature points of the top and bottom seem to connect as the aspect ratio decreases, which means that less inside area of the contour has small error. In fact, very few areas on the inside seem to have an error below 0.1, while the outside stays consistent with much of the area having error below 0.1. For purposes of using the algorithm as an indicator of eigenvalues, this is not a problem. As long as the error on either sides of the contour is below 0.5, then there is a clear difference in the value of \hat{Q} for a pole inside the contour and for a pole outside the contour. However,

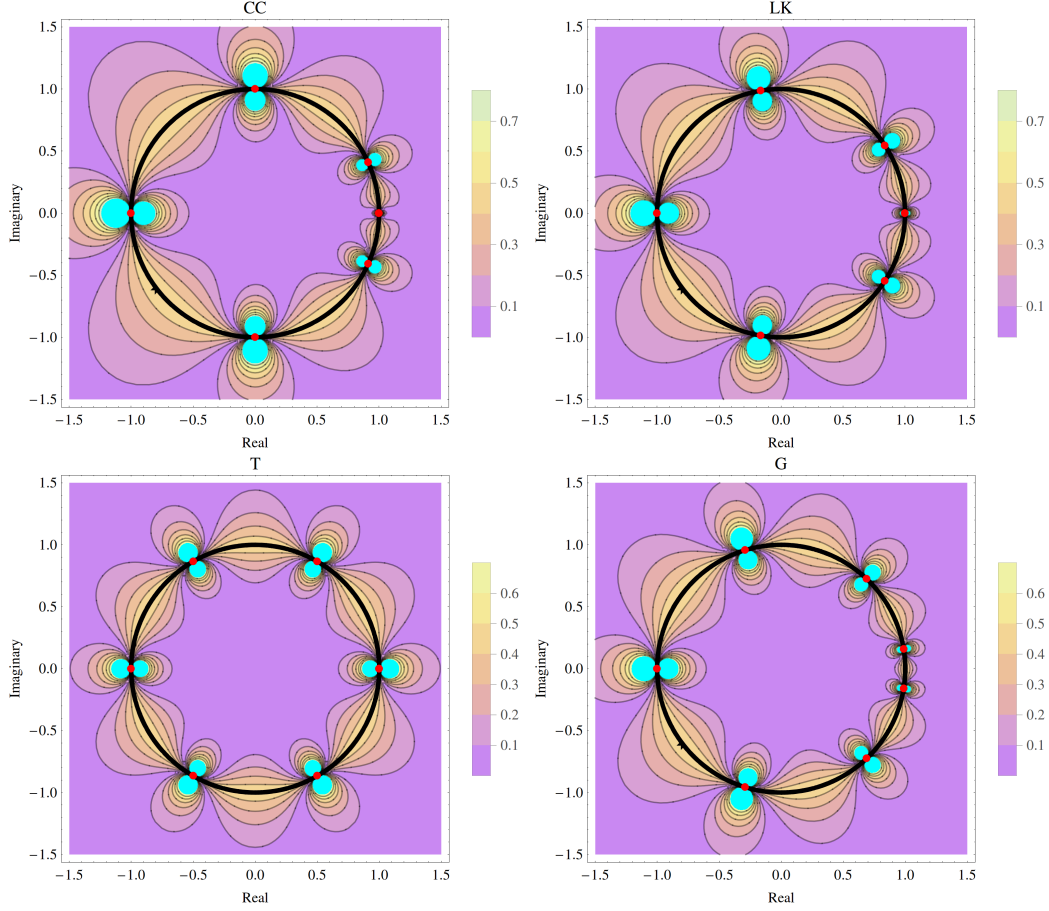
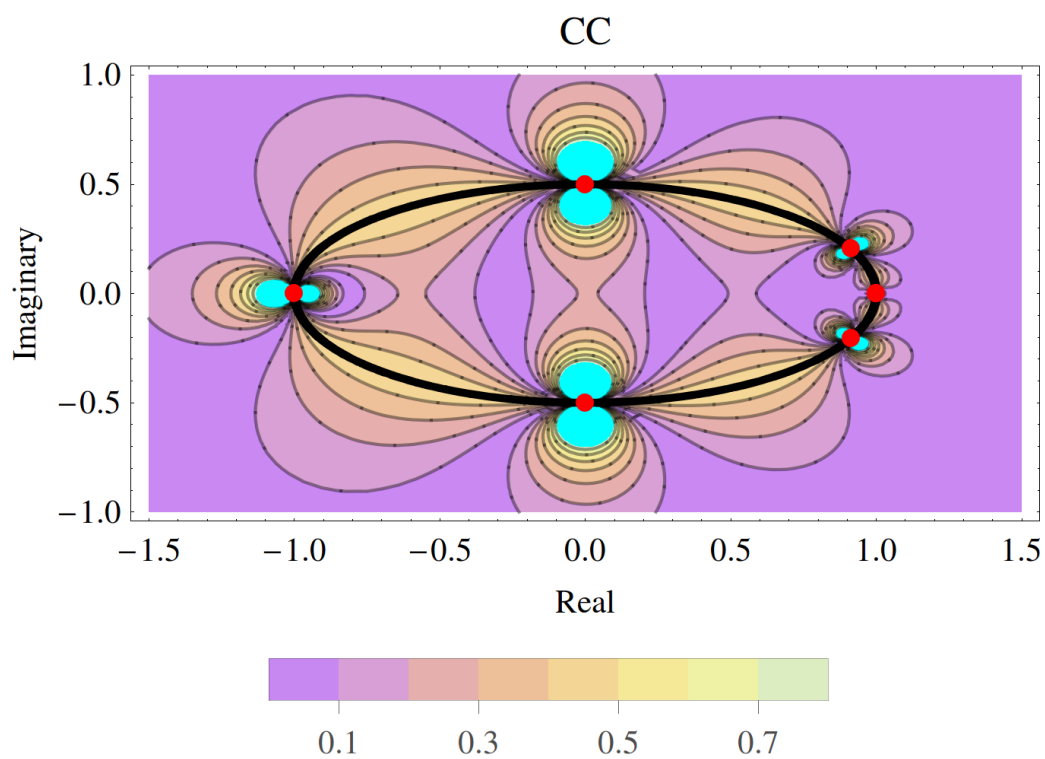


Figure 4.14: Comparison of quadrature error: Aspect Ratio = 1, $q = 7$, complex poles, circle contour

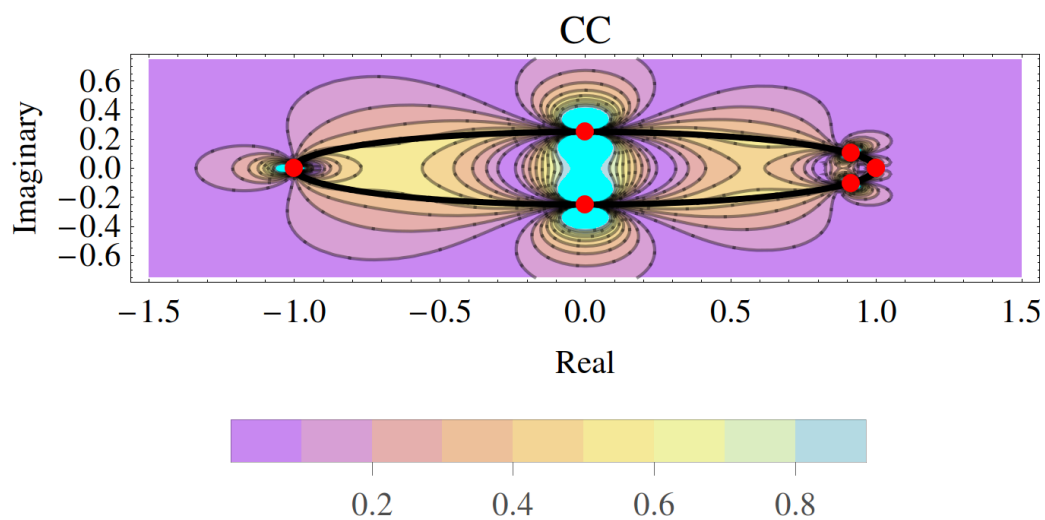
figures 4.14 and 4.15 show that the maximum contour for Clenshaw-Curtis is 0.8 for aspect ratios of 1 and 0.5, and 0.9 for an aspect ratio of 0.25. This means that at worst the value of \hat{Q} could be 0 ± 0.8 and 1 ± 0.8 which clearly shows there could be some overlap in the possible values.

The same story is told when looking at the figure for Gauss-Legendre (4.16), Trapezoid (4.17), and Lobatto-Kronrod (4.17). The maximum contour level is above 0.5 which is a problem when using the plots as indicators. Smaller aspect ratios also cause higher errors,

another issue.



(a) Aspect Ratio 0.5



(b) Aspect Ratio 0.25

Figure 4.15: Dependence on aspect ratio: Clenshaw-Curtis Quadrature, $q = 7$, complex poles, circle contour

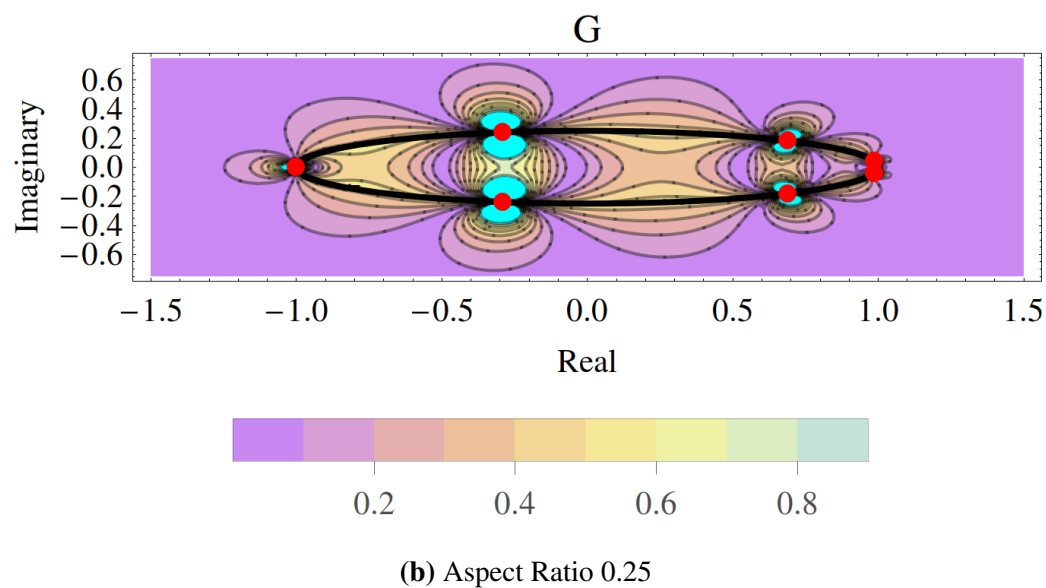
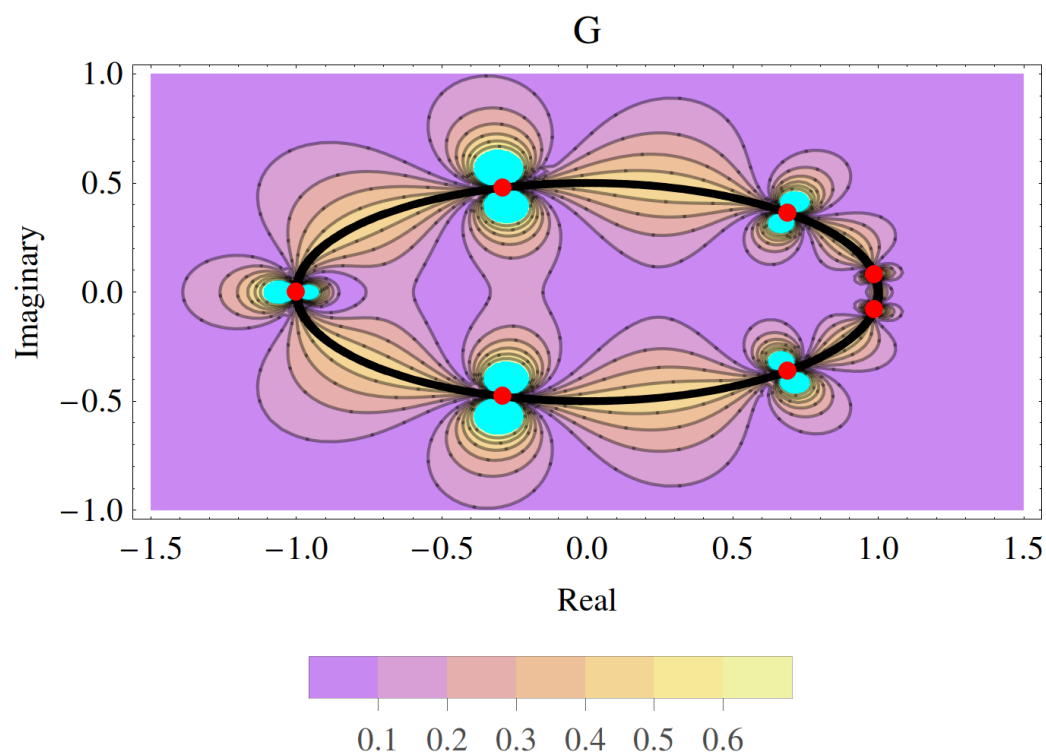
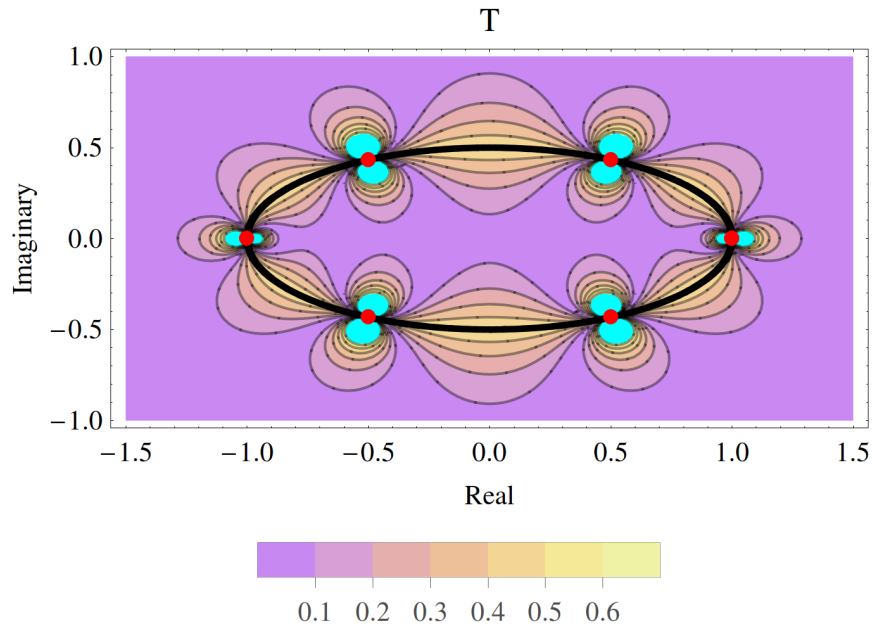
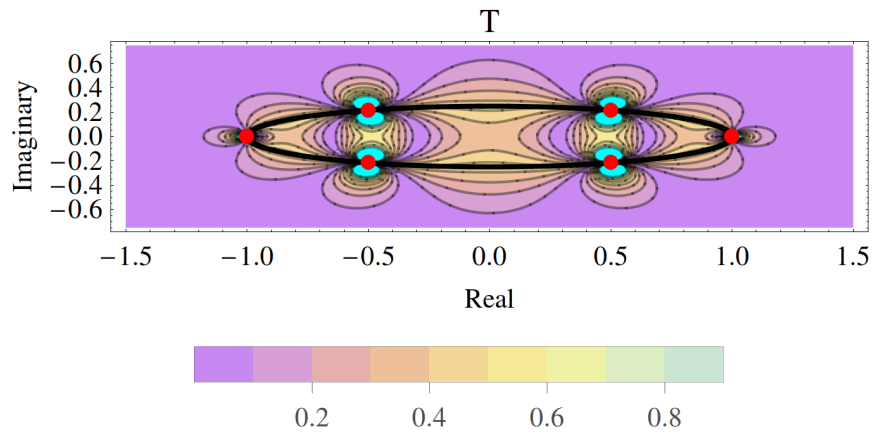


Figure 4.16: Dependence on aspect ratio: Gauss-Legendre Quadrature, $q = 7$, complex poles, circle contour

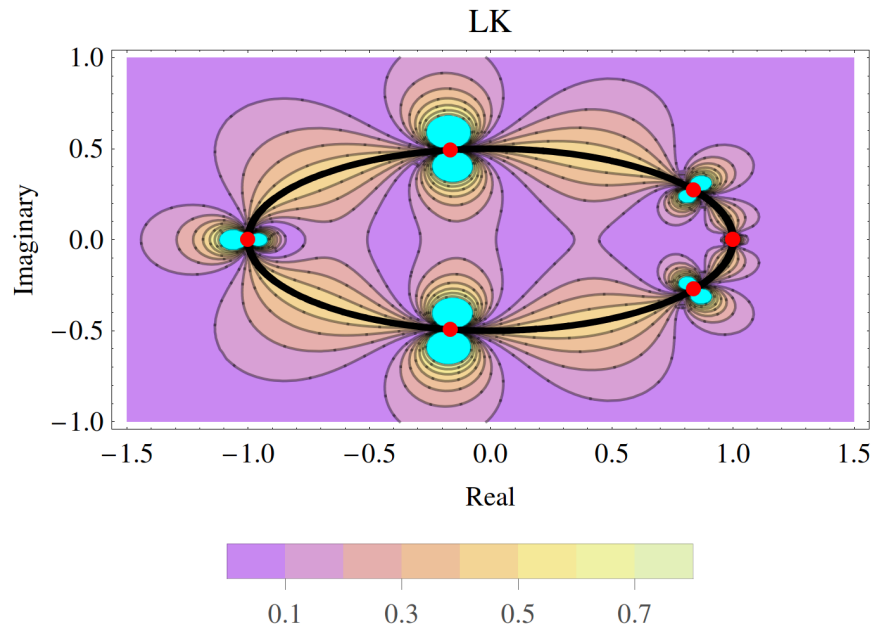


(a) Aspect Ratio 0.5

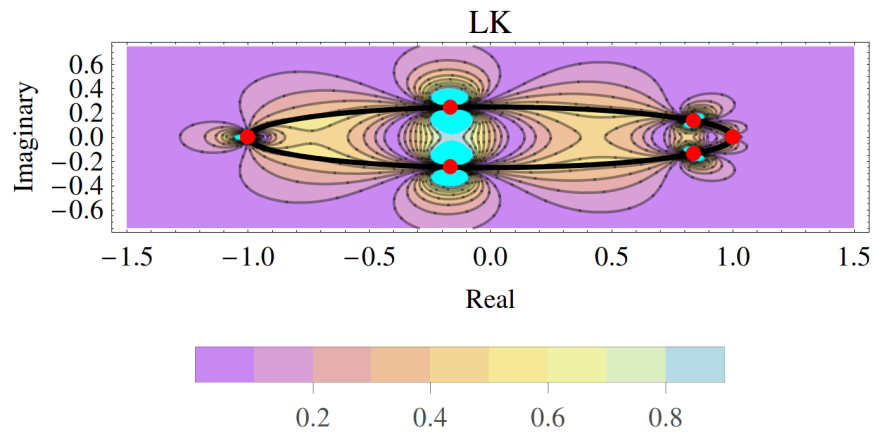


(b) Aspect Ratio 0.25

Figure 4.17: Dependence on aspect ratio: Trapezoid Quadrature, $q = 7$, complex poles, circle contour



(a) Aspect Ratio 0.5



(b) Aspect Ratio 0.25

Figure 4.18: Dependence on aspect ratio: Lobatto-Kronrod Quadrature, $q = 7$, complex poles, circle contour

4.4 Testing Quadrature on Rectangular Contours

For rectangular contours, we are again looking at accuracy and whether there is a large difference in the value of \hat{Q} for poles inside and outside the contours. A third thing to look at is whether rectangles will have smaller error when looking at complex poles, since ellipses had high amounts of error.

We can numerically approximate the integral over the contour by the Midpoint Rule:

$$\int_0^1 f(\gamma(t)) \gamma'_i(t) dt \approx \sum_{i=1}^4 f(\gamma(0.5)) \gamma'_i(t) \quad (4.12)$$

Trapezoid Rule:

$$\int_0^1 f(\gamma(t)) \gamma'_i(t) dt \approx \sum_{i=1}^4 0.5 \left(f(\gamma(0)) + f(\gamma(1)) \right) \gamma'_i(t) \quad (4.13)$$

Simpson's Rule:

$$\int_0^1 f(\gamma(t)) \gamma'_i(t) dt \approx \sum_{i=1}^4 \frac{1}{6} \left(f(\gamma(0)) + 4f(\gamma(0.5)) + f(\gamma(1)) \right) \gamma'_i(t) \quad (4.14)$$

Simpson's Rule is known to be more accurate than the Trapezoid Rule and Midpoint Rule over polynomials.

4.4.1 Real Poles

Figure 4.19 shows the value of various quadrature schemes over the real axis and the unit square. The goal is to have a value of 1 for pole locations between -1 and 1 , and a value of 0 for pole locations outside. Midpoint rule has a gradual taper from 1 to 0, which is not accurate. Trapezoid and Simpson's rule both have asymptotes near the edge of the contour. If what matters is just a sharp difference for pole locations inside versus pole locations outside, then Midpoint rule should not be used at all.

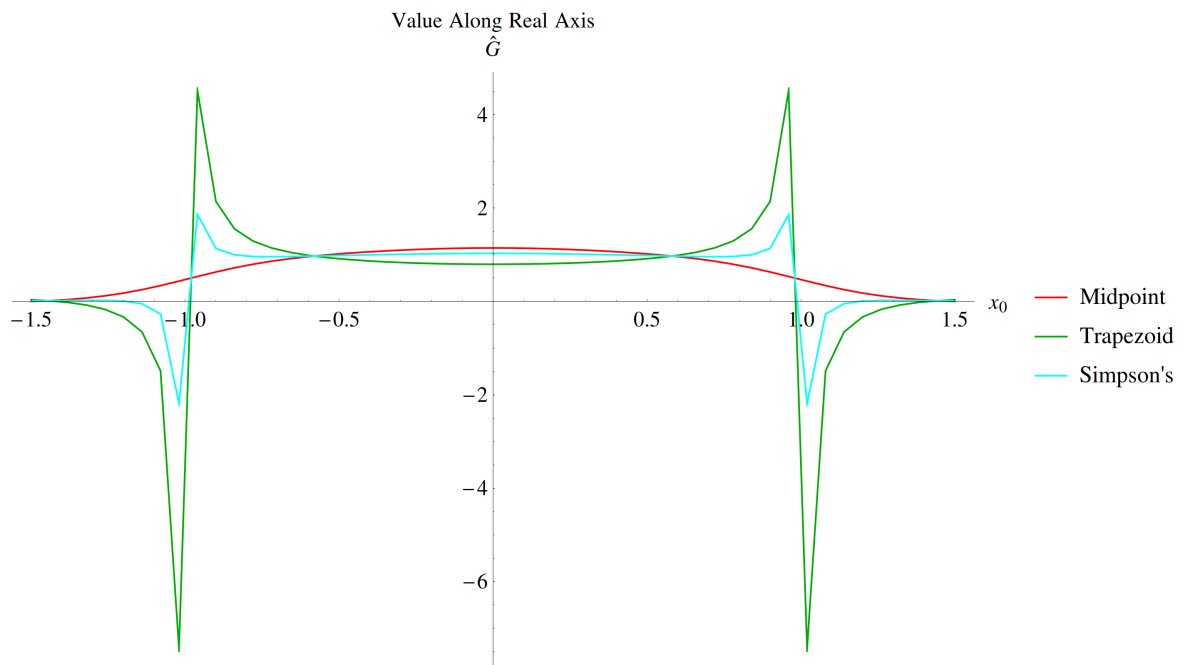


Figure 4.19: Quadrature comparison: Aspect Ratio = 1, real eigenvalues, rectangle contour

Figure 4.20 shows for a particular pole location $x_0 = 0$ the result of the integration for Midpoint, Trapezoid, and Simpson's Rules at different aspect ratios. Since this location

is inside the contour, the expected value is 1 (the dashed line on the plot). As expected, Simpson's Rule is the most accurate though it does not get the result exactly. At an aspect ratio of 1 all three schemes are at their closest to the expected value of 1.

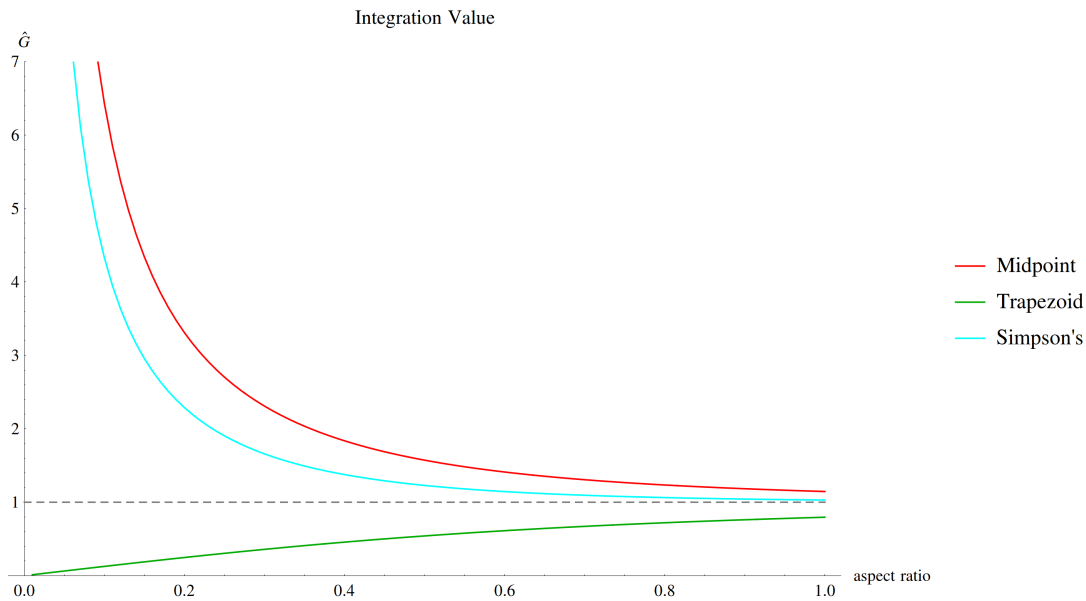


Figure 4.20: Comparisons of quadrature for different aspect ratios: real eigenvalues, rectangle contour, $x_0 = 0$

4.4.2 Complex-Conjugate Poles

For complex-conjugate eigenvalues, Simpson's Rule is the best choice as it has the smallest error on the inside of the contour, as seen in Figure 4.21. The maximum error contour for Simpson's Rule is 0.07, while for Midpoint and Trapezoid it is 0.09. This is not good when looking for a difference between poles inside and poles outside. If we look at the edges of the contour, we want the color to be consistent at the transition from both sides. This tells us that the difference between the two is consistent with what we expect and

we are able to use the scheme. Notice that both Midpoint and Trapezoid have some areas where the color of the contour changes as we transition from inside to outside the rectangle. This means we should not use these schemes when using our algorithm as an eigenvalue indicator. Simpson's Rule will still work, and it has a very large area with an error below 0.1.

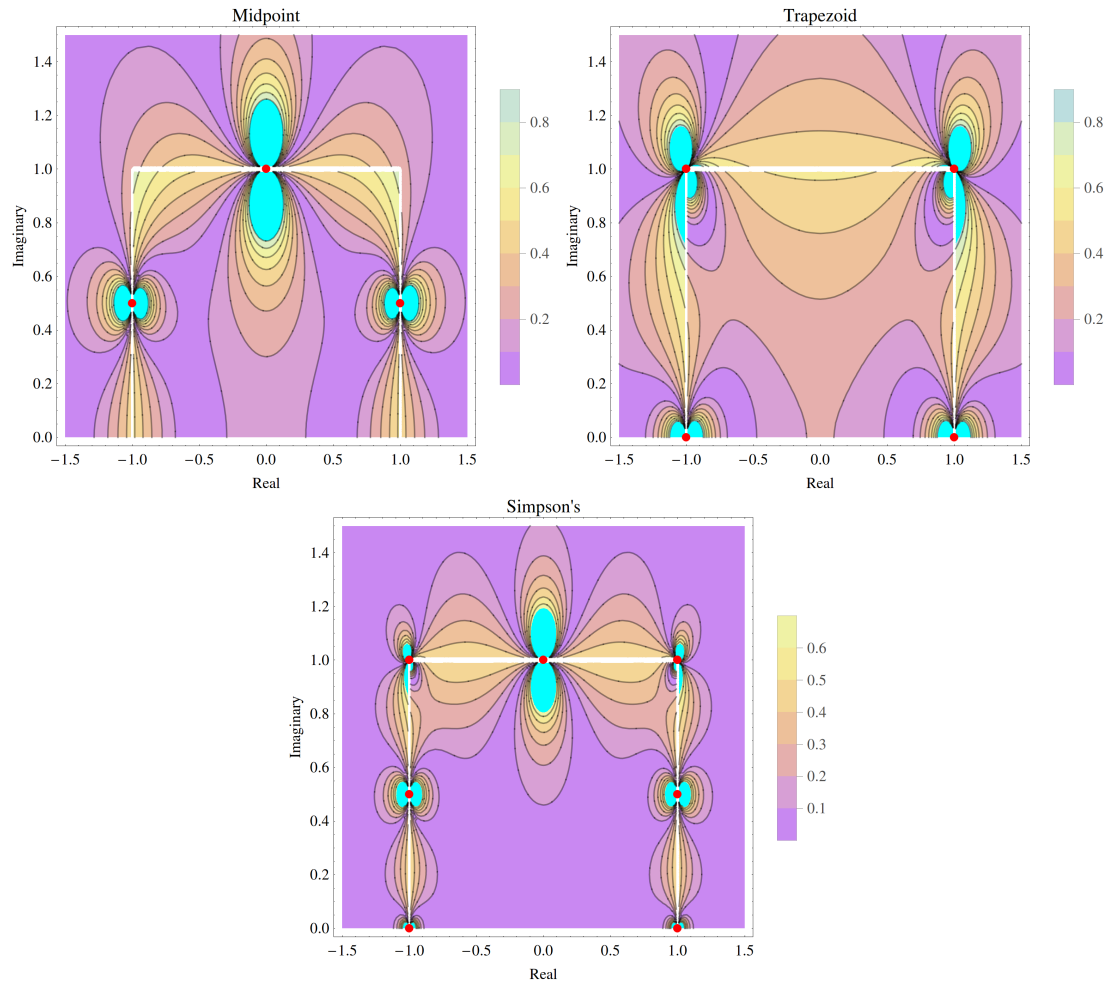


Figure 4.21: Comparison of quadrature error: Aspect Ratio = 1, complex-conjugate eigenvalues, rectangle contour

4.4.3 Complex Poles

Figure 4.22 shows the contour plots of the error for the three schemes. Simpson's Rule is the only one with an error less than 0.1 on the very inside. Simpson's Rule is also the only one with an error close to zero for the bulk of the outside of the contour as well, which is also desirable. If we look at the transition from inside the contour to outside, Simpson's rule has consistent colors which means there is enough of a difference between the inside and outside that we can use it as an indicator of eigenvalues.

Figure 4.23 shows what happens when we cut r_y in half. All three schemes suffer from the points influencing each other and causing "bands" of high error. If we compare this to our ellipses, the ellipse works much better at the same aspect ratio because the points do not influence each other yet. Based on the results, it seems that ellipses would be better to work with than rectangles. And additional thing to look at is the maximum contour. For Midpoint the maximum is 1, for Trapezoid it is 1.1, and for Simpson's it is 0.7. In regards to the transition from inside to outside the contour, only Simpson's Rule is consistent with the amount of error.

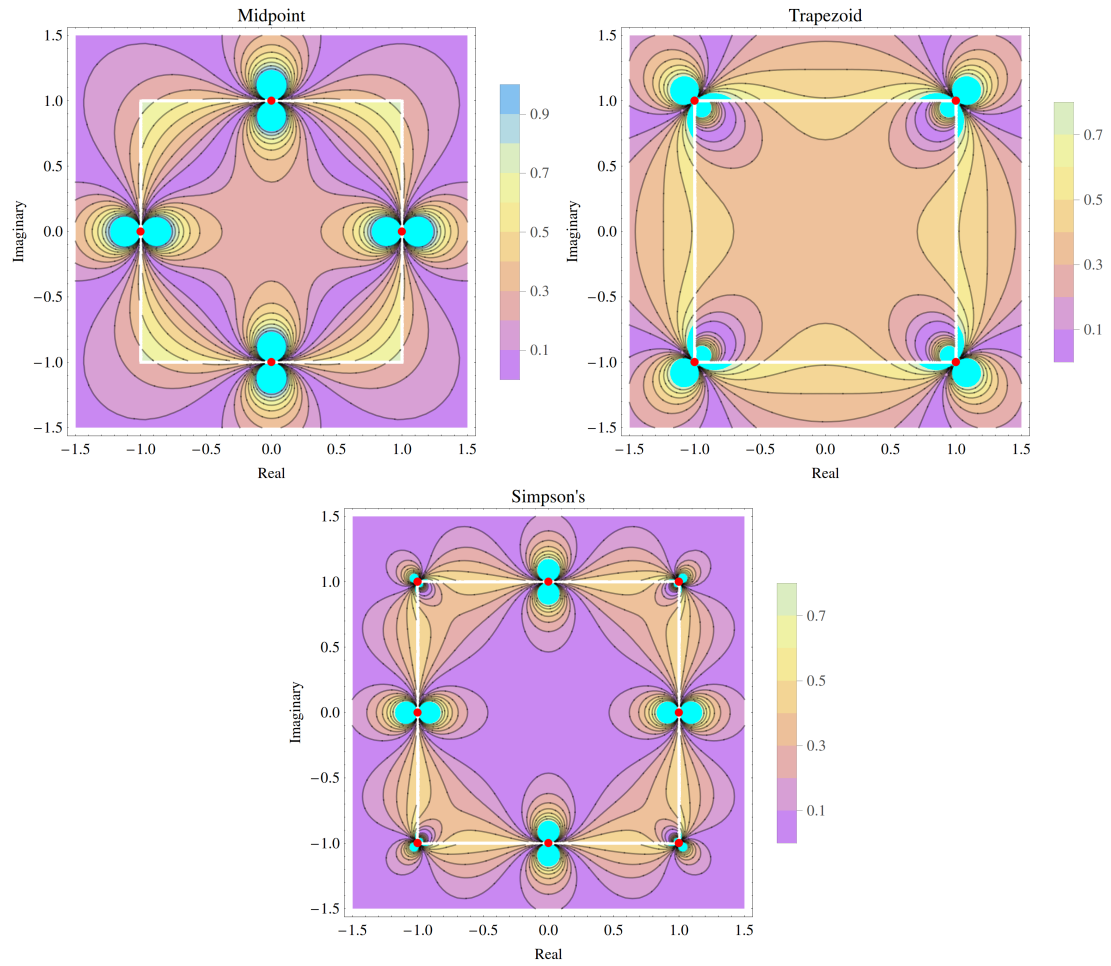


Figure 4.22: Comparison of quadrature error: Aspect Ratio = 1, complex eigenvalues, rectangle contour

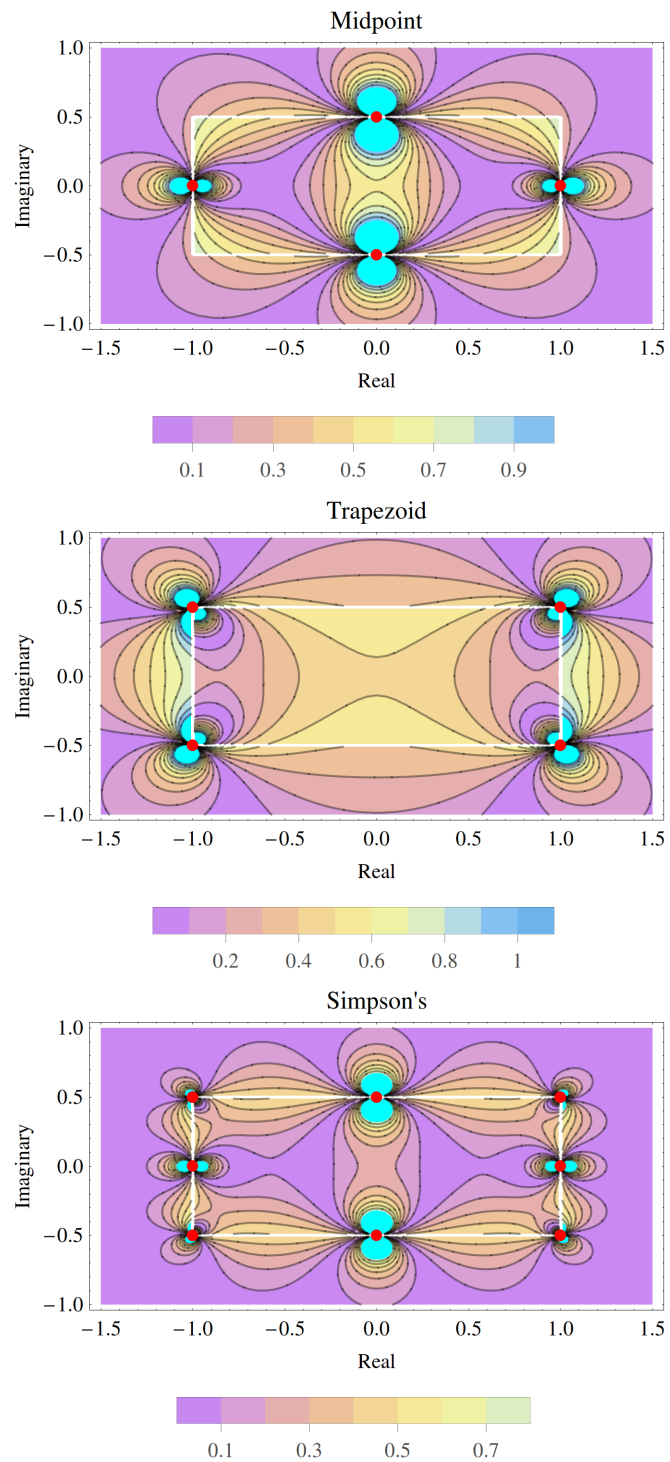


Figure 4.23: Comparison of quadrature error: Aspect Ratio = 0.5, complex eigenvalues, rectangle contour

4.5 Testing Quadrature on Diamond Contours

4.5.1 Real Poles

On the diamond Simpson's Rule gets more accurate results along the real axis than Trapezoid or Midpoint. Figure 4.24 demonstrates this. As with the rectangles, Midpoint is not good to use because there is no sharp transition of value from inside the contour to outside the contour. Even if Trapezoid is not accurate (about 0.3 instead of 1), it still is sharply contrasted with the value outside the contour, so we can still use this scheme when running our algorithm as an indicator of eigenvalues.

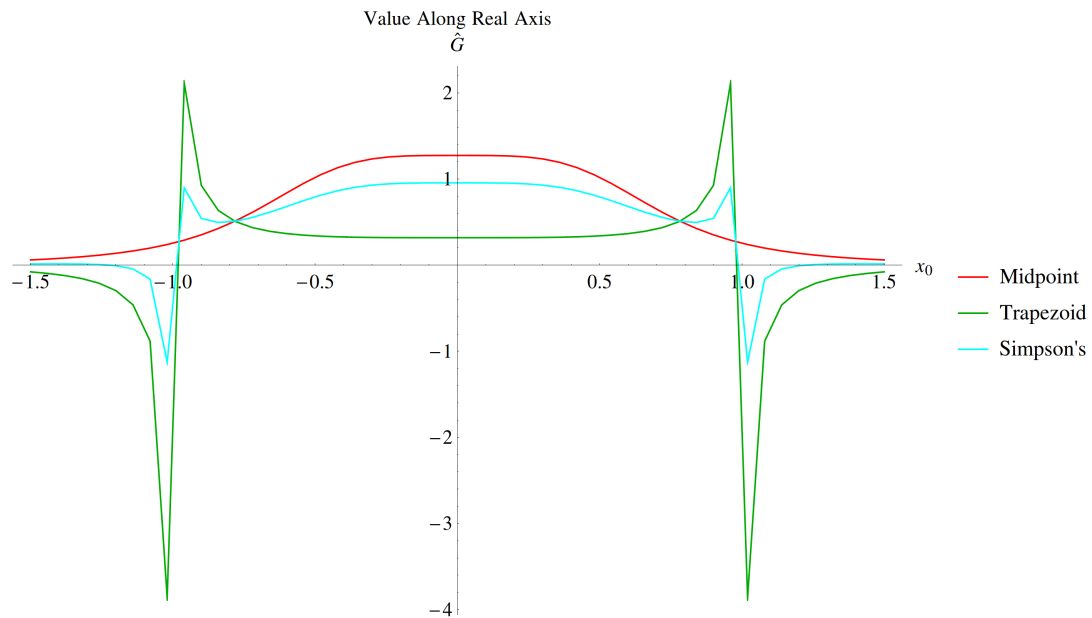


Figure 4.24: Quadrature comparison: Aspect Ratio = 1, real eigenvalues, diamond contour

Figure 4.25 shows that integrating over a diamond contour is more accurate than a rectangle: Simpson's Rule reaches the expected value quicker, and both Midpoint and Trapezoid hit the expected value at some point. Using $x_0 = 0$, Simpson's Rule gets near 1 quite quickly, as opposed to the rectangle where it did not reach 1. It appears that Midpoint gets the correct value for an aspect ratio around 0.45, while Trapezoid gets the correct value for an aspect ratio near 0.15, and Simpson's Rule for an aspect ratio near 0.6.

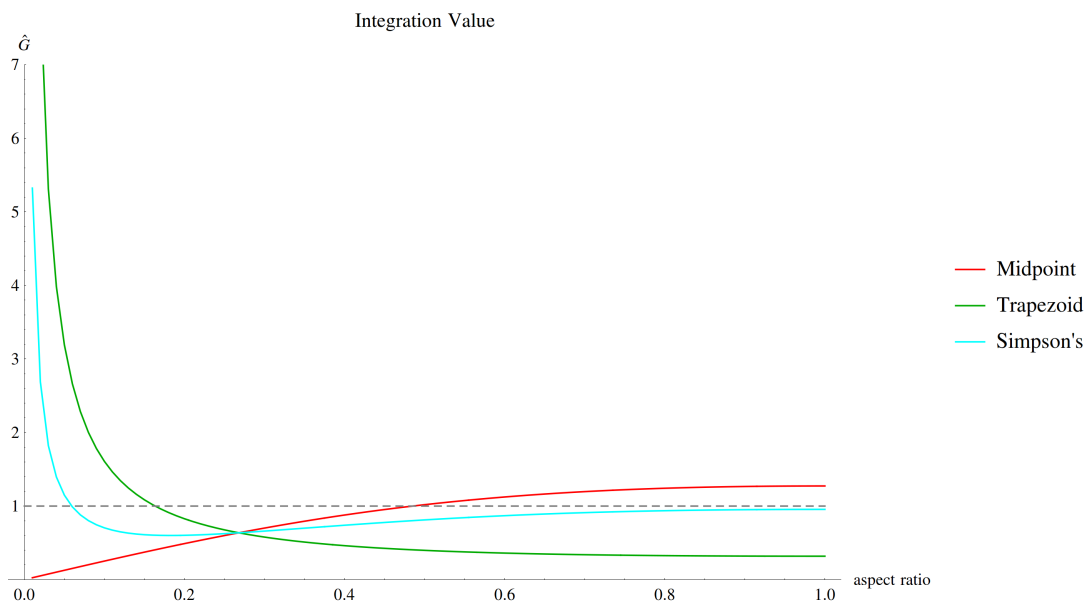


Figure 4.25: Comparisons of quadrature for different aspect ratios: real eigenvalues, diamond contour, $x_0 = 0$

Figure 4.26 shows the three quadrature schemes for an aspect ratio of 0.6. As expected from the plot over all aspect ratios, Simpson's Rule gets a quadrature value of 1 on the inside of the contour. This aspect ratio can still be used as an indicator of the presence of an eigenvalue inside the contour, as long as the scheme is Simpson's Rule or Trapezoid. Midpoint has a gradual decline from the inside to the outside which is not desirable. The

same comments apply for an aspect ratio of 0.45 as seen in Figure 4.27.

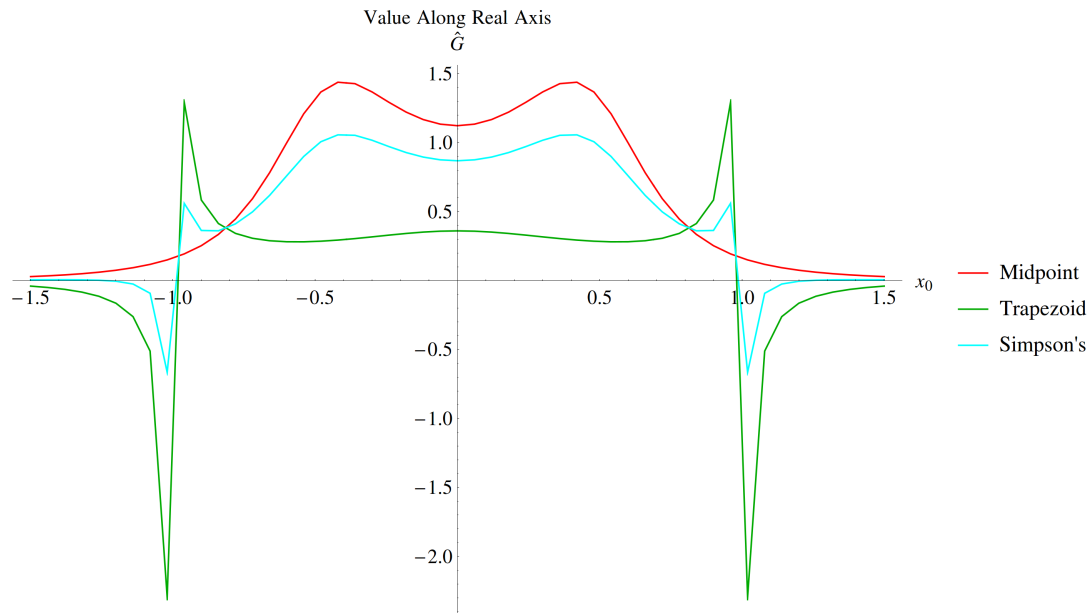


Figure 4.26: Quadrature comparison: Aspect Ratio = 0.6, real eigenvalues, diamond contour

At an aspect ratio of 0.15 as seen in Figure 4.28 Trapezoid achieves a value of 1 at the very middle of the contour, but we get further out it is near 0 which is not desirable as that would falsely indicate no eigenvalue is inside the contour. Midpoint still has the gradual transition problem, and Simpson's rule has this issue as well. This shows that this aspect ratio is not good to use, and that for our purposes we want to use something larger.

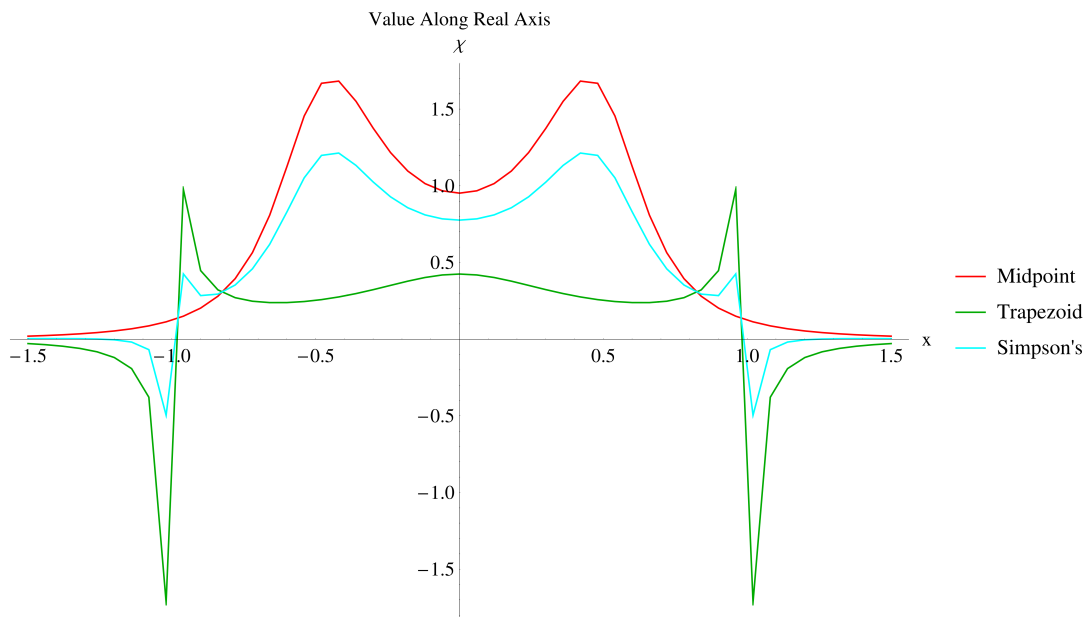


Figure 4.27: Quadrature comparison: Aspect Ratio = 0.45, real eigenvalues, diamond contour

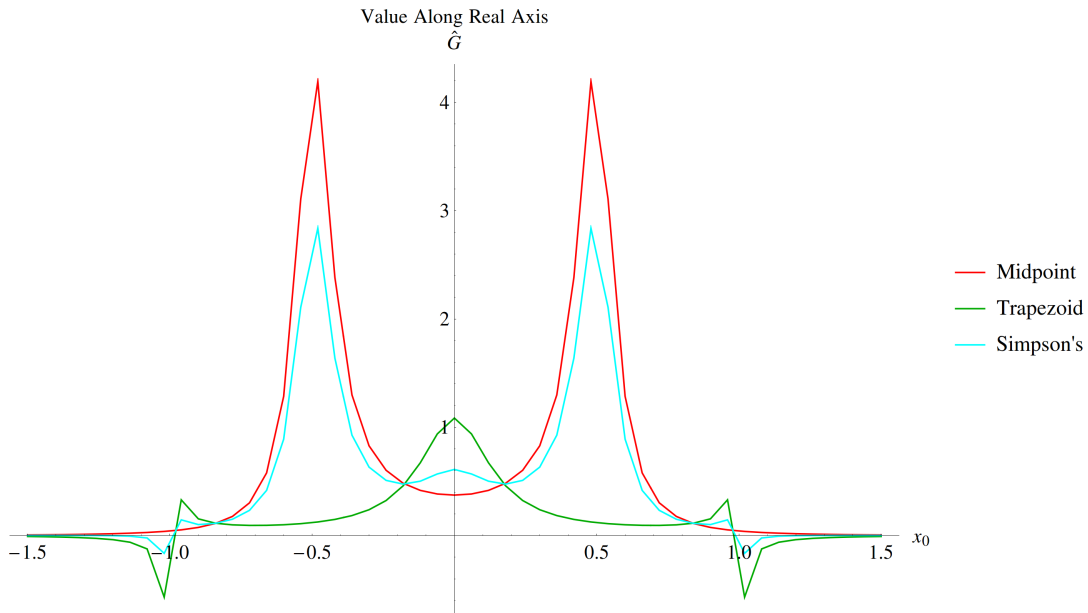


Figure 4.28: Quadrature comparison: Aspect Ratio = 0.15, real eigenvalues, diamond contour

4.5.2 Complex-Conjugate Poles

For complex-conjugate poles, Simpson's Rule again is the best quadrature scheme as it gets a smaller error on the inside of the contour. Figure 4.29 shows this. Trapezoid is definitely not good to use, as the error inside the contour is over 0.6. Additionally, the transition from inside to outside has a color change in the contour. Midpoint has that same problem, though at least there are a few areas inside the contour with an error below 0.1.

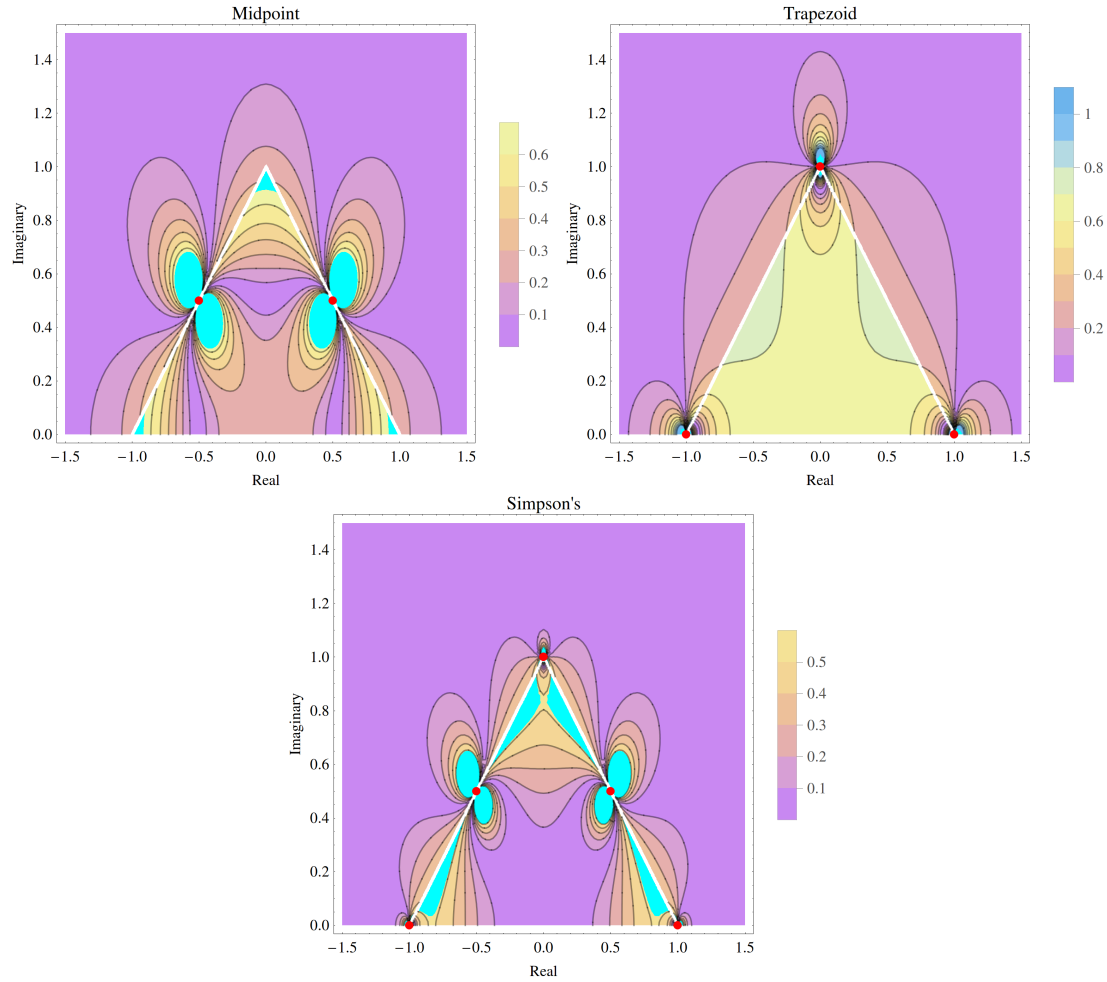


Figure 4.29: Comparison of quadrature error: Aspect Ratio = 1, complex-conjugate eigenvalues, diamond contour

4.5.3 Complex Poles

For the diamond, the error is again only under 0.1 when using Simpson's Rule. Figure 4.30 shows the resulting contours of the error. Midpoint does not have consistency in error when crossing the contour, but Trapezoid almost does. Simpson's Rule has consistence in error when transitioning from the inside to the outside, and the maximum error is 0.6 which is

much smaller than the other quadrature schemes.

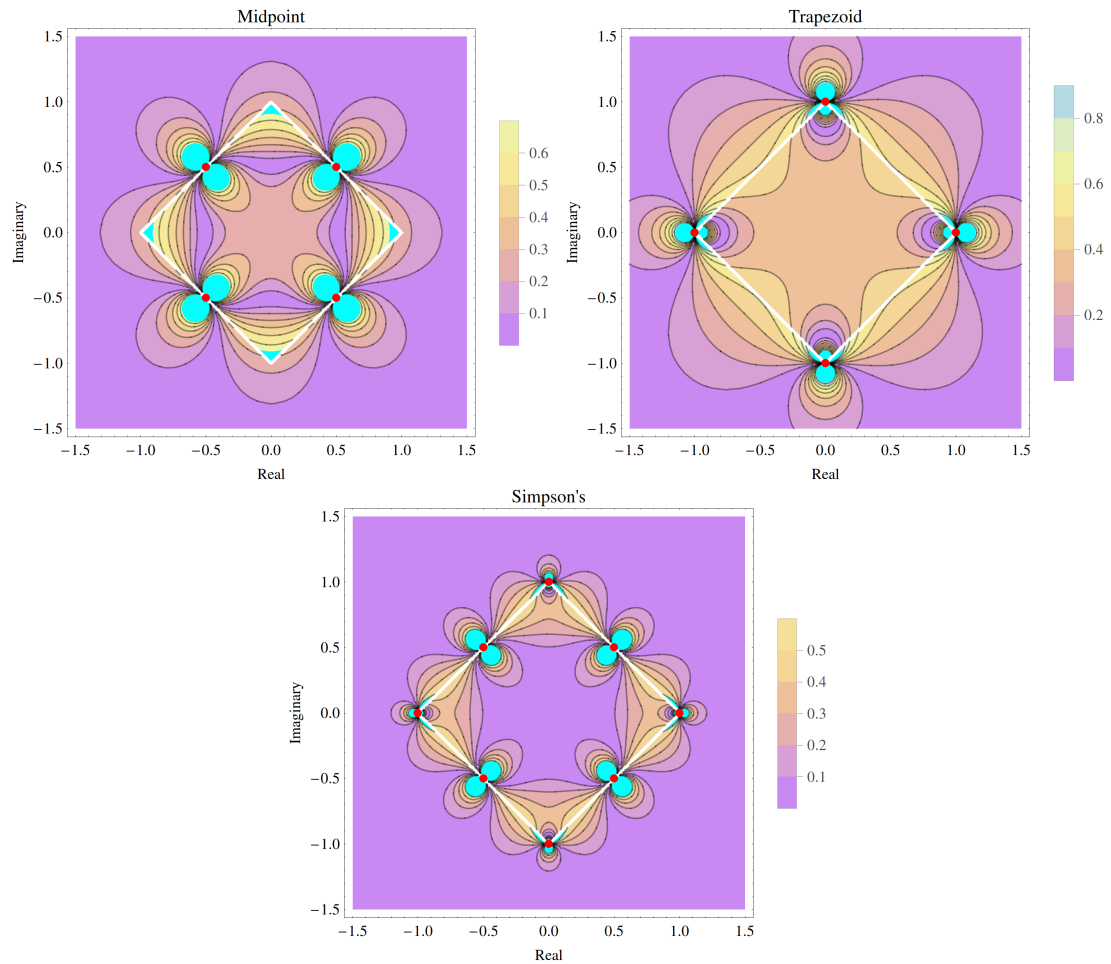


Figure 4.30: Comparison of quadrature error: Aspect Ratio = 1, complex eigenvalues, diamond contour

As with the rectangles, making the vertical distance smaller does not help the accuracy of the integration (see Figure 4.31). However, the advantage of the diamond over the rectangles is that for Simpson's Rule you can still get accuracy when you look at the middle of the interval. For the rectangle this was not the case because at zero there was influencing going on in the points above and below the real-axis.

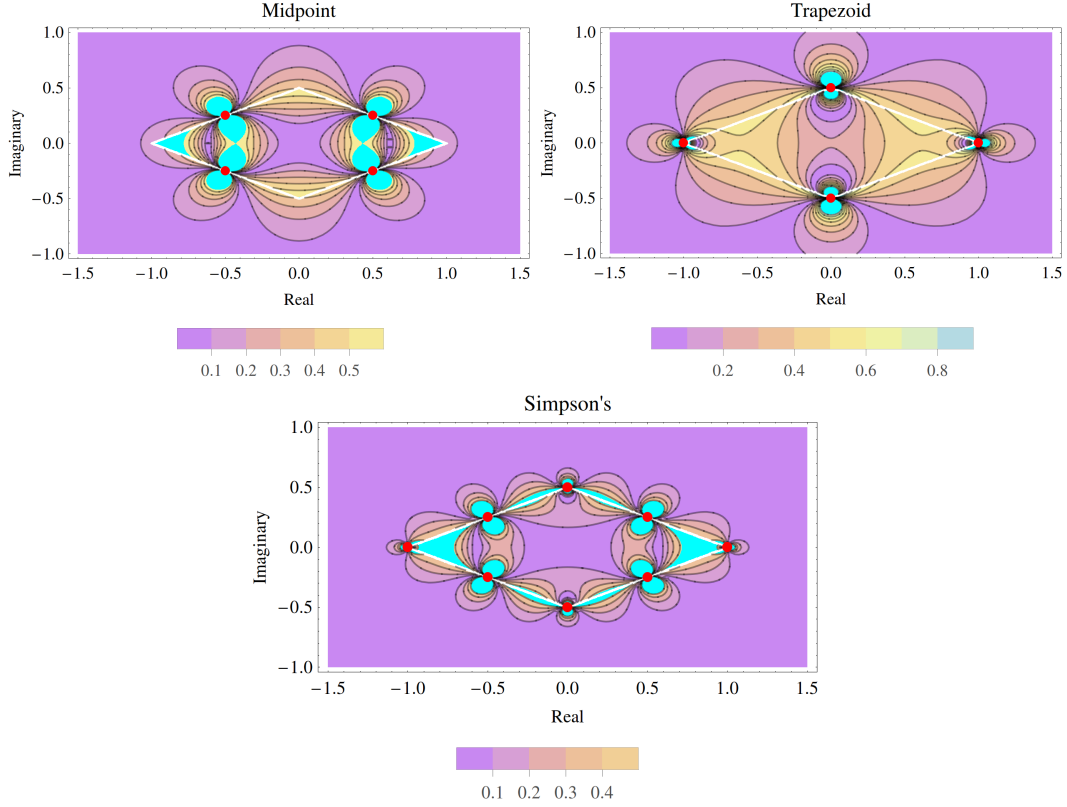


Figure 4.31: Comparison of quadrature error: Aspect Ratio = 0.5, complex eigenvalues, diamond contour

4.6 Conclusions

If you are looking for real eigenvalues, then when using the recursive FEAST as an indicator of eigenvalue locations circular contours are ideal. There is the sharpest contrast from inside to outside the contour when using circles, and the accuracy is better as well. One of the quadrature schemes for the circle, it could be argued to use Trapezoid, but any of the schemes seem to work well enough for the purposes of an indicator.

If it is known that you have complex-conjugate eigenvalues, then follow the same advice as for real eigenvalues: use circles, and any quadrature scheme.

When looking for complex eigenvalues, diamond contours using Simpson's Rule would be best because it has the smallest maximum error. However, circles work just as well as long as the aspect ratio is 1, and rectangles work as long as it is Simpson's Rule.

If your eigenvalues are both real and complex, follow the advice for when looking for complex eigenvalues.

No strict recommendations can be made because there are good results for all contours and various quadrature schemes. As long as the algorithm is run as a way to indicate eigenvalue locations there is flexibility in what schemes and contours to use.

Chapter 5

Conclusions

The FEAST algorithm as originally described by Eric Polizzi [7] works well as long as the number of eigenvalues in an interval are known, and matrices are only real symmetric or Hermitian. Modifying the algorithm to use matrices orthogonal to the quadrature matrix \hat{P} improves the algorithm in that fewer incorrect eigenvalues are found. However, the need to know how many eigenvalues are in the interval is not eliminated.

Creating a recursive variation on the algorithm eliminates the need to know how many eigenvalues are in the desired region.

The FEAST algorithm can be generalized to find complex eigenvalues in addition to real eigenvalues. The best contour for real eigenvalues and recursion is a diamond, as it can find any eigenvalue type accurately, unlike rectangles that work but not as well on real

eigenvalues.

When the purpose of the algorithm is to find location of eigenvalues, it really does not matter which contour shape or quadrature scheme is used. With the exception of Midpoint Rule on all shapes, and Trapezoid on the rectangles and diamonds, all other schemes tested work fine with the purpose of FEAST as an indicator. The choice then is up to the user as far as goals: is more accuracy preferred? Is it known whether the eigenvalues are real or complex? Do you want something that can be easily divided up for recursion and parallelizing purposes? The clear result of the research is that the FEAST algorithm can be extended and improved upon in a way that allows the user to have options in the coding details.

Further work can be done to modify the recursive FEAST algorithm to run in parallel on the GPU.

References

- [1] Howard Anton, Irl Bivens, and Stephen Davis. *Calculus: Early Trancendentals*. John Wiley & Sons, Inc., Hoboken, New Jersey, 7th edition, 2002.
- [2] Wolf-Jurgen Beyn. An integral method for solving nonlinear eigenvalue problems. *Linear Algebra and its Applications*, 436:3839–3863, 2012.
- [3] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.
- [4] Wolfram Research Inc. *Mathematica Edition: Version 9.0*. Wolfram Research, Inc., 2010.
- [5] Erwin Kreyszig. *Introductory Functional Analysis With Applications*. John Wiley & Sons, Inc., Windsor, Canada, 1978.
- [6] Tristan Needham. *Visual Complex Analysis*. Oxford University Press, New York, 1997.

- [7] Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79:115112–115117, 2009.
- [8] Eric Polizzi. *A High-Performance Numerical Library for Solving Eigenvalue Problems: FEAST Solver v2.0 User's Guide*, Mar 2012. <http://arxiv.org/abs/1203.4031>.
- [9] Steven Roman. *An Introduction to Linear Algebra with Applications*. CBS College Publishing, New York, 1985.
- [10] Ping Tak Peter Tang and Eric Polizzi. Subspace iteration with approximate spectral projection. Feb 2013. <http://arxiv.org/abs/1302.0432v2>.
- [11] Lloyd N. Trefethen and David Bau, III. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.
- [12] David S. Watkins. *Fundamentals of Matrix Computations*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2010.